ABSTRACT

SCHATZ, KARA MARIE. Domain- and Task-Aware Knowledge Graphs: From Construction to Completion. (Under the direction of Rada Chirkova).

Knowledge graphs are semantic networks that encode real world data as relationships (edges) between entities (nodes). In recent years, knowledge graphs have become popular in many domains because they can flexibly represent complex, heterogeneous data. Moreover, knowledge graphs can enable a variety of data analysis and knowledge discovery applications. Since domain scientists have routinely stored their data in other formats, e.g., relational databases, spreadsheets, figures, and text, there is a growing need for the construction and curation of domain knowledge graphs.

Challenges may arise during the construction and curation processes, as well as during analysis of the resulting graphs. For example, converting data in various formats to the knowledge graph format while maintaining semantic relationships both within and between data sets is not a trivial task. Moreover, automatic knowledge graph construction is inherently error-prone and calls for subsequent curation. Due to the sheer amount of available data, the sizes of modern knowledge graphs are growing rapidly, which has presented computational challenges to analytics and knowledge-discovery techniques. Additionally, knowledge graphs are inherently incomplete, which naturally leads to the challenge of knowledge graph completion: identifying which missing information is true and should be added to the graphs.

Towards addressing these challenges, this dissertation presents a knowledge-graph usability pipeline called *the four C's pipeline*. This pipeline takes domain scientists through four phases with their data: *construction*, *curation*, *compression*, and *completion*. Four domain-agnostic approaches are developed and discussed to implement this pipeline. The approaches accept inputs from domain scientists to enable appropriate handling and processing of the data. Together these approaches form a domain-agnostic, yet *domain-aware* pipeline for domain knowledge graphs. Beyond the work in this dissertation, future directions of research can be pursued to further improve the pipeline.

Domain- and Task-Aware Knowledge Graphs: From Construction to Completion

by
Kara Marie Schatz

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2024

APPROVED BY:

Christopher Healey	Steffen Heber
Alexander Tropsha	Rada Chirkova Chair of Advisory Committee

DEDICATION

To my parents, my sisters, and all others who have encouraged me to work towards my dream.

BIOGRAPHY

Kara Schatz completed her Bachelor's degree in Mathematics and Computer Science at Xavier University in May 2020. In August 2020, she joined the Department of Computer Science at North Carolina State University to pursue her Ph.D. Her research interests include knowledge graphs, knowledge discovery, data management, and data science. In particular, she is interested in increasing the usability of knowledge graphs and analytical methods that will assist domain scientists in their research.

ACKNOWLEDGEMENTS

I sincerely appreciate my advisor, Dr. Rada Chirkova, who has been an indispensable source of encouragement and guidance in both professional and personal matters. She has shown constant dedication to my growth as a person and an academic, for which I am incredibly grateful. Thank you for shaping me into a strong researcher, writer, and presenter, and for your constant advocacy.

I am very grateful for Dr. Alexander Tropsha, who in reality is my co-advisor despite an institutional technicality, for his unrelenting confidence in my abilities and his invaluable insights. Thank you for the biomedical expertise that gives my work purpose and impact.

I would also like to thank the other members of my committee, Dr. Christopher Healey and Dr. Steffen Heber, whose time, insights, thoughtful questions, and constructive feedback have been incredibly important for my dissertation.

I am grateful for my other co-authors for their crucial contributions to this work: Jon-Michael Beasley, Dr. Alexey Gulyuk, Dr. Pei-Yu Hou, Dr. Daniel Korn, Dr. Cleber Melo-Filho, and Dr. Yaroslava Yingling. Thank you also to my other colleagues Dr. Jing Ao, Ezio Mei, and Nahed Abu Zaid for their guidance, feedback, and collaboration.

I also thank Chris Bizon and the entire ARAGORN team at the Renaissance Computing Institute for their development of ROBOKOP and for countless insightful discussions.

I am deeply appreciative of my advisors and professors at Xavier University who turned me into a mathematician and a computer scientist, and who led me to pursue my Ph.D. Thanks also to the requirements of the mathematics degree at Xavier, without which I never would have been exposed to computer science.

Finally, I cannot adequately express my gratitude towards my parents, Mark and Leah; my sisters/role models/inspiration, Anna, Emma, Ella, and Nora; my partner, Anthony; and all my loved ones who have unknowingly inspired me to continue everyday.

This work was completed with support from the National Science Foundation under Grant No. CBET-2019435, from Research Triangle Institute International under Project No. 0282103.100.007, and from the National Institutes of Health under Grant No. OT2TR003441 and Grant No. GM146615.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

In recent years, knowledge graphs (KGs) have become a popular model for storing big data in many domains, as they provide an intuitive, semi-structured format for storing vast, heterogeneous data. KGs encode real-world data as relationships (edges) between entities (nodes). These data take the form of KG *subject-predicate-object* (s, p, o) *triples*. Here, the subject s and object o are specific KG nodes representing real-world entities, and the predicate p indicates the specific real-world relationship held between s and o. Knowledge graphs also provide mechanisms for annotating entities, predicates, and triples with desired metadata. For years, domain scientists have routinely stored their data in other formats, e.g., relational databases, spreadsheets, figures, and text. Now, realizing the strengths of the KG data model, there is a growing need for KG construction and curation efforts in the domain sciences.

The flexibility of the triple format makes KGs well suited for storing data in a variety of application domains; they also enable a variety of data analysis and knowledge discovery applications, see Noy et al. (2019); Sheth et al. (2019); Zou (2020) for overviews. For example, domain scientists in the biomedical field can use KGs to capture real-world knowledge about the various interactions between biomedical entities such as drugs and diseases. Then, they can use these KGs to support drug-repurposing efforts (Pushpakom et al. 2019), a problem of growing importance in today's society.

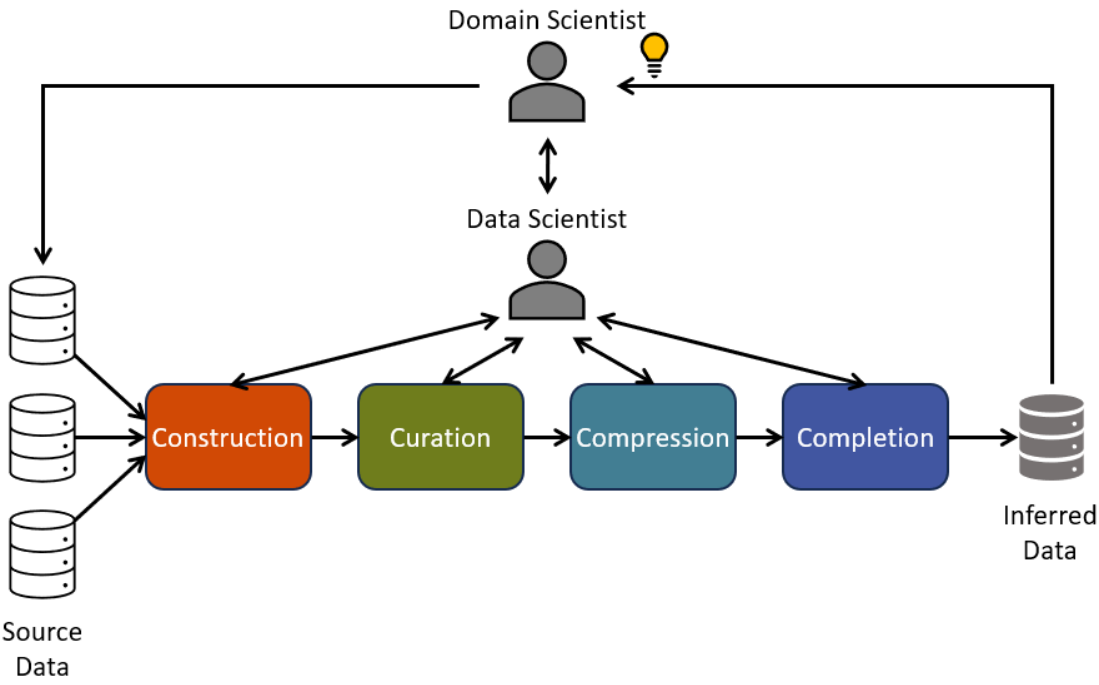


Figure 1.1: A pipeline for domain scientists seeking to transform their data into a knowledge graph and infer new knowledge from the transformed data. We refer to this as *the four C's pipeline* for *knowledge-graph construction*, *curation*, *compression*, and *completion*.

1.1 Motivation for This Dissertation

To enable domain scientists to leverage the advantages of knowledge graphs (KGs) for their data-analysis tasks, this dissertation aims to integrate disparate heterogeneous data from multiple sources into unified KGs that can potentially enable richer analytics not supported by the sources taken in isolation. Moreover, we aim to generate KGs that are domain- and task-aware, meaning that they appropriately handle and encode data according to the domain and task of interest. Overall, analytics on these KGs can potentially accelerate scientific discovery from data for domain scientists, thus contributing to their research progress.

1.2 Pipeline for Domain Scientists from Knowledge-Graph Construction to Completion

To employ knowledge graphs (KGs) for knowledge discovery and other data-analysis tasks, domain scientists can follow the pipeline illustrated in Figure 1.1. This pipeline takes domain scientists through *the four C's of knowledge graphs*: *construction*, *curation*, *compression*, and

completion. Due to the intricate nature of many data domains, domain-specific knowledge may be necessary (Sheth et al. 2019), so domain scientists can provide invaluable insights at each stage of this pipeline. Therefore, the pipeline would benefit from their involvement as humans-in-the-loop at all stages, so that the techniques used can reflect the insights, needs, and wishes of the domain scientists.

The first stage of the four C's pipeline is *construction*. In this stage, data from disparate sources can be integrated into a single, analytics-enabling KG. The data can range from structured to unstructured, and can come from various sources including experimental-data collections, spreadsheets, reports, web pages, and even other KGs (Sheth et al. 2019). KG construction is a challenging task (Kejriwal 2019; Wu et al. 2019), and in many cases, different data formats and data sources must be handled individually to ensure proper ingestion of the data. Moreover, the data semantics must be taken into account during translation, so that the KG data will accurately capture the data present in the disparate sources. Insights from domain scientists as humans-in-the-loop can be extremely beneficial for maintaining the data semantics. In addition, the data must be ingested in such a way that the appropriate connections between different sources could be made. These connections can provide insights to domain scientists beyond what would be elucidated in the disparate sources, making the integrated KG an invaluable resource for their analyses. For example, researchers who have a common goal, but are conducting different experiments, may want to integrate their data such that connections between their experimental results will be highlighted for corroboration.

The second stage of the four C's pipeline is *curation*. In this stage, the constructed KG is curated for the domain- and task-specific needs of domain scientists. Since automatic KG construction is inherently error-prone (Sheth et al. 2019), this stage is crucial for ensuring the usability of the KG. First, the KG is transformed into a format convenient for use by domain scientists, e.g., a KG management system. Depending upon software compatibility, KG conversion to the desired format may be nontrivial. Next, the KG must be enriched with any metadata that is desired by domain scientists for their analyses. After all, one of the advantages of the KG format is its seamless storage of metadata, which can be attributed to specific nodes and edges in the graph. Examples of useful metadata include knowledge (data) sources, synonyms for entity or predicate names, predicate qualifiers, and supporting evidence for KG relationships. This stage of the pipeline is also the appropriate time for domain scientists and KG users to decide who needs to be given access to the KG and how access will be granted, e.g., whether all collaborators will access a universal copy or download a copy for personal use. Providing public access to the KG can accelerate knowledge discovery, as many teams may be able to analyze data that may have not been available otherwise (Arzberger et al. 2004).

The third stage of the four C's pipeline is *compression*. In this stage, the curated KG is

compressed to enable more efficient downstream KG analyses. Although the KG format is well suited for data-analytics and knowledge-discovery approaches, see, e.g., Ahmadi et al. (2020); Lajus et al. (2020); Sadeghian et al. (2019); Wang et al. (2017); and Wang and Li (2015), the sizes of modern KGs are growing rapidly, which unfortunately leads to costly downstream analytics with respect to both time and space. Thus, KG compression has become an indispensable technique for managing KG sizes and the costs of analysis. Compressed KGs, typically referred to as KG summaries, can be lossy or lossless and can be tailored to specific use cases, see, e.g., Huang and Lai (2006); Jin and Koutra (2017); Jin et al. (2019); Koutra et al. (2014); Sacenti et al. (2022); Safavi et al. (2019); Song et al. (2016, 2018); Tran et al. (2020); and Yu et al. (2021). Analysts must take care to apply a KG-compression technique suitable for their needs. For example, lossy compression may impact the accuracy of downstream analyses, and personalized, targeted KG summaries may not be useful for diverse analyses.

The fourth and final stage of the four C's pipeline is *completion*. In this stage, the KG is enriched with inferred triples, i.e., triples previously missing from the KG for which evidence is found (Chen et al. 2020; Shen et al. 2022). These triples may have been previously known, but were from sources beyond those used in the *construction* and *curation* stages, or these triples may not have been previously known at all, i.e., they are newly discovered knowledge. In either case, explainable approaches to KG completion, see Kotonya and Toni (2020a) and Paulheim (2017) for overviews, are useful for domain scientists, as they provide reasoning for the inferred triples, enabling experts to assess the quality of the inferences or conduct further testing. For example, experts working in drug repurposing may choose to conduct clinical trials to test the accuracy of inferred *drug-treats-disease* relationships that have strong supporting explanations. This stage results in high-quality inferred data that can provide invaluable insights and propel future research for domain scientists.

1.3 Challenges

The four C's pipeline shown in Figure 1.1 can be beneficial for domain scientists seeking to use knowledge graphs (KGs) for knowledge discovery and other data-analytics tasks. Unfortunately, in real-world scenarios, these tasks are nontrivial, and domain scientists can face challenges accomplishing them on their own. In many cases, domain scientists could take advantage of tools that ensure proper handling of their domain data, i.e., according to the data semantics, at each stage of the pipeline. In this section we list challenges that may arise and that our proposed pipeline allows us to address.

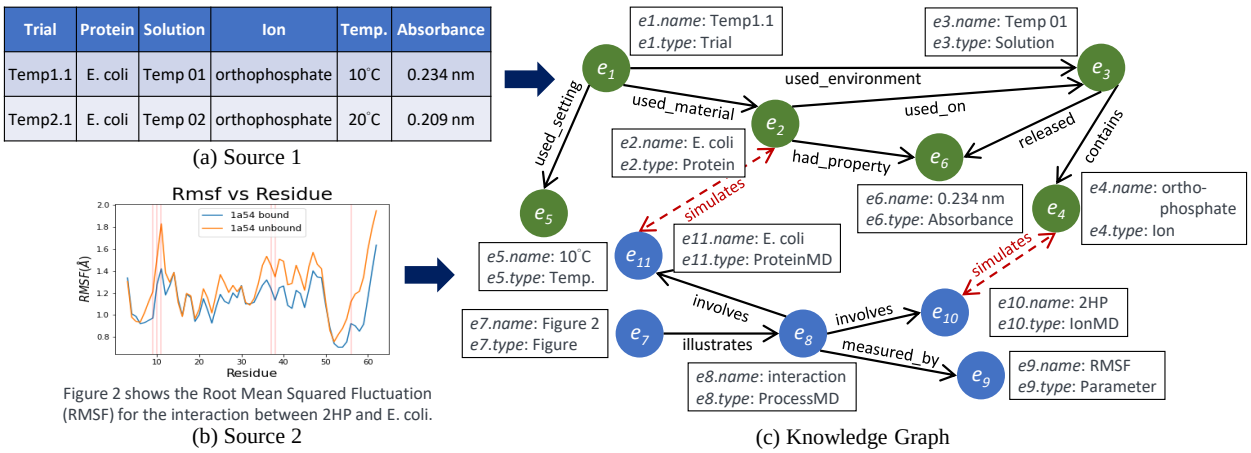


Figure 1.2: The motivating example for the knowledge graph construction stage of the four C's pipeline, showcasing a scenario in which data of different types from multiple sources (a)-(b) need to be converted into a single unified knowledge graph (c). Source 1 (a) provides *spreadsheet data*, while Source 2 (b) provides *images* and *regularized text* data. A knowledge graph capturing and connecting these data is shown in (c). The dashed edges in (c) provide examples of relationships between entities across the data sources.

1.3.1 Challenges for Knowledge-Graph Construction

While KGs have been on the rise, and many domain scientists are beginning to recognize their strengths, beginning to use KGs for data storage is not a trivial task. At the same time, domain data are in many cases routinely stored in other formats, e.g., as spreadsheets, text, or figures. While such formats can be familiar and intuitive to users, storing the same data in KGs can open the door to more complex types of analysis, such as rule mining and machine learning. Moreover, combining heterogeneous data from multiple sources into a single unified KG could lead to even richer analytics not supported by the sources taken in isolation. As such, the KG format can be preferable to other data-storage formats in many scenarios. Converting domain data to the KG format for the benefit of domain scientists is the focus of the first stage of the four C's pipeline: *construction*.

Consider a motivating example arising from a use case that we work with in Chapter 4 of this dissertation. Figures 1.2(a)-(b) show fragments of the large-scale data coming from two materials-science teams working in the Science and Technologies for Phosphorus Sustainability (STEPS) Center.[1] While both teams study interactions between phosphate-binding proteins *(PBPs)* and phosphate ions in solvents, they do not use the exact same materials, nor do they use the same experimental procedures or settings. Moreover, they do not use the same storage format for their data: The first team stores their data as *spreadsheets* in (Data) Source 1, see

[1] `https://steps-center.org/`

Figure 1.2(a), while the second team stores data as *regularized text* and *images* in Source 2, see Figure 1.2(b).

The research teams would like to improve the utilization of their large-scale scientific and experimental data, by integrating the data into a single unified KG. Figure 1.2(c) shows one such possible KG, which would allow the researchers to accelerate scientific discovery compared to what could be supported by their isolated source data. The desired integration process would involve conversion of the heterogeneous source data into the KG format. It would also involve combining the resulting KG fragments in a way that would ensure overlap in the shared entities, with potential addition of extra connections across the converted sources, see the dashed edges in Figure 1.2(c).

Integrating their data into a KG might not be trivial for the research teams to accomplish on their own. Further, adding the extra connections across the converted sources might be a challenge in case the teams are not very familiar with each other's projects. These issues could make it difficult for unassisted domain scientists to integrate their data effectively into the unified KG format that could enable richer analytics.

Given enough time and resources, domain scientists could certainly solve their KG-integration problem in a one-off way for their particular purpose. At the same time, their being able to use instead a predefined automated workflow for integrating heterogeneous data from multiple sources into a single unified KG could significantly alleviate the time and resource burden, while potentially resulting in higher-quality KG data conducive to accelerating scientific discovery. Ideally, such a workflow would allow the scientists to input their data in familiar formats and to control the KG ontology. Furthermore, it would output an integrated KG that would reflect the data-analysis expectations of the scientists, allowing them to make adjustments as needed in the integration process.

To the best of our knowledge, most KG-construction approaches are not analogous to ours, as their efforts focus only on textual data, see, e.g., Bosselut et al. (2019); Martinez-Rodriguez et al. (2018); and Wu et al. (2019), and/or are domain-specific, see, e.g., Chen et al. (2018); Elhammadi et al. (2020); Hao et al. (2021); Huang et al. (2020); Li et al. (2020a,b); Luan et al. (2018); and Wang et al. (2018). The approach of Asprino et al. (2023), which converts data in multiple formats to the KG format, is domain independent, but does not directly involve domain experts as humans-in-the-loop. The KG-construction procedure of Zhang et al. (2023), which also accepts non-textual data inputs from users and is applicable to multiple domains, is complementary to our work, as it only supports data in the JSON and audio formats.

1.3.2 Challenges for Knowledge-Graph Curation

Unfortunately, even when constructed with domain-scientist needs in mind, KGs are often not immediately usable in their provided form, which may lead to vast amounts of their data being underexplored and underutilized. Especially for research groups conducting a wide variety of research tasks, certain KG tasks may have specific format or data requirements that must be resolved after construction. Three common features that potentially make KGs unusable are: (1) the format in which they are provided, (2) the presence of unreadable or missing data, and (3) the public accessibility of the KGs. Resolving these usability issues is the focus of the second stage of the four C's pipeline: *curation*.

To better understand and address these issues, we now consider the three features in detail. The issues that arise due to the provided KG format are related to the convenience of use and the user's ability to modify the data. Not all KGs are publicly available; those that are publicly available are typically provided in one of two formats: (i) a graph data-management system (DBMS) endpoint, e.g., ROBOKOP (Bizon et al. 2019) and Hetionet (Himmelstein et al. 2017), or as (ii) a series of plain-text files containing the KG triples and, optionally, the metadata, e.g., the biomedical KG called Drug Repurposing Knowledge Graph *(DRKG)* (Ioannidis et al. 2020). Advantages of DBMS endpoints include convenience of data access via graph query languages. However, users may want to also modify the data for their use case for many reasons, e.g., extracting subsets that meet their needs, fixing errors that they find, adding their own metadata, or adding inferred or missing information. In this light, a disadvantage of using DBMS endpoints is that users have little or no freedom to modify the data due to it being a public copy, which restricts them to dealing with the KG in its provided state.

The plain-text-file format of a KG has exactly opposite advantages and disadvantages. That is, the data are inconvenient to access due to the lack of query support, so they may not be immediately usable. However, the (user's personal copy of the) KG is modifiable and integrable with existing systems. A possible alternative to plain-text files is for KG sources to provide downloadable KG database dump files, so that users could easily load up a personal copy in their own DMBS. Unfortunately, this approach can also run into issues, as database dump files from one DBMS may not be compatible with other DBMSs, or even other versions of the same DBMS. Thus, KGs in the plain-text-file format are the most universally useful, as they can be modified or integrated into existing systems if users so choose, and users can import them into their preferred DMBS. With access to a DBMS endpoint, such plain-text files can be generated by systematically querying the entire KG. (Note that that would add to the initial setup overhead.) Whether plain-text files are provided to the users or generated by them, the users must address the issue of the inconvenient data access before they are able to use the KG.

Second, numerous issues can arise due to unreadable or missing KG data. Typically, KGs are

built via automated or semi-automated extraction of structured, semi-structured, or unstructured data from a variety of sources, e.g., from web pages or from other databases, including other KGs (Sheth et al. 2019). While automatic construction facilitates quick creation of large KGs, it is an inherently imperfect process that is likely to result in unknown data errors (Sheth et al. 2019). Some of these errors are negligible, e.g., missing spaces between words or trimming words improperly, while others hinder the usability of the KG, e.g., not decoding text or importing acronyms without their corresponding full text. An even more troublesome issue is that some KGs could be missing valuable metadata that was either not extracted or available from the data sources during the KG construction. In some cases, missing metadata can prevent users from productively using the KGs. For example, if entities are missing their standard names, then query results may be uninterpretable. Perhaps more troubling is when entities or triples are missing source information, as this makes query results unsupported and possibly unconvincing, since the data cannot be tracked or verified (Weikum 2021). If important metadata is not provided, users may spend significant amounts of time manually identifying information that is necessary for their use case, or may even have to abandon the use of the KG altogether. Due to the potential imperfection of KG data, KG refinement has become a popular research topic (Paulheim 2017); however, most approaches focus on adding missing KG triples, rather than missing metadata. One of the many advantages of using KGs is their seamless storage of metadata attributed to the appropriate nodes and edges, but when the metadata is missing or error ridden, it can greatly limit the value of the KG. Thus, it is crucial for users to be able to fix unreadable data and add missing data, especially as it may be necessary for their domain and task.

Finally, public access to KGs (or lack thereof) raises issues for KG use and collaboration. Constructing a new KG, or even just cleaning an available KG, is a challenging, arduous task (Kejriwal 2019; Wu et al. 2019). Therefore, providing public access to curated and cleaned KGs can drastically reduce construction and cleaning efforts, or even eliminate them altogether. Providing public access to the KGs can also accelerate knowledge discovery, as it allows many teams to analyze data that may have not been available otherwise (Arzberger et al. 2004). Moreover, it can lead to synergy between different data-analytics projects. Indeed, with many teams using the same data, their results and findings may align with, or build off of, each other. Overall, universal access to data is important for data analytics and knowledge discovery. Unfortunately, many KGs are not publicly available, which limits the options that users have when searching for data sets. Even in the case of sensitive or confidential KG data that cannot be shared publicly, researchers are likely still interested in granting access within their teams and to their collaborators. Thus, making the graph accessible to multiple users is still of interest in this case.

To the best of our knowledge, no previous KG work is analogous to ours. One project (Huaman and Fensel 2021) that is tangentially related to our work has similar goals of improving the quality of KGs. The KG-curation framework of Huaman and Fensel (2021) aims to clean and enrich KGs with respect to the accuracy of the KG data, unlike our work, which assumes that the KG data are already accurate. While our work also aims to clean and enrich KGs, the focus is specifically on preparing them for domain- and task-relevant applications.

1.3.3 Challenges for Knowledge-Graph Compression

The flexible triple format for representing data makes KGs well suited for data-analytics and knowledge-discovery approaches in domains where the data being analyzed are necessarily complex, see, e.g., Ahmadi et al. (2020); Lajus et al. (2020); Sadeghian et al. (2019); Wang et al. (2017); and Wang and Li (2015). At the same time, the growing sizes of KGs and the irregularity of the data in domains such as bioinformatics and healthcare have presented challenges to scaling analytics and knowledge-discovery methods to such contexts, as the data are typically too large and complex to be processed efficiently. The computational burden on the methods can be alleviated by making the approaches more scalable and/or by reducing the sizes of the KGs being analyzed. As data reduction is becoming an indispensable technique in many domains, the latter path, that is, improving the efficiency of KG analytics via compressing the graphs in question, is the focus of the third stage of the four C's pipeline: *compression*.

Consider a motivating example, which has given rise to the bioinformatics use case with which we work in this dissertation. In *drug repurposing* (Pushpakom et al. 2019), biomedical researchers aim to identify new uses for existing drugs. This is traditionally a lengthy and labor-intensive process, in which generating specific "drug-treats-disease" (treatment) hypotheses and verifying them via clinical trials are both time-consuming components. Rule mining of biomedical KGs has been used as a way of speeding up the hypothesis-generation stage of drug repurposing, see Schatz et al. (2021). Unfortunately, rule mining for drug repurposing cannot scale up to extremely large-scale KGs such as ROBOKOP[2] (Bizon et al. 2019). In such cases, reducing the size of the KG could make a difference in whether rule mining on the KG could be used for hypothesis generation in drug repurposing.

To generate treatment hypotheses for drug repurposing, rule mining applied to a biomedical KG has to process multiple regions of the KG data, with some regions describing well-studied biomedical entities and others describing less-explored entities, e.g., data for common diseases as opposed to data for rare diseases, with the latter being an important focus for drug-repurposing studies (Fetro and Scherman 2020; Roessler et al. 2021). Existing KG-

[2] `https://robokop.renci.org`

summarization approaches for data reduction in KGs, see Bonifati et al. (2020); Čebirić et al. (2019); and Liu et al. (2018) for overviews, aim to generate concise and meaningful representative summaries of the original KG data whose use can improve the rule-mining efficiency. At the same time, information in biomedical KGs about underexplored entities, including rare diseases, could be lost in the process of summarizing the graphs using such approaches in the scope of our motivating example. As a result, the important goal of identifying treatments for such diseases might not be attainable if inference rules were mined on such graph summaries, rather than on the original KGs.

To the best of our knowledge, among the KG-summarization techniques that have been developed, see, e.g., Huang and Lai (2006); Jin and Koutra (2017); Jin et al. (2019); Koutra et al. (2014); Sacenti et al. (2022); Safavi et al. (2019); Song et al. (2016, 2018); Tran et al. (2020); and Yu et al. (2021), many have been designed for a specific purpose or application, so the summary KGs may not be a good fit for other purposes. Further, existing KG-summarization techniques may be prone to loss of important information, which may be undesirable for downstream analytics tasks such as rule mining. On the other hand, lossless KG summarization would be ideal for applications such as these, because the summary KGs could ensure efficiency while also being effective.

1.3.4 Challenges for Knowledge-Graph Completion

Knowledge-graph completion (Chen et al. 2020; Shen et al. 2022) is the process of adding new information to KGs. The new information can be information available from other sources but not present in the KG (*known unknowns*) or information previously unknown by any sources (*unknown unknowns*); the latter enables knowledge discovery from data. Explainable fact-checking (Kotonya and Toni 2020a) and link prediction (Paulheim 2017) are approaches for KG completion that focus on adding new data- and reasoning-supporting information to the existing data, while also providing explanations, i.e., other data that support the truth of the new information. Interest in these approaches has been growing in part due to the rise of large-scale domain KGs. With KGs being inherently incomplete, explainable fact-checking and link prediction can be used to address the problem of inferring new domain *facts* in the form of new explanation-supported edges in the KG. Adding such edges to enrich KGs is the focus of the fourth and final stage of the four C's pipeline: *completion*.

In this work we consider KG completion in the context of it supporting drug discovery as an impactful use case, though the problem and the approach that we introduce are of general appeal. Drug discovery is an area of biomedical research and development that aims to identify potential new treatments for diseases (Hughes et al. 2011). This process has two

broad phases: (1) identification of drug candidates, and (2) clinical trials of the candidates to test their viability. Two major approaches to the drug-identification step are (i) painstaking synthesis and development of new chemical entities with the desired bioactivity, and (ii) *drug repurposing* (Pushpakom et al. 2019). The latter efforts make use of existing knowledge about drugs, such as their experimental characteristics and known mechanisms of action, which can be retrieved from KGs.

Currently, biomedical experts spend significant effort on generating and evaluating candidate *drug-treats-disease* hypotheses. The reason is, the amount of information to process is overwhelming, with many aspects to consider, such as a drug's potential side effects, a multitude of complex biological processes that involve the drug, or the degree to which the drug manages the disease or its symptoms. Automating the hypothesis-generation and evaluation processes with fact-checking tools would enable generation of otherwise unreachable hypotheses and facilitate innovation in the presence of modern large-scale biomedical KGs, while alleviating expert efforts.

In this work, we consider the problem of generating viable *drug-treats-disease* hypotheses for drug repurposing, by leveraging information available from KGs via explainable fact-checking, a popular technique for KG completion. That is, we are interested in extracting from the KG *explanations* for the hypotheses, i.e., KG facts supporting the hypotheses, and in then evaluating the truth of the hypotheses in order to identify the strongest drug candidates. In this work we use several publicly available KGs that store information usable in drug repurposing, namely ROBOKOP (Bizon et al. 2019), DRKG (Ioannidis et al. 2020), and Hetionet (Himmelstein et al. 2017).

To the best of our knowledge, most state-of-the-art fact-checking and link-prediction tools (Ciampaglia et al. 2015; Gad-Elrab et al. 2019; Popat et al. 2017; Raedt et al. 2007; Shiralkar et al. 2017; Shu et al. 2019) have not been developed with the biomedical domain in mind. In fact, Nakov et al. (2021) observes that fact-checking research lacks domain-user feedback, but asserts that such feedback would certainly be useful. As a result, although drug-repurposing hypothesis generation is an impactful data-mining application, it cannot be accomplished with current data-mining tools. First, the explanations produced by the tools that we have studied are in formats that are not readily understood by biomedical experts. Second, these tools return explanations that are based on specific entities represented in KGs. In drug repurposing, where large numbers of hypotheses are typically considered, this approach does not scale well. Further, quality metrics used in some existing explanation-evaluation tools make assumptions that do not necessarily hold in the biomedical domain, see Section 2.4.2 for the details. Some metrics also evaluate explanation derivations as opposed to evaluating the explanations, and provide evaluations that are hypothesis specific (local) without considering the other KG data

(global).

To the best of our knowledge, only one study (Kotonya and Toni 2020b) published thus far has addressed biomedical hypothesis checking. We find that the method of Kotonya and Toni (2020b) faces some of the challenges discussed above. For example, their explanations are paragraphs, which take longer to understand than KG patterns and are hypothesis specific. In addition, their explanation-evaluating metrics provide local evaluations that do not consider other domain evidence.

1.4 Our Contributions

In this dissertation, we propose domain-agnostic approaches to address the needs of domain scientists and the challenges they face during the four stages of the knowledge-graph (KG) usability pipeline. To enable appropriate processing of domain-specific data, the approaches accept inputs from domain scientists that provide protocols for semantically correct handling and processing of the data. Therefore, together these approaches comprise a domain-agnostic, yet *domain-aware* pipeline for domain KGs from construction to completion.

1.4.1 Integrating Heterogeneous Data Into Analytics-Enabling Knowledge Graphs

To address the *construction* stage of the four C's pipeline, in this dissertation we propose a domain-agnostic workflow called *BUILD-KG* for integrating heterogeneous scientific and experimental data from multiple sources into a single unified KG that can potentially enable richer analytics, see Chapter 4. By design, BUILD-KG is broadly applicable, accepting input data in popular structured and unstructured storage formats. To enable appropriate processing of domain-specific data, it accepts inputs from domain scientists regarding the semantics and handling of the data, in an effort to ensure that the resulting KG will be accurate and useful for their needs. This makes BUILD-KG *domain agnostic* and *domain aware* at the same time. Moreover, the workflow is designed to be carried out with end users as humans-in-the-loop. In this way, BUILD-KG enables domain scientists to facilitate the KG construction and verify that the end result will align with their expectations, potentially enabling acceleration of scientific discovery.

Moreover, we have implemented the BUILD-KG workflow, and provide details on our experience with applying it to scientific and experimental research data from the STEPS Center,[3] with STEPS researchers involved as humans-in-the-loop. The STEPS Center is comprised of

[3] `https://steps-center.org/`

several research teams united by the common theme of studying phosphorus sustainability. As such, they are particularly interested in the insights that may be provided by the "junction points" (relationships) between their independently collected data. Thus, their data serve as an ideal test case for our proposed workflow.

Our specific contributions are as follows:

- We propose a domain-agnostic, human-in-the-loop workflow called BUILD-KG to construct KGs from structured and unstructured data according to domain experts' specifications, which makes BUILD-KG domain aware;
- Within BUILD-KG, we introduce a collection of conversion procedures for three popular data types: spreadsheet data, images with annotations, and regularized text data;
- We propose a BUILD-KG methodology for combining multiple heterogeneous data sets into a unified KG;
- We outline our implementation of the BUILD-KG workflow, and report on our experiences with applying it to scientific and experimental STEPS-center data; and
- We report on our experiences working with STEPS researchers, and provide tips on involving domain scientists as humans-in-the-loop in the BUILD-KG workflow.

1.4.2 Workflow for Domain- and Task-Sensitive Curation of Knowledge Graphs

To address the *curation* stage of the four C's pipeline, in this dissertation we introduce a workflow for domain- and task-sensitive curation of KGs called *Knowledge Graph Curation for Usability, Readability, and Accessibility for Task Expectations (KG-CURATE)*, see Chapter 5. The proposed workflow is conducive to resolving issues that arise with respect to the three features discussed in Section 1.3.2 by: (1) transferring KG data from plain-text files into a queryable graph DBMS, (2) finding and extracting necessary metadata, and then adding it to the KG, and (3) making the resulting KG publicly available via cloud services. Additionally, the workflow includes a phase for data cleaning to ensure that the KG and metadata are understandable and readable for users. Because domain- and task-specific needs directly impact the quality and usability of KGs (Sheth et al. 2019), the proposed workflow is a human-in-the-loop workflow involving domain experts and anticipated users of the KG. These stakeholders can provide invaluable guidance into the workflow, to ensure that the resulting KG fits the needs of the domain and task at hand.

Moreover, we have implemented the KG-CURATE workflow, and provide details on our experience with applying it in the biomedical domain to the Drug Repurposing Knowledge Graph (DRKG) (Ioannidis et al. 2020). DRKG was built for machine-learning analysis; the graph is represented via plain-text files, and is not publicly available in a queryable graph DBMS. Additionally, it is missing some potentially valuable information for other use cases, e.g., entity names. Thus, DRKG serves as a perfect test case for our proposed workflow.

Our specific contributions are as follows:

- We propose a domain-agnostic, human-in-the-loop workflow called KG-CURATE to curate KGs in a domain- and task-sensitive manner, according to domain experts' and anticipated users' specifications;
- Within KG-CURATE, we address issues with (1) the format in which KGs are provided, (2) the presence of unreadable or missing data in KGs, and (3) the public accessibility of the KGs;
- We propose a KG-CURATE methodology for verifying the readability and understandability of the KG data;
- We discuss our implementation of the KG-CURATE workflow, and report on our experiences with applying it to the biomedical KG called DRKG; and
- We outline the involvement of domain scientists and anticipated users as humans-in-the-loop in the KG-CURATE workflow, and report on our experiences working with biomedical data and biomedical researchers.

1.4.3 Lossless Knowledge-Graph Compression for Improving the Efficiency and Effectiveness of Rule Mining

To address the *compression* stage of the four C's pipeline, in this dissertation we introduce a domain- and application-independent approach called GAME+: Graph Abstraction for Mining Efficiency, see Chapter 6, that produces a compressed abstract KG and serves as a preprocessing step in preparing a KG for a range of analytics tasks. The goals of the GAME+ approach in generating an abstract (summarized) KG from the given KG of interest are twofold: (i) to *improve the efficiency* of downstream KG-analytics, specifically rule mining, compared to the original KG by effectively reducing the KG size,[4] while (ii) preserving all the information from the input KG in the summarized KG, to ensure *effectiveness* of the downstream analytics. That is, the

[4] In the extreme, the sizes of the KGs output by GAME+ could enable analytics even in cases in which the input KGs are too large for the purpose.

GAME+ approach would provide a *lossless* summary KG. In the context of rule mining, which is our particular focus, these goals translate into generating an abstract KG on which inference rules can be mined much faster than on the original KG, while providing *losslessness* of the mined rules. We define losslessness for rule mining as the rule-mining process being able to obtain from the abstract KG generated by GAME+ all of the inference rules that can be mined from the original input KG.

Intuitively, GAME+ first clusters nodes based on a given similarity-scoring function for KG nodes, and then fuses the nodes in each cluster together, along with their incident edges, such that the KG data sizes are reduced via elimination of repetitive KG triples. Together, the two stages of KG summarization in the GAME+ approach aim to address the challenges of scaling rule mining to big-graph data in the presence of data irregularity, while delivering on the lossless preservation of the rules that can be mined from the summarized KG output. In fact, our experimental results suggest that our proposed KG-abstraction approach can substantially reduce the sizes of the original KGs, leading to much faster performance of rule mining on the resulting abstract KGs, while also enabling mining on the abstract KGs of all of the inference rules that can be obtained from the original KGs.

Our specific contributions are as follows:

- We formalize the problem of lossless KG abstraction (summarization) for downstream analytics, with a particular focus on rule mining;
- We introduce a domain- and application-independent approach called GAME+, which generates a reduced KG that can be lossless for rule mining while improving the rule-mining efficiency;
- We prove that our KG abstraction approach is lossless by demonstrating that the original KG can be recovered from the abstract KG;
- We present the results of a case study performed by a biomedical expert, which suggest that the node-clustering phase of the proposed approach can cluster nodes that are meaningfully related and that the resulting clusters can be insightful to biomedical experts; and
- We report experimental results on three large-scale biomedical KGs (DRKG (Ioannidis et al. 2020), Hetionet (Himmelstein et al. 2017), and ROBOKOP (Bizon et al. 2019)); the results indicate that the proposed approach is promising for generating reduced KGs that are lossless for efficient rule mining; in particular, we perform data reductions at various summarization levels, to understand the trade-off between the improvements in mining efficiency and the effectiveness of the rule-mining outcomes.

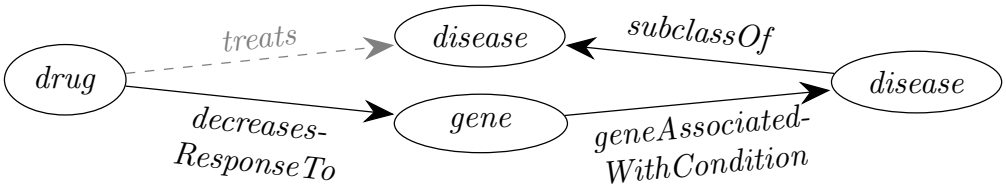


Figure 1.3: A knowledge-graph pattern (nodes linked by solid edges) that serves as an explanation for a drug-disease relationship (dashed edge), as well as a clinical outcome pathway (Korn et al. 2022) revealing to biomedical experts the mechanistic explanation as to how the drug acts on the disease. (Each p-labeled edge between nodes x and y represents the triple (x, p, y).)

1.4.4 Using Extraction and Evaluation of Explanations for Drug Repurposing on Knowledge Graphs

To address the *completion* stage of the four C's pipeline, in this dissertation we propose a scalable automated approach called KGEG: Knowledge Graph Explanation and Evaluation Generation, see Chapter 7, that extracts and evaluates supporting explanations for inferred KG triples. To address the challenges faced when adopting existing KG-completion tools for drug repurposing, our proposed approach generates explanations based on facts supporting the given hypothesis, that is, explanations that are modeled after existing biomedical concepts and supplemented with data-supported evaluation metrics. As its explanation format, our approach uses *knowledge-graph (KG) patterns*, i.e., graphs composed of entity- and predicate-type labels rather than specific entities, see Figure 1.3 for an example. In the context of fact-checking, KG patterns serve as explanatory pathways that link the subject and object of the given hypothesis, providing evidence of the relationship between them. Biomedical experts find KG patterns easy to understand, as they are closely related to the concept of *clinical outcome pathways (COPs)* (Korn et al. 2022), defined as sequences of biological interactions that explain mechanisms of drug action, see Figure 1.3 for an illustration.

In addition to being easy for experts to understand, KG patterns are general enough to apply to multiple hypotheses. As a result, we can treat KG patterns as inference rules for `drug-treats-disease` relationships, thus avoiding the time-consuming generation of explanations specific to each hypothesis. Moreover, our experimental results suggest that our KGEG approach is efficient and tunable, and that it can be applied to relationships beyond `drug-treats-disease`. We also introduce metrics that consider the existing KG data as evidence for or against each explanation, thus providing global rather than local evaluations. As such, we posit that our collaboration with domain experts in the context of this study can be a step towards the human-in-the-loop ideal suggested in Nakov et al. (2021).

Our specific contributions are as follows:

- We pose drug-repurposing hypothesis generation as a fact-checking problem and propose an explainable automated fact-checking solution;
- We introduce knowledge-graph (KG) patterns as a novel explanation format, with an algorithm for deriving them for any KG hypothesis;
- We treat KG-pattern-based explanations as inference rules that are applicable to multiple hypotheses once derived;
- We introduce three data-supported scoring metrics for reliable evaluation of explanations for KG hypotheses;
- We present the results of two biomedical-expert user studies, which suggest that our explanations can be understandable and reasonable to experts working in the field of drug repurposing, thus validating the design of our explanations; and
- Our experimental results on the biomedical KGs ROBOKOP (Bizon et al. 2019), DRKG (Ioannidis et al. 2020), and Hetionet (Himmelstein et al. 2017) suggest that our proposed metrics can be accurate and useful on large-scale biomedical KGs. We also demonstrate the efficiency, extensibility, and tunability of our explanation-extraction KGEG approach, and show that it can be a reliable hypothesis classifier.

1.5 Dissertation Outline

This dissertation aims to provide a domain-agnostic, yet domain-aware approach for handling each stage of the four C's pipeline. The rest of this dissertation is organized as follows. In Chapter 2, we review related work for each stage of the pipeline, and in Chapter 3, we provide the necessary preliminaries. In Chapter 4, we present our proposed domain-agnostic, human-in-the-loop workflow BUILD-KG for addressing the problem of integrating heterogeneous data into analytics-enabling KGs, which would be used in the first pipeline stage: *construction*. In Chapter 5, we present our proposed KG-CURATE workflow for domain- and task-sensitive curation of KGs, which would be used in the second pipeline stage: *curation*. In Chapter 6, we present our proposed knowledge graph abstraction approach GAME+ for compressing KGs to improve the feasibility of analytics, which would be used in the third pipeline stage: *compression*. In Chapter 7, we present our proposed KGEG approach for extracting and evaluating explanations for inferred KG triples, which would be used in the fourth pipeline stage:

completion. Finally, in Chapter 8, we summarize the contributions of this dissertation and discuss future directions for potentially extending this work.

CHAPTER

2

RELATED WORK

This chapter presents a survey of related work in the following directions: knowledge graph construction (Section 2.1), knowledge graph curation (Section 2.2), knowledge graph compression (Section 2.3), and knowledge graph completion via fact-checking explanation derivation and evaluation (Section 2.4).

2.1 Knowledge Graph Construction

In this dissertation, the problem of integrating heterogeneous structured and unstructured data into analytics-enabling knowledge graphs, addressed in Chapter 4, is most closely related to the topic of knowledge graph (KG) construction, see, e.g., Asprino et al. (2023); Kejriwal (2019); Wu et al. (2019); Martinez-Rodriguez et al. (2018); Bosselut et al. (2019); Ye et al. (2022); Zhang et al. (2023). KG construction is a complex process, with approaches ranging from fully manual to semi-automatic to fully automatic. Since fully manual construction is not scalable and fully-automated construction is error prone, most KG-construction approaches, including the one proposed in this dissertation, are semi-automatic.

The most common format of source data in KG construction is text, with many approaches, such as Bosselut et al. (2019); Martinez-Rodriguez et al. (2018); Wu et al. (2019), supporting textual inputs only. These approaches rely on machine learning, most commonly natural-

language processing *(NLP)*, to extract data. In contrast, our BUILD-KG workflow is specifically designed to handle a variety of input types, including some of the most popular data-storage formats, so that domain scientists can use BUILD-KG to convert their data to the KG format while continuing their regular data collection. The domain-independent approach of Asprino et al. (2023), which also converts multiple data formats to the KG format, does not directly involve domain experts, so it is unclear how much control they would have over the handling of their data. Moreover, Asprino et al. (2023) does not discuss how overlapping portions of disparate data are handled; in contrast, our approach covers this scenario, as these "junction points" can be crucial for domain scientists in their KG exploration. The KG-construction procedure of Zhang et al. (2023), which also accepts non-textual data inputs from users and is applicable to multiple domains, is complementary to our work, as it only supports data in the JSON and audio formats.

Domain-specific KG construction has become popular due to the nuances present in domain-specific data (Kejriwal 2019). Domain-specific KGs have been generated in many domains, including geosciences (Wang et al. 2018), education (Chen et al. 2018), science (Luan et al. 2018), medical records (Li et al. 2020b), e-commerce (Li et al. 2020a), power-grid equipment (Huang et al. 2020), finances (Elhammadi et al. 2020), and surveying and remote sensing (Hao et al. 2021). In contrast, while our use case is in the materials-science *(MS)* domain, our proposed BUILD-KG workflow is entirely domain agnostic. Moreover, our workflow allows scientists in a variety of domains to input their existing data and easily specify their desired KG ontology, which makes our approach domain aware, while at the same time enabling richer analytics on the output KGs.

To handle textual data, our BUILD-KG workflow includes an NLP component that performs named entity recognition *(NER)* (Nadeau and Sekine 2007). Despite the recent advances in pretrained NER models, most are trained on common-sense and common-knowledge corpora, see, e.g., WikiBERT (Pyysalo et al. 2021) and bert-base-NER (Devlin et al. 2019; Tjong Kim Sang and De Meulder 2003). Such general models may not always perform well on domain-specific texts. Pretrained language models have also been built in some scientific domains, e.g., Lee et al. (2020); Sung et al. (2022) in the biomedical domain and Walker et al. (2021); Gupta et al. (2022) in the MS domain. However, in our MS use case, these MS pretrained language models cannot achieve a high prediction accuracy, due to the variety in language used by scientists in different subdomains of MS. Thus, in our proposed BUILD-KG workflow, to complete the NER step we use the NLP tool `flair` (Akbik et al. 2019), as it allows users to easily build their own language model.

2.2 Knowledge Graph Curation

In this dissertation, we introduce the problem of domain- and task-sensitive curation of knowledge graphs, addressed in Chapter 5. To the best of our knowledge, there are no other projects analogous to ours that focus specifically on knowledge graphs (KGs). The most closely related work is that of Huaman and Fensel (2021), which proposes a framework for KG curation, and has similar goals of improving KG quality. The framework focuses on cleaning and enriching KGs with respect to the accuracy of the KG data. In contrast, our work assumes that the KG data are already accurate, and instead focuses on cleaning and enriching KGs specifically to prepare them for domain- and task-relevant applications. Therefore, the problem and framework of Huaman and Fensel (2021) are only tangentially related to our work.

Some other related topics include KG construction and KG refinement. For a discussion of KG construction techniques, see Section 2.1. Our proposed workflow for KG curation is intended for use on already constructed KGs. KG refinement (Paulheim 2017), sometimes called KG link prediction, focuses on improving the coverage and correctness of KGs, i.e., adding missing data and/or removing incorrect data. For KG refinement, "data" refers specifically to KG triples. In contrast, our proposed workflow for KG curation adds entity- and triple-metadata, rather than KG triples. Moreover, in our curation work we assume that the given KG is clean with respect to the KG triples, and only has formatting or readability issues, which are handled by our proposed workflow. For a review of link prediction approaches, see Section 2.4.1.

2.3 Knowledge Graph Compression

In this dissertation, the problem of lossless knowledge graph (KG) abstraction (compression), addressed in Chapter 6, is closely related to KG summarization work (Čebirić et al. 2019; Liu et al. 2018; Bonifati et al. 2020), which has received much attention in recent years. KG-summarization approaches have been developed for purposes including graph visualization (Huang and Lai 2006), graph cleaning (Tran et al. 2020), recommender systems (Sacenti et al. 2022), efficient query evaluation (Song et al. 2016, 2018), other graph exploration (Jin and Koutra 2017; Jin et al. 2019; Yu et al. 2021), and KG-size reduction (Koutra et al. 2014; Jin et al. 2019). There have also been efforts towards generating domain-specific graph summaries (Jin and Koutra 2017) and personalized graph summaries for users (Safavi et al. 2019). Unfortunately, many KG-summarization approaches may result in the loss of valuable information for KG analytics such as rule mining. In contrast, our proposed approach can enable effective KG rule mining due to its losslessness.

The KG-summarization approaches most similar to ours are those of Huang and Lai (2006);

Tran et al. (2020). The approach of Huang and Lai (2006) introduces the concepts of abstract nodes and abstract edges, which are similar to the concepts of consolidated nodes and consolidated edges that we introduce in Section 3.2. (We opt for the term "consolidated," as it better captures the purpose of these nodes and edges in our proposed approach.) In addition, the approach of Huang and Lai (2006) uses the same node-similarity function as the one considered in this dissertation, see Section 6.3.1.3. At the same time, the summarization approach of Huang and Lai (2006) applies to KGs whose edge labels are all the same, unlike our approach, which is applicable to KGs that can have any number of different edge labels. In addition, the approach of Huang and Lai (2006) uses the common k-means algorithm for node clustering, whereas we plan to introduce our own node-clustering algorithms. Two key challenges arise when using k-means for KG summarization: It relies both on predefined numbers of clusters and predefined seed nodes. As appropriate values for these parameters are difficult to determine in many scenarios, our proposed node-clustering approaches do not rely on such parameters, see Section 6.2.2. Moreover, the approach of Huang and Lai (2006) specifically targets improving KG visualization, whereas we aim to reduce the size of the KG for downstream analytical tasks, such as rule mining.

The approach of Tran et al. (2020) also generates abstract KGs, however, the goal of their work is to perform graph cleaning, specifically, consistency checking, which is accomplished via their abstract KGs. Because the approach of Tran et al. (2020) focuses on resolving ontological inconsistencies in the data, their abstract KGs rise entirely to the ontological level and do not preserve instance information. While such an abstraction is sufficient for consistency checking, it is insufficient for many other downstream analysis tasks, such as the task of rule mining, which is our focus in this dissertation.

2.4 Knowledge Graph Completion

In this dissertation we address the problem of knowledge graph (KG) completion via explainable hypothesis generation in Chapter 7, with a particular focus on the biomedical domain as our use case. This problem consists of two major components: explanation derivation and explanation evaluation. In Sections 2.4.1–2.4.2, we explore the related work in these areas, respectively.

2.4.1 Explanation Derivation

Explanation generation in fact checking and link prediction has recently received significant attention in the field of knowledge discovery, see, e.g., Popat et al. (2017, 2018); Shu et al. (2019); Yang et al. (2019); Lu and Li (2020); Wu et al. (2020); Atanasova et al. (2020); Kotonya and

Toni (2020b); Raedt et al. (2007); Ahmadi et al. (2019); Gad-Elrab et al. (2019); Lin et al. (2018); Shiralkar et al. (2017); Ciampaglia et al. (2015); Shi and Weninger (2016); Huang et al. (2022); Vedula and Parthasarathy (2021); Samarinas et al. (2021). Arguably, the most popular approach to explanation generation is evaluation of textual sources via deep learning, where a wide variety of models have been used, e.g., LSTMs (Popat et al. 2018; Yang et al. 2019; Huang et al. 2022; Vedula and Parthasarathy 2021), GRUs (Lu and Li 2020; Shu et al. 2019), CRFs (Popat et al. 2017), decision trees (Wu et al. 2020), and BERT-based systems (Atanasova et al. 2020; Kotonya and Toni 2020b; Samarinas et al. 2021).

Rule-based approaches for finding hypothesis-supporting paths or subgraphs in KGs (Raedt et al. 2007; Ahmadi et al. 2019; Gad-Elrab et al. 2019; Lin et al. 2018), as well as other approaches for finding KG paths (Shiralkar et al. 2017; Ciampaglia et al. 2015; Shi and Weninger 2016), are also popular. The approaches of Shiralkar et al. (2017); Ciampaglia et al. (2015); Shi and Weninger (2016) are restricted to KG paths, whereas our approach makes no restriction on the type of subgraphs that can serve as explanations. In addition, Raedt et al. (2007) requires enumeration of all known facts and rules, making it effectively intractable on large-scale KGs.

Backward chaining is a standard method in artificial intelligence (Russell and Norvig 2021) used in many applications to form proofs by combining inference rules. We use it in our approach to find explanations over biomedical KGs for drug-repurposing hypotheses. The fact-checking approach of Gad-Elrab et al. (2019) adopts backward chaining in a different application domain; their criteria for acceptable explanations, including (i) small explanation size, (ii) using small collections of rules for explanation generation, and (iii) using both KG facts and text facts, turn out to not be applicable to our case. Indeed, as we do not use text sources, criterion (iii) does not apply. In regard to (i) and (ii), as confirmed by biomedical experts on our team, longer and more detailed explanations can often be stronger in drug repurposing, since they might describe better the biological processes and relevant entity interactions; our key criterion is that explanations be biomedically accurate and reasonable.

In addition to the variety of derivation methods, the knowledge sources used also vary. Some projects use text-based sources, such as text corpora (Gad-Elrab et al. 2019; Vedula and Parthasarathy 2021; Samarinas et al. 2021), web articles (Popat et al. 2018, 2017; Ahmadi et al. 2019), social-media posts (Lu and Li 2020; Wu et al. 2020), or news data (Shu et al. 2019; Yang et al. 2019); others use KGs (Ahmadi et al. 2019; Gad-Elrab et al. 2019; Shiralkar et al. 2017; Ciampaglia et al. 2015; Shi and Weninger 2016; Vedula and Parthasarathy 2021). We use KGs due to their data being already structured and often well curated, making them cleaner and more reliable.

Additionally, the derived explanations can take many forms, including text summaries (Atanasova et al. 2020; Kotonya and Toni 2020b; Vedula and Parthasarathy 2021), article clips

(Popat et al. 2017; Samarinas et al. 2021), highlighted words in clips (Popat et al. 2018; Lu and Li 2020; Wu et al. 2020; Shu et al. 2019; Yang et al. 2019), sets of inference rules and facts (Raedt et al. 2007; Ahmadi et al. 2019; Gad-Elrab et al. 2019; Lin et al. 2018), and KG paths or subgraphs (Shiralkar et al. 2017; Ciampaglia et al. 2015; Shi and Weninger 2016; Gad-Elrab et al. 2019). Our explanations take the form of KG patterns; to the best of our knowledge, we are the first to consider this explanation format. In particular, this is the only explanation format that can be used to explain other hypotheses beyond those for which the explanations were generated.

For a discussion of Kotonya and Toni (2020b), please see Section 1.4.4.

2.4.2 Explanation Evaluation

Some explanation-derivation methods come with metrics for evaluating the explanations that they produce, see, e.g., Kotonya and Toni (2020b); Atanasova et al. (2020); Raedt et al. (2007); Gad-Elrab et al. (2019); Ciampaglia et al. (2015); Shiralkar et al. (2017); Shi and Weninger (2016).

Metrics assessing text summaries evaluate the textual content locally (Kotonya and Toni 2020b; Atanasova et al. 2020), whereas our proposed metrics incorporate other data for a global evaluation. Metrics evaluating KG paths focus on node degree (Ciampaglia et al. 2015), feature vectors (Shi and Weninger 2016), and relational similarity (Shiralkar et al. 2017), and evaluate specific path instances. In contrast, we focus on KG pattern prevalence and co-occurrence with the predicate of the candidate hypothesis, enabling an evaluation without a specific pattern instance.

Although the fact-checking approach of Nakashole and Mitchell (2014) does not produce explicit explanations, its metric for evaluating the truth of a fact assigns objectivity scores to supporting documents that contribute to a candidate fact's believability score. We use KGs and evaluate KG patterns, as opposed to documents; however, we consider the goals of the approach of Nakashole and Mitchell (2014) to be related to ours.

Two existing metrics (Raedt et al. 2007; Gad-Elrab et al. 2019) can be adapted to scoring explanations in our proposed format, see Section 7.3.2.3. Among them, Raedt et al. (2007) scores explanations by multiplying the weights of the rules used, relying upon the assumption of their independence. This assumption is not met in the biomedical domain and possibly others. The approach of Gad-Elrab et al. (2019) uses a heuristic explanation-scoring metric called confidence, which is based on the number and order of the rules required to generate the explanation pattern. The underlying assumption is that each individual rule can be used for perfectly reliable inference. This assumption does not necessarily apply to the biomedical domain and possibly others.

Further, the metrics of Raedt et al. (2007); Gad-Elrab et al. (2019) inherently assign lower

scores as the patterns get larger; however, biomedical experts on our team have confirmed that larger explanation patterns are often stronger due to their increased specificity. In addition, the metrics of Raedt et al. (2007); Gad-Elrab et al. (2019) focus on evaluating the process of generating an explanation, rather than the actual explanation itself. A side effect is that the same explanation can receive different scores if it can be derived in different ways. In contrast, our proposed metrics provide data-supported evaluations of explanations regardless of how they were derived, while not making domain-dependent assumptions.

CHAPTER

3

PRELIMINARIES

This chapter presents necessary preliminaries for this dissertation.

3.1 Knowledge Graphs

We define a *knowledge graph (KG)* $\mathcal{G}$ as a 5-tuple $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, where $\mathcal{N}$ is the set of *entities (nodes)*, $\mathcal{T}$ is the set of *entity-types*, $\phi : \mathcal{N} \to \mathcal{T}$ is the *entity-type labeling function*, $\mathcal{P}$ is the set of *predicates (edge labels)*, and $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{P} \times \mathcal{N}$ is the set of *triples (edges)*. Each triple $(s, p, o) \in \mathcal{L}$ represents the real-world fact that the *subject* s has a *relationship of type* p with the *object* o. For example, in the biomedical domain, the triple *(metformin*, `treats`, *type 2 diabetes)* represents the real-world fact that "the drug *metformin* `treats` the disease *type 2 diabetes*," and in the materials science domain, the triple (*Solution 01*, `contains`, *orthophosphate*) represents the fact that "the solution with ID 01 contains orthophosphate."

3.2 Abstract Knowledge Graphs

We define an *abstract knowledge graph (AKG)* $\mathcal{G}^A = (\mathcal{N}^A, \mathcal{T}^A, \phi^A, \mathcal{P}^A, \mathcal{L}^A)$ as a KG in which an entity $e \in \mathcal{N}^A$ or triple $(s, p, o) \in \mathcal{L}^A$ can semantically represent a group of entities or triples.

We call these groups *consolidated entities (nodes)* and *consolidated triples (edges),* respectively. For example, the disease node *diabetes* in an AKG may semantically represent three specific diseases: *type 1 diabetes, type 2 diabetes,* and *gestational diabetes*. Furthermore, the triple (*diabetes*, `subclass_of`, *pancreatic disorder*) in an AKG may semantically represent all of the following triples: (1) (*type 1 diabetes*, `subclass_of`, *pancreatic disorder*), (2) (*type 2 diabetes*, `subclass_of`, *pancreatic disorder*), and (3) (*gestational diabetes*, `subclass_of`, *pancreatic disorder*). In an AKG, the specific instances of a consolidated entity or triple are not present, as they are already represented by the consolidated entity or triple.

3.3 Metapaths

We define a *metapath* M of a KG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$ as an ordered list of triple types $M = \{(e_0, p_1, e_1), (e_1, p_2, e_2), \ldots, (e_{n-1}, p_n, e_n)\}$. A *triple type* is a triple (s, p, o), where $p \in \mathcal{P}$ and s, o are variables. That is, a triple type is a triple consisting of variables instead of distinct entities from $\mathcal{N}$. The number of triple types in a metapath is referred to as the *length* of the metapath. An example of a metapath M of length 2 from the ROBOKOP[1] KG is

$$\{(x, \texttt{affects\_activity\_of}, y), (y, \texttt{gene\_associated\_with\_condition}, z)\}.$$

A metapath of an AKG $\mathcal{G}^A$ is defined analogously.

3.4 Inference Rules

We define *inference rules* as *Horn rules*, which are comprised of *atoms*. In the case of KG Horn rules, atoms take the form of (s, p, o) triples, where $p \in \mathcal{P}$ and s, o are either variables or specific entities from $\mathcal{N}$. A Horn rule r consists of a single *head* atom H and a set of *body* atoms $\{B_1, B_2, \ldots, B_n\}$ and is denoted by

$$r : H \Leftarrow B_1 \wedge B_2 \wedge \cdots \wedge B_n. \tag{3.1}$$

The semantics of such a Horn rule are that the body of the rule, $body(r)$, may imply the head of the rule, $head(r)$. In this way, Horn rules generated via KG rule mining are used for a variety of tasks, e.g., link prediction or KG refinement. The number of atoms in the body of a rule is referred to as the *length* of the rule.

[1] `https://robokop.renci.org`

An example of a Horn rule r of length 2 from the ROBOKOP[1] KG is

$$r : (x, \texttt{treats}, z) \Leftarrow (x, \texttt{affects\_activity\_of}, y) \\ \wedge (y, \texttt{gene\_associated\_with\_condition}, z).$$

This rule captures the biomedical reasoning that a drug x may treat a disease z if x affects the activity of a gene y associated with the condition z.

3.5 Explanations

We define *explanations* as Horn rules of the form

$$E : (e_0, p, e_1) \Leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_n. \tag{3.2}$$

In an explanation E of this form, the *explanation pattern* $A = \{a_1, \ldots, a_n\}$ is a set of atoms using variables for their components s and o, p is the *target predicate*, and e_0, e_1 are variables used in A. We say that the explanation pattern A in E *explains* the triple (e_0, p, e_1). Consider an example:

$$E : (e_0, \mathit{treats}, e_1) \Leftarrow (e_0, \mathit{treats}, e_2) \wedge (e_3, \mathit{biomarkerFor}, e_2) \wedge (e_3, \mathit{biomarkerFor}, e_1).$$

An *entity-typed explanation* E' is an explanation E that has been enriched with entity-type specifications for each entity. E.g., an entity-typed version of the above explanation E is

$$\begin{aligned} E' : (e_0 : \mathit{drug}, \mathit{treats}, e_1 : \mathit{disease}) \Leftarrow & (e_0 : \mathit{drug}, \mathit{treats}, e_2 : \mathit{disease}) \wedge \\ & (e_3 : \mathit{gene}, \mathit{biomarkerFor}, e_2 : \mathit{disease}) \wedge \\ & (e_3 : \mathit{gene}, \mathit{biomarkerFor}, e_1 : \mathit{disease}). \end{aligned}$$

CHAPTER

4

INTEGRATING HETEROGENEOUS DATA INTO ANALYTICS-ENABLING KNOWLEDGE GRAPHS

This chapter has been published in:

> **Kara Schatz**, Pei-Yu Hou, Alexey V. Gulyuk, Yaroslava G. Yingling, Rada Chirkova, "BUILD-KG: Integrating Heterogeneous Data Into Analytics-Enabling Knowledge Graphs," in *2023 IEEE International Conference on Big Data (Big Data)*, pp. 2965–2974, IEEE, December 2023.

In this chapter, we introduce our proposed BUILD-KG workflow for integrating heterogeneous data into analytics-enabling knowledge graphs in a domain-agnostic and domain-aware manner, with domain scientists as humans-in-the-loop. We propose that this workflow be used during the first stage of the four C's pipeline presented in Chapter 1 (see Figure 1.1): *construction*.

This chapter is organized as follows. We provide the problem statement for our work, in Section 4.1. In Section 4.2 we introduce the proposed BUILD-KG workflow, illustrating it in Section 4.3 with a materials-science use casefrom the Science and Technologies for Phosphorus

Sustainability (STEPS) Center.[1] In Section 4.4 we describe the role of humans-in-the-loop in the BUILD-KG workflow. We conclude in Section 4.5.

Relevant preliminaries for this chapter can be found in Section 3.1.

4.1 Problem Statement

We now introduce the formal problem statement for the problem addressed by our BUILD-KG workflow: Given a set $\mathscr{D}$ of heterogeneous data files, integrate the data in $\mathscr{D}$ into a unified knowledge graph (KG) $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$ that could enable richer analytics not supported by the sources taken in isolation.

To limit the scope of this general problem for this paper, we focus on three distinct data types for the data files in $\mathscr{D}$: (1) *spreadsheet data*, (2) *images with annotations*, and (3) *regularized text data*. These data types have been chosen due to their popularity among domain scientists for storing their data. *Spreadsheet data* are relational-type data stored in data sheets in the spreadsheet format. Data sheets contain data on objects stored as rows; the columns headers indicate the specific components of the data objects. These components have relationships between them that are specified in triple sheets and can be encoded as KG edges. *Images with annotations* consist of sets of figures, graphics, etc. that are annotated with additional sets of properties of interest (metadata) stored in annotations files. *Regularized text data* consist of sets of sentences that share a similar structure, in the sense that similar entities or entities of the same type appear in the same general location in the sentence. This parallel structure allows for a "universal form" to be extracted, such that each sentence in the set is an instantiation of the universal form. The entities in each sentence have relationships between them that are specified in triple sheets and can be encoded as KG edges. More detailed descriptions of these data types can be found in Section 4.2.

4.2 The BUILD-KG Workflow

We now introduce our proposed domain-agnostic BUILD-KG workflow for integrating heterogeneous data into a knowledge graph *(KG)* that could enable richer analytics not supported by the sources taken in isolation. In Sections 4.2.1–4.2.3 we present procedures for constructing KGs from *spreadsheet data*, *images with annotations*, and *regularized text data*, respectively. Then, in Section 4.2.4 we outline the procedure for combining data in multiple formats into a unified KG.

[1] `https://steps-center.org/`

4.2.1 Converting Spreadsheet Data into the KG Format

We present here construction of a KG from *spreadsheet data*, the first data type that we consider in this work. In the proposed domain-agnostic conversion procedure, we require the data to be in the format described in Section 4.2.1.1. The output KG and its ontology are discussed in Section 4.2.1.2, and the conversion procedure is presented in Section 4.2.1.3.

4.2.1.1 The Input Data

The proposed procedure accepts spreadsheets in a specific format, which can be arrived at in collaboration with domain scientists, see Section 4.4 for an illustration. The data in the desired format are stored in two spreadsheets: (1) a *data sheet* D, and (2) a *triple sheet* T. The data sheet D is similar to a relation, in that each row in D corresponds to a single data object. Each cell represents an entity in the resulting KG; related entities appear in the same row and will have corresponding edges in the resulting KG. If the same entity name appears in multiple rows, then it will correspond to a single entity in the resulting KG. This situation arises when an entity contributes to multiple data objects, and therefore has relationships with other entities from multiple rows.

The triple sheet T has three columns, one for each of subject, predicate, and object. Each row of T describes a *triple type* in the resulting KG, that is, a (*sType*, p, *oType*) triple, in which *sType*, *oType* $\in \mathscr{T}$ represent respectively the subject and object entity-types corresponding to column names in the data sheet D, and $p \in \mathscr{P}$ is the predicate indicating the relationship type between them. As such, the triple sheet T enumerates the relationships between the entities present in the data sheet D. See Section 4.3.3 for an illustration.

4.2.1.2 The Output KG

The KG $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$ returned by the conversion procedure of Section 4.2.1.3 contains the data from the data sheet D according to the format specified by the triple sheet T. The entity set $\mathscr{N}$ of $\mathscr{G}$ consists of all the unique entries in D, and the entity-types $\mathscr{T}$ of $\mathscr{G}$ are the column headers of D. The entity-type labeling function ϕ in $\mathscr{G}$ maps each entry in D to its corresponding column header, and the predicate set $\mathscr{P}$ consists of all the entries from the second column of T. Finally, the set of triples $\mathscr{L}$ consists of a single triple of each triple type (*sType*, p, *oType*) in T for each data object d in D, where the subject and object are the entries from d in columns *sType* and *oType*, and the predicate is p.

Algorithm 1: Converting spreadsheet data into a KG

Input : Data sheet D and triple sheet T.
Output : A KG $\mathcal{G}$ containing the data from D in the format specified by T.

```
1  G ← ∅; // initialize the KG G
   // create a node of G for each entity:
2  for row ∈ D do
3      for column ∈ D do
4          e ← D[row][column]; t ← column.name;
5          G ← G ∪ create node of type t for e;
   // create all the edges of G:
6  for sType, p, oType ∈ T do
7      for row ∈ D do
8          s ← row[sType]; o ← row[oType];
9          G ← G ∪ create edge of type p from s to o;
10 return G;
```

4.2.1.3 The Conversion Procedure

Once the data are in the proper format, they can be converted into a KG in a straightforward manner, as outlined in Algorithm 1. The algorithm takes as inputs the data sheet D and the triple sheet T formatted as specified in Section 4.2.1.1, and outputs the KG $\mathcal{G}$ described in Section 4.2.1.2. The procedure works in two stages: (i) creating the nodes of the KG $\mathcal{G}$, and (ii) creating the edges of $\mathcal{G}$.

First, to create all the nodes of $\mathcal{G}$, the algorithm extracts from the data sheet D (lines 2–5) each entity e along with its type t (line 4), and creates the corresponding node of type t in the KG $\mathcal{G}$ (line 5). If the Neo4j system (The Neo4j Team 2022c) is used for storing and processing the KG data, this process can be implemented via the following Cypher (The Neo4j Team 2022b) query:

$$\texttt{MERGE}\,(\texttt{n}:t)\,\texttt{SET}\,\texttt{n.id} = e$$

In Cypher, the keyword `MERGE` is used to either identify an already existing node pattern in the graph, or to create a new node pattern if one does not exist. Thus, the above query will create a node `n` in $\mathcal{G}$ of type t with the unique identifier e only if a node corresponding to e does not already exist in $\mathcal{G}$. In this way, Algorithm 1 guarantees that only a single node will be created for each unique entity in the data sheet D.

Next, to create all the edges in the KG $\mathcal{G}$, Algorithm 1 makes a triple of each triple type

in T for each row in the data file D. To this end, the algorithm loops through the rows of T (lines 6–9), where each row specifies a subject type *sType*, a predicate p, and an object type *oType*. Then, the algorithm loops through the rows of D (lines 7–9), extracting from each row the subject s and the object o from the *sType* and *oType* columns (line 8). Finally, it creates an edge in the KG $\mathscr{G}$ of type p from s to o (line 9). Line 9 can be implemented using the following Cypher query:

$$\texttt{MATCH}\,(\texttt{n}_1),(\texttt{n}_2)$$
$$\texttt{WHERE}\,\texttt{n}_1.\texttt{id} = s\ \texttt{AND}\,\texttt{n}_2.\texttt{id} = o$$
$$\texttt{MERGE}\,(\texttt{n}_1)-[\texttt{r}:p]\rightarrow(\texttt{n}_2)$$

This query first identifies the nodes $\texttt{n}_1$ and $\texttt{n}_2$ corresponding to the subject s and object o, and then creates an edge with the predicate type p from $\texttt{n}_1$ to $\texttt{n}_2$.

Finally, Algorithm 1 returns the KG $\mathscr{G}$ that has been populated with the data from the input data sheet D according to the format specified by the input triple sheet T (line 10).

4.2.2 Converting Images with Annotations into the KG Format

We now describe KG construction from *images with annotations*. To enable a domain-agnostic conversion procedure, we require that the annotations (image properties) be provided in a specific format described in Section 4.2.2.1. The output KG and its ontology are discussed in Section 4.2.2.2, and the conversion procedure is presented in Section 4.2.2.3.

4.2.2.1 The Input Data

For this data type, the input consists of (1) a set I of one or more image files, and (2) an annotations file A. The image data in I can be in any of the popular image formats, e.g., JPG or PNG. The annotations file A consists of a single row for each image file in I. The column names in A are the property names provided by the annotations, and the row entries are the corresponding property values for the image. While the properties and values can be customized, for uniformity we recommend using consistent property names for similar images. See Section 4.3.4 for an illustration.

4.2.2.2 The Output KG

After executing the conversion procedure, see Section 4.2.2.3, the resulting KG $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$ contains the set of images I and the data from the annotations file A. The entity set $\mathscr{N}$

of $\mathscr{G}$ consists of the images in I, along with all the property values in A. The entity types $\mathscr{T}$ are `Image` and the column headers of A, and the entity-type labeling function ϕ maps each image to `Image` and each property value to its corresponding column header. The predicate set $\mathscr{P}$ = { `has_property` }. The set $\mathscr{L}$ consists of triples (s, `has_property`, o), where s is an image from I and o is a property value from the corresponding row in A.

It is possible that some use cases may require additional triples to be present in the output KG $\mathscr{G}$, specifically, triples that relate property values to one another. In this case, the desired relationships can be encoded in a triple sheet T^* according to the spreadsheet data format specified in Section 4.2.1.1, with T^* serving as an optional third input.

4.2.2.3 The Conversion Procedure

This conversion procedure takes as inputs the set of images I and the annotations file A described in Section 4.2.2.1, and outputs the KG $\mathscr{G}$ described in Section 4.2.2.2. To construct $\mathscr{G}$, the procedure adapts the inputs I and A to the spreadsheet data format described in Section 4.2.1.1, and then invokes Algorithm 1.

To adapt the files in I and A to the spreadsheet data format, the procedure adds to A a column with the `Image` header. The values in this column are the image-file names for each row. The resulting annotations file A' is the data sheet D that is input into Algorithm 1. For the triple sheet T that is also required as an Algorithm 1 input, the procedure generates a sheet with one row for each column in the annotations file A; the subject of the row is `Image`, the predicate is `has_ property`, and the object is the column name from A.

As discussed in Section 4.2.2.2, it is possible that additional triples may be desired. In this case, the optional triple sheet T^* generated for the extra triples can be concatenated with the triple sheet T before invoking Algorithm 1.

Once the data sheet D and the triple sheet T have been generated, they are passed as inputs to Algorithm 1, which generates the KG $\mathscr{G}$ outlined in Section 4.2.2.2.

4.2.3 Converting Regularized Text Data into the KG Format

We now describe construction of a KG from *regularized text data*. To handle the lack of uniformity present in textual data, we require that the text be regularized as described in Section 4.2.3.1, to facilitate accurate triple extraction via named entity recognition (NER) (Nadeau and Sekine 2007). The output KG and its ontology are discussed in Section 4.2.3.2, and the conversion procedure, including the use of NER, is presented in Section 4.2.3.3.

4.2.3.1 The Input Data

The input data consist of (1) a set S of regularized sentences, some of which are tagged sentences $S^T \subset S$, and (2) a triple sheet T formatted as described in Section 4.2.1.1. *Regularized* means that the sentences share a similar structure. E.g., consider the sentences:

- Figure 1 shows the RMSD for the interaction between 2HP and E. coli in clean water.
- Figure 3 shows the radius of gyration for the interaction between dihydrogen phosphate and E. coli in multi-ion water.

The shared structure can be specified as "`<Figure>` shows the `<Parameter>` for the `<MD process>` between `<Ion>` and `<Protein>` in `<Solvent>`."

Despite recent advances in NER performance, it still has limitations, especially with domain-specific terminology (Marrero et al. 2013). Therefore, we require that the input sentences be regularized this way to maximize the performance of the NER step of Section 4.2.3.3, resulting in higher-quality KGs. We do not expect such regularization to be burdensome to domain scientists, as they may already tend to write sentences with similar structures in their work. We have confirmed this expectation with the scientists who provided the data for our use case.

We require some of the sentences $S^T \subset S$ to be tagged, to serve as the training data for the NER model. By using an NER model, we relieve domain scientists from the arduous task of tagging all the sentences in S. Tagging NER data consists of assigning a label to each token (word) in the text indicating the named entity that the token corresponds to. Entities in the sentence that consist of multiple tokens, e.g., *clean water*, have the first token labeled as `B-tokenName`, which stands for the "beginning" of the entity, and the subsequent tokens labeled as `I-tokenName`, which stands for the "inside" of the entity. The tag `O` (other) is used for any token that does not correspond to a named entity in the text.

The triple sheet T used in the procedure is in the format described in Section 4.2.1.1. The entries in the subject and object columns of T must correspond to named entity types from the tagged sentences. See Section 4.3.5 for an example.

4.2.3.2 The Output KG

The conversion procedure of Section 4.2.3.3 outputs a KG $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$ that contains the named entities from the sentences S connected by the relationships specified in the triple sheet T. The entity set $\mathscr{N}$ consists of the named entities in S, and the entity types $\mathscr{T}$ are the tags from the tagged sentences (aside from `O`). The entity-type labeling function ϕ maps each named entity to the tag assigned by the NER model, and the predicate set $\mathscr{P}$ consists of all the entries from the second column of the triple sheet T. The set of triples $\mathscr{L}$ consists of a

single triple of each possible triple type (*sType*, p, *oType*) in T for each sentence in S, where the subject and object are the *sType*- and *oType*-tagged entities in the sentence, and the predicate is p.

4.2.3.3 The Conversion Procedure

The procedure takes as inputs the set of sentences S, with some $S^T \subset S$ tagged, and the triple sheet T, see Section 4.2.3.1, and outputs a KG $\mathscr{G}$, see Section 4.2.3.2. To construct $\mathscr{G}$, an NER model is trained on S^T and used to extract named entities from S. Then, Algorithm 1 is invoked on the result of converting the outputs to the spreadsheet-data format described in Section 4.2.1.1.

If the set $S^T \subset S$ is too small for successful training of a NER model, then S^T can be synthetically enlarged by automatically generating many sentences with similar structure. This generation can be done by substituting entity tokens in the regularized sentence structure with random words from a dictionary provided by the domain scientists. We used this technique to enlarge the training data set for our use case.

The trained model is used to provide tags for the remaining sentences in $S \setminus S^T$. Then, a data sheet D is generated from the complete set of tagged sentences S, just as for the spreadsheet data format, see Section 4.2.1.1. The column headers of D are the NER tags (aside from `O`), and each row corresponds to a sentence in S, with the `t`-tagged entities in S appearing in the columns of D with column header `t`. The resulting data sheet D and the triple sheet T are passed as inputs to Algorithm 1, which returns the output KG $\mathscr{G}$.

4.2.4 Assembling a KG from Multiple Data Types and Data Sets

We now discuss the scenario in which a KG is created by combining data of multiple types. The KG can be constructed by using our proposed procedures for the corresponding data types, see Sections 4.2.1–4.2.3. The data from individual sources can be converted one source at a time into a single unified KG, by executing Algorithm 1 on each source and using the same KG $\mathscr{G}$ for all the runs. To ensure that the data align and connect properly, one must pay particular attention to the terminology used across the sources. Terms representing the same entity/concept should be unified across the sources, such that the `MERGE` queries executed during the runs of Algorithm 1 will identify pre-existing entities when appropriate instead of creating new entities, see Section 4.2.1.3 for a discussion of the semantics of `MERGE` in Cypher.

In this scenario, we recommend maintaining data provenance when constructing the KG. Keeping data provenance is useful in general, but when assembling a KG from multiple data sources, it can be imperative down the line, e.g., for data cleaning or verification. Thus, we

recommend maintaining as provenance the specific data set (input files) from which each KG triple originated. This can be achieved by adding the `provenance` property to each triple that would store the name of the data set from which the triple originated. The Cypher-query implementation of line 9 of Algorithm 1 can be modified to accomplish this by adding an appropriate SET clause.

In addition to ensuring that the same nodes are used when entities are shared across the input data sets, domain scientists might also choose to add to the KG extra edges connecting nodes across the converted source data, to express extra relationships based on domain semantics. Our proposed BUILD-KG workflow makes this possible with additional inputs. Domain scientists can input a set of specific (s, p, o) triples that would connect entities s and o via predicate p across the converted source data. These triples can be added to the KG using the command of line 9 in Algorithm 1.

4.3 The Use Case with Materials Science Data

In this section we outline our implementation of the proposed BUILD-KG workflow, and illustrate its application to a use case in the materials-science *(MS)* domain. Section 4.3.1 describes the tools used in our implementation of the BUILD-KG workflow, and Section 4.3.2 presents the source data for the use case. Sections 4.3.3–4.3.5 describe our instantiations of the workflow for the use-case MS *spreadsheet data*, *images with annotations*, and *regularized text data*, respectively. Finally, Section 4.3.6 describes the process of assembling a single unified knowledge graph (KG) from the three use-case data sets.

4.3.1 Tools Used in our Implementation of BUILD-KG

Implementing the BUILD-KG workflow requires a graph data-management system (DBMS), a graph query language, and a programming language. In our implementation, we used the Neo4j graph DBMS (The Neo4j Team 2022c) with the Cypher graph query language (The Neo4j Team 2022b). The workflow was implemented in Python (Van Rossum and Drake 2009), including its publicly available package `py2neo` (Small 2019) for connecting to Neo4j and executing Cypher queries within Python code. For the named entity recognition (NER) model, we used the `flair` (Akbik et al. 2019) package in Python. As training the NER model requires a data corpus and a tagger, for our use case we built the corpus by using the `ColumnCorpus` object in `flair`, with `SequenceTagger` from `flair` as the tagger. To train the `SequenceTagger`, we used the popular word-embedding model GloVe (Pennington et al. 2014).

4.3.2 The Source Data in the Materials Science Use Case

As a use case for testing the proposed BUILD-KG workflow, we used data provided by researchers in the Science and Technologies for Phosphorus Sustainability (STEPS) Center. STEPS is sustainability driven, with a focus on creating systems that would allow successful capture of phosphates from a variety of environments. To build such systems, one needs to consider materials with high affinity to phosphate binding, low toxicity, and amenability to being easily cleaned or destroyed. Possible solutions include soft materials, such as bio-inspired proteins found in plants or microorganisms.

In our use-case data, the Escherichia coli (E. coli) phosphate-binding protein was used as a test model. This protein was investigated in various environments for its capability to bind with several forms of phosphate ions. We used data sets generated by experimental and computational E. coli tests. Merging the data sets into a KG would enable domain scientists to identify "junction points" and direct links between the data that may not be easily identifiable as the data sets grow in sizes. This would allow domain scientists to perform deeper analyses of the phosphate-binding properties of E. coli.

The data sets generated by STEPS scientists are as follows:

4.3.2.1 Data Set 1: Spreadsheet Data

This data set was obtained via conducting a series of experiments with the E. coli protein. The data consist of ultraviolet–visible (UV-Vis) spectrophotometer measurements summarized in spreadsheets. These measurements include absorbance, through which one can derive the phosphate capture rate via comparisons to the baseline. This series of experiments measured the capture rate under several conditions: temperature gradient, pH variations, ionic strength variations (concentration of the salt ions KCl present in the test solution), as well as in several system configurations: different amounts of the E. coli protein introduced in the test solution, different dihydrogen phosphate ion concentrations, several water matrix models, different types of nanoparticles used as the carrier, and various numbers of cycles in which the proteins were used.

4.3.2.2 Data Set 2: Images with Annotations

This data set was generated via analysis and summary of the raw data from Data Set 1 described in Section 4.3.2.1. It contains various graphs and charts, as well as data on the process kinetics and related isotherms, which were derived from the raw data points.

4.3.2.3 Data Set 3: Regularized Text Data

The data in this set were derived from atomistic simulations of the E.coli phosphate-binding protein carried out via all-atom molecular dynamics (MD) simulations. In the simulations, the test protein interacted with various phosphate ion types, and several values were computationally observed at the atomistic level, including binding rate, binding strength, speed of binding, and the particular regions of the protein affected by the presence of ions. This approach provides some insight into the kinetics of the binding process and supplements the experimental results.

The data for the MD simulations consist of input files (systems configurations, scripts, etc.), output data (files representing the dynamics of the systems across the simulation), and summary data (parameters derived from the systems via analysis). We used the summary data, which include various figures, together with their captions and descriptions.

4.3.3 Constructing a KG from the MS Spreadsheet Data

In this section we discuss our experience of constructing a KG from the *spreadsheet data* in our MS use case. Those data come from Data Set 1, see Section 4.3.2.1 for the details.

4.3.3.1 The Input Data

To arrive at the required input format specified in Section 4.2.1.1, we used the data sheet D provided by Data Set 1, and generated a triple sheet T with the help of domain scientists. Table 4.1 shows a fragment of the data sheet D, which contains data about the experimental trials outlined in Section 4.3.2.1. For each trial, there is a single data object (row) containing information describing the trial, e.g., the trial ID, the protein used, and the solution used. For example, the first row of Table 4.1 details the trial *Temp1.1*. This trial used the protein *E. coli* on solution *Temp 01*, which contained *clean water* and *orthophosphate* at the original concentration of 1.12 *mg/L*; the trial used the temperature setting of $10°C$ and resulted in the absorbance of 0.234 *nm*.

Table 4.2 shows a fragment of the triple sheet T, which specifies the triple types desired in the resulting KG. Each row contains the subject, predicate, and object for a single triple type. E.g., the first row specifies the triple type (*Trial*, `used_protein`, *Protein*), with the meaning that each *Trial* and the corresponding *Protein* are linked by the `used_protein` relationship.

Table 4.1: Fragment of the data sheet from Data Set 1 (see Section 4.3.2.1) ingested with the data in Table 4.2 by the conversion procedure of Section 4.2.1.3 for spreadsheet data. Each row describes a single data object, and the columns represent different entities.

Trial	Protein	Solution	Solvent	Ion	Temp. ($°C$)	Absorbance (nm)	Original Conc. (mg/L)
Temp1.1	E. coli	Temp 01	clean water	ortho-phosphate	10	0.234	1.12
Temp1.2	E. coli	Temp 01	clean water	ortho-phosphate	10	0.240	1.12
Temp1.3	E. coli	Temp 01	clean water	ortho-phosphate	10	0.226	1.12

Table 4.2: Fragment of the triple sheet for Data Set 1 (see Section 4.3.2.1) ingested with the data in Table 4.1 by the conversion procedure of Section 4.2.1.3 for spreadsheet data. Each row describes a triple type of the form (*subjectType*, *p*, *objectType*).

Subject	Predicate	Object
Trial	`used_protein`	Protein
Trial	`used_environment`	Solution
Protein	`used_on`	Solution
Protein	`had_property`	Absorbance
Solution	`contains`	Solvent
Solution	`contains`	Ion

4.3.3.2 The Output KG

The inputs D and T determine both the ontology and the contents of the resulting KG $\mathscr{G}_1 = (\mathscr{N}_1, \mathscr{T}_1, \phi_1, \mathscr{P}_1, \mathscr{L}_1)$. The components of the KG $\mathscr{G}_1$ resulting from the inputs outlined in Section 4.3.3.1 are as follows. The entity set $\mathscr{N}_1$ consists of all the unique entries in D, that is, $\mathscr{N}_1 = \{$ *Temp1.1*, *Temp1.2*, *Temp1.3*, *E. coli*, *Temp 01*, *clean water*, *orthophosphate*, 10, 0.234, 0.240, 0.226, 1.12 $\}$. The entity-types $\mathscr{T}_1$ are the column headers of D, so $\mathscr{T}_1 = \{$ `Trial`, `Protein`, `Solution`, `Solvent`, ..., `Original Conc. (mg/L)` $\}$. The entity-type labeling function ϕ_1 maps each entry in D to its corresponding column header, e.g., ϕ_1 : *Temp1.1* $\rightarrow$ `Trial` and ϕ_1 : *Temp 01* $\rightarrow$ `Solution`. The predicate set $\mathscr{P}_1$ consists of all the entries from the second column of T, i.e., $\mathscr{P}_1 = \{$ `used_protein`, `used_environment`, ..., `contains` $\}$. Finally, the set of triples $\mathscr{L}_1$ consists of a single triple of each triple type (*sType*, *p*, *oType*) in T for each data object d in D, where the subject and object are the entries from d in columns $sType$ and $oType$, and the predicate is p. For example, (*Temp 1.1*, `used_protein`, *E. coli*) $\in \mathscr{L}_1$.

4.3.3.3 The Conversion Process

We implemented the spreadsheet-conversion procedure of Algorithm 1, see Section 4.2.1.3, in Python (Van Rossum and Drake 2009) using the `py2neo` (Small 2019) package to connect to our graph database stored in Neo4j (The Neo4j Team 2022c). Using the data sheet D and the triple sheet T of Section 4.3.3.1 as its inputs, Algorithm 1 returned the KG $\mathscr{G}_1 = (\mathscr{N}_1, \mathscr{T}_1, \phi_1, \mathscr{P}_1, \mathscr{L}_1)$ described in Section 4.3.3.2.

4.3.4 Constructing a KG from the MS Images with Annotations

In this section we discuss our experience of constructing a KG from the *images with annotations* in our MS use case. The data come from Data Set 2, see Section 4.3.2.2 for the details.

4.3.4.1 The Input Data

To align with the required input format specified in Section 4.2.2.1, we used the set of images I provided by Data Set 2, and compiled all the annotations into a single annotations file A with one row of annotations per file in I. Figure 4.1 shows a sample image from the set I of images in Data Set 2; this is a graph generated by MS researchers to summarize their experimental results. Table 4.3 shows the annotations for the image of Figure 4.1, which comprise a single line in the annotations file A. The annotations, i.e., properties of interest of the image, include in this case *title*, *x-axis*, *y-axis*, *description*, and *meaning*. E.g., the first annotation indicates that the image *title* is "Removal vs. Temperature."

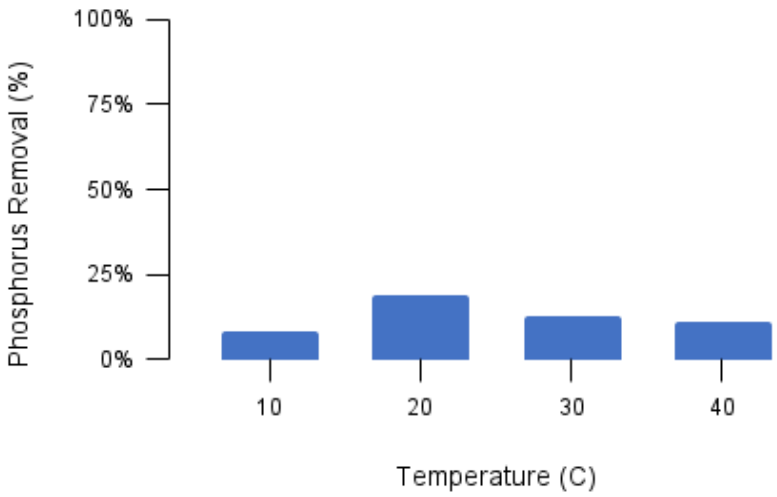


Figure 4.1: An image from Data Set 2 described in Section 4.3.2.2, which was ingested by the conversion procedure of Section 4.3.4 for images with annotations, along with the annotations shown in Table 4.3.

Table 4.3: The annotations from Data Set 2 described in Section 4.3.2.2 for the image shown in Figure 4.1. The annotations have been used as inputs to the conversion procedure of Section 4.2.2.3 for images with annotations.

title	x-axis	y-axis	description	meaning
Removal vs. Temperature	temperature	removal rate (%)	Removal rate as a function of temperature (the temperature-dependence of removal)	The amount of phosphate that is removed at different temperatures

Table 4.4: The data sheet generated when executing the conversion procedure of Section 4.2.2.3 for images with annotations; the procedure used as inputs the image shown in Figure 4.1 and its annotations shown in Table 4.3.

Image	title	x-axis	y-axis	description	meaning
Figure 4.1	Removal vs. Temperature	temperature	removal rate (%)	Removal rate as a function of temperature (the temperature-dependence of removal)	The amount of phosphate that is removed at different temperatures

4.3.4.2 The Output KG

The components of the KG $\mathscr{G}_2 = (\mathscr{N}_2, \mathscr{T}_2, \phi_2, \mathscr{P}_2, \mathscr{L}_2)$ resulting from the inputs discussed in Section 4.3.4.1 are as follows. The entity set $\mathscr{N}_2$ consists of the images in I and the unique entries in A, that is, $\mathscr{N}_2 = \{$ *Figure 4.1*, *Removal vs. Temperature*, *temperature*, *removal rate (%)*, ...$\}$. The entity types $\mathscr{T}_2$ are `Image` and the column headers of A: $\mathscr{T}_2 = \{$ `Image`, `title`, x`-axis`, y`-axis`, `description`, `meaning` $\}$. The entity-type labeling function ϕ_2 maps each image to `Image` and each property value to its corresponding column header, e.g., ϕ_2 : *Figure 4.1* $\rightarrow$ `Image` and ϕ_2 : *temperature* $\rightarrow$ x`-axis`. The predicate set $\mathscr{P}_2 = \{$ `has_property` $\}$. The set of triples $\mathscr{L}_2$ consists of triples (s, `has_property`, o), where s is an image from I and o is a property value from the corresponding row in A. E.g., (*Figure 4.1*, `has_property`, *temperature*) $\in \mathscr{L}_2$.

4.3.4.3 The Conversion Process

Following the procedure of Section 4.2.2.3, we converted the set of images I and the annotations file A of Section 4.3.4.1 into the spreadsheet data format, obtaining a data sheet D and a triple sheet T. The adaptation procedure, described in Section 4.2.2.3, was implemented as a Python (Van Rossum and Drake 2009) script. Table 4.4 shows the data sheet D generated for the inputs I and A, see Section 4.3.4.1. D consists of the image annotations A with an additional column indicating the image corresponding to the annotations. Table 4.5 shows a fragment of the triple sheet T, which contains a single row for each column in the annotations file A. We passed these inputs D and T into our implementation of Algorithm 1 of Section 4.3.3.3, which returned the KG $\mathscr{G}_2 = (\mathscr{N}_2, \mathscr{T}_2, \phi_2, \mathscr{P}_2, \mathscr{L}_2)$ described in Section 4.3.4.2.

4.3.5 Constructing a KG from the MS Regularized Text Data

In this section we discuss our experience of constructing a KG from *regularized text data* in our MS use case. These data come from Data Set 3, see Section 4.3.2.3 for the details.

Table 4.5: Fragment of the triple sheet generated when executing the conversion procedure of Section 4.2.2.3 on the image of Figure 4.1 and its annotations shown in Table 4.3. Each row describes a triple type of the form (*subjectType*, *p*, *objectType*).

Subject	Predicate	Object
Image	`has_property`	title
Image	`has_property`	x-axis
Image	`has_property`	y-axis

Table 4.6: Fragment of the triple sheet for the Data Set 3 of Section 4.3.2.3, used as an input to the conversion procedure of Section 4.2.3.3 for regularized text data. Each row describes a triple type of the form (*subjectType*, *p*, *objectType*).

Subject	Predicate	Object
Figure	`illustrates`	Process
Process	`measured_by`	Parameter
Process	`involves`	IonMD
Process	`involves`	ProteinMD
Process	`occurs_in`	SolventMD

4.3.5.1 The Input Data

To align with the required input format of Section 4.2.3.1, we used the set S of regularized sentences from Data Set 3, tagging some of the sentences $S^T \subset S$. For example, the sentence "Figure 1 shows the RMSD for the interaction between 2HP and E. coli in clean water" in S has been converted to the following sequence of (Token, Tag) pairs in S^T: [(Figure, B-Figure), (1, I-Figure), (shows, O), (the, O), (RMSD, B-Parameter), (for, O), (the, O), (interaction, B-ProcessMD), (between, O), (2HP, B-IonMD), (and, O), (E, B-ProteinMD), (coli, I-ProteinMD), (in, O), (clean, B-SolventMD), (water, I-SolventMD)]. The tags indicate the named entities in the sentence and their types. E.g., the first two tokens in the example form a named entity of type Figure, while the third and fourth tokens are not named entities.

We also generated a triple sheet T with the help of domain scientists; a fragment of T is shown in Table 4.6. The triple sheet specifies the triple types desired in the resulting KG. Each row of the sheet contains the subject, predicate, and object for a single triple type. For example, the first row denotes the triple type (*Figure*, `illustrates`, *Process*), meaning that each *Figure* and the corresponding *Process* have an `illustrates` relationship between them.

4.3.5.2 The Output KG

The components of the KG $\mathscr{G}_3 = (\mathscr{N}_3, \mathscr{T}_3, \phi_3, \mathscr{P}_3, \mathscr{L}_3)$ resulting from the sample inputs provided in Section 4.3.5.1 are as follows. The entity set $\mathscr{N}_3$ consists of the named entities in S, that is,

Table 4.7: The data sheet generated by the conversion procedure of Section 4.2.3.3 for regularized text data for the sample tagged sentence of Section 4.3.5.1 from the Data Set 3 of Section 4.3.2.3.

Figure	Parameter	Process	IonMD	ProteinMD	SolventMD
Figure 1	RMSD	interaction	2HP	E. coli	clean water

$\mathcal{N}_3 = \{$ *Figure 1*, *RMSD*, *interaction*, *2HP*, *E. coli*, *clean water* $\}$. The entity types $\mathcal{T}_3$ are the tags from the tagged sentences (aside from `O`), so $\mathcal{T}_3 = \{$ `Figure`, `Parameter`, `ProcessMD`, `IonMD`, `ProteinMD`, `SolventMD` $\}$. The entity-type labeling function ϕ_3 maps each named entity to the tag that it is assigned by the trained NER model, e.g., ϕ_3 : *RMSD* $\rightarrow$ `Parameter` and ϕ_3 : *2HP* $\rightarrow$ `IonMD`. The predicate set $\mathcal{P}_3$ consists of all the entries from the second column of the triple sheet T, i.e., $\mathcal{P}_3 = \{$ `illustrates`, `measured_by`, `involves`, `occurs_in` $\}$. Finally, the set of triples $\mathcal{L}$ consists of a single triple of each possible triple type (*sType*, p, *oType*) in T for each sentence in S, where the subject and object are the *sType*- and *oType*-tagged entities in the sentence, and the predicate is p. For example, (*interaction*, `measured_by`, *RMSD*) $\in \mathcal{L}_3$.

4.3.5.3 The Conversion Process

Following the conversion procedure of Section 4.2.3.3, we first trained a named entity recognition (NER) model on the set of tagged sentences S^T synthetically enhanced by automatically generating sentences with similar structure, see Section 4.3.5.1 for an example. See Sections 4.3.1 and 4.2.3.3 for more details about our NER model and the process for generating synthetic sentences.

Next, we used the trained model to provide tags for the sentences $S \setminus S^T$. Then we adapted the complete set of tagged sentences S to the format of the data sheet D used for the spreadsheet data input (see Section 4.2.1.1). To this end, we implemented the adaptation procedure described in Section 4.2.3.3 as a Python (Van Rossum and Drake 2009) script. Table 4.7 shows the data sheet D that was generated for the sample sentence and tags given in Section 4.3.5.1. D consists of a single row for the sentence; each entry is a named entity from the sentence, and each named entity appears in the column corresponding to its tag (type).

Finally, we passed the newly generated data sheet D and the triple sheet T (see Section 4.3.5.1) to our implementation of Algorithm 1 of Section 4.3.3.3. The run of Algorithm 1 returned the KG $\mathcal{G}_3 = (\mathcal{N}_3, \mathcal{T}_3, \phi_3, \mathcal{P}_3, \mathcal{L}_3)$ described in Section 4.3.5.2.

4.3.6 Assembling a KG from Multiple Data Types and Data Sets

For our use case, it would be valuable to the domain scientists if the KGs $\mathscr{G}_1$, $\mathscr{G}_2$, and $\mathscr{G}_3$ were unified into a single KG. As discussed in Section 4.2.4, this can be accomplished by building a single KG $\mathscr{G}$ using the BUILD-KG conversion process for each data type and each data set. Executing the conversion processes of Sections 4.3.3.3, 4.3.4.3, and 4.3.5.3 resulted in the KG $\mathscr{G}$ with the contents of the KGs $\mathscr{G}_1$, $\mathscr{G}_2$, and $\mathscr{G}_3$ described in Sections 4.3.3.2, 4.3.4.2, and 4.3.5.2, respectively, with $\mathscr{G}$ having the unified ontology of the three KGs. Further, each node or edge that is shared between $\mathscr{G}_1$, $\mathscr{G}_2$, and $\mathscr{G}_3$ was mapped by BUILD-KG into the same node or edge in $\mathscr{G}$.

Following the recommendations given in Section 4.2.4, we consulted domain scientists to ensure that the three data sets in our use case would use common terminology where appropriate. Additionally, we maintained data provenance when assembling $\mathscr{G}$. To this end, each triple (edge) t in $\mathscr{G}$ has a property t.`provenance` that stores the ID of one of the three source data sets from which that triple originated.

The domain scientists were also looking for additional relationships that would connect entities across the data sets, thus enabling richer analytics on $\mathscr{G}$ than what the data sets or even $\mathscr{G}_1$, $\mathscr{G}_2$, and $\mathscr{G}_3$ could allow in isolation. To this end, the scientists provided a set of (s, p, o) triples to connect such entities; s and o would come from different data sets but have the relationship p between them in $\mathscr{G}$. An example of such a triple is (*2HP*, `simulates`, *orthophosphate*), where *2HP* is a node of type `IonMD` in Data Set 3 (see Section 4.3.5.1) and *orthophosphate* is a node of type `Ion` in Data Set 1 (see Section 4.3.3.1). The triple, see Figure 1.2, indicates that the `IonMD` *2HP* is used to simulate the `Ion` *orthophosphate* in molecular-dynamics simulations. We added the triples to $\mathscr{G}$ via the Cypher query implementing line 9 of Algorithm 1.

These steps completed our BUILD-KG conversion and unification process for the use case. We provided the resulting KG $\mathscr{G}$ to the STEPS scientists for their further use and exploration.

4.4 Humans-in-the-Loop in the Workflow

We now detail the preprocessing steps that should be taken in order to properly format the data that are to be integrated by our proposed BUILD-KG workflow. This preprocessing involves close collaboration with domain scientists as humans-in-the-loop, and is needed for achieving full understanding of their data and for ensuring that the data are captured in the knowledge graph *(KG)* as accurately as possible.

To facilitate productive collaboration with domain scientists, we recommend the following four data-preprocessing steps:

Table 4.8: Sample semantic sentences generated based on a semantic description of Data Set 1 presented in Section 4.3.2.1.

Defines	Semantic sentence
Entity-type	E. coli is a Protein.
Entity-type	Water containing orthophosphate is a Solution.
Predicate	An experimental trial resulted in 0.234 nm of absorbance.
Predicate	A solution had an original concentration of 1.12 mg/L.
Predicate	The protein E. coli was used on solution Temp 01.

1. Obtain raw data from the domain scientists, along with semantic descriptions of the data elements;

2. Generate sample semantic *subject-predicate-object* sentences based on the semantic descriptions;

3. Ask the domain scientists to validate or correct the semantic sentences; and

4. Based on the semantic sentences, format the raw data such that they would meet the formatting requirements presented in Sections 4.2.1.1, 4.2.2.1, and 4.2.3.1.

Examples of the data received from the STEPS scientists in Step 1 are shown in Figures 1.2(a)-(b). From these data, we generated semantic *subject-verb-object* sentences to show the scientists (Step 2). The second column of Table 4.8 shows some sample semantic sentences that were generated based on the semantic descriptions of the spreadsheet data from Data Set 1 (see Section 4.3.2.1) given by the scientists. The first two sentences in Table 4.8 are used to define entity-types in the resulting KG. They take the general form "`<EntityName>` is a `<EntityType>`." The last three sentences in Table 4.8 are used to define predicates in the resulting KG. They take the general form "`<Subject> <Predicate> <Object>`."

After the generation of the semantic sentences, domain scientists validate them in Step 3 to ensure that the data are understood and captured correctly before the KG construction. Based on the entity-type sentences, the scientists can verify the entities selected from the data sets and the types assigned to the entities. Based on the predicate sentences, the scientists can verify the related entities and the relationship names. Verifying and correcting these items enables the scientists to control the data in the resulting KG and its ontology, so that it would align with their expectations and needs. In this step, we found it useful to generate with the STEPS scientists a dictionary of domain-specific terminology, to enable us to align the terminology used across the data sets. The dictionary allowed us to build the ontology more efficiently. It also cleared up some misunderstandings about the terms used. E.g., instead of using a single

node of type *Solution* for the liquids used in the experiments, the scientists requested that we use nodes of types *Solvent* and *Ion* to separate the contents of the liquids.

Steps 2 and 3 can be repeated as many times as necessary to produce an integrated KG that satisfies the needs of the domain scientists. Such a KG will have a proper structural and semantic representation of the original data, and will provide insightful connections (junction points) across the data. Once this iterative process is done, in Step 4 the raw data can be reformatted to match the BUILD-KG input-format requirements. In our use case, the semantic sentences shown in Table 4.8 were used to generate the spreadsheet data-type inputs shown in Tables 4.1 and 4.2 and discussed in Section 4.3.3.1.

4.5 Summary

In this chapter we introduced a domain-agnostic workflow called BUILD-KG for integrating heterogeneous scientific and experimental data from multiple sources into a single unified knowledge graph (KG) that can enable richer data analytics. BUILD-KG is broadly applicable, accepting input data in popular structured and unstructured storage formats. It is also designed to be carried out with end users as humans-in-the-loop, which makes BUILD-KG domain aware. We presented the BUILD-KG workflow, reported on our experiences with applying it to data in the materials-science domain, and provided suggestions on involving domain scientists in BUILD-KG as humans-in-the-loop. We posit that our proposed BUILD-KG workflow can enable domain scientists to seamlessly convert their data into the KG format, unify their data with those shared by other domain scientists, and then apply to the resulting KG data-analysis tasks, such as rule mining and machine learning, that may not be supported by the data sources taken in isolation. In this way, the BUILD-KG workflow can make KGs more accessible to domain scientists, thus encouraging greater use and exploration, while increasing collaboration and richness of research results. Our future work includes expanding the proposed collection of conversion procedures within BUILD-KG to include other popular data-storage formats.

CHAPTER

5

WORKFLOW FOR DOMAIN- AND TASK-SENSITIVE CURATION OF KNOWLEDGE GRAPHS

This chapter has been published in:

> **Kara Schatz**, Daniel Korn, Alexander Tropsha, and Rada Chirkova, "Workflow for Domain- and Task-Sensitive Curation of Knowledge Graphs, with Use Case of DRKG," in *2022 IEEE International Conference on Big Data (Big Data)*, pp. 3692–3701, IEEE, December 2022.

In this chapter we introduce our proposed KG-CURATE: Knowledge Graph Curation for Usability, Readability, and Accessibility for Task Expectations workflow for domain- and task-sensitive curation of knowledge graphs, with domain experts and anticipated knowledge-graph users as humans-in-the-loop. We propose that this workflow be used during the second stage of the four C's pipeline presented in Chapter 1 (see Figure 1.1): *curation*.

This chapter is organized as follows. In Section 5.1 we provide the formal problem statement for our work. In Section 5.2 we introduce the objectives and tools for the proposed workflow, along with the specific details of our test case in the biomedical domain. In Section 5.3 we detail the KG-CURATE workflow phases and their application to the biomedical knowledge

graph called Drug Repurposing Knowledge Graph (DRKG) (Ioannidis et al. 2020), followed in Section 5.4 by suggestions of user involvement in the workflow. Finally, in Section 5.5 we discuss the challenges and lessons learned, and we conclude in Section 5.6.

Relevant preliminaries for this chapter can be found in Section 3.1.

5.1 Problem Statement

We now introduce the formal problem statement that our proposed KG-CURATE workflow addresses. Given (i) knowledge-graph (KG) source files in the plain-text-file format, containing the sets of KG entities $\mathcal{N}$, predicates $\mathcal{P}$, and triples $\mathcal{L}$, along with *entity- and triple-metadata*, and (ii) sets $\mathcal{U}_{\mathcal{N}}$ and $\mathcal{U}_{\mathcal{L}}$ of *user-desired entity- and triple-properties*, respectively, that contain *tests of satisfaction*, our goals are to (a) create a KG $\mathcal{G} = (\mathcal{N}, \mathcal{P}, \mathcal{L})$[1] in a graph data-management system (Neo4j (The Neo4j Team 2022c) in our implementation), (b) enrich $\mathcal{G}$ with $\mathcal{U}_{\mathcal{N}}$ and $\mathcal{U}_{\mathcal{L}}$, and (c) ensure that $\mathcal{G}$ is accessible to multiple users.

To make satisfaction of the goals verifiable, *tests of satisfaction* are used. In this work, we use *tests of satisfaction* that require the properties of $\mathcal{U}_{\mathcal{N}}$ and $\mathcal{U}_{\mathcal{L}}$ to be applied to a certain proportion of the KG entities and triples. That is, each test indicates a percentage t_p of the entities (or triples) for which the user envisions the property p. We say that goal (b) of the problem statement has been satisfied as long as t_p% of the entities (or triples) have the desired property.

Note that this formulation of the problem allows the satisfaction of the goals to be verified in a very direct and natural way. The satisfaction of goal (a) is verifiable by simply checking that all entities in $\mathcal{N}$ and all triples in $\mathcal{L}$ are present in the KG $\mathcal{G}$ as nodes and edges. As mentioned, the satisfaction of goal (b) is verifiable via tests of satisfaction. Lastly, the satisfaction of goal (c) is verifiable by directly testing that an external machine can access the KG.

5.2 The KG-CURATE Workflow: Objectives, Tools, and Test-Case Details

We now describe the objectives of the KG-CURATE workflow, see Section 5.2.1, the necessary tools used for its implementation, see Section 5.2.2, and the details of our test case based on the DRKG graph, see Section 5.2.3.

[1] In this chapter we do not directly consider the entity-types $\mathcal{T}$ or the entity-type labeling function $\phi : \mathcal{N} \to \mathcal{T}$ of a knowledge graph $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$ since not all knowledge-graph sources provide these data. If such data were available, then the proposed KG-CURATE workflow could be naturally extended to this case.

5.2.1 Objectives

To address the issues mentioned in Section 1.3.2 and captured by the problem statement in Section 5.1, the following steps are to be performed:

1. Represent all the triples from the knowledge-graph (KG) source files in a graph data-management system;
2. Extract desired metadata provided by the KG source and add it to the KG;
3. Map entity identifiers to desired attributes/metadata that were not provided by the KG source;
4. Ensure that all the information in the KG be readable and understandable; and
5. Make the resulting KG cloud accessible.

Each of these steps comprises a single phase of the proposed KG-CURATE workflow and is explained in greater detail in Section 5.3.

5.2.2 The Tools

To complete the KG-CURATE workflow, some technological tools are required: a graph data-management system (DBMS), a graph query language, a programming language, and a cloud service. There is a wide variety of options for each of these; we based our choices on several criteria, including public availability, ease of use, and support availability.

For our graph DBMS, we chose Neo4j (The Neo4j Team 2022c). The main reason for this selection is that Neo4j is publicly available and is fairly straightforward to set up and troubleshoot, thanks to the extensive online documentation and support. Moreover, several programming languages provide packages that enable access to Neo4j databases.

For the graph query language, we chose Cypher (The Neo4j Team 2022b), which is compatible with Neo4j. Other graph query languages that could be considered include SPARQL (Harris et al. 2013), Gremlin (Apache TinkerPop 2023), and nGQL (NebulaGraph 2023).

We chose Python (Van Rossum and Drake 2009) as the programming language. There were two key reasons for this choice. The first is that Python has a publicly available package, `py2neo` (Small 2019), for using Neo4j and Cypher within Python code. The second is that Python also has a publicly available package, `requests` (Reitz 2022), for making HTTP requests. We use `requests` in Phase 3 of the proposed workflow, see Section 5.3.3 for the details.

For the cloud service, we chose Google Cloud (Google LLC 2022). The key reason for this choice is that it is one of the most publicly available, ready-to-use cloud services that exists. It

also provides a vast number of options that allow users to customize their cloud instances to their specific needs. It involves minimal setup and maintenance, making it easy to use even for those who are not familiar with using cloud services. Finally, there is a good amount of troubleshooting support for Google Cloud.

5.2.3 Test-Case Details

Having outlined the general steps and tools needed for accomplishing the proposed KG-CURATE workflow, we now discuss the specific details of testing the workflow on the use case of DRKG. The process is generalizable to other KGs and tools; as such, we provide the specific details of this test case as a template.

5.2.3.1 Drug Repurposing Knowledge Graph

As a test case, we applied our KG-CURATE workflow to the biomedical KG Drug Repurposing Knowledge Graph (DRKG) (Ioannidis et al. 2020). It contains 97,238 nodes of 13 different types and 5,874,261 triples of over 107 different predicate types. Each node represents a biological entity of a certain type, e.g., *anatomy*, *compound*, *disease*, *gene*, or *symptom*. Triples represent biological relationships between these entities, with the predicate type indicating the type of relationship, e.g., *Anatomy-expresses-Gene*, *Compound-treats-Disease*, and *Gene-regulates-Gene*.

The source files. We now describe the DRKG source files.[2] The KG source website provides a brief description of the source files.[3] There are three key files that comprise DRKG: (i) a *triple file* containing all the (s, p, o) triples in the tab-separated format, see Section 5.3.1 for details; (ii) an *entity-source mapping file* that provides the original source for each entity; and (iii) a *predicate glossary file* that provides for each predicate a description of its meaning, along with other properties of the predicate, e.g., the original source and the subject and object types connected by the predicate.

The workflow inputs. In alignment with the inputs to the problem statement of Section 5.1, the specific KG-CURATE inputs with respect to DRKG are as follows:

(i) The DRKG source files containing the sets of KG entities $\mathscr{N}$, predicates $\mathscr{P}$, and triples $\mathscr{L}$, along with entity- and triple-metadata.

[2] See `https://github.com/gnn4dr/DRKG#download-drkg`.
[3] `https://github.com/gnn4dr/DRKG#drkg-dataset`

Table 5.1: A snippet from the DRKG source file (Ioannidis et al. 2020) that contains the knowledge-graph triples. Each row represents a single triple; the columns represent the triple subject, predicate, and object.

Compound:: DB00222	DRUGBANK::ddi-interactor-in::Compound:Compound	Compound:: DB00331
Compound:: DB00331	DRUGBANK::target::Compound:Gene	Gene:: 2819
Compound:: DB00331	DRUGBANK::target::Compound:Gene	Gene:: 5564
Compound:: DB00331	Hetionet::CbG::Compound:Gene	Gene:: 5564
Compound:: DB00331	Hetionet::CtD::Compound:Disease	Disease:: DOID:9352
Compound:: DB00222	Hetionet::CtD::Compound:Disease	Disease:: DOID:9352

(ii) The set $\mathscr{U}_{\mathscr{N}} = \{source, name, type, InChIKey, SMILES\}$ of user-desired entity-properties and the set $\mathscr{U}_{\mathscr{L}} = \{source, connectedEntityTypes, interactionType, description\}$ of user-desired triple-properties. *InChIKey* refers to the International Chemical Identifier key (Heller et al. 2013), a standard identifier for chemical substances; *SMILES* refers to the Canonical Simplified Molecular-Input Line-Entry System (SMILES) string of a chemical substance (Weininger 1988), which is a standard way to represent its molecular structure. Each property p has a percentage t_p for its test of satisfaction (see Section 5.1); $t_p = 95\%$ for all properties, except *InChIKey* and *SMILES*, for which it is 50%.

5.2.3.2 Sample Input and Output Data

Table 5.1 shows a snippet from the DRKG *triple file*, which is the source file containing the KG triples (Ioannidis et al. 2020). While information dense, e.g., the entity identifiers, entity types, predicates and predicate sources are all included in this file, these source data are not very usable in their current state and require learning specialized, unexplained notation.

Figure 5.1 shows the same data, but in a more intuitive, immediately readable format, that is, as a KG with entity (node) metadata added. (Triple, i.e., edge, metadata was omitted to reduce clutter.) Figure 5.1 serves as an example of the desired output data from our proposed KG-CURATE workflow.

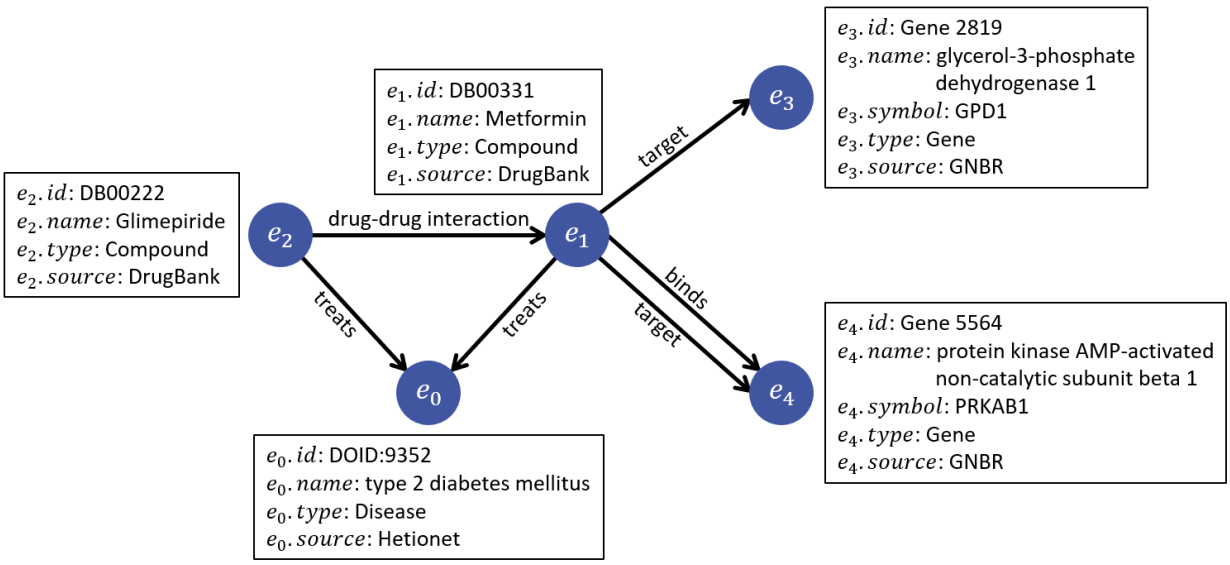


Figure 5.1: The DRKG triples presented in Table 5.1, shown here in the knowledge-graph format with additional entity (node) metadata.

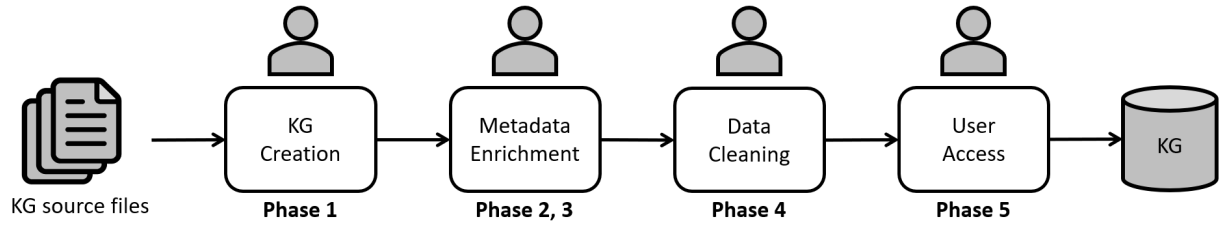


Figure 5.2: An overview of the proposed knowledge-graph curation workflow KG-CURATE: Knowledge Graph Curation for Usability, Readability, and Accessibility for Task Expectations. The workflow transforms input knowledge graph source files into a complete, cleaned knowledge graph. Each box represents a specific step of the workflow and is labeled with the workflow phase(s) that address that step. See Sections 5.3.1–5.3.5 for specific details about the five phases. We recommend treating the workflow as a human-in-the-loop workflow, working directly with domain experts and anticipated users of the knowledge graph to ensure high-quality results.

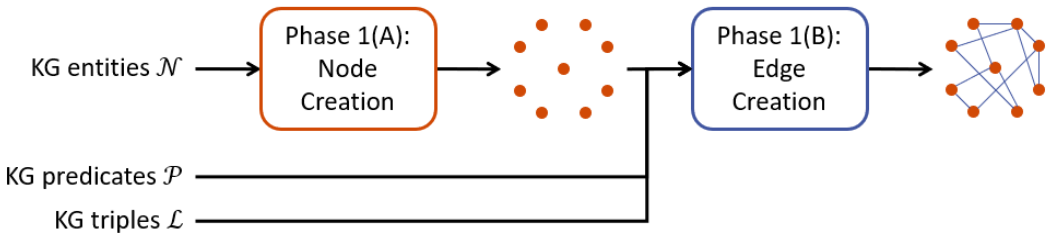


Figure 5.3: Phase 1: Knowledge-graph entity (node) and triple (edge) creation. This phase has three inputs: the knowledge graph (KG) entities $\mathcal{N}$, predicates $\mathcal{P}$, and triples $\mathcal{L}$. The output is a knowledge graph containing the corresponding nodes and edges.

5.3 The Phases of the KG-CURATE Workflow

We now introduce our proposed KG-CURATE (Knowledge Graph Curation for Usability, Readability, and Accessibility for Task Expectations) workflow. It consists of five phases, one for each objective presented in Section 5.2.1. Figure 5.2 provides an overview of the workflow, showing the four key steps, along with the phases that address each step. The workflow takes as an input a set of knowledge-graph (KG) source files and transforms those into a complete, cleaned KG via the five phases with input from humans-in-the-loop, e.g., domain experts and anticipated users of the KG. See Sections 5.3.1–5.3.5 for the details of each of the five phases.

5.3.1 Phase 1: Knowledge-Graph Node and Edge Creation

Phase 1 addresses the goal of creating a KG $\mathcal{G} = (\mathcal{N}, \mathcal{P}, \mathcal{L})$, i.e., goal (a) of the problem statement of Section 5.1. Figure 5.3 outlines this phase, which has two subphases: 1(A) create all the nodes in the KG, and 1(B) create all the KG edges.

To accomplish the two subphases, Phase 1 of the workflow requires three inputs: the KG entities $\mathcal{N}$, predicates $\mathcal{P}$, and triples $\mathcal{L}$. These sets can be extracted from the KG source files. E.g., KG triples are frequently stored in a file with delimiter-separated values listing one (s, p, o) triple per line, so extracting the sets $\mathcal{N}$, $\mathcal{P}$, and $\mathcal{L}$ is a simple string-parsing task. In Phase 1(A), a single KG node is created for each entity $e \in \mathcal{N}$; in Phase 1(B), an edge of predicate p is created between nodes s and o for each triple $(s, p, o) \in \mathcal{L}$. The output of Phase 1 is a KG $\mathcal{G} = (\mathcal{N}, \mathcal{P}, \mathcal{L})$ without metadata, which satisfies goal (a) of the problem statement in Section 5.1.

We now detail the Phase 1 process with respect to our test KG, DRKG. The process is generalizable to other KGs and tools; we provide the specific details of our test case as a template. As the input, we used the sets $\mathcal{N}$, $\mathcal{P}$, and $\mathcal{L}$ of DRKG entities, predicates, and triples, which we had generated from the DRKG *triple file*, see Section 5.2.3.1. The triple file lists one

triple per line in tab-separated format, i.e., the triple (s, p, o) is formatted as "s <*tab*> p <*tab*> o <*newline*>." We generated $\mathscr{N}$ from the first and third columns, $\mathscr{P}$ from the second column, and $\mathscr{L}$ from all three columns.

To complete Phase 1(A), we execute the following *node-creation query* for each entity $e \in \mathscr{N}$:[4]

```
CREATE (n)
SET n.id = e
RETURN n
```

This query creates in the KG a new, isolated node that has the unique identifier e; we refer to this node as the e-node. In future queries, the e-node can be found by its unique id e.

To complete Phase 1(B), we execute the following *edge-creation query* for each triple $(s, p, o) \in \mathscr{L}$:

```
MATCH (n_1),(n_2)
WHERE n_1.id = s AND n_2.id = o
CREATE (n_1)-[r : p]->(n_2)
RETURN r
```

This query first finds the nodes $\mathtt{n}_1$ and $\mathtt{n}_2$ corresponding to the subject s and object o, respectively. Then, it creates an edge with the predicate label p from $\mathtt{n}_1$ to $\mathtt{n}_2$.

Unfortunately, numerous unpredictable issues can arise during this phase, preventing the creation of some nodes or edges, e.g., DBMS connection timeout or failure, entity- or triple-parsing errors, accidental duplicate identifiers, unexpected query-formatting issues due to abnormal input data, and so on. In our test case, we experienced DBMS connection timeouts due to the large number of queries that had to be issued in Phases 1(A) and 1(B). Fortunately, overcoming such unpredictable issues is straightforward, by maintaining as safeguards the set $\mathscr{N}_u^1$ of unsuccessfully created entities and the set $\mathscr{L}_u^1$ of unsuccessfully created triples. If a Phase-1(A) node-creation query or a Phase-1(B) edge-creation query returns no results, this means that the entity or triple was not created successfully and is to be added to the corresponding set $\mathscr{N}_u^1$ or $\mathscr{L}_u^1$.

After completing a single round of Phase 1(A), i.e., executing the Phase-1(A) node-creation query for each entity in $\mathscr{N}$ once, the set of unsuccessful entities $\mathscr{N}_u^1$ needs to be resolved. This is done by iteratively (i) analyzing the unsuccessful entities to solve any connection, parsing,

[4]All the queries are shown in the Cypher query language.

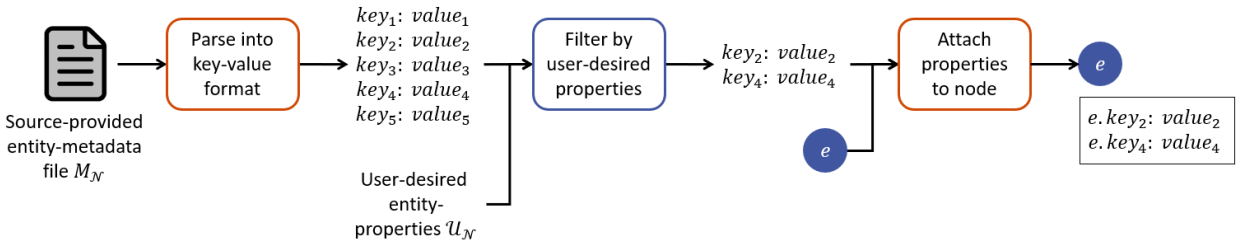


Figure 5.4: Phase 2(A): adding source-provided metadata to the knowledge-graph nodes; for simplicity, the process is shown only for a single entity e. The inputs to this phase are (i) the knowledge graph entities $\mathcal{N}$ – in this case, just a single entity $e \in \mathcal{N}$, (ii) the source-provided entity-metadata file $M_{\mathcal{N}}$, and (iii) the user-desired entity-properties $\mathcal{U}_{\mathcal{N}}$. Each box represents a single step of the process, and the inputs and outputs of each step are shown in between. The final output is the entity e, which has been enriched by the appropriate source-provided metadata. The complete Phase 2(A) outputs the entire enriched entity set.

or other error that arose during the initial round, and (ii) running another round of Phase 1(A) on just the set $\mathcal{N}_u^1$ to retry the node creation. Each subsequent round creates a new set of unsuccessful entities $\mathcal{N}_u^1$. Once the set $\mathcal{N}_u^1$ is empty, Phase 1(A) is complete.

Phase 1(B) can begin only after Phase 1(A) is complete. Otherwise, the `MATCH` clause of some Phase-1(B) queries might fail to find their s-nodes and/or o-nodes. Once complete, Phase 1(A) has created a KG node for each of the entities in $\mathcal{N}$, and the Phase-1(B) edge-creation queries are guaranteed to find their s- and o-nodes. Phase 1(B) is executed in the same iterative manner as Phase 1(A) until the set $\mathcal{L}_u^1$ is empty.

After completing Phases 1(A) and 1(B), the basic KG structure has been built, i.e., the graph/network structure is complete, without any metadata present. For DRKG, the graph that we obtained via Phase 1 has 97,238 nodes of 13 different types and 5,874,261 triples over 107 different predicate types, which matches the size reported in Section 5.2.3.1. This satisfies goal (a) of the problem statement of Section 5.1 and concludes Phase 1.

5.3.2 Phase 2: KG Enrichment with Source-Provided Metadata

Phase 2 partially addresses goal (b) of the problem statement of Section 5.1: enrich $\mathcal{G}$ with the user-desired entity- and triple-properties $\mathcal{U}_{\mathcal{N}}$ and $\mathcal{U}_{\mathcal{L}}$. Similarly to Phase 1, Phase 2 has two subphases: addition of source-provided metadata to the KG nodes in 2(A) and to the KG edges in 2(B).

To accomplish these subphases, Phase 2 requires multiple inputs: the KG entities $\mathcal{N}$, predicates $\mathcal{P}$, and triples $\mathcal{L}$; the user-desired entity-properties $\mathcal{U}_{\mathcal{N}}$ and triple-properties $\mathcal{U}_{\mathcal{L}}$; and the source-provided entity-metadata file $M_{\mathcal{N}}$ and triple-metadata file $M_{\mathcal{L}}$. In Phase 2(A), $M_{\mathcal{N}}$

is parsed and then filtered to include only entity-properties from $\mathscr{U}_{\mathscr{N}}$. Then, this metadata is attached to the corresponding KG entities (nodes). In Phase 2(B), the same process is undertaken for $M_{\mathscr{L}}$ with respect to $\mathscr{U}_{\mathscr{L}}$. The output of Phase 2 is a KG $\mathscr{G}$ that has been enriched by $\mathscr{U}_{\mathscr{N}} \cap P_{M_{\mathscr{N}}}$ and $\mathscr{U}_{\mathscr{L}} \cap P_{M_{\mathscr{L}}}$, where $P_{M_{\mathscr{N}}}$ and $P_{M_{\mathscr{L}}}$ are the entity- and triple-properties contained in $M_{\mathscr{N}}$ and $M_{\mathscr{L}}$. This KG partially satisfies goal (b) of the problem statement in Section 5.1, which will be completed in Phase 3.

We now detail the Phase 2 process with respect to our test KG DRKG. Again, the process is generalizable to other KGs and tools. As inputs, we used the sets $\mathscr{N}$, $\mathscr{P}$, and $\mathscr{L}$ of DRKG entities, predicates, and triples, the user-desired entity-properties $\mathscr{U}_{\mathscr{N}}$ and triple-properties $\mathscr{U}_{\mathscr{L}}$ detailed in Section 5.2.3, and the entity- and triple-metadata files $M_{\mathscr{N}}$ and $M_{\mathscr{L}}$. $M_{\mathscr{N}}$ is the *entity-source mapping file*, and $M_{\mathscr{L}}$ is the *predicate glossary file* (see Section 5.2.3.1 for the details).

Figure 5.4 outlines the Phase 2(A) process for a single entity. In this phase, we parse the file $M_{\mathscr{N}}$, which contains for each entity $e \in \mathscr{N}$ the values corresponding to entity-properties $P_{M_{\mathscr{N}}}$. The file is parsed into a key-value format in which for each entity $e \in \mathscr{N}$ we create $e_{prop} = \{key_1 : value_1, \ldots, key_m : value_m\}$, where $key_i \in P_{M_{\mathscr{N}}}$ and $value_i$ is the appropriate property value for e. In $P_{M_{\mathscr{N}}} = \{source, type\}$ for DRKG, *source* indicates the original source of each entity, and *type* indicates the class of the entity, e.g., *compound*, *disease*, or *side effect*. Once the source-provided entity-metadata is parsed into the key-value format, we filter it by the user-desired set of entity-properties. That is, we eliminate unnecessary metadata, keeping in e_{prop} only the values of the properties in $P_{M_{\mathscr{N}}} \cap \mathscr{U}_{\mathscr{N}}$.

Next, for each entity $e \in \mathscr{N}$ and its corresponding metadata $e_{prop} = \{key_1 : value_1, \ldots, key_m : value_m\}$, we execute the following *node-metadata attachment query*:

```
MATCH (n)
WHERE n.id = e
SET n+= {key_1 : value_1, ..., key_m : value_m}
RETURN n.key_1, ..., n.key_m
```

This query first finds the appropriate e-node, and then adds to it a new property for each key-value pair in e_{prop}.

Phase 2(B) proceeds similarly to Phase 2(A). First, the file $M_{\mathscr{L}}$ is parsed into key-value pairs such that for each triple $t = (s, p, o) \in \mathscr{L}$, we have $t_{prop} = \{key_1 : \texttt{value}_1, \ldots, key_m : value_m\}$. Here, $key_i \in P_{M_{\mathscr{L}}}$ and $value_i$ is the appropriate property value for t. For DRKG, $P_{M_{\mathscr{L}}} = \{source, connectedEntityTypes, interactionType, description\}$. Again, we filter the extracted properties so that t_{prop} will contain only values for the properties in $P_{M_{\mathscr{L}}} \cap \mathscr{U}_{\mathscr{L}}$.

Finally, for each triple $t = (s, p, o) \in \mathscr{L}$ and its corresponding metadata $t_{prop} = \{key_1 : value_1, \ldots, key_m : value_m\}$, we add the metadata to the KG by executing the following *edge-metadata attachment query:*

$$\texttt{MATCH}\,(\texttt{n}_1)-[\texttt{r} : p]->(\texttt{n}_2)$$
$$\texttt{WHERE}\,\texttt{n}_1.\texttt{id} = s\ \texttt{AND}\,\texttt{n}_2.\texttt{id} = o$$
$$\texttt{SET}\,\texttt{r}+= \{key_1 : value_1, \ldots, key_m : value_m\}$$
$$\texttt{RETURN}\,\texttt{r}.key_1, \ldots, \texttt{r}.key_m$$

This is similar to the Phase-2(A) *node-metadata attachment query*, except instead of finding a node, we find the entire (s, p, o)-triple in the KG. Then, instead of adding properties to a node, we add properties to the relationship edge.

Just as in Phase 1, many issues could arise during this phase preventing metadata from being attached to its corresponding entity or triple, see Section 5.3.1. In particular, in our test case we faced query-formatting issues due to abnormal input data, e.g., special characters and illegal characters. Thus, we recommend the same safeguards here as in Phase 1: a set $\mathscr{N}_u^2$ of entities whose metadata-attachment was unsuccessful and a set $\mathscr{L}_u^2$ of triples whose metadata-attachment was unsuccessful.

With the help of $\mathscr{N}_u^2$ and $\mathscr{L}_u^2$, the node- and edge-property values returned by Phase-2(A) and Phase-2(B) queries can be checked against the correct values to verify if the metadata was attached successfully. If it was not, then the node or edge is added to the corresponding unsuccessful set. After analyzing the unsuccessful sets and solving any issues that arose, additional rounds of Phase 2(A) and Phase 2(B) should be executed until $\mathscr{N}_u^2$ and $\mathscr{L}_u^2$ are empty. In our test case, we replaced illegal characters with legal characters and modified the query formatting to allow for special characters.

Unlike Phase 1, Phases 2(A) and 2(B) need not be sequential. In fact, executing them concurrently could speed up Phase 2 of the workflow. After completing Phases 2(A) and 2(B), the basic KG structure produced by Phase 1 has been enriched with the metadata provided by the KG source, i.e., $\mathscr{U}_{\mathscr{N}} \cap P_{M_{\mathscr{N}}}$ and $\mathscr{U}_{\mathscr{L}} \cap P_{M_{\mathscr{L}}}$. We then observed that these properties were added to 100% of the entities or triples, thus passing the tests of satisfaction, see Section 5.2.3 for details. This partially satisfies goal (b) of the problem statement of Section 5.1 and concludes Phase 2.

5.3.3 Phase 3: KG Enrichment with External Metadata

Phase 3 also partially addresses goal (b) of the problem statement of Section 5.1. In Phase 3 we fill in the knowledge gaps from the source data, i.e., we enrich the KG entities and triples

with the remaining properties from the sets $\mathscr{U}_{\mathscr{N}}$ and $\mathscr{U}_{\mathscr{L}}$ that were not provided by the KG source. This is accomplished mainly via application programming interface (API) calls and web scraping. In our DRKG test case, the entire set $\mathscr{U}_{\mathscr{L}}$ was provided by the source. Thus, in this section we focus only on adding the remaining properties from $\mathscr{U}_{\mathscr{N}}$. It is straightforward to extend the process to adding triple-properties.

To accomplish Phase 3, the following inputs are required: the KG entities $\mathscr{N}$ and the remaining user-desired entity-properties $\mathscr{U}_{\mathscr{N}} \setminus P_{M_{\mathscr{N}}}$ that were not provided by the KG source (see Section 5.2.3). For each property key $k \in \mathscr{U}_{\mathscr{N}} \setminus P_{M_{\mathscr{N}}}$, there are four main steps:

(i) Identify the original source s_e for each entity $e \in \mathscr{N}$;

(ii) Access the original source s_e;

(iii) Extract the value v of the desired property k from the original source s_e; and

(iv) Attach key-value pair (k, v) as a property of the e-node.

Towards accomplishing Step (i), we note that in many cases the KG source provides the original source of each entity. If so, then the original source s_e of the entity e was attached to the e-node in Phase 2, see Section 5.3.2. Let `source` be the property key whose value is s_e. Then s_e can be retrieved by executing the following *node-source retrieval query*:

```
MATCH (n)
WHERE n.id = e
RETURN n.source
```

For our test case, while the DRKG files did provide entity sources, we found that these sources were not helpful: Some were just references to papers that used data sources containing the entity, while others were KGs or databases that were not publicly accessible. So, instead of using these sources, we had to do a source search, just as if the original source of each entity was not provided. To be adequate, sources need to contain sufficient entity-specific information. We recommend identifying large domain-specific databases, as these are likely to contain information about many entities in $\mathscr{N}$, thereby minimizing the number of sources that must be dealt with during this phase. For DRKG, we identified several such sources, e.g., the National Center for Biotechnology Information (NCBI),[5] Chemical Entities of Biological

[5] `https://www.ncbi.nlm.nih.gov/`

Interest,[6] DrugBank,[7] the Human Disease Ontology,[8] and Medical Subject Headings (MeSH).[9]

For Step (ii), an appropriate access method for each original source must be identified. Many sources provide public-facing APIs, e.g., MeSH.[10] If there is a provided API call that retrieves the value v of the desired property k, then the access method is that API call. The result of such an API call contains a response message in a standard format, e.g., Extensible Markup Language (XML) or JavaScript Object Notation (JSON). If a source does not provide a public-facing API or if there is no appropriate API call, as is the case of, e.g., NCBI, then the access method is to scrape the web page containing the value v of the desired property k. This can be accomplished via a Hypertext Transfer Protocol (HTTP) `GET` request of the web page. Such a request provides the HTML source code of the web page.

For Step (iii), a custom parsing method must be implemented for each source. The method parses the results of the API call or the HTTP `GET` request to extract just the value v of the desired property k. Python, our programming language of choice, has nice packages for parsing, e.g., `Beautiful Soup` (Richardson 2020). For each entity $e \in \mathcal{N}$, the appropriate API call or HTTP `GET` request is made, and then the corresponding parsing method is used to extract the desired value v.

The parsing methods written for our sources follow the format outlined in Algorithm 1.[11] For each entity source s_e, we identify a *startFlag* and an *endFlag*, i.e., the source text that marks the start and end of the desired property value; these serve as inputs to the method. We make a `GET` request for s_e; the result is stored in *response* (see line 1). Then we identify the index of *startFlag* in *response* (see line 2), and trim *response* such that it starts immediately after the *startFlag* (see line 3). Next, we identify the index of *endFlag* in *response* (see line 4), and trim *response* such that it ends immediately before the *endFlag* (see line 5). Finally, the extracted value v is returned (see line 6).

For Step (iv), the newly extracted key-value pair (k, v) is attached to the e-node as a property

[6] `http://www.ebi.ac.uk/chebi/`
[7] `https://go.drugbank.com/`
[8] `http://www.disease-ontology.org`
[9] `https://id.nlm.nih.gov/mesh/`
[10] `https://id.nlm.nih.gov/mesh/swagger/ui`
[11] The algorithm is shown for the case in which the access method is an HTTP `GET` request. If the access method is an API call, then line 1 is simply replaced by the appropriate API call.

Algorithm 2: Extract property value from source

Input : Entity source s_e, property value start flag *startFlag*, and property value end flag *endFlag*.

Output : The value v of property k for entity e.

1 *response* $\leftarrow$ `GET` s_e;
2 $i_{start} \leftarrow$ index of *startFlag* in *response*;
3 *response* $\leftarrow$ *response*$[\ i_{start} + length(startFlag) :]$;
4 $i_{end} \leftarrow$ index of *endFlag* in *response*;
5 $v \leftarrow$ *response*$[\ : i_{end}\ ]$;
6 **return** v;

via the following *node-metadata attachment query*:

```
MATCH (n)
WHERE n.id = e
SET n.k = v
RETURN n.k
```

As with Phases 1 and 2, see Sections 5.3.1 and 5.3.2, some issues may arise, preventing the success of Phase 3. In fact, Phase 3 is more prone to issues, because external sources are likely less uniform and organized than the KG source files are. To this end, we propose the same safeguard as in Phases 1 and 2: maintaining a set $\mathcal{N}_u^3$ of entities for which Phase 3 was unsuccessful at any step. As before, issues that arose should be solved, and then additional rounds of Phase 3 should be executed until $\mathcal{N}_u^3$ is empty. In our test case with DRKG, we experienced similar issues to those faced in Phases 1 and 2, so we solved them in the same manner.

Phase 3 is repeated for each property that remains in the sets of user-desired properties. In many cases, it is straightforward to extract multiple property values at once, which may speed up Phase 3. In our test case, we added the property $name$ to about 98% of the entities in DRKG; the remaining 2% of the entities were either unnamed in every identified source, or we were unable to identify a source that recognized the entity identifier. We also added the properties *InChIKey* and *SMILES* to about 60% of the *compound* entities. Thus, by our evaluation the tests of satisfaction were passed for each of these properties, see Section 5.2.3 for details.

Phase 3 is complete once the set $\mathcal{N}_u^3$ is empty upon completing Steps (i)–(iv). The result is a KG that has been completely enriched with the user-desired entity-properties $\mathcal{U}_{\mathcal{N}}$ and triple-properties $\mathcal{U}_{\mathcal{L}}$. At this point, goal (b) of the problem statement of Section 5.1 has been

satisfied, and Phase 3 is concluded.

5.3.4 Phase 4: Readability and Understandability Verification

After the KG has been built and the desired metadata has been added, the KG must be reviewed for readability and understandability. Unfortunately, in Phases 1–3 (see Sections 5.3.1–5.3.3) some of the ingested data may be "dirty." E.g., the terminology may be poor or irregular, unknown acronyms may be present, or some string parsing may have been done improperly. Phase 4 focuses on data cleaning, i.e., on identifying and resolving these anomalies to improve the quality of the KG. After all, the more readable and understandable the information is, the more useful the KG will be.

Clearly, it would require painstaking manual effort to review all the information in the KG for cleanliness. Instead, we recommend reviewing a representative sample of the data. At the very least, such a sample should include: (i) nodes of each type, (ii) predicates of each type, and (iii) nodes from each original source. In some cases, this may not be an exhaustive list of requirements for a representative sample, depending on the data domain and other data features.

If issues are found when reviewing the representative sample, then a few other related elements of the KG should be examined, e.g., predicates of the same type or nodes from the same original source, to determine if the issues appear to be widespread. If they do not, then just the singular instance should be fixed. It the issues do appear elsewhere, then we recommend executing another round of the corresponding Phase (1, 2, or 3) for that group of nodes or edges.

In our test case with DRKG, we reviewed a representative sample and identified two key features of the KG that needed improvement to increase readability and understandability. The first issue was with the predicate names. In the DRKG source, 60 of the predicate names are non-intuitive acronyms. (An example of an unreadable acronym predicate from DRKG is *CtD*, which stands for *Compound-treats-Disease*.) These cannot be readily understood when they appear in query answers, so we sought to remedy this issue. Fortunately, the DRKG source files provided the original source for each predicate type. This allowed us to go to the original source for each of the acronym predicates to identify the full predicate name. We changed each acronym predicate p in the KG to its full name p' via the following *predicate-modification query*:

$$\texttt{CALL apoc.refactor.rename.type}(p, p').$$

This query uses the `apoc` library (The Neo4j Team 2022a) of Neo4j.

The second feature that we analyzed was the potential alignment of DRKG with the Biolink ontology (Unni et al. 2022). Aligning a KG to an ontology is advantageous for several reasons, e.g., it can enable automated reasoning over the data, and can also enable the graph to be compared to other data sources that align with the same ontology. Thus, users familiar with the ontology and its terms will have a better understanding of the KG. The Biolink ontology is used by other biomedical databases, e.g., the ROBOKOP KG (Bizon et al. 2019), so we elected to use it as well. Initially, only eight DRKG predicates had defined mappings to the Biolink ontology. However, thanks to our collaboration with the Biolink ontology developers, the Biolink ontology now includes mappings to 22 DRKG predicates. We are continuing to work with the Biolink ontology developers to map the remaining DRKG predicates to the ontology.

These two features of DRKG can serve as examples of features that can be analyzed and enhanced in Phase 4, but they are certainly not exhaustive. The specific features to focus on in Phase 4 can be both domain- and task-specific.

While Phase 4 will hopefully resolve most issues with KG readability and understandability, we also recommend establishing a protocol for users to submit errors, issues, and other feedback. Since only a sample of the KG would have been reviewed during Phase 4, this will allow for greater coverage of the KG, thus enabling further improvements.

5.3.5 Phase 5: Making the Knowledge Graph Accessible

Phase 5 addresses goal (c) of the problem statement of Section 5.1. This final phase of the workflow entails making the KG accessible. Ideally, the KG will be accessible to the general public for their use. However, some data could be sensitive or confidential; in this case, we focus on making sure that the KG is accessible to all project members and critical parties. One of the simplest ways to do this is by uploading the KG to a cloud service. For our test case, we selected Google Cloud (Google LLC 2022), see Section 5.2.2. First, on a Google Cloud machine we set up Neo4j (The Neo4j Team 2022c), our graph DBMS of choice (see Section 5.2.2). Then we set up the version of DRKG that was created during Phases 1–4, see Sections 5.3.1–5.3.4, by creating and uploading a database dump file.[12] Finally, we shared with our users the Google Cloud machine information and the Neo4j login credentials, which gave them 24/7 access to the KG. Alternatively, the graph instance could be made completely publicly available by removing any required authentication.

[12]Instructions for using database dump files in Neo4j can be found at `https://neo4j.com/docs/operations-manual/current/backup-restore/`.

5.4 Human-in-the-Loop in the Workflow

In the proposed KG-CURATE workflow, there are many points at which involving anticipated users and/or domain experts could be very beneficial. For example, Phases 1 and 2 require parsing the knowledge graph (KG) source and metadata files, see Sections 5.3.1 and 5.3.2. Consulting anticipated users and domain experts could provide a greater understanding of the source-file formats, while also revealing which parts of the data are of particular interest or importance. Feedback could even help with understanding how to read certain acronyms or entity labels, as the format used in the source files may be standard in the specific field. In Phases 1 and 2 of our DRKG test case, we consulted with domain experts regarding the source-file formatting to understand the format of entity identifiers and to confirm the meaning of the triple-properties. Moreover, the users and experts provided the sets of user-desired entity- and triple-properties, $\mathscr{U}_{\mathscr{N}}$ and $\mathscr{U}_{\mathscr{L}}$, used in Phases 2–3, and aided in evaluating the tests of satisfaction.

Phase 3 of KG-CURATE (see Section 5.3.3) provides many opportunities to consult a human-in-the-loop. In Step (i) of Phase 3, if the original entity sources are not provided by the KG source, then domain experts are likely capable of identifying common, reliable sources. In Steps (ii)–(iii) of Phase 3, users and experts could help identify the specific property of interest. In those cases where a property has multiple commonly-used names, consulting a user or expert can help to clarify what information is desired. Moreover, the users and/or experts could assist in parsing out specific property values, by indicating which portions are important or which terminology is preferred. In Phase 3 of the DRKG test case, we involved domain experts in each of these instances. They were able to recognize entity identifiers and thus point us to appropriate sources. They also helped us to identify property values and suggested the best format to store each property.

Phase 4 of KG-CURATE (see Section 5.3.4) is also well suited for a human-in-the-loop style approach. Anticipated users and domain experts are uniquely qualified to identify issues with the KG data. In particular, they could assist with any issues regarding terminology and acronyms, e.g., adjusting terminology to conform to domain standards, or translating acronyms for readability. Overall, anticipated users and domain experts will best know what changes are necessary, as their KG readability and understanding is the most crucial for making the KG usable. In Phase 4 of our DRKG test case, domain experts and users pointed out to us the two issues that we then addressed: (1) changing non-intuitive acronyms to their full forms, and (2) mapping DRKG predicates to the Biolink ontology. According to the experts and users, addressing these issues made the KG more useful to them.

In summary, to achieve the highest-quality KG, we recommend treating the KG-CURATE

workflow as a human-in-the-loop workflow, gaining insight and suggestions from anticipated users and/or domain experts whenever possible.

5.5 Challenges and Lessons Learned

We now discuss the challenges that we faced in implementing and testing our proposed KG-CURATE workflow, as well as the lessons learned during the experience.

5.5.1 Identifying Original Sources for the KG Entities

In the proposed workflow, Step (i) of Phase 3 requires identifying original sources for each knowledge graph (KG) entity, see Section 5.3.3 for details. Unfortunately, if the original entity sources are not provided in the KG source, finding a source can prove to be challenging without domain-expert involvement. This is especially the case if the KG source does not provide a description or understandable metadata for the entity. This was the case for DRKG, since entities were provided with an identifier but no name and, in many cases, no original source. In our experience, many of the entity identifiers were difficult to work with for a number of reasons, e.g., the notation was not understandable, the identifier was retired or replaced, or the entity had multiple identifier mappings. Fortunately, we were able to alleviate multiple of these issues through collaboration with domain experts. For example, the experts were familiar with the identifier notation, so they were able to help us understand the meaning, which then led us to reliable entity sources. Moreover, being aware of the retired-identifier and multiple-identifier phenomena, they recommended we update these entity identifiers to the current identifiers. This made it much easier to find other entity sources and metadata.

5.5.2 Dealing with Conflicting Terminology

In the biomedical domain and, conceivably, in other domains, there are many terms with synonyms that are used interchangeably. This can cause confusion at several points in the workflow, e.g., not realizing that the KG source-provided metadata includes a certain property, finding original entity sources that have different entity-type distinctions, and not being able to identify an entity property from the original entity source. Fortunately, again, domain experts tend to be familiar with these interchangeable terms; thus, they can provide invaluable guidance in aligning the terminology.

5.5.3 Recommendations for Knowledge-Graph Curators

From our experience with implementing and testing the KG-CURATE workflow on DRKG, we have identified a few actions that KG curators can take to make their KGs more usable, and thus more impactful. The first of these is to provide publicly available, version-labeled KG database dump files. If users are comfortable with the compatible DBMS, the dump file can completely eliminate the need for Phase 1 of the proposed workflow, see Section 5.3.1. Labeling database dump versions allows for changes made to the KG to be tracked over time, which is useful in many applications, e.g., link prediction (Paulheim 2017). The second recommendation is to maintain all the original-source metadata during KG curation. If entities, predicates, or triples are imported from original sources that provide metadata, that metadata should be imported as well. This can allow users to save time during Phase 3 (or eliminate the need for Phase 3 altogether) since more metadata can be covered by Phase 2. Similarly, maintaining the original entity source as metadata is valuable, since this can reduce the time spent on Step (i) of Phase 3. Our final recommendation is to provide detailed documentation regarding the KG data and source-file formatting. This could save much of the time that users would otherwise spend attempting to decipher source-file formatting, KG formatting, and entity-metadata meaning.

5.6 Summary

In this chapter we proposed a workflow called KG-CURATE for setting up a clean, public instance of a knowledge graph (KG). We also detailed our specific work with the biomedical KG DRKG (Ioannidis et al. 2020), which was our test case for the proposed workflow. Moreover, we discussed the challenges we faced, as well as the lessons we learned from our experience. Based on our experience, we have also provided recommendations for KG curators to make their KGs more usable and impactful. Our proposed workflow enables KG users to enhance existing KGs such that they are well suited for their intended domains and use cases. We also recommend that the KG-CURATE workflow be treated as a case of human-in-the-loop, since anticipated users and/or domain experts can provide invaluable insight into the KG data and design. We posit that the proposed workflow can enable KG users to strategically and systematically prepare KGs for their intended use cases, which can increase the productivity of their projects, as well as the quality of their results. Most importantly, the proposed workflow can increase the usefulness of KGs, thus encouraging greater use and exploration. Future work includes analysis of the impact of each workflow phase.

CHAPTER

6

LOSSLESS KNOWLEDGE-GRAPH COMPRESSION FOR IMPROVING THE EFFICIENCY AND EFFECTIVENESS OF RULE MINING

Portions of this chapter have been published in:

> **Kara Schatz**, Alexander Tropsha, Rada Chirkova, "GAME: Improving Efficiency and Effectiveness of Knowledge-Graph Rule Mining via Data Reduction," in *2023 IEEE International Conference on Big Data (Big Data)*, pp. 4248–4257, IEEE, December 2023.

In this chapter we introduce our proposed approach, called Graph Abstraction for Mining Efficiency (GAME+), that addresses the problem of lossless knowledge-graph compression, with the goal of enabling efficient, yet effective, knowledge-graph analytics. That is, analytics on the compressed knowledge graphs can be performed quicker, but just as effectively as on the original knowledge graphs. We propose that this approach be used during the third stage of the four C's pipeline presented in Chapter 1 (see Figure 1.1): *compression*.

This chapter is organized as follows. In Section 6.1 we formalize the problem of lossless KG

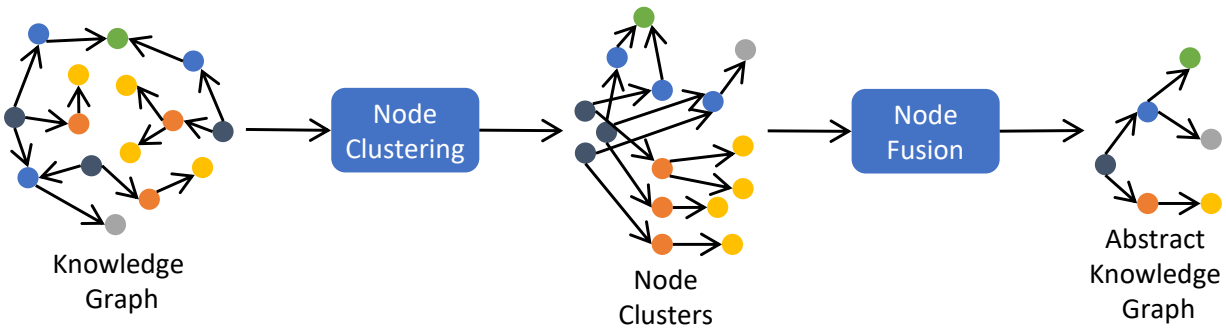


Figure 6.1: An overview of the proposed GAME+ approach for knowledge-graph abstraction (summarization): GAME+ reduces the input knowledge graph into an abstract (summarized) knowledge graph via the phases of node clustering and node fusion. See Sections 6.2.2–6.2.3 for specific details about these two phases.

compression. In Section 6.2 we introduce the proposed GAME+ approach, and in Section 6.3 we present and discuss our experimental results, followed by our case-study results in Section 6.4. Finally, we conclude in Section 6.5.

Relevant preliminaries for this chapter can be found in Sections 3.1–3.4.

6.1 Problem Statement

We now introduce the formal problem statement for *lossless knowledge-graph (KG) abstraction (summarization) for rule mining*. From a given KG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, a node-similarity function $S : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$, and a real-valued node-similarity threshold σ, construct an AKG $\mathcal{G}^A = (\mathcal{N}^A, \mathcal{T}^A, \phi^A, \mathcal{P}^A, \mathcal{L}^A)$ such that $\mathcal{G}^A$ (i) is smaller in size than $\mathcal{G}$, thus enabling more efficient rule mining than $\mathcal{G}$, and (ii) is obtained from $\mathcal{G}$ in a *lossless* manner, that is, $\mathcal{G}$ can be completely reconstructed from $\mathcal{G}^A$, and rule mining on $\mathcal{G}^A$ can recover all the rules that can be obtained from the original graph $\mathcal{G}$.

6.2 The Proposed GAME+ Approach

We now introduce our proposed Graph Abstraction for Mining Efficiency (GAME+) approach, which addresses the problem formulated in Section 6.1. Please see Figure 6.1 for a high-level overview of the GAME+ approach.

Algorithm 3: The GAME+ KG-Abstraction Approach.

Data: KG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, node-similarity function $S : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$, node-similarity threshold σ, parameter c that determines which node-clustering approach will be called as a subroutine.

Result: AKG $\mathcal{G}^A = (\mathcal{N}^A, \mathcal{T}^A, \phi^A, \mathcal{P}^A, \mathcal{L}^A)$.

1 $\mathcal{C} \leftarrow \emptyset$; // set of node clusters
// Phase 1: node clustering:
2 **if** $c = relaxed$ **then**
3 $\quad \mathcal{C} \leftarrow RelaxedNodeClustering(\mathcal{G}, S, \sigma)$;
4 **else**
5 $\quad \mathcal{C} \leftarrow StrictNodeClustering(\mathcal{G}, S, \sigma)$;
// Phase 2: node fusion:
6 $\mathcal{G}^A \leftarrow NodeFusion(\mathcal{G}, \mathcal{C})$;
7 **return** $\mathcal{G}^A$;

6.2.1 The Proposed Knowledge-Graph Abstraction Framework

Our proposed GAME+ approach is outlined in Algorithm 3. It accepts as inputs a knowledge graph *(KG)* $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, a node-similarity function $S : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$, a node-similarity threshold σ, and a parameter $c \in \{relaxed, strict\}$ that determines which node clustering approach will be used. On completion, Algorithm 3 outputs an AKG $\mathcal{G}^A = (\mathcal{N}^A, \mathcal{T}^A, \phi^A, \mathcal{P}^A, \mathcal{L}^A)$ (line 7).

The GAME+ approach consists of two main phases: (1) *node clustering* and (2) *node fusion*. In the node-clustering phase, GAME+ clusters the nodes in the input KG $\mathcal{G}$ according to the given similarity function S and threshold σ, see Section 6.2.2 for the details. The nature of the clustering depends on the choice of the node-clustering approach (*relaxed* or *strict*), governed by the input parameter c (lines 2–5). These clustering approaches are presented in Sections 6.2.2.1 and 6.2.2.2, respectively. In the node-fusion phase, GAME+ fuses the clustered nodes together, along with their incident edges, to form consolidated nodes (entities) and edges (triples) (line 6), see Section 6.2.3 for the details.

6.2.2 The Node-Clustering Phase of the GAME+ Approach

The proposed node-clustering algorithms are based on node similarity; they work with any given *similarity function* $S : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$ that assigns a real-valued *similarity score* to each pair of nodes from $\mathcal{N}$. In this work, we have used the *1-hop neighborhood similarity function*, see Section 6.3.1.3. The algorithms also accept as an input a node-similarity threshold σ; we say that two nodes n, m are σ-similar if their similarity score meets or exceeds this threshold, i.e.,

$S(n,m) \geq \sigma$.

We present two node-clustering algorithms that differ in their incorporation of σ. The *relaxed node-clustering algorithm* ensures that each node in a cluster C is σ-similar to *at least one* other node in C (see Section 6.2.2.1), whereas the *strict node-clustering algorithm* ensures that each node in a cluster C is σ-similar to *all* the other nodes in C (see Section 6.2.2.2). Note that these two algorithms are equivalent when $\sigma = 1.0$. Selection of the desired node-clustering approach enables control over the nature of the resulting node clusters. The two approaches present a natural tradeoff between the strictness of the node-similarity threshold and the time taken to perform node clustering. See Sections 6.2.2.1–6.2.2.2 for more details on this tradeoff.

6.2.2.1 The Relaxed Node-Clustering Approach

Algorithm 4: Phase 1 of GAME+: relaxed node clustering.

Data: KG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, node-similarity function $S : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$, node-similarity threshold σ.
Result: Set $\mathcal{C}$ of node clusters for the nodes in $\mathcal{G}$.

$\mathcal{C} \leftarrow \emptyset$; `// initializing the set of node clusters`
for $n \in \mathcal{N}$ **do**
  $\mathcal{N}_n^\sigma \leftarrow \{m \in \mathcal{N} \; s.t. \; S(n,m) \geq \sigma\}$;
  if $\mathcal{N}_n^\sigma \neq \emptyset$ **then**
    `// identify a cluster` C `for` n`:`
    if *n is in some cluster in $\mathcal{C}$* **then**
      `// case 1:` n `is already in a cluster:`
      $C \leftarrow$ cluster in $\mathcal{C}$ containing n;
    else if $\exists \; n' \in \mathcal{N}_n^\sigma \; s.t. \; n'$ *is σ-similar to n and is in some $C' \in \mathcal{C}$* **then**
      `// case 2: some nodes that are` σ`-similar to` n `are already in clusters:`
      $m \leftarrow$ the clustered node most similar to n;
      $C \leftarrow$ cluster in $\mathcal{C}$ containing m;
    else
      `// case 3: neither` n `nor its` σ`-similar nodes are in a cluster yet:`
      $C \leftarrow$ create new empty cluster for n;
      $\mathcal{C} \leftarrow \mathcal{C} \cup C$;
    `// populate the identified cluster` C`:`
    $C \leftarrow C \cup \{n\} \cup \{m \in \mathcal{N}_n^\sigma \; s.t. \; m$ is not in any $C' \in \mathcal{C}$ and m is σ-similar to $n\}$;
return $\mathcal{C}$;

Our first node-clustering approach, referred to as the *relaxed approach*, is outlined in Algorithm 4. Intuitively, the approach assigns each node in the KG $\mathscr{G}$ to a cluster if possible, in a greedy manner according to the similarity scoring. Algorithm 4 takes as inputs a KG $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$, a node-similarity function $S : \mathscr{N} \times \mathscr{N} \to \mathbb{R}$, and a node-similarity threshold σ. It outputs a set of node clusters $\mathscr{C}$ for $\mathscr{G}$, where each node n in a cluster $C \in \mathscr{C}$ is σ-similar to *at least one* other node in the cluster C.

The set of node clusters $\mathscr{C}$ is first initialized as the empty set (line 1) and is built up throughout the algorithm. The main loop of Algorithm 4 (lines 2–13) iterates through each node n in the KG $\mathscr{G}$. First, the similarity function S is used to identify all the nodes in $\mathscr{G}$ that are σ-similar to n; we denote the set of such nodes by $\mathscr{N}_n^\sigma$ (line 3). If n has no σ-similar nodes, then it does not belong in any cluster, as there are no nodes with which it can be clustered. However, if n has at least one σ-similar node in $\mathscr{N}_n^\sigma$ (line 4), then n and its σ-similar nodes belong in some cluster, which will be identified in lines 5–11.

Overall, there are three possible cases for identifying a cluster for n and its σ-similar nodes:

1. In case 1 (line 5), n has already been placed in a cluster during a previous iteration of the main loop. Thus, the appropriate cluster C for n and its σ-similar nodes is the cluster that contains n (line 6).

2. In case 2 (line 7), n has not yet been clustered, but at least one of its σ-similar nodes $n' \in \mathscr{N}_n^\sigma$ has been clustered. So, the appropriate cluster C for n and its σ-similar nodes is the cluster that contains the node m that is the most similar to n (lines 8–9).

3. In case 3, neither n nor any of its σ-similar nodes have been clustered (line 10). In this case, n and its σ-similar nodes belong in a completely new cluster C (line 11), which is added to the set of all clusters $\mathscr{C}$ (line 12).

Once the appropriate cluster C has been identified, n and all its non-clustered σ-similar nodes are added to C (line 13). Any nodes that are σ-similar to n and are already in a cluster are not affected here since no nodes should be placed in more than one cluster.

After the main loop is completed, each node of $\mathscr{G}$ has been examined and placed into an appropriate cluster if possible, that is, if they have any σ-similar nodes in $\mathscr{G}$. At this point, the resulting set of clusters $\mathscr{C}$ is output (line 14).

The runtime complexity of Algorithm 4 is dominated by either the square $|\mathscr{N}|^2$ in the total number of nodes in the KG $\mathscr{G}$, or by the runtime complexity $S(|\mathscr{N}|)$ of computing all the pairwise node-similarity scores via the similarity function S, in the case where $S(|\mathscr{N}|)$ asymptotically dominates $|\mathscr{N}|^2$:

Theorem 1. *The runtime complexity of Algorithm 4 is* $S(|\mathscr{N}|) + O(|\mathscr{N}|^2)$.

Proof. Let $N = |\mathcal{N}|$, and let $S(N)$ be the complexity of computing all the pairwise node-similarity scores via the similarity function S. So, computing the similarity scores for all the executions of line 3 takes $S(N)$ total time. In addition, there are N iterations of the main loop (lines 2–13). In each iteration, it takes N steps to find all the σ-similar nodes for a node n (line 3), and it takes $M = |\mathcal{N}_n^\sigma|$ steps, in the worst case, to find the appropriate cluster C for n (lines 5–11). Overall, this takes $S(N) + N(N+M) = S(N) + O(N^2)$ time. □

Observe that, by the greedy nature of Algorithm 4, there are cases where the clusters generated may not completely satisfy the similarity threshold σ. That is, for a cluster C, some node pairs will pairwise meet or exceed σ while others will not. Consider an example.

Example 1. *Let the input KG $\mathcal{G}$ contain (at least) three nodes $n_1, n_2, n_3 \in \mathcal{N}$, with the following pairwise similarity scores for these nodes computed via node-similarity function S:*

$$S(n_1, n_2) = 0.8 \text{ and } S(n_1, n_3) = S(n_2, n_3) = 0.9.$$

Moreover, let these nodes have a similarity score of 0 with all the other nodes in $\mathcal{G}$. Finally, let the similarity threshold be $\sigma = 0.9$. Upon invoking Algorithm 4 on these inputs, the set of node clusters $\mathcal{C}$ is initialized to the empty set, and the main loop (lines 2–13) begins. We will consider the three iterations of the main loop corresponding to the nodes n_1, n_2, and n_3.

1. *In the first iteration, n_1 is the node being considered. The set $\mathcal{N}_{n_1}^{0.9} = \{n_3\}$ is built (line 3), and since it is not empty, a cluster C must be identified for n_1. To this end, the three cases are checked to identify the appropriate cluster for n_1. Since n_1 is not yet in a cluster, and its only σ-similar node n_3 is not yet in a cluster, cases 1 and 2 do not apply. Thus, according to case 3 (lines 10–12), a new cluster C is created and added to the set of all clusters $\mathcal{C}$. Next, C is populated with n_1 and $n_3 \in \mathcal{N}_{n_1}^{0.9}$ (line 13), producing $C = \{n_1, n_3\}$.*

2. *In the second iteration of the main loop, n_2 is the node being considered. The set $\mathcal{N}_{n_2}^{0.9} = \{n_3\}$ is built (line 3), and since it is not empty, a cluster C must be identified for n_2. Again, the three cases are checked to identify the appropriate one. Since n_2 is not yet in a cluster, case 1 does not apply. However, one of its σ-similar nodes, i.e., n_3, is in the cluster C. Therefore, this cluster is identified for n_2, according to case 2 (lines 7–9). Then, n_2 is added to C (line 13), producing $C = \{n_1, n_2, n_3\}$.*

3. *In the third iteration of the main loop, n_3 is the node being considered. The set $\mathcal{N}_{n_3}^{0.9} = \{n_1, n_2\}$ is built (line 3), and since it is not empty, a cluster C must be identified for n_3 by checking the three possible cases. Since n_3 is already in cluster C, case 1 applies here (lines*

5–6). Since n_3 and $n_1, n_2 \in \mathcal{N}_{n_3}^{0.9}$ are already in the cluster C, no new nodes are added to C (line 13).

Algorithm 4 (line 14) outputs the final cluster $C = \{n_1, n_2, n_3\} \in \mathscr{C}$. Among the nodes in C, n_1 and n_3 are σ-similar, as are n_2 and n_3. However, n_1 and n_2 are not σ-similar. Thus, the nodes of C are not all pairwise σ-similar.

As shown in the above example, while every node in a cluster C will be σ-similar to at least one other node in C, they may not all be pairwise σ-similar. Since our overall goal is to reduce the size of the given KG, we consider σ to be a guideline for the extent to which the KG should be reduced, rather than a strict threshold. In this way, the clusters whose nodes are not all pairwise σ-similar are not problematic. After all, clusters at any threshold level can be used in the next phase of the approach, so this does not impact usability.

6.2.2.2 The Strict Node-Clustering Approach

Algorithm 5: Phase 1 of GAME+: strict node clustering.

Data: KG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, node-similarity function $S : \mathcal{N} \times \mathcal{N} \to \mathbb{R}$, node-similarity threshold σ.
Result: Set $\mathscr{C}$ of node clusters for the nodes in $\mathcal{G}$.

1 $\mathscr{C} \leftarrow \emptyset$; // initializing the set of node clusters
2 $\mathcal{Q} \leftarrow initializeQueue(\mathcal{N})$; // max priority queue keyed on the number of σ-similar nodes
3 **while** $\mathcal{Q} \neq \emptyset$ **do**
4 $\quad n \leftarrow \mathcal{Q}.pop()$;
5 $\quad \mathcal{Q}_n^\sigma \leftarrow \{m \in \mathcal{Q}\ s.t.\ S(n,m) \geq \sigma\}$;
6 $\quad$ **if** $\mathcal{Q}_n^\sigma \neq \emptyset$ **then**
7 $\quad\quad$ **while** $\exists x, y \in \mathcal{Q}_n^\sigma\ s.t.\ S(x,y) < \sigma$ **do**
$\quad\quad\quad$ // some nodes in $\mathcal{Q}_n^\sigma$ are not σ-similar
8 $\quad\quad\quad z \leftarrow$ node with least total similarity in $\mathcal{Q}_n^\sigma$;
9 $\quad\quad\quad \mathcal{Q}_n^\sigma.remove(z)$;
$\quad\quad$ // $\mathcal{Q}_n^\sigma$ is now pairwise σ-similar
10 $\quad\quad C \leftarrow \{n\} \cup \mathcal{Q}_n^\sigma$; // create new cluster
11 $\quad\quad \mathscr{C} \leftarrow \mathscr{C} \cup C$;
12 $\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \setminus C$; // remove clustered nodes from the queue
13 **return** $\mathscr{C}$;

Our second node-clustering approach, referred to as the *strict approach*, is outlined in

Algorithm 5. Intuitively, this approach also assigns each node in the KG $\mathscr{G}$ to a cluster if possible, in a greedy manner according to the similarity scoring. However, this strict approach includes extra functionality to ensure that the nodes in each cluster are pairwise σ-similar. Algorithm 5 takes the same three inputs as Algorithm 4: a KG $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$, a node-similarity function $S : \mathscr{N} \times \mathscr{N} \to \mathbb{R}$, and a node-similarity threshold σ. It outputs a set of clusters $\mathscr{C}$ for $\mathscr{G}$, where each node n in a cluster $C \in \mathscr{C}$ is σ-similar to *all* the other nodes in the cluster C.

The set of node clusters $\mathscr{C}$ is first initialized as the empty set (line 1) and is built up throughout the algorithm. Algorithm 5 uses a max priority queue $\mathscr{Q}$ of the nodes in $\mathscr{G}$ that is keyed on the number of σ-similar nodes (line 2). To this end, all similarity scores are computed when building the queue $\mathscr{Q}$, and saved for easy retrieval.

The main loop of Algorithm 5 (lines 3–12) iterates through the nodes in the max priority queue $\mathscr{Q}$. First, the node with the most σ-similar nodes is popped off the queue (line 4), and the similarity function S is used to identify all the nodes in $\mathscr{Q}$ that are σ-similar to n; we denote the set of such nodes by $\mathscr{Q}_n^\sigma$ (line 5). If n has no σ-similar nodes remaining in $\mathscr{Q}$, then it does not belong in any cluster, as there are no remaining nodes with which it can be clustered. However, if n has at least one σ-similar node in $\mathscr{Q}_n^\sigma$ (line 6), then an appropriate cluster will be generated for n and at least some of its σ-similar nodes in lines 7–10.

The inner loop of Algorithm 5 (lines 7–9) ensures that all nodes in $\mathscr{Q}_n^\sigma$ are pairwise σ-similar. This is accomplished by repeatedly removing from $\mathscr{Q}_n^\sigma$ the node z with the least total similarity to the other nodes in $\mathscr{Q}_n^\sigma$ (lines 8–9). This continues until all the nodes remaining in $\mathscr{Q}_n^\sigma$ are pairwise σ-similar, at which point this inner loop (lines 7–9) ends.

Next, a cluster C is built that contains n and the now pairwise σ-similar nodes in $\mathscr{Q}_n^\sigma$ (line 10). These newly clustered nodes are then removed from the queue $\mathscr{Q}$ (line 12) so they will no longer be considered in future iterations of the main loop, ensuring that they will not be placed into more than one cluster. Finally, the newly generated cluster C is added to the set of all clusters $\mathscr{C}$ (line 11).

After the main loop is completed, each node of $\mathscr{G}$ has been examined and placed into an appropriate cluster if possible. At this point, the resulting set of clusters $\mathscr{C}$ is output (line 13).

The runtime complexity of Algorithm 5 is dominated by either the total number of nodes in the KG $\mathscr{G}$ to the power of four $|\mathscr{N}|^4$, or by the runtime complexity $S(|\mathscr{N}|)$ of computing all the pairwise node-similarity scores via the similarity function S, in the case where $S(|\mathscr{N}|)$ asymptotically dominates $|\mathscr{N}|^4$:

Theorem 2. *The runtime complexity of Algorithm 5 is* $S(N) + O(N^4)$.

Proof. Let $N = |\mathscr{N}|$, and let $S(N)$ be the complexity of computing all the pairwise node-similarity scores via the similarity function S. So, computing the similarity scores to generate

the max priority queue takes $S(N)$ total time. In addition, building the max priority queue $\mathcal{Q}$ takes $O(N \log N)$ time (line 2).

In the worst case, all clusters would have only 2 nodes. In this case, there would be $\frac{N}{2}$ iterations of the main loop (lines 3–12) with $I = N - 2i$ nodes remaining in the queue $\mathcal{Q}$ before the ith iteration ($0 \leq i \leq \frac{N}{2} - 1$). In the ith iteration, it takes $I - 1$ steps to find in $\mathcal{Q}$ all the σ-similar nodes to the node n (line 5). In the worst case, n is σ-similar to all the remaining $I - 1$ nodes, but none of them are σ-similar to each other. Thus, there would be $I - 1$ iterations of the inner loop (lines 7–9) with $J = I - 1 - j$ nodes remaining in $\mathcal{Q}_n^\sigma$ before the jth iteration ($0 \leq j \leq I - 2$). Each iteration of the inner loop takes $J * (J - 1)$ steps because for each of the J nodes remaining in $\mathcal{Q}_n^\sigma$, we have to look at its $J - 1$ similarity scores with the other nodes. The overall time taken is:

$$
\begin{aligned}
& S(N) + O(N \log N) + \sum_{i=0}^{\frac{N}{2}-1} \left[I + \sum_{j=0}^{I-2} J * (J-1) \right] \\
&= S(N) + O(N \log N) + \frac{1}{24} N^2 (N^2 + 2) \\
&= S(N) + O(N \log N) + O(N^4) \\
&= S(N) + O(N^4).
\end{aligned}
$$

□

Recall that the runtime complexity of the *relaxed node-clustering approach* of Algorithm 4 (see Theorem 1) is worst-case quadratic in $|\mathcal{N}|$ or dominated by the runtime complexity $S(|\mathcal{N}|)$ of computing all the pairwise node-similarity scores. Thus, the *strict node-clustering approach* of Algorithm 5 is worse due to the time required to strictly adhere to the node-similarity threshold σ.

As discussed, the defining feature of Algorithm 5 is its guarantee that the nodes within each generated cluster will be pairwise σ-similar. This is accomplished via the inner loop of lines 7–9. Consider the behavior of Algorithm 5 on the example presented in Example 1 of Section 6.2.2.1.

Example 2. *Let the input KG $\mathcal{G}$ contain (at least) three nodes $n_1, n_2, n_3 \in \mathcal{N}$, with the following pairwise similarity scores for these nodes computed via the node-similarity function S:*

$$S(n_1, n_2) = 0.8 \text{ and } S(n_1, n_3) = S(n_2, n_3) = 0.9.$$

Moreover, let these nodes have a similarity score of 0 *with all the other nodes in $\mathcal{G}$. Finally, let the similarity threshold be $\sigma = 0.9$. Upon invoking Algorithm 5 on these inputs, the set of nodes*

clusters $\mathscr{C}$ is initialized to the empty set, the max priority queue $\mathscr{Q}$ is initialized with the set of nodes $\mathscr{N}$ in $\mathscr{G}$, and the main loop (lines 3–12) begins. We will consider the iterations of the main loop corresponding to the subset $\{n_3, n_1, n_2\}$ of the queue $\mathscr{Q}$.

1. *In the first iteration, n_3 is popped off the queue $\mathscr{Q}$ as it has the most σ-similar nodes (2). The set $\mathscr{Q}^{\sigma}_{n_3} = \{n_1, n_2\}$ is built, and since it is not empty, a cluster C must be built for n_3. Before building the appropriate cluster, the inner loop (lines 7–9) verifies that all the nodes in $\mathscr{Q}^{\sigma}_{n_3}$ are pairwise σ-similar. In this case, $n_1, n_2 \in \mathscr{Q}^{\sigma}_{n_3}$ are not σ-similar, i.e., $S(n_1, n_2) = 0.8 < \sigma$, so n_2 is removed from $\mathscr{Q}^{\sigma}_{n_3}$ (lines 8–9), leaving $\mathscr{Q}^{\sigma}_{n_3} = \{n_1\}$. At this point, all nodes in $\mathscr{Q}^{\sigma}_{n_3}$ are pairwise σ-similar (trivially), so a new cluster is created and populated with n_3 and $\mathscr{Q}^{\sigma}_{n_3}$, producing $C = \{n_1, n_3\}$ (line 10). Next, C is added to the set of all clusters $\mathscr{C}$ (line 11), and the newly clustered nodes are removed from the queue $\mathscr{Q}$, producing $\mathscr{Q} - \{n_2\}$ (line 12).*

2. *In the next iteration, n_2 is popped off the queue $\mathscr{Q}$ as it is the only node remaining in the subset of $\mathscr{Q}$ that we are considering. The set $\mathscr{Q}^{\sigma}_{n_2} = \emptyset$ is built from the remaining nodes in $\mathscr{Q}$. Since this set is empty, lines 7–12 are skipped, and no cluster is built for n_2.*

Algorithm 5 (line 13) outputs the final cluster $C = \{n_1, n_3\} \in \mathscr{C}$. Among the nodes in C, n_1 and n_3 are σ-similar. Thus, the nodes of C are all pairwise σ-similar.

As shown in the above example, the *strict node-clustering approach* ensures that all nodes within a cluster are pairwise σ-similar as a result of the inner loop. Thus, the *strict node-clustering approach* has a higher runtime complexity than the *relaxed node-clustering approach*, but it provides clusters that strictly adhere to the threshold σ, where the *relaxed approach* does not.

6.2.3 The Node-Fusion Phase of the GAME+ Approach

We now introduce the proposed node-fusion approach; it is a method for creating consolidated nodes and edges from the node clusters generated in the preceding phase of the GAME+ approach. The algorithm creates a single consolidated node for each cluster by fusing together all the nodes in that cluster. Edges incident to these nodes become consolidated edges. The fusion eliminates repetitive KG triples and similar nodes that may be superfluous, thereby reducing the size of the input KG. Importantly, provenance is maintained for the consolidated nodes and edges, ensuring the losslessness of the fusion process and enabling full reconstruction of the original input KG from the output AKG. To this end, we present two algorithms: (1) the *forward node-fusion algorithm* for generating the consolidated AKG $\mathscr{G}^A$ from the input KG $\mathscr{G}$ (see Section 6.2.3.1) and (2) the *inverse node-fusion algorithm* for reconstructing the KG $\mathscr{G}$ from the AKG $\mathscr{G}^A$ (see Section 6.2.3.2).

For the node-fusion phase of GAME+, we use the notion of a 1-hop neighborhood of KG nodes, defined as follows. The *1-hop neighborhood* of a node n in KG $\mathcal{G}$, denoted $nbrhd(n)$, is the set of all nodes adjacent to n in $\mathcal{G}$, along with the predicates of the edges that connect the nodes to n. We differentiate between the 1-hop in- and out-neighborhoods of n. The *1-hop in-neighborhood* of n, denoted $nbrhd^{\leftarrow}(n)$, is the set of subject nodes and predicates for the triples in which n is the object, see Eq. (6.1); the *1-hop out-neighborhood* of n, denoted $nbrhd^{\rightarrow}(n)$, is the set of predicates and object nodes for the triples in which n is the subject, see Eq. (6.2).

$$nbrhd^{\leftarrow}(n) = \{(m,p) | (m,p,n) \in \mathcal{G}\}. \tag{6.1}$$

$$nbrhd^{\rightarrow}(n) = \{(p,m) | (n,p,m) \in \mathcal{G}\}. \tag{6.2}$$

Then the 1-hop neighborhood of n in $\mathcal{G}$ is the union of the 1-hop in-neighborhood of n and its 1-hop out-neighborhood:

$$nbrhd(n) = nbrhd^{\leftarrow}(n) \cup nbrhd^{\rightarrow}(n). \tag{6.3}$$

Each element (m,p) of $nbrhd^{\leftarrow}(n)$ is called an *in-neighbor*, and each element (p,m) of $nbrhd^{\rightarrow}(n)$ is called an *out-neighbor*.

6.2.3.1 The Forward Node-Fusion Algorithm

We now introduce the proposed *forward node-fusion algorithm*, outlined in Algorithm 6. This phase of the GAME+ approach takes as inputs a KG $\mathcal{G}$ and the set of clusters $\mathcal{C}$ generated for $\mathcal{G}$ in the preceding phase, see Section 6.2.2. The output of this phase and of the overall GAME+ approach is an AKG $\mathcal{G}^A$ for $\mathcal{G}$.

The algorithm initializes the AKG $\mathcal{G}^A$ to the input KG $\mathcal{G}$ (line 1), and then iterates over $\mathcal{G}^A$ by replacing some of its nodes and edges with provenance-enhanced consolidated nodes and edges in accordance with the clusters created in the preceding phase of GAME+.

The main loop of Algorithm 6 (lines 2–20) considers a single cluster C in each iteration, and performs three main steps for each cluster C. First, a consolidated node C^A is created (line 3) and added to the current AKG $\mathcal{G}^A$ (line 5). C^A stores the nodes in C, i.e., those that are semantically represented by C^A, via the the property C^A.`prov` (line 4). If the Neo4j data-management system (The Neo4j Team 2022c) is used for storing and processing the KG data, lines 3–5 can be executed via the following Cypher (The Neo4j Team 2022b) query:

```
CREATE (a:ConsolidatedNode)
SET a.id = C^A AND a.prov = C
```

Algorithm 6: Phase 2 of GAME+: node fusion.

Data: KG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, set of node clusters $\mathcal{C}$.
Result: AKG $\mathcal{G}^A = (\mathcal{N}^A, \mathcal{T}^A, \phi^A, \mathcal{P}^A, \mathcal{L}^A)$.

$\mathcal{G}^A \leftarrow \mathcal{G}$;
for $C \in \mathcal{C}$ **do**
    // Create a consolidated node with provenance:
    $C^A \leftarrow$ new consolidated node;
    C^A.prov $\leftarrow C$;
    $\mathcal{N}^A \leftarrow \mathcal{N}^A \cup C^A$;
    // Create triples with provenance for the consolidated node:
    for $(m,p) \in \bigcup_{n \in C} nbrhd^{\leftarrow}(n)$ **do**
        if $m \in C$ **then**
            $t \leftarrow (C^A, p, C^A)$;
        else
            $t \leftarrow (m, p, C^A)$;
        t.prov $\leftarrow \{(m,n) \mid \forall n \in C \ s.t. \ (m,p,n) \in \mathcal{L}\}$;
        $\mathcal{L}^A \leftarrow \mathcal{L}^A \cup t$;
    for $(p,m) \in \bigcup_{n \in C} nbrhd^{\rightarrow}(n)$ **do**
        if $m \in C$ **then**
            $t \leftarrow (C^A, p, C^A)$;
        else
            $t \leftarrow (C^A, p, m)$;
        t.prov $\leftarrow \{(n,m) \mid \forall n \in C \ s.t. \ (n,p,m) \in \mathcal{L}\}$;
        $\mathcal{L}^A \leftarrow \mathcal{L}^A \cup t$;
    // Remove clustered nodes and incident edges:
    $\mathcal{G}^A \leftarrow \mathcal{G}^A \setminus$ nodes in C and their incident edges;
return $\mathcal{G}^A$;

Next, consolidated edges incident to C^A are created and added to $\mathscr{G}^A$ in two batches: (i) the in-edges, i.e., those with C^A as the object, and (ii) the out-edges, i.e., those with C^A as the subject. The loop in lines 6–12 creates a *consolidated in-edge* incident to C^A for each in-neighbor incident to any node in the cluster C. There are two cases to consider when creating a consolidated in-edge to represent the in-neighbor (m, p):

1. In the first case (line 7), the subject m of the in-neighbor (m, p) is also in the cluster C; we refer to such in-edges as *collapsed in-edges*. These edges must be handled appropriately to avoid edge contraction (or removal) from the graph, so a self-loop $t = (C^A, p, C^A)$ is created as the corresponding consolidated in-edge in the AKG $\mathscr{G}^A$ (line 8).

2. In the second case (line 9), the subject m of the in-neighbor (m, p) is not in the cluster C, so the consolidated in-edge $t = (m, p, C^A)$ is created (line 10).

In either case, the property $t.\texttt{prov}$ stores the pairs of adjacent nodes (m, n) for all nodes n in the cluster C for which (m, p, n) is in the input KG $\mathscr{G}$, i.e., those nodes n that are incident to the in-neighbor (m, p) (line 11). These consolidated in-edges are then added to the current AKG $\mathscr{G}^A$ (line 12). Lines 10–12 (case 2) can be executed via the following Cypher query:[1]

$$\begin{aligned}
&\texttt{MATCH}\,(\texttt{a}),(\texttt{b})\\
&\texttt{WHERE}\,\texttt{a.id} = m\ \texttt{AND}\ \texttt{b.id} = C^A\\
&\texttt{CREATE}\,(\texttt{a})-[\texttt{r} : p]\rightarrow(\texttt{b})\\
&\texttt{SET}\,\texttt{r.prov} = \{(m, n) \mid \forall n \in C\ s.t.\,(m, p, n) \in \mathscr{L}\}
\end{aligned}$$

The loop in lines 13–19 handles the out-edges in the same manner. Finally, the nodes from the cluster C are removed from the AKG $\mathscr{G}^A$, along with all their incident edges (line 20). This can be executed via the following Cypher query for each clustered node $n \in C$:

$$\begin{aligned}
&\texttt{MATCH}\,(\texttt{b})\\
&\texttt{WHERE}\,\texttt{b.id} = n\\
&\texttt{DETACH DELETE}\,(\texttt{b})
\end{aligned}$$

Upon completion of the main loop, the AKG $\mathscr{G}^A$ contains consolidated nodes representing the nodes of each cluster, as well as consolidated edges representing edges incident to these nodes. Algorithm 6 (line 21) outputs this AKG $\mathscr{G}^A$.

[1] Modifying the `WHERE` clause of this Cypher query to $\texttt{WHERE}\,\texttt{a.id} = C^A\ \texttt{AND}\ \texttt{b.id} = C^A$ would make it appropriate for the case of the collapsed in-edges in lines 8, 11–12.

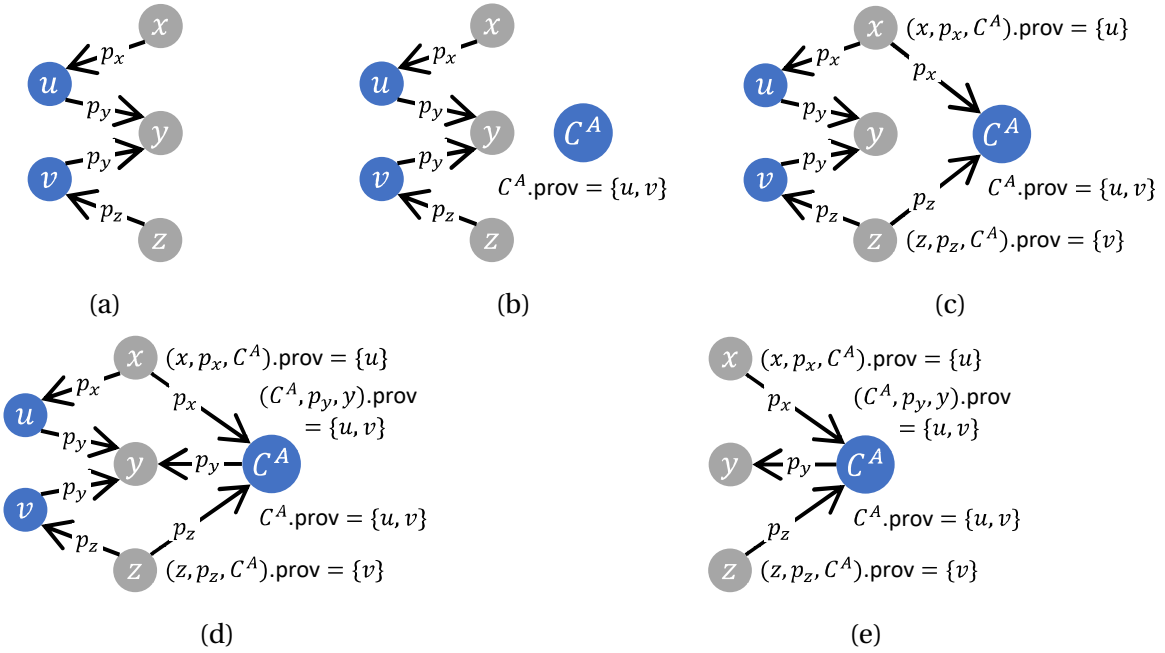


Figure 6.2: An example of the node-fusion process that takes place in the main loop of Algorithm 6. See Section 6.2.3.1 and Example 3 for the details.

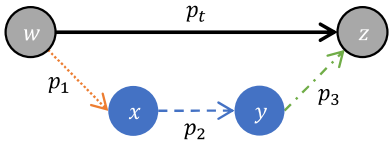
We now present an example, sketched in Figure 6.2, of the node-fusion process that takes place in Algorithm 6.

Example 3. *Let the input KG $\mathcal{G}$ contain at least five nodes $u, v, x, y, z \in \mathcal{N}$ and at least the following four triples: $(x, p_x, u), (u, p_y, y), (z, p_z, v)$, and (v, p_y, y); see Figure 6.2a for a depiction of this subgraph. Suppose the node-clustering phase of GAME+ has created a cluster $C = \{u, v\} \in \mathcal{C}$ for $\mathcal{G}$. Upon invoking Algorithm 6 on these inputs, the AKG $\mathcal{G}^A$ is initialized with the contents of $\mathcal{G}$. Then, the main loop (lines 2–20) starts.*

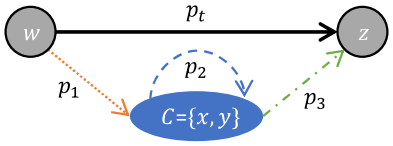
Consider the iteration of the main loop corresponding to the cluster C. Lines 3–5 create in the AKG $\mathcal{G}^A$ a consolidated node C^A that represents nodes u and v in the KG $\mathcal{G}$ with C^A.`prov` $= \{u, v\}$*, see Figure 6.2b. Next, the following triples are created for C^A using the in- and out-neighborhoods of the clustered nodes u and v.*

$$nbrhd^{\leftarrow}(u) = \{(x, p_x)\}$$
$$nbrhd^{\rightarrow}(u) = \{(p_y, y)\}$$
$$nbrhd^{\leftarrow}(v) = \{(z, p_z)\}$$
$$nbrhd^{\rightarrow}(v) = \{(p_y, y)\}$$

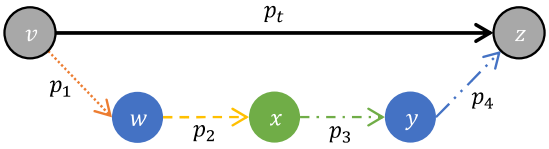
The loop in lines 6–12 of Algorithm 6 creates a consolidated in-edge (m, p, C^A) for each



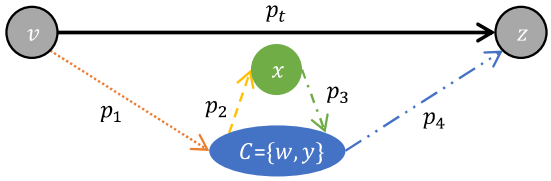
(a) An example KG from which an inference rule of length 3 could be mined.



(b) An example AKG that could be generated from the KG in Figure 6.3a.



(c) An example KG from which an inference rule of length 4 could be mined.



(d) An example AKG that could be generated from the KG in Figure 6.3c.

Figure 6.3: Example knowledge graphs *(KGs)* and abstract knowledge graphs *(AKGs)* from which various inference rules could be mined. (b) and (d) present AKGs that could be generated from (a) and (c), respectively, using the GAME+ approach. Shorter inference rules could be mined on these AKGs in addition to the inference rules that could be mined from the corresponding KGs. These examples highlight the ability of GAME+ to preserve in the AKGs all the KG metapaths, which also creating shorter pathways in the AKGs.

$(m,p) \in \bigcup_{n \in C} nbrhd^{\leftarrow}(n) = nbrhd^{\leftarrow}(u) \cup nbrhd^{\leftarrow}(v) = \{(x,p_x),(z,p_z)\}$. *In the context of this example, this creates in* $\mathcal{G}^A$ *two consolidated in-edges* $t_1 = (x,p_x,C^A)$ *and* $t_2 = (z,p_z,C^A)$, *representing respectively the triples* (x,p_x,u) *and* (z,p_z,v) *in* $\mathcal{G}$ *with* $t_1.\texttt{prov} = \{(x,u)\}$ *and* $t_2.\texttt{prov} = \{(z,v)\}$, *see Figure 6.2c.*

The loop in lines 13–19 of Algorithm 6 creates a consolidated out-edge (C^A,p,m) *for each* $(p,m) \in \bigcup_{n \in C} nbrhd^{\rightarrow}(n) = nbrhd^{\rightarrow}(u) \cup nbrhd^{\rightarrow}(v) = \{(p_y,y)\}$. *In the context of this example, this creates in* $\mathcal{G}^A$ *a consolidated out-edge* $t_3 = (C^A,p_y,y)$ *for* (u,p_y,y) *and* (v,p_y,y) *in* $\mathcal{G}$ *with* $t_3.\texttt{prov} = \{(u,y),(v,y)\}$, *see Figure 6.2d.*

Finally, line 20 removes from $\mathcal{G}^A$ *the nodes* u *and* v, *along with the triples* (x,p_x,u), (u,p_y,y), (z,p_z,v), *and* (v,p_y,y). *Figure 6.2e shows the resulting subgraph in the final AKG* $\mathcal{G}^A$.

An important feature of the proposed node-fusion approach is its ability to create new metapaths in the AKG, while still maintaining in the AKG all the original metapaths present in the input KG. This feature enables inference-rule mining on the AKGs to not only produce all inference rules that could be mined on the input KG, but to also produce new inference rules that could not be mined on the input KG, making the inference-rule mining potentially more effective (see Section 6.3.5 for details). Consider the example KG shown in Figure 6.3a. From

this KG, the following inference rule could be mined:

$$(w, p_t, z) \Leftarrow (w, p_1, x) \wedge (x, p_2, y) \wedge (y, p_3, z) \tag{6.4}$$

Next, consider the example AKG shown in Figure 6.3b, which was generated from the KG shown in Figure 6.3a by fusing the nodes x and y into a single consolidated node. From this AKG, the inference rule in Eq. 6.4 could be mined, along with the following additional inference rule:

$$(w, p_t, z) \Leftarrow (w, p_1, C) \wedge (C, p_3, z)$$

By fusing x and y together, a shorter path from w to z was created, and thus a shorter inference rule. In this manner, extra inference rules can be mined from the AKGs generated by the GAME+ approach, while still recovering all inference rules that could be mined from the input KG.

Another example of such a path-shortening scenario is shown in Figures 6.3c and 6.3d. The following inference rule could be mined from the KG in Figure 6.3c:

$$(v, p_t, z) \Leftarrow (v, p_1, w) \wedge (w, p_2, x) \wedge (x, p_3, y) \wedge (y, p_4, z) \tag{6.5}$$

The inference rule in Eq. 6.5 could also be mined from the AKG in Figure 6.3d, along with this inference rule:

$$(v, p_t, z) \Leftarrow (v, p_1, C) \wedge (C, p_4, z)$$

6.2.3.2 The Inverse Node-Fusion Algorithm

As discussed, a key feature of our proposed node fusion approach is its invertibility, i.e., there exists an inverse approach that produces the original KG $\mathscr{G}$ and set of clusters $\mathscr{C}$ when given the AKG $\mathscr{G}^A$. The proposed inverse-fusion approach is outlined in Algorithm 7, which takes as input the AKG $\mathscr{G}^A = (\mathscr{N}^A, \mathscr{T}^A, \phi^A, \mathscr{P}^A, \mathscr{L}^A)$ generated in the forward-fusion phase of GAME+, see Section 6.2.3.1. The output of the inverse-fusion approach is the original KG $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$ and set of clusters $\mathscr{C}$ input to the inverse-fusion approach.

The algorithm first initializes the KG $\mathscr{G}$ to the input AKG $\mathscr{G}^A$ (line 1), and then iterates over $\mathscr{G}$ by replacing the consolidated nodes and edges with the appropriate specific KG nodes and edges in accordance with the `prov` property attached to each consolidated node and edge in the node-fusion phase of GAME+. Also, the set of clusters $\mathscr{C}$ is initialized as the empty set (line 2), and the consolidated nodes are retrieved from $\mathscr{G}^A$ as the set $\mathscr{C}^A$ (line 3). Line 3 can be

Algorithm 7: Inverse node fusion.

Data: AKG $\mathcal{G}^A = (\mathcal{N}^A, \mathcal{T}^A, \phi^A, \mathcal{P}^A, \mathcal{L}^A)$.
Result: KG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, set of node clusters $\mathcal{C}$.

$\mathcal{G} \leftarrow \mathcal{G}^A$;
$\mathcal{C} \leftarrow \emptyset$;
$\mathcal{C}^A \leftarrow$ the consolidated nodes from $\mathcal{G}^A$;
for $C^A \in \mathcal{C}^A$ **do**
    // Create KG nodes from C^A:
    $C \leftarrow C^A$.prov;
    $\mathcal{C} \leftarrow \mathcal{C} \cup C$;
    $\mathcal{N} \leftarrow \mathcal{N} \cup C$; // create a node for each node in C
    // Create KG triples for the newly added nodes:
    for $(m, p) \in nbrhd^{\leftarrow}(C^A)$ **do**
        $t \leftarrow (m, p, C^A)$;
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{(m, p, n) \mid \forall (m, n) \in t.\text{prov}\}$;
    for $(p, m) \in nbrhd^{\rightarrow}(C^A)$ **do**
        $t \leftarrow (C^A, p, m)$;
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{(n, p, m) \mid \forall (n, m) \in t.\text{prov}\}$;
    // Remove consolidated node and incident edges:
    $\mathcal{G} \leftarrow \mathcal{G} \setminus C^A$ and its incident edges;
return $\mathcal{G}$, $\mathcal{C}$;

executed via the following Cypher query:

```
MATCH (a:ConsolidatedNode)
RETURN a
```

The main loop of Algorithm 7 (lines 4–14) considers a single consolidated node C^A in each iteration, and performs three main steps for each consolidated node C^A that invert those from Algorithm 6. First, a cluster C is created as the set of nodes stored in C^A.prov (line 5) and added to the set of all clusters $\mathscr{C}$ (line 6). Also, a KG node is created for each node n in the cluster C and added to the current KG $\mathscr{G}$ (line 7). Line 7 can be accomplished by executing the following Cypher query for each node $n \in C$:

```
CREATE (a)
SET a.id = n
```

Next, KG edges incident to the nodes in the cluster C are added to $\mathscr{G}$ in two batches: (i) the in-edges, i.e., those with a node $n \in C$ as the object, and (ii) the out-edges, i.e., those with a node $n \in C$ as the subject. The loop in lines 8–10 creates in-edges incident to the appropriate subset of nodes from C for each in-edge incident to C^A. The triple t corresponding to the in-edge (m, p, C^A) of C^A (line 9) contains in t.prov the set of nodes in C having the same in-edge in the original KG. Line 9 can be executed via the following Cypher query:

```
MATCH (a)−[r : p]→(b)
WHERE a.id = m AND b.id = C^A
RETURN r
```

A corresponding in-edge (m, p, n) is added to the current KG $\mathscr{G}$ for each such pair of nodes $(m, n) \in t$.prov (line 10). Line 10 can be accomplished by executing the following Cypher query for each pair of nodes $(m, n) \in t$.prov:

```
MATCH (a),(b)
WHERE a.id = m AND b.id = n
CREATE (a)−[r : p]→(b)
```

The loop in lines 11–13 handles the out-edges of C^A in the same manner. Finally, the consolidated node C^A is removed from the KG $\mathscr{G}$, along with all its incident edges (line 14).

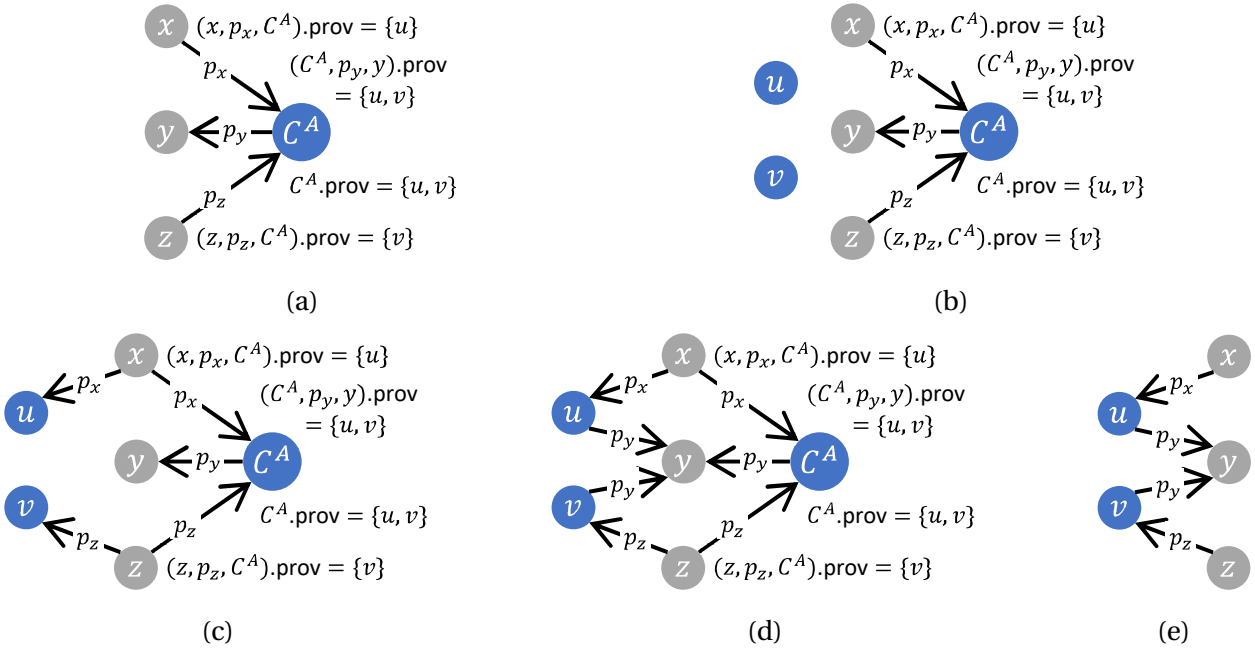


Figure 6.4: An example of the inverse node-fusion process that takes place in the main loop of Algorithm 7. See Section 6.2.3.2 and Example 4 for the details.

Upon completion of the main loop, the KG $\mathcal{G}$ no longer contains consolidated nodes and edges; rather it contains all the specific KG nodes and edges that were previously represented by the consolidated nodes and edges in $\mathcal{G}^A$. Moreover, the set of clusters $\mathscr{C}$ contains all the clusters that were used to build $\mathcal{G}^A$ in the node-fusion phase of GAME+. Algorithm 7 (line 15) outputs this KG $\mathcal{G}$ and the set of clusters $\mathscr{C}$.

We now present am example, sketched in Figure 6.4, of the inverse node-fusion process that takes place in Algorithm 7, using the output from the node-fusion process presented in Example 3 of Section 6.2.3.1.

Example 4. *Let the input AKG $\mathcal{G}^A$ contain at least the consolidated node $C^A \in \mathcal{N}^A$ with $C^A.\texttt{prov} = \{u, v\}$, and at least the following three triples: (x, p_x, C^A), (C^A, p_y, y), and (z, p_z, C^A) with:*

$$(x, p_x, C^A).\texttt{prov} = \{(x, u)\}$$
$$(C^A, p_y, y).\texttt{prov} = \{(u, y), (v, y)\}$$
$$(z, p_z, C^A).\texttt{prov} = \{(z, v)\}$$

See Figure 6.4a for a depiction of this subgraph. Upon invoking Algorithm 7 on this input, the KG $\mathcal{G}$ is initialized with the contents of $\mathcal{G}^A$, the set of clusters $\mathscr{C}$ is initialized as the empty set,

and the set of consolidated nodes $\mathscr{C}^A$ is retrieved from the input AKG $\mathscr{G}^A$. Then, the main loop (lines 4–14) starts.

Consider the iteration of the main loop corresponding to the consolidated node C^A. Lines 5–7 create a new cluster $C = \{u, v\}$ of the nodes represented by C^A and stored in C^A.`prov`*, add this cluster to the set $\mathscr{C}$ of all clusters, and create in $\mathscr{G}$ the KG nodes u and v, see Figure 6.4b. Next, the following triples are created for the nodes u, v using the in- and out-neighborhoods of the consolidated node C^A.*

$$nbrhd^{\leftarrow}(C^A) = \{(x, p_x), (z, p_z)\}$$
$$nbrhd^{\rightarrow}(C^A) = \{(p_y, y)\}$$

The loop in lines 8–10 of Algorithm 7 creates in $\mathscr{G}$ appropriate in-edges for each $(m, p) \in nbrhd^{\leftarrow}(C^A) = \{(x, p_x), (z, p_z)\}$, according to the `prov` *of each edge. In the context of this example, this creates two in-edges: (1) (x, p_x, u) because $(x, p_x, C^A).$*`prov` $= \{(x, u)\}$ *and (2) (z, p_z, v) because $(z, p_z, C^A).$*`prov` $= \{(z, v)\}$*, see Figure 6.4c.*

The loop in lines 11–13 creates in $\mathscr{G}$ appropriate out-edges for each $(p, m) \in nbrhd^{\rightarrow}(C^A) = \{(p_y, y)\}$. In the context of this example, this creates two out-edges: (u, p_y, y) and (v, p_y, y) because $(C^A, p_y, y).$`prov` $= \{(u, y), (v, y)\}$*, see Figure 6.4d.*

Finally, line 14 removes from $\mathscr{G}$ the consolidated node C^A, along with the consolidated triples (x, p_x, C^A), (z, p_z, C^A), and (C^A, p_y, y). Figure 6.4e shows the resulting subgraph in the final KG $\mathscr{G}$, which matches the original input from Example 3 depicted in Figure 6.2a

6.2.4 Making Incremental Updates to Abstract Knowledge Graphs

We now discuss how to make incremental updates to an AKG $\mathscr{G}^A$ when the original input KG $\mathscr{G}$ is modified. The goals here are twofold: (i) to ensure that $\mathscr{G}^A$ remains in sync with $\mathscr{G}$ over time, while (ii) making the modifications more efficiently than rerunning the entire GAME+ approach. (The algorithm modifications that are required to accommodate the incremental updates are straightforward and are omitted for the sake of brevity.) There are four basic modification operations on $\mathscr{G}$ to consider:

1. a node n is added,
2. an edge (triple) $e = (s, p, o)$ is added,
3. an edge (triple) $e = (s, p, o)$ is removed, or
4. a node n is removed.

For operation (1), when a node n is added to the original input KG $\mathscr{G}$, an appropriate cluster C for n must be identified (if possible). To this end, the node-clustering phase of GAME+ (either Algorithm 4 or Algorithm 5) can be run with the existing AKG $\mathscr{G}^A$, the original node-similarity function S, and the original node-similarity threshold σ as inputs. The key modification is to allow only a single iteration of the main loop (lines 2–13 of Algorithm 4 or lines 3–12 of Algorithm 5), i.e., one iteration to consider the new node n. If no cluster is identified for n, then n is added directly to $\mathscr{G}^A$.[2] Otherwise, if a cluster C is identified for n, then the node-fusion phase of GAME+ (Algorithm 6) can be run with the existing AKG $\mathscr{G}^A$ and the set of clusters $\mathscr{C} = \{C\}$ as inputs. If n is clustered with a consolidated node m, then the values of the `prov` properties of m and its incident edges are included in the `prov` properties of the nodes and edges created by this run of Algorithm 6.

For operation (2), when an edge $e = (s, p, o)$ is added to the original input KG $\mathscr{G}$, a corresponding edge is added to the existing AKG $\mathscr{G}^A$. If s has been fused into a consolidated node s^A in $\mathscr{G}^A$, then e is added to $\mathscr{G}^A$ as a consolidated edge $e^A = (s^A, p, o)$ with $e^A.\texttt{prov} = \{(s, o)\}$. If e^A already exists, then (s, o) is added to $e^A.\texttt{prov}$. Analogous steps are taken if o has been fused into a consolidated node o^A in $\mathscr{G}^A$.

For operation (3), when an edge $e = (s, p, o)$ is removed from the original input KG $\mathscr{G}$, the corresponding edge is removed from the existing AKG $\mathscr{G}^A$. If e has been fused into the consolidated edge e^A in $\mathscr{G}^A$, then (s, o) is removed from $e^A.\texttt{prov}$. If $e^A.\texttt{prov} = \emptyset$, then e^A is removed from $\mathscr{G}^A$.

For operation (4), when a node n is removed from the original input KG $\mathscr{G}$, the corresponding node is removed from the existing AKG $\mathscr{G}^A$.[3] If n has been fused into a consolidated node n^A in $\mathscr{G}^A$, then n is removed from $n^A.\texttt{prov}$. If $n^A.\texttt{prov} = \emptyset$, then n^A is removed from $\mathscr{G}^A$.

When multiple of these operations occur on the original input KG, they can be handled sequentially or in four batches: one batch to handle the instances of each operation. If a large number of changes are made, it may be more beneficial to rerun the GAME+ approach on the input KG $\mathscr{G}$.

6.3 Experimental Results

We now present the results of our experiments on three large-scale biomedical knowledge graphs *(KGs)*. In summary, our results suggest that our GAME+ approach for generating abstract KGs *(AKGs)* can be an effective strategy for reducing the sizes of the input KGs. Specifically, (i) performing rule mining on such AKGs can be substantially faster than on the original KGs. At

[2] If n has incident edges, these are added by operation (2).

[3] If n has incident edges, these are removed by operation (3).

Table 6.1: Summary statistics of the knowledge graphs used in the experiments.

Knowledge graph	Entities ($\|\mathscr{N}\|$)	Entity types ($\|\mathscr{T}\|$)	Predicates ($\|\mathscr{P}\|$)	Triples ($\|\mathscr{L}\|$)
DRKG	97,238	13	99	5,874,344
ROBOKOP	412,889	12	84	3,332,026
Hetionet	45,158	11	24	2,250,197

the same time, (ii) rule mining on the AKGs can recover all of the rules that can be mined on the original KGs since the abstraction is lossless. Moreover, the relaxed clustering approach can be more efficient than the strict clustering approach, while potentially being more effective.

6.3.1 Experimental Setup

6.3.1.1 Implementation

We have implemented the GAME+ approach and all the programs used for running the experiments in Python 3.9 (Van Rossum and Drake 2009). To interact with the data sets, we used the `py2neo` Python package (Small 2019). The experiments were conducted on a computer with an AMD Ryzen 7 2700X CPU processor running at 3.7 GHz with 16GB of RAM and Windows version 10.

6.3.1.2 Data Sets

As mentioned earlier, in the experiments we used three large-scale biomedical KGs: the Drug Repurposing Knowledge Graph *(DRKG)* (Ioannidis et al. 2020), Reasoning Over Biomedical Objects linked in Knowledge Oriented Pathways *(ROBOKOP)*[4] (Bizon et al. 2019), and *Hetionet*[5] (Himmelstein et al. 2017). All three graphs were hosted in Neo4j (The Neo4j Team 2022c) version 4.4. Please see Table 6.1 for some details on the KG sizes. For DRKG, the data were not available in a format compatible with Neo4j, so we used the version of the graph curated as discussed in Schatz et al. (2022). For ROBOKOP, we removed from the original graph all `related_to` and `subclass_of` triples, as these edges dominate all others in the node-similarity calculations, thereby skewing the results. For all three graphs, we also removed all isolated nodes, i.e., those with degree 0, as these nodes cannot participate in node clustering or in rule mining, and are therefore uninteresting.

[4] `http://robokopkg.renci.org/browser/`
[5] `https://neo4j.het.io/browser/`

6.3.1.3 Node-Similarity Function

In this work, we use the *1-hop neighborhood similarity function* as input to the approach, though any real-valued node-similarity function could be used in its place. The *1-hop neighborhood similarity function* $S(n,m)$ (Eq. 6.6) is the Jaccard similarity coefficient (Jaccard 1912) of the 1-hop neighborhoods (Eq. 6.3) of the two input nodes $n,m \in \mathcal{N}$.

$$S(n,m) = \frac{|nbrhd(n) \cap nbrhd(m)|}{|nbrhd(n) \cup nbrhd(m)|} \tag{6.6}$$

This metric indicates the similarity between the 1-hop neighborhoods of the two nodes, and is based on the intuition that similar nodes have similar relationships encoded in the KG $\mathcal{G}$.

In some scenarios, it may not make sense for nodes of different types, i.e., $\phi(n) \neq \phi(m)$, to be similar. If this is the case, then 0 can be assigned as the similarity score for nodes of different types, and Eq. 6.6 can be used for nodes of the same type.[6] In particular, we used this scheme in our experiments.

6.3.1.4 Rule Mining

In our experiments, we apply KG rule mining to the generated AKGs, using rule mining as a sample KG analytics task that could benefit from the speedup that KG reduction enables. Rule mining approaches, e.g., Ahmadi et al. (2020); Lajus et al. (2020); Sadeghian et al. (2019); Wang and Li (2015), explore KGs in a systematic manner, extracting inference rules along the way.

For the rule mining needed for our experiments, we ran AMIE (Lajus et al. 2020), the state-of-the-art approach in KG rule mining, on the three input KGs (see Section 6.3.1.2) and on each of the AKGs generated. To enable comparison between the rule sets, we wanted *all* rules to be output, so we ran AMIE with no pruning.[7]

6.3.2 Effectiveness in Reducing Graph Sizes

In this experiment we tested the effectiveness of the proposed GAME+ approach at reducing the size of the input KG, and, thus, the viability of the approach as a KG-reduction technology. In summary, the AKGs generated by the GAME+ approach are substantially smaller than the input KG, indicating that our approach can provide effective KG-size reduction.

For the experiment, we used each of the three input KGs (see Section 6.3.1.2) with the 1-hop neighborhood similarity function, various similarity thresholds, and both the relaxed and strict clustering approaches. The specific inputs into Algorithm 3 that we used were (i) one of DRKG,

[6] In KGs where a predicate has distinct subject and object types, this behavior occurs naturally.
[7] We set the thresholds for each metric to 0 and disabled Skyline pruning.

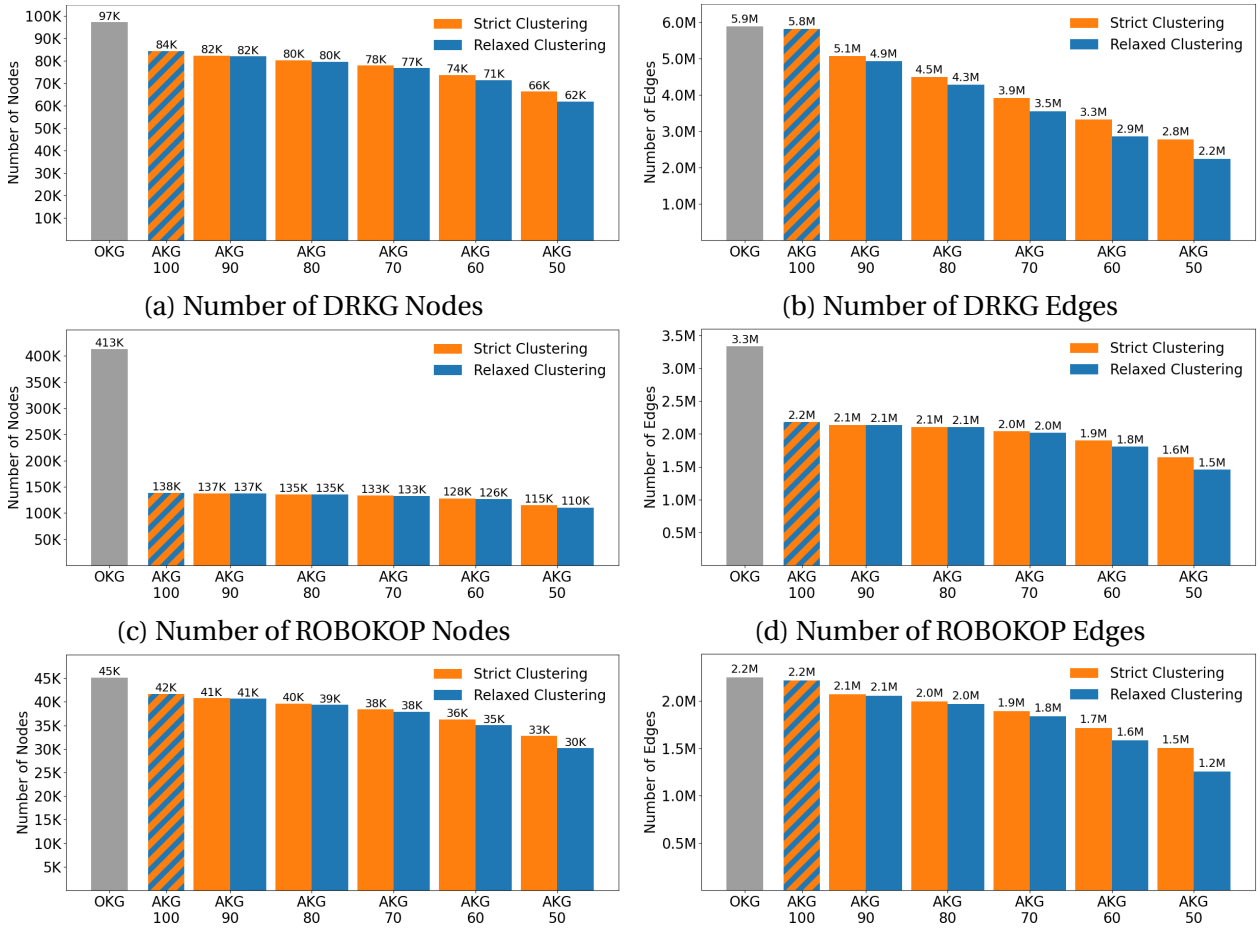
ROBOKOP, and Hetionet as the KG $\mathscr{G}$, (ii) the 1-hop neighborhood-similarity function (Eq. 6.6) as the node-similarity function S, one of $\{100\%, 90\%, 80\%, 70\%, 60\%, 50\%\}$ in each experiment as the similarity threshold σ, and one of $\{relaxed, strict\}$ as the parameter c that dictates the desired node-clustering approach. We measured the sizes of the resulting AKGs using the numbers of nodes and edges in each graph.

Figure 6.5 shows the numbers of nodes and edges in the input KGs and in each of the generated AKGs.[8] We can see that each of the AKGs is smaller than the original KG, in terms of the numbers of both nodes and edges, with the sizes of the AKGs decreasing along with the values of the similarity threshold σ. The size reduction observed varies between the three input graphs, reflecting the varied levels of redundancy in each graph. The most substantial reduction is seen on ROBOKOP (Figures 6.5c–6.5d) where even with the strictest similarity threshold of $\sigma = 100\%$, the resulting AKG has only about one-third the number of nodes and two-thirds the number of edges of the original KG.

Additionally, we observe that the relaxed and strict node-clustering approaches produce AKGs of similar sizes when the similarity threshold σ is high. As the value of σ decreases, the discrepancy between the clustering approaches increases, with the relaxed approach consistently producing smaller AKGs than the strict clustering approach. As discussed in Section 6.2.2, the relaxed algorithm is less precise than the strict algorithm, however, it is much faster in terms of the time complexity (see Theorems 1 and 2). Thus, we deduce that the relaxed algorithm is preferred since it achieves comparable results in less time. (See Section 6.3.3 for an experimental analysis of the efficiency.)

It should be noted that generating an AKG at $\sigma = 100\%$ is a special case, in which nodes are only clustered and fused when they have exactly identical 1-hop neighborhoods. Thus, such abstraction eliminates only truly redundant nodes and edges. In the case of the ROBOKOP KG (Figures 6.5c–6.5d), we observed three main contributing factors to the drastic size reduction seen at $\sigma = 100\%$. First, multiple of the biomedical databases that ROBOKOP includes are hierarchical in nature. As a result, there are many parent and child nodes in the KG that have the same 1-hop neighborhoods, since they interact similarly with other biomedical entities. Second, there are many chemicals in existence that have nearly the same makeup and behavior. However, individual chemicals are represented in ROBOKOP by separate nodes in the graph, yielding many similar nodes. Third, in the common phenomenon known as node promiscuity, some entities are overexplored and thus have many connections in KGs, while others are underexplored and have few connections (Kleinberg 1999; Newman 2018). Underexplored entities are more likely to be clustered in our approach, because it is easier

[8] Recall that the relaxed and strict node-clustering algorithms are equivalent when $\sigma = 100\%$. Thus, we report these results only once.



(a) Number of DRKG Nodes

(b) Number of DRKG Edges

(c) Number of ROBOKOP Nodes

(d) Number of ROBOKOP Edges

(e) Number of Hetionet Nodes

(f) Number of Hetionet Edges

Figure 6.5: The numbers of nodes ((a), (c), and (e)) and of edges ((b), (d), and (f)) both in the input knowledge graphs DRKG (a)–(b), ROBOKOP (c)–(d), and Hetionet (e)–(f) used in our experiments and in several abstract knowledge graphs generated by the proposed GAME+ approach using both the relaxed and strict node clustering approaches, see Section 6.3.2. The x-axes indicate the graphs, with OKG referring to the original input knowledge graph and AKGσ referring to the abstract knowledge graph generated with the node-similarity threshold σ%. The y-axis indicates the number of nodes in (a), (c), and (e) and the number of edges in (b), (d), and (f).

for smaller neighborhoods to have a high similarity. However, with both DRKG (Figures 6.5a–6.5b) and Hetionet (Figures 6.5e–6.5f), we do not observe such a drastic size reduction at high similarity levels. As these factors may vary between KGs, we expect that some other KGs may or may not have such a drastic size reduction at high similarity thresholds, depending on the levels of redundant information present. After all, node similarity reflects redundancies.

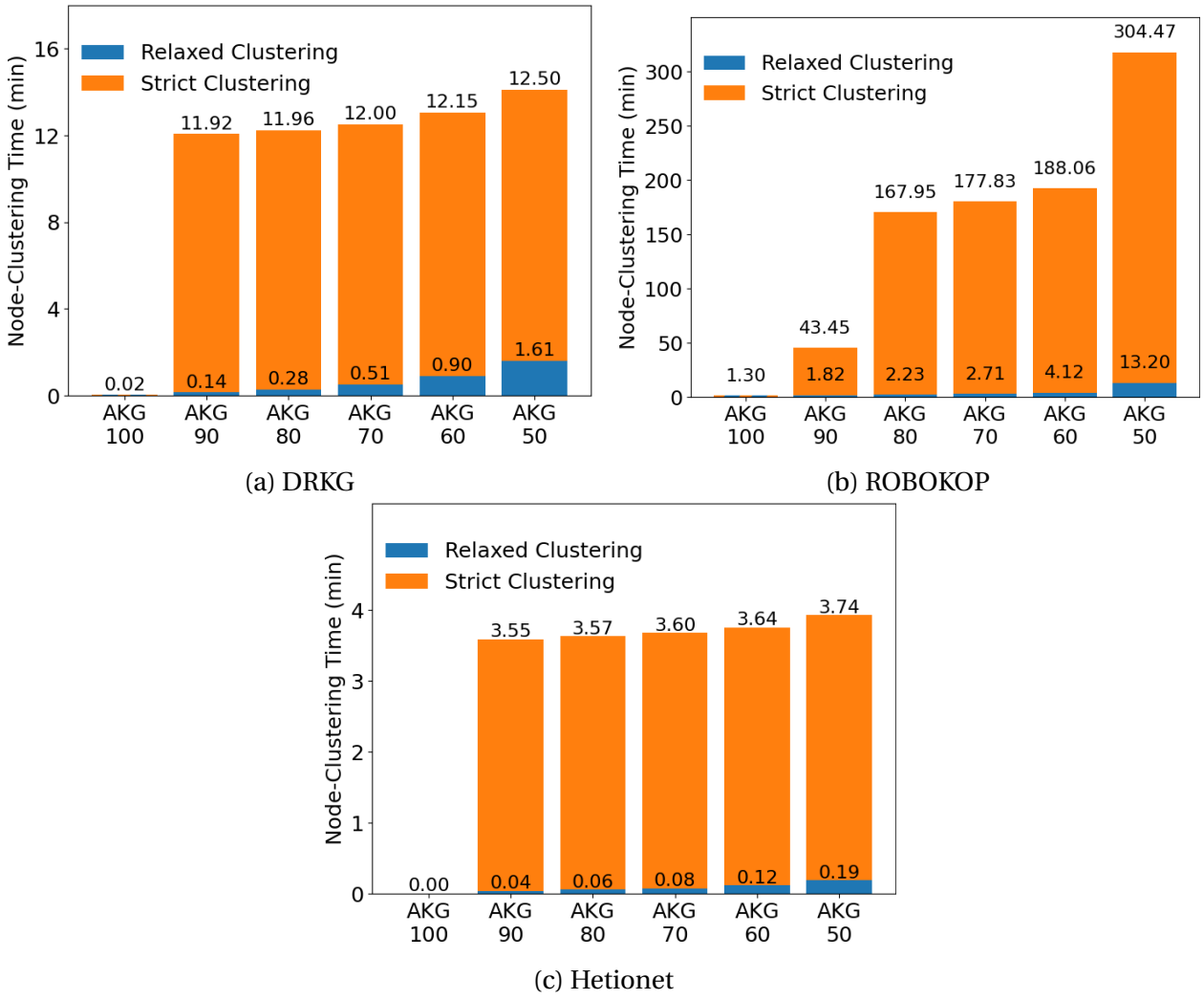
Viewing abstraction as a technique for eliminating KG redundancy, we take this analysis as evidence that these KGs contain some redundant information, and that reduction ability of the proposed GAME+ approach can effectively reduce the sizes of these KGs.

6.3.3 Efficiency of Node Clustering

In this experiment we tested the efficiency of the two proposed node-clustering approaches: relaxed and strict. Theorems 1–2 of Section 6.2.2 presented the worst-case theoretical time complexity of the two approaches. In summary, our experimental results confirm the theoretical result that the relaxed node-clustering approach is more efficient than the strict approach. As seen in Section 6.3.2, both node-clustering approaches provide comparable, effective compression, so we deem the relaxed approach to be more useful given its efficiency.

For the experiment, we recorded the time elapsed during the node-clustering phase of GAME+ when generating the AKGs in the experiment described in Section 6.3.2. Figure 6.6 shows the time taken to cluster the nodes when generating each AKG. We can see that the relaxed node-clustering algorithm was substantially faster regardless of the input KG $\mathscr{G}$ or the input node-similarity threshold σ. In fact, on DRKG (Figure 6.6a), ROBOKOP (Figure 6.6b), and Hetionet (Figure 6.6c) the time taken for relaxed clustering was less than 13%, 5%, and 5%, respectively, of the time taken for strict clustering, making relaxing clustering an advantageous option. We take these results as confirmation of the theoretical results presented in Theorems 1–2.

In addition, we observe that the time taken for the clustering on the three input KG does, in fact, reflect the number of nodes in each graph. Hetionet (Figure 6.6c) has the fewest nodes of the three graphs (with 45,158), and it also displayed the fastest node-clustering performance, with the relaxed node-clustering algorithm taking fewer than 12 seconds in each case, and the strict node-clustering algorithm taking about $3-4$ minutes in each case. Next, DRKG (Figure 6.6a) has just over twice as many nodes as Hetionet (with 97,238), and it displayed a slower node-clustering performance than Hetionet (but faster than ROBOKOP). The time taken with the relaxed approach was less than 1 minute and 37 seconds in all cases, whereas the time taken with the strict approach was almost 12 minutes or more all cases. Finally, ROBOKOP (Figure 6.6b) has the most nodes of the three graphs by far (with 412,889), which is reflected in



(a) DRKG

(b) ROBOKOP

(c) Hetionet

Figure 6.6: The times taken to complete the node-clustering phase of GAME+ on the input knowledge graphs DRKG (a), ROBOKOP (b), and Hetionet (c), see Section 6.3.3. The x-axes indicate the graphs, with AKGσ referring to the abstract knowledge graph generated with the node-similarity threshold σ%. The y-axis indicates the node-clustering time in minutes.

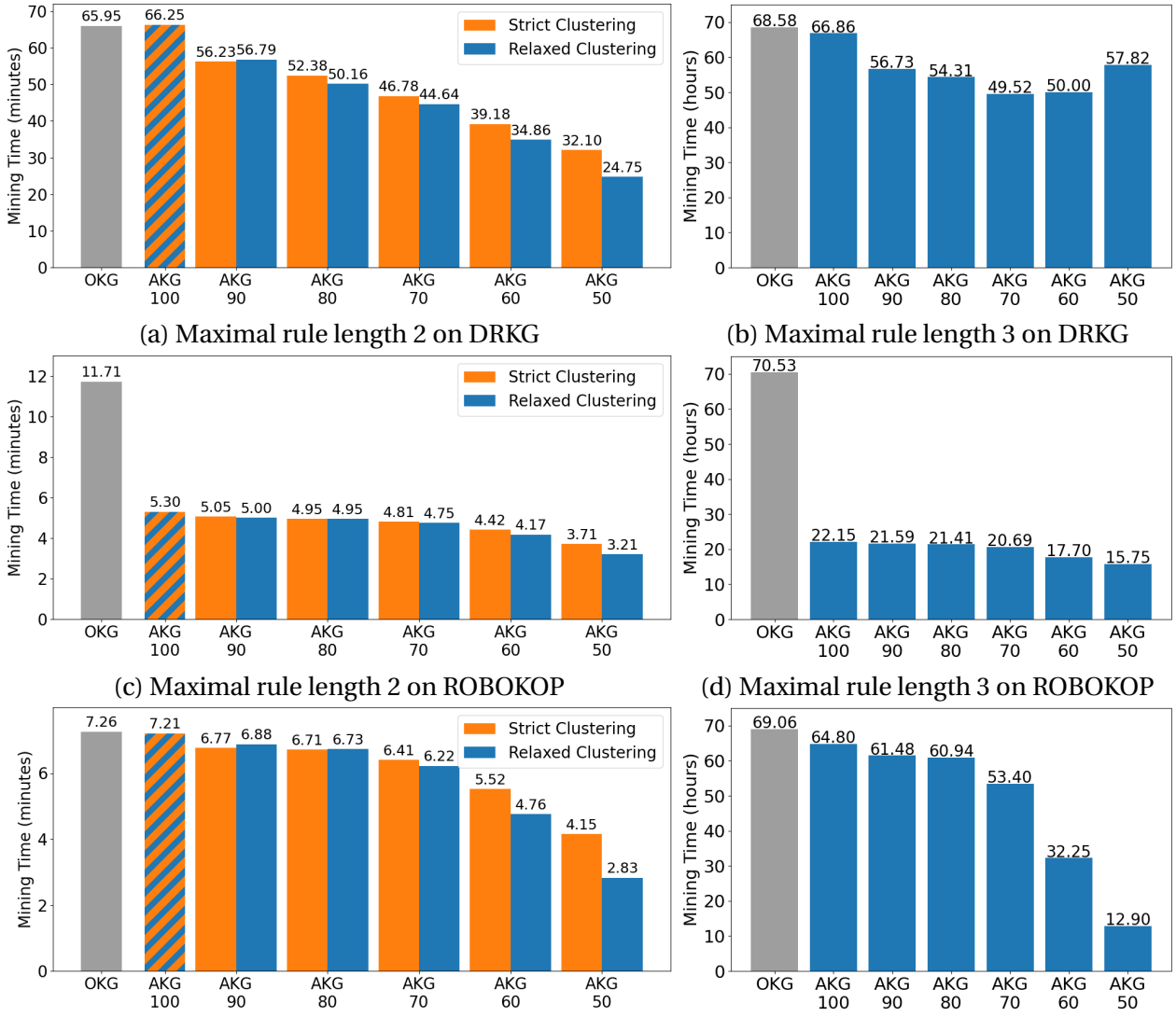
its node-clustering times being much higher than those of the other two graphs. On ROBOKOP, the relaxed approach took about 2−14 minutes, and the strict approach took about 45 minutes to 5 hours. Especially on the larger graphs, we can see the clear advantage of using the relaxed node-clustering algorithm. That is, as the number of nodes in the KGs increases, the time saved by the relaxed approach increases as well.

6.3.4 Efficiency of Rule Mining

In this experiment we tested the impact of the GAME+ KG-abstraction approach on one of downstream KG-analytics tasks, rule mining (Lajus et al. 2020; Ahmadi et al. 2020; Wang and Li 2015; Sadeghian et al. 2019). As one of the key motivations for KG abstraction is to speed up KG analytics, we test the significance of the impact here. In summary, mining rules on the AKGs generated by our GAME+ approach was faster than mining rules on the original KG. These results indicate that abstraction can be an effective technique to speed up rule mining and potentially other downstream KG analytics tasks.

We mined rules on each of the three input KGs (see Section 6.3.1.2) and on the AKGs generated via the experiment described in Section 6.3.2. We mined all the rules using AMIE (Lajus et al. 2020), see Section 6.3.1.4 for details. We looked at two scenarios: obtaining rules of maximal length 2 and of maximal length 3. We mined rules of maximal length 2 for each of the predicate types in DRKG, ROBOKOP, and Hetionet. That is, we mined rules with each predicate in the head of the rule. Due to the time required to mine rules of length 3, we mined rules of maximal length 3 for 52 of the 99 DRKG predicate types, 32 of the 84 ROBOKOP predicate types, and all 24 of the Hetionet predicate types. Additionally, we mined rules of maximal length 3 on each of the three input KGs and on the AKGs generated via the relaxed node-clustering approach *(relaxed AKGs)*, but not those generated via the strict node-clustering approach *(strict AKGs)*. Based on the sizes of these graphs (see Section 6.3.2) and on the results from mining rules of maximal length 2 (discussed below), we anticipate that the results from mining rules of maximal length 3 on the strict AKGs would be comparable to those results on the relaxed AKGs.

Figure 6.7 shows the total time taken to mine rules of maximal lengths 2 and 3 on the DRKG (Figures 6.7a–6.7b), ROBOKOP (Figures 6.7c–6.7d), and Hetionet (Figures 6.7e–6.7f) KGs and on the AKGs. We can see that mining rules on the AKGs generated by our GAME+ approach was faster than mining rules on the input KGs. In most cases, we also observe that the trends in the rule-mining time reflect the trends seen with the graph sizes in Section 6.3.2. For instance, the most substantial mining-time reduction is seen on ROBOKOP (Figures 6.7c–6.7d), where mining rules on the AKGs was always significantly faster – less than half the amount of time – than on ROBOKOP. As expected, there is usually a decrease in the time taken to perform rule



(a) Maximal rule length 2 on DRKG

(b) Maximal rule length 3 on DRKG

(c) Maximal rule length 2 on ROBOKOP

(d) Maximal rule length 3 on ROBOKOP

(e) Maximal rule length 2 on Hetionet

(f) Maximal rule length 3 on Hetionet

Figure 6.7: The runtimes for rule mining on the input knowledge graphs DRKG (a)–(b), ROBOKOP (c)–(d), and Hetionet (e)–(f) and on the abstract knowledge graphs generated by the proposed GAME+ approach, see Section 6.3.4. The x-axes indicate the graphs, with OKG referring to the original input knowledge graph and AKGσ referring to the abstract knowledge graph generated with the node-similarity threshold σ%. The y-axis indicates the mining time in minutes in (a), (c), and (e) and in hours in (b), (d), and (f).

mining with the decrease of the similarity threshold σ. The only exceptions to this trend are in the cases of mining rules of maximal length 3 on the AKGs generated from DRKG by the proposed GAME+ approach with node-similarity thresholds $\sigma = 60\%$ and $\sigma = 50\%$ (Figure 6.7b).[9] These cases are the result of mining rules for a small handful of DRKG predicates that lie in dense areas of the graph. We still take these as good results because these times are still less than the rule-mining time on the original DRKG graph.

Additionally, we observe that the relaxed and strict AKGs generated with high values of the similarity threshold σ provided similar speed improvements when mining rules of maximal length 2 (Figures 6.7a, 6.7c, and 6.7e). However, in most cases, the time taken to mine rules on the relaxed AKGs was less than on the strict AKGs. As was observed with the graph sizes (see Section 6.3.2), as the value of σ decreases, the improved performance on the relaxed AKGs becomes more clear. As evidenced by Theorems 1–2 and by the results presented in Section 6.2.2.2, it is faster to generate the relaxed AKGs than the generate the strict AKGs. Since rule mining performance is also faster on the relaxed AKGs, we again see a clear advantage of the relaxed node-clustering approach.

Rule mining on the AKGs for rules of maximal length 3 (Figures 6.7b, 6.7d, and 6.7f) took more than a day in many cases. We still take this as a good result, because these times are, in several cases, substantially less than the rule-mining time on the original input KGs, e.g., mining rules on the AKGs generated from ROBOKOP saved up to 78% of the rule-mining time on the original ROBOKOP KG, and mining rules on the AKGs generated from Hetionet saved up to 81% of the rule-mining time on the original Hetionet KG. Moreover, in our experience, analytics tasks rarely require one to mine rules for all predicates in the KG. A more typical use case would involve mining rules for a subset of predicates that is relevant to the given task. For example, for the drug-repurposing task discussed in Section 1.3.3, researchers may want to focus their rule-mining efforts specifically on predicates that can provide insights on drug-disease interactions, drug-target interactions, and target-disease interactions. Such predicates in the ROBOKOP KG include `contraindicated_for`, `has_phenotype`, and `gene_associated_with_condition`. In scenarios in which rules are mined for a subset of all the KG predicates, we anticipate that the speedup secured by the proposed GAME+ approach would enable rules to be mined within reasonable amounts of time, e.g., hours – not days.

6.3.5 Effectiveness of Rule Mining

In this experiment, we tested the effectiveness of KG rule mining on the AKGs generated by the proposed GAME+ approach. We understand effectiveness as losslessness, that is, being

[9]The reasons for this phenomenon are discussed in Section 6.3.5 as they are largely related to the rule-mining effectiveness, i.e., the number of rules mined.

Table 6.2: The numbers of missing rules, i.e, those mined on the input knowledge graphs DRKG (a), ROBOKOP (b), and Hetionet (c), but *not* on the abstract knowledge graphs generated by the proposed GAME+ approach, see Section 6.3.5. AKGσ refers to the abstract knowledge graph generated with the node-similarity threshold σ%. The results for abstract knowledge graphs generated with relaxed node clustering are identical to those generated with strict node clustering.

(a) DRKG

Graph	Missing Rules	
	Length 2	Length 3
AKG100	0	0
AKG90	0	0
AKG80	0	0
AKG70	0	0
AKG60	0	0
AKG50	0	0

(b) ROBOKOP

Graph	Missing Rules	
	Length 2	Length 3
AKG100	0	0
AKG90	0	0
AKG80	0	0
AKG70	0	0
AKG60	0	0
AKG50	0	0

(c) Hetionet

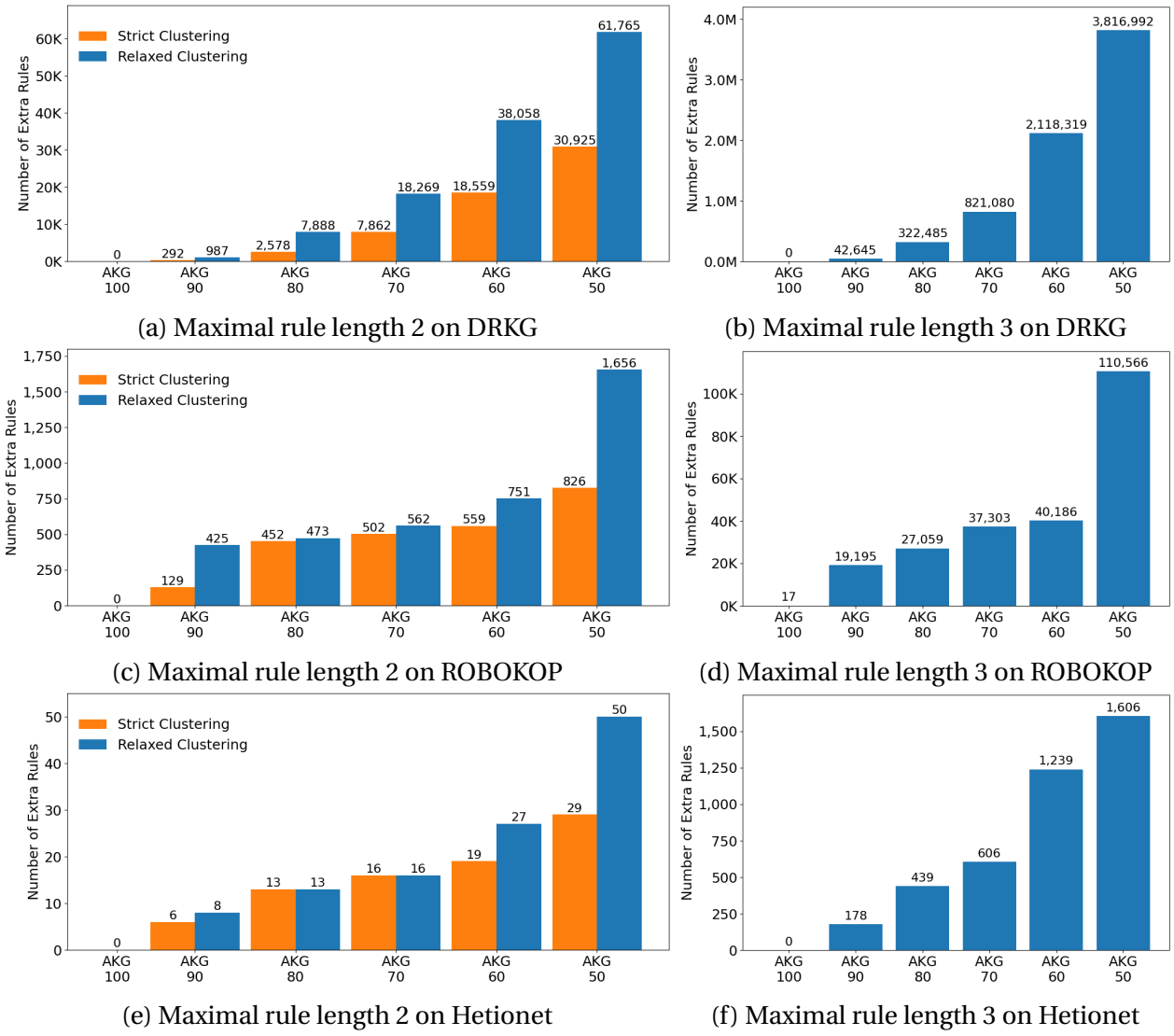
Graph	Missing Rules	
	Length 2	Length 3
AKG100	0	0
AKG90	0	0
AKG80	0	0
AKG70	0	0
AKG60	0	0
AKG50	0	0

able to mine on the AKGs all of the rules that could be obtained from the original KG. This way, reducing the given KG would not lead to significant loss of information or predictive power for rule mining. In summary, the results suggest that rule mining on the AKGs generated by the GAME+ approach can be potentially more effective than on the original KGs. Combined with the results presented in Section 6.3.4, with the proposed approach we see potential for rule mining to be much more efficient and effective on AKGs as compared with the original KGs.

We analyzed the rule sets generated via the experiment described in Section 6.3.4, by comparing and contrasting the rules in each set. We denote by $\mathscr{R}^{\mathscr{G}}$ the rules mined on the KG $\mathscr{G}$. Perfectly effective rule mining on AKGs would generate *all* the rules mined on the original graph. To this end, for the rule set $\mathscr{R}^{\mathscr{G}^A}$ mined on the AKG $\mathscr{G}^A$, we identified the number of rules in the rule set $\mathscr{R}^{\mathscr{G}}$ mined on the original KG that are *missing* from $\mathscr{R}^{\mathscr{G}^A}$, i.e., $|\mathscr{R}^{\mathscr{G}} \setminus \mathscr{R}^{\mathscr{G}^A}|$.[10] Moreover, we identified the number of rules in $\mathscr{R}^{\mathscr{G}^A}$ that are *extra* rules beyond those in $\mathscr{R}^{\mathscr{G}}$, i.e., $|\mathscr{R}^{\mathscr{G}^A} \setminus \mathscr{R}^{\mathscr{G}}|$.

From the DRKG, ROBOKOP, and Hetionet input KGs, we mined a total of 182,055 rules, 31,593 rules, and 590 rules of maximal length 2, respectively, and 2,695,918 rules, 1,108,254 rules, and 9,156 rules of maximal length 3, respectively. For the rule sets generated from the AKGs considered in Section 6.3.4, Table 6.2 shows the number of missing rules when compared to the rule sets obtained from the corresponding input KG. (The numbers of missing rules

[10]We found that some rules that could be found in the input KGs were not in the AMIE rule-mining output. To resolve this, we manually checked the KGs for each "missing" rule to see whether or not it was present. We report here as *missing* only those rules that we were not able to recover in this way.



(a) Maximal rule length 2 on DRKG

(b) Maximal rule length 3 on DRKG

(c) Maximal rule length 2 on ROBOKOP

(d) Maximal rule length 3 on ROBOKOP

(e) Maximal rule length 2 on Hetionet

(f) Maximal rule length 3 on Hetionet

Figure 6.8: The numbers of extra rules, i.e, those mined on the abstract knowledge graphs generated by the proposed GAME+ approach, but *not* on the input knowledge graphs DRKG (a), ROBOKOP (b), and Hetionet (c), see Section 6.3.5. AKGσ refers to the abstract knowledge graph generated with the node-similarity threshold σ%.

from AKGs generated with relaxed and strict clustering are identical, so only one set of results is displayed.) Observe that all of the rules sets mined on the AKGs are missing no rules at all, indicating that no inference rules are lost due to the abstraction process. This lossless rule mining suggests the AKGs generated by the proposed GAME+ approach can yield results that are just as effective as those on the original input KGs.

Interestingly, rule mining on almost all the AKGs generated by the proposed GAME+ approach produced more rules than rule mining on the original input KGs. Figure 6.8 displays the number of extra rules mined on each AKG when compared to the rule sets obtained from the corresponding input KG. Observe that the numbers of extra rules increase with the decrease in the value of the node-similarity threshold σ. Moreover, the numbers of extra rules obtained from the relaxed AKGs is consistently higher than the numbers of extra rules obtained from the strict AKGs, which is a direct result of the increased compression provided by the relaxed AKGs, see the discussion of Figure 6.5 in Section 6.3.2. One cause for the additional rules emerging on the AKGs is that the proposed KG-reduction approach can shorten KG paths. That is, a rule of length 3 in the original graph may have been shortened to a rule of length 2 in the abstract graphs if two atoms in the body of the rule were clustered together and then reduced into a consolidated node. Thus, the reduction provided by the GAME+ approach can bring more paths into the length 2 and length 3 ranges than the respective numbers in the original graph. See Figure 6.3 and the corresponding discussion in Section 6.2.3.1 for examples. This could provide domain scientists with rules that would otherwise not be discovered, potentially making the rule mining even more effective.

We specifically notice that the AKGs generated from DRKG yielded significantly more rules than the original DRKG graph. After analyzing DRKG, we have found that this is due to the density of the graph. In particular, most of the extra rules are for a handful of predicates that connect *gene* entities, and these entities lie in the densest areas of the graph. Moreover, the density and the vast number of extra rules are the causes of the rule-mining performance observed on these AKGs, see Figure 6.7b. Recall that in general, the time taken to perform rule mining decreases with the decrease of the similarity threshold σ. The only exceptions are when mining rules of maximal length 3 on the AKGs generated from DRKG with $\sigma = 0.6$ and $\sigma = 0.5$. In all other instances, we observed that the time saved as a result of the graph compression outweighed the time lost by mining extra rules. In these two cases, the tradeoff went in the opposite direction. We still take these as good results because the increased effectiveness of the rule-mining task could potentially yield important insights for domain scientists.

6.3.6 Discussion

We now summarize the key takeaways of our experimental results. The first insight is that our proposed GAME+ approach generates lossless AKGs that can be substantially smaller in size than the input KGs, see Section 6.3.2. This indicates that the approach can be a useful KG-reduction technology. Additionally, in Section 6.3.3 we saw experimental evidence confirming the theoretical results of Theorems 1–2 (see Section 6.2.2) that the relaxed node-clustering approach is more efficient than the strict node-clustering approach. Further, in Section 6.3.4 we saw that performing rule mining on the AKGs generated by GAME+ can be much more efficient than on the original KGs, suggesting that the proposed KG reduction can enable more efficient KG analytics. Finally, the results from Section 6.3.5 suggest that rule mining performed on the AKGs generated by GAME+ can be potentially more effective than rule mining on the original KGs. Overall, these results suggest that our proposed GAME+ approach can be useful for reducing the sizes of KGs, by enabling potentially more efficient and effective KG analytics in (at least) the case of rule mining.

6.4 Case-Study Results

We now present the results of a case study analyzing the meaningfulness of the clusters generated by the proposed GAME+ approach on the ROBOKOP knowledge graph *(KG)*.[11] In summary, our results suggest that clustering nodes according to the 1-hop neighborhood similarity function (see Eq. 6.6 of Section 6.3.1.3) can generate clusters whose nodes have a semantically meaningful relationship. That is, the structural similarities captured by the 1-hop neighborhood similarity function can reflect the semantic similarities between nodes

For the case study, we had a biomedical expert on our team analyze several node clusters generated by the node-clustering phase of the proposed GAME+ approach. From the node clusters generated via the experiment described in Section 6.3.2, we randomly selected one cluster of each of the 12 ROBOKOP node types. Due to the manual efforts required for this case study, we focused on the runs of the GAME+ approach that used the relaxed node-clustering approach with node-similarity threshold $\sigma \in \{100\%, 90\%, 80\%, 70\%, 60\%, 50\%\}$. Thus, we randomly selected 72 total clusters (one of each node type from each of the six runs), and presented them to the expert. We instructed them to consider how semantically meaningful each cluster of nodes is, from a biomedical perspective. That is, they analyzed whether or not the nodes of each cluster are meaningfully related. They were asked to rate each cluster according to the

[11] Since the case-study analysis was manual and time consuming, we focus solely on the ROBOKOP KG in this section.

Table 6.3: A summary of the ratings assigned by the biomedical expert on our team to the node clusters generated by node-clustering phase of the proposed GAME+ approach, where clusters with expert rating (1) are the least semantically meaningful and clusters with expert rating (5) are the most semantically meaningful, see Section 6.4.

Similarity threshold (σ)	# 5s	# 4s	# 3s	# 2s	# 1s	Median
100%	0	4	4	2	2	3
90%	0	7	1	2	2	4
80%	1	8	2	1	0	4
70%	1	10	0	1	0	4
60%	2	4	5	1	0	3.5
50%	2	4	1	1	4	3.5
Totals	6 (8.0%)	37 (51.4%)	13 (18.1%)	8 (11.1%)	8 (11.1%)	

following scale:

1. No meaningful relationship.
2. Weakly meaningful relationship.
3. Meaningful relationship.
4. Strongly meaningful relationship.
5. Nodes represent identical entities.

We also asked the expert to provide brief justifications for their ratings as a qualitative evaluation of the clusters.

Table 6.3 shows, for each node-similarity threshold σ, the number of examined clusters that received each rating from (1)-(5), along with the median rating. We first notice that only eight out of the 72 clusters (11%) were given a rating of (1), which indicates that there is no meaningful relationship between the nodes in those clusters. This means that the other 64 of the 72 clusters (89%) were found to be meaningful in some way, indicating that the clusters generated during the node-clustering phase of the GAME+ approach can be valuable for domain experts. In fact, a majority of the clusters ($43/72 = 60\%$) were given ratings of (4) or (5), which indicate a strongly meaningful relationship between the nodes in the clusters or that the nodes represent identical entities. Moreover, we notice that the median rating is between 3 and 4 for all values of the node-similarity threshold σ, which suggests that the node clusters generated by the GAME+ approach can be meaningful at various levels of the threshold σ.

Table 6.4: Node clusters of various node types generated by the node-clustering phase of the proposed GAME+ approach that received various meaningfulness ratings according to the biomedical expert on our team; (a), (b), and (c) show clusters rated (5), (4), and (3), respectively, see Section 6.4.

(a) A cluster of type *disease* that received an expert meaningfulness rating of (5).

	Node Names
1.	X-linked intellectual disability, Porteous type
2.	X-linked intellectual disability, Sutherland-Haan type

(b) A cluster of type *phenotypic feature* that received an expert meaningfulness rating of (4).

	Node Names
1.	Aplasia cutis congenita on trunk or limbs
2.	Aplasia cutis congenita over posterior parietal area

(c) A cluster of type *gene* that received an expert meaningfulness rating of (3).

	Node Names
1.	ZNF271P
2.	ZNF807P
3.	ZNF663P

We next present a few clusters, along with their ratings, and explore the qualitative evaluations provided by the biomedical expert on our team. Table 6.3 presents three example clusters of different node types that received different ratings.

First, Table 6.4a shows a cluster of two nodes of type *disease* that received a meaningfulness rating of (5). This cluster consists of two types of *X-linked intellectual disabilities* known as *Porteous syndrome* and *Sutherland-Haan syndrome*. Both syndromes fall under *Renpenning syndrome* and display the same effects. Therefore, these nodes are nearly the same entity, and are meaningfully related.

Next, Table 6.4b shows a cluster of two nodes of type *phenotypic feature* that received a meaningfulness rating of (4). This cluster consists of *aplasia cutis congenita on trunk or limbs* and *aplasia cutis congenita over posterior parietal area*. These nodes represent the same skin defect, i.e., *aplasia cutis congenita*, just found on different parts of the body, i.e., the *trunk* (torso) or *limbs* and the *posterior parietal area* of the head. Therefore, they are meaningfully related.

Finally, Table 6.4c shows a cluster of three nodes of type *gene* that received a meaningfulness rating of (3). This cluster consists of the genes *ZNF271P*, *ZNF807P*, and *ZNF663P*, which all belong to the family of *zinc finger proteins*, and are meaningfully related in that way. However, there are many zinc finger proteins, and these three in particular are contained in different chromosomes, so they are not as strongly related as the nodes in the other clusters discussed.

Overall, these results suggest that the node-clustering phase of the proposed GAME+ approach can generate via the 1-hop neighborhood similarity function (see Eq. 6.6 of Section 6.3.1.3) node clusters that are semantically meaningfully related. Therefore, the output node clusters can be insightful for domain scientists using the GAME+ approach.

6.5 Summary

With the sizes of domain knowledge graphs *(KGs)* often being too large to be processed efficiently, graph-data reduction has become an indispensable technique for improving the efficiency of downstream analytics. In this context, we studied the trade-off between the performance and accuracy of one type of KG analytics, inference-rule mining, on reduced (summarized) KGs as compared with the original KGs. Toward addressing the trade-off challenge, we introduced a domain- and task-independent KG-summarization approach called GAME+ that generates reduced abstract KGs from a given graph. The GAME+ approach aims to (i) improve rule-mining efficiency by reducing the size of a given graph, while (ii) maintaining rule-mining effectiveness by preserving as much information as possible. To achieve these aims, GAME+ uses similarity between KG nodes to identify and eliminate repetitive KG triples. Our experimental results characterize the impact of GAME+ data reduction on KG rule mining, suggesting that the GAME+ approach can preserve all the information from the given (original) graph, while potentially significantly improving the rule-mining efficiency and effectiveness. In addition, our case study results suggest that the node clusters generated by the node-clustering phase of GAME+ can be semantically meaningful and insightful for domain scientists.

We anticipate that the KG reduction provided by the proposed GAME+ approach can enable efficient, yet effective, results for diverse applications and use cases. As part of our future work, we are interested in testing the GAME+ approach with other domains, similarity functions, and analytical tasks. Additionally, we would like to explore the potential benefits of allows for different similarity thresholds to be used for different entity-types.

CHAPTER

7

USING EXTRACTION AND EVALUATION OF EXPLANATIONS FOR DRUG REPURPOSING ON KNOWLEDGE GRAPHS

Portions of this chapter have been published in:

> **Kara Schatz**, Cleber Melo-Filho, Alexander Tropsha, and Rada Chirkova, "Explaining Drug-Discovery Hypotheses Using Knowledge-Graph Patterns," in *2021 IEEE International Conference on Big Data (Big Data)*, pp. 3709–3716, IEEE, December 2021.

In this chapter we introduce our proposed approach, called Knowledge Graph Explanation and Evaluation Generation (KGEG), for addressing the problem of extracting and evaluating explanations for knowledge-graph hypotheses, i.e., inferred triples. We propose that this approach be used during the fourth stage of the four C's pipeline presented in Chapter 1 (see Figure 1.1): *completion*.

This chapter is organized as follows. In Section 7.1 we formalize the problem, and in Section 7.2 we introduce the proposed KGEG approach. In Sections 7.3 and 7.4 we present the results of our experiments and biomedical expert user studies, respectively. We explore limitations of our work in Section 7.5. In Section 7.6 we conclude and suggest future work.

Relevant preliminaries for this chapter can be found in Sections 3.1 and 3.4–3.5.

7.1 Problem Statement

We now introduce the formal problem statement for explanation extraction and evaluation for knowledge graph (KG) hypotheses. Given (i) a KG $\mathscr{G} = (\mathscr{N}, \mathscr{T}, \phi, \mathscr{P}, \mathscr{L})$, (ii) a set of Horn rules $\mathscr{R}$, and (iii) a *hypothesis triple* $q = (e_0, p, e_1)$ with $p \in \mathscr{P}$ and $e_0, e_1 \in \mathscr{N}$, our goals are to (a) generate with $\mathscr{G}$ and $\mathscr{R}$ a set $\mathscr{E}$ of possible (entity-typed) explanations for q, and (b) assign to each $E \in \mathscr{E}$ a reliable, understandable, and data-supported score representing the extent to which E justifies the truth of q.

7.2 The Proposed KGEG Approach

We now introduce an approach, KGEG: Knowledge Graph Explanation and Evaluation Generation, that addresses both goals in the problem statement of Section 7.1: explanation extraction, see Section 7.2.1 for the details, and explanation evaluation, see Section 7.2.2 for the details.

7.2.1 Explanation Extraction in KGEG

7.2.1.1 Algorithm

In the problem statement of Section 7.1, explanation extraction requires three inputs: a knowledge graph (KG) $\mathscr{G}$, a set of Horn rules $\mathscr{R}$, and a hypothesis triple $q = (e_0, p, e_1)$. Each serves a specific purpose: $\mathscr{G}$ serves as a source of facts that we can use to find supporting evidence for the explanations being built; $\mathscr{R}$ allows us to rewrite current explanations using known inference rules; and the hypothesis q focuses the search and provides a starting point.

Our explanation-finding algorithm is based on backward chaining (Russell and Norvig 2021), that is, to find explanations for the input (goal) hypothesis q the algorithm works backwards from q. That is, the first explanation we consider is the hypothesis q itself. Then, we repeatedly (i) attempt to find the current explanation in the KG $\mathscr{G}$, and (ii) generate new possible explanations by rewriting atoms in the current explanation. To rewrite an atom a, we apply an appropriate rule $r \in \mathscr{R}$, replacing a with $body(r)$ to infer a. The process continues until either (1) all the explanation atoms are grounded (i.e., can be found) in $\mathscr{G}$, or (2) there are no rules in $\mathscr{R}$ that can facilitate further rewritings.

We now detail the main framework of our explanation-derivation approach KGEG: Knowledge Graph Explanation Generation, see Algorithm 8. Its first three inputs are the inputs in the

problem statement of Section 7.1: KG $\mathcal{G}$, set of Horn rules $\mathcal{R}$, and hypothesis triple $q = (e_0, p, e_1)$. We add one more input, maximum search depth d, to ensure termination by restricting the number of recursive rule rewritings allowed. Algorithm 8 outputs a set of entity-typed explanations $\mathcal{E}$ for the input hypothesis q, where each explanation $E \in \mathcal{E}$ is of the form described in Section 3.5.

Algorithm 8: Deriving explanations

Input : Knowledge graph $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \phi, \mathcal{P}, \mathcal{L})$, set of Horn rules $\mathcal{R}$, hypothesis triple $q = (e_0, p, e_1)$, maximum search depth d.

Output : Set $\mathcal{E}$ of $\mathcal{G}$- and $\mathcal{R}$-generated explanations for q.

```
1  ℰ ← ∅; // set of explanations found for q
2  q.found ← false; q.depth ← 0; // initialize attributes for q
3  𝒬 ← {q ⇐ q}; // initialize queue of possible explanations to
       consider
4  while 𝒬 ≠ ∅ do
5      E ← some explanation from 𝒬; // select an explanation to consider
6      if ∄ atom a ∈ E s.t. a.found = false then
7          ℰ ← ℰ ∪ imposeEntityTypes(E); // successful entity-typed
               explanation
8      else
9          a ← some atom from E s.t. a.found = false; // select an atom to
               consider
           // first, ground E with a
10         𝒬 ← 𝒬 ∪ groundAtom(E, a, 𝒢); // add E back to 𝒬 if ground with
               a
           // next, rewrite a in E
11         𝒬 ← 𝒬 ∪ rewriteAtom(E, a, ℛ, d); // add all rewritings of a in E
               to 𝒬
12 return ℰ;
```

Algorithm 8 implements a form of iterative backward chaining (Russell and Norvig 2021) that is suited for our application. Standard iterative backward chaining proceeds as follows. A queue is maintained, with possible explanations to be considered. Initially, it contains solely the input hypothesis (line 3). In each iteration of the main loop (lines 4–11), an explanation is removed from the queue for processing (line 5). If data are found to support it, the explanation is grounded (line 10). Further, if data exist to prove the explanation, then *true* is output and the process terminates, indicating that an explanation was found. Otherwise, the explanation is

rewritten (if possible) using rules, and the rewritings are added to the queue (line 11). If there is nothing left on the queue and the process has not terminated, then *false* is output, indicating that no explanation was found.

We have modified this approach to suit the needs of our problem. First, to each atom a of an explanation E, we attach two attributes: $a.found$ and $a.depth$. The value of $a.found$ is *true* if we have found a grounding for E with a in $\mathcal{G}$, i.e., the explanation pattern with a exists in $\mathcal{G}$; otherwise, $a.found$ is *false*. The value of $a.depth$ for an atom a in explanation E represents the number of recursive rewritings required for a to appear in E. E.g., all atoms that appear in E after the first rewriting have depth 1, and atoms that appear in E by rewriting an atom of depth 1 have depth 2. Initially, $q.found$ and $q.depth$ are set to *false* and 0, respectively (line 2).

Maintaining $a.found$ allows us to consider only one atom (that has not been found) with each iteration of the main loop (line 9), rather than the entire explanation E. The atom is first grounded with the found atoms of E (line 10) to determine if the partial pattern exists, and then rewritten within E (line 11). The subprocedures $groundAtom$ and $rewriteAtom$ handle these two steps, respectively, see below for the details. Since an explanation pattern can exist only if all its partial patterns also exist, this helps to eliminate useless patterns early (without sacrificing completeness) and, thus, limits the number of large queries applied to the KG $\mathcal{G}$.

The subprocedure $groundAtom$ (line 10) queries $\mathcal{G}$ for a subgraph containing (i) all the found atoms of E, which we know can be grounded in $\mathcal{G}$, and (ii) a, which is the atom that we are interested in finding. If there is such a subgraph, called *grounding*, then $a.found$ is marked as *true*, and E is returned by $groundAtom$. If no groundings exist, then $groundAtom$ returns $\emptyset$.

The subprocedure $rewriteAtom$ (line 11) uses the rules from $\mathcal{R}$ to replace a in E with other atom(s). First, we generate the set of rules $\mathcal{R}_a$ consisting of those rules from $\mathcal{R}$ whose head predicate matches the predicate of a, i.e., those rules that can be used to infer a. For each such rule $r \in \mathcal{R}_a$, we generate a new explanation by replacing a with $body(r)$ in E. For each atom $b \in body(r)$, we set $b.found = false$ and $b.depth = a.depth + 1$. After generating one rewriting for each rule in $\mathcal{R}_a$, these rewritings are returned by $rewriteAtom$. If $\mathcal{R}_a = \emptyset$, then $rewriteAtom$ returns $\emptyset$.

Maintaining $a.depth$ allows us to impose a depth limit on the backward-chaining process. Before performing rule rewritings, the procedure $rewriteAtom$ of line 11 compares $a.depth$ with the maximum search depth d. The rewriting process proceeds if and only if $a.depth < d$. Moreover, we return all possible explanations within the depth limit d, rather than stopping after a single explanation is found. To this end, we maintain an output set $\mathcal{E}$, initially an empty set (line 1), which stores successful explanations, i.e., those whose atoms are all found (lines

6–7).[1] Once the queue $\mathcal{Q}$ is empty, all the successful explanations of the hypothesis q within d have been added to the output set $\mathcal{E}$, which is output (line 12) as the result of Algorithm 8.

Finally, instead of returning explanations with just variables as entities, the algorithm returns entity-typed explanations, see Section 3.5. The procedure $imposeEntityTypes$ (line 7) queries the explanation E against the KG $\mathcal{G}$ and returns all possible entity-typings of E in $\mathcal{G}$.

7.2.1.2 Example

The following example outlines the explanation-extraction process. Suppose we have the following inputs:

$$\begin{aligned}
\mathcal{G} = \{&(\textit{benazepril}, \textit{treats}, \textit{diabetic nephropathy}),\\
&(\textit{kidney failure}, \textit{biomarkerFor}, \textit{diabetic nephropathy}),\\
&(\textit{kidney failure}, \textit{biomarkerFor}, \textit{chronic kidney disease})\};\\
\mathcal{R} = \{&(u, \textit{treats}, w) \Leftarrow (u, \textit{treats}, v) \wedge (v, \textit{hasPhenotype}, w),\\
&(x, \textit{hasPhenotype}, y) \Leftarrow (z, \textit{biomarkerFor}, x) \wedge (z, \textit{biomarkerFor}, y)\};\\
q = \quad &(\textit{benazepril}, \textit{treats}, \textit{chronic kidney disease}).
\end{aligned}$$

As mentioned, the first explanation E that we consider is $q \Leftarrow q$. Our goal is to show that the right-hand side of E is true, so that we can conclude the left-hand side to be true as well. Since $q \notin \mathcal{G}$, E is not yet a successful explanation. Therefore, we rewrite E with rules in $\mathcal{R}$ by finding rules whose heads unify with our right-hand side goal q. Notice that q unifies with the head of the first rule r_1 under the substitution $\sigma_1 = \{u/\textit{benazepril}, w/\textit{chronic kidney disease}\}$. Since $body(r_1) = (u, \textit{treats}, v) \wedge (v, \textit{hasPhenotype}, w)$ allows inference of $head(r_1) = (u, \textit{treats}, w)$, we can rewrite q in the right-hand size of E to get:

$$E : q \Leftarrow (\textit{benazepril}, \textit{treats}, v) \wedge (v, \textit{hasPhenotype}, \textit{chronic kidney disease}).$$

Finding groundings in $\mathcal{G}$ for each atom in E would complete the inference of q, and we would have a successful explanation. Since E cannot yet be grounded in $\mathcal{G}$, we continue working backwards and repeat the rewriting process. We can choose any atom to rewrite; let us choose $(v, \textit{hasPhenotype}, \textit{chronic kidney disease})$, which can be rewritten by the second rule r_2:

$$E : q \Leftarrow (\textit{benazepril}, \textit{treats}, v) \wedge (z, \textit{biomarkerFor}, v) \wedge (z, \textit{biomarkerFor}, \textit{chronic kidney disease}).$$

[1] Before adding E to $\mathcal{E}$, the endpoints of E are replaced with fresh variables.

Now E can be entirely grounded in $\mathcal{G}$ by the substitution $\sigma_2 = \{v/\textit{diabetric nephropathy}, z/\textit{kidney failure}\}$. Therefore, E is a successful explanation for our hypothesis q. Before outputting E, we perform the substitution $\sigma_{var} = \{\textit{benazepril}/e_0, \textit{chronic kidney disease}/e_1\}$, which results in

$$E : (e_0, \textit{treats}, e_1) \Leftarrow (e_0, \textit{treats}, v) \wedge (z, \textit{biomarkerFor}, v) \wedge (z, \textit{biomarkerFor}, e_1).$$

Now the explanation pattern is general and can be potentially reused for other specific hypotheses of the form $(e_0, \textit{treats}, e_1)$. Additionally, we impose entity-types on E, producing one entity-typed explanation:

$$\begin{aligned} E' : (e_0 : \textit{drug}, \textit{treats}, e_1 : \textit{disease}) \Leftarrow & (e_0 : \textit{drug}, \textit{treats}, v : \textit{disease}) \wedge \\ & (z : \textit{disease}, \textit{biomarkerFor}, v : \textit{disease}) \wedge \\ & (z : \textit{disease}, \textit{biomarkerFor}, e_1 : \textit{disease}). \end{aligned}$$

We stop our example here. However, recall that we do not stop after finding the first explanation. In particular, there are still atoms of E that can be rewritten to produce new explanations to consider.

7.2.2 Explanation Evaluation in KGEG

We now present our KGEG solution for assigning to each explanation E that is output by Algorithm 8 of Section 7.2.1 a reliable, understandable, data-supported score representing the extent to which E justifies the truth of the hypothesis q.

Our KGEG solution includes explanation-evaluation metrics that have been inspired by metrics used in rule mining in knowledge graphs (KGs). This area of research, a relative of association-rule mining, has been gaining popularity, see, e.g., Ahmadi et al. (2020); Lajus et al. (2020); Ortona et al. (2018). Several metrics have been introduced for evaluating such rules. We adopt two of the strongest and most widely accepted of these metrics, as well as their combination, for evaluating our KGEG explanations. The metrics that we consider are confidence, head coverage, and the harmonic mean, or F1-score, of confidence and head coverage.

Before defining the metrics, we introduce the terminology. Let E be an explanation of the form

$$E : (e_0, p, e_1) \Leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_n. \tag{7.1}$$

We define the *set of groundings of* E on KG $\mathcal{G}$ as follows.

$$groundings(E) = \{(e_0\sigma, e_1\sigma) \mid \forall \sigma \; such \; that \; A\sigma \in \mathcal{G}\}. \tag{7.2}$$

Here, $A = \{a_1, a_2, \ldots, a_n\}$ is the explanation pattern of E. That is, the set of groundings of E is induced by the set of all substitutions σ that place A into the KG $\mathcal{G}$; σ is then applied to the head of E to obtain the groundings. The *support* of E *in* $\mathcal{G}$ is then defined as the number of groundings of E:

$$support(E) = |groundings(E)|. \tag{7.3}$$

7.2.2.1 Confidence

The first metric that we introduce is *confidence* (Agrawal et al. 1993), defined as

$$conf(E) = \frac{support(E')}{support(E)}. \tag{7.4}$$

Here, E is $(e_0, p, e_1) \Leftarrow a_1 \wedge \cdots \wedge a_n$ and E' is $(e_0, p, e_1) \Leftarrow a_1 \wedge \cdots \wedge a_n \wedge (e_0, p, e_1)$.

The confidence of explanation E is defined as the conditional probability that the hypothesis (e_0, p, e_1) can be grounded in KG $\mathcal{G}$, given that the explanation pattern $A = \{a_1, \ldots, a_n\}$ of E can be grounded in $\mathcal{G}$. Thus, it is a representation of how likely (e_0, p, e_1) is to be true based on A being true. As a result of our considering explanations to be inference rules, confidence can also be viewed as a proportion of the inferences of the rule $(e_0, p, e_1) \Leftarrow a_1 \wedge \cdots \wedge a_n$ that are in the KG $\mathcal{G}$. Overall, it is effectively a measure of how often the explanation pattern A is associated with (e_0, p, e_1) in $\mathcal{G}$ and, therefore, how much we trust A to explain (e_0, p, e_1).

Consider, e.g., the explanation E from Section 3.5, duplicated below.

$$E : (w, \textit{treats}, z) \Leftarrow (w, \textit{treats}, x) \wedge (y, \textit{biomarkerFor}, x) \wedge (y, \textit{biomarkerFor}, z).$$

Then, in the ROBOKOP KG[2] used in our experiments (Section 7.3), $support(E') = 144$ and $support(E) = 204$, thus $conf(E) = 0.706$. In other words, for all the groundings of A in $\mathcal{G}$, 70.6% of them appeared with $(w, \textit{treats}, z)$ as well. Therefore, we would expect that whenever we see the pattern A, there is a 70.6% chance that $(w, \textit{treats}, z)$ is also true.

7.2.2.2 Head Coverage

The next proposed metric is that of head coverage (Lajus et al. 2020), defined as follows:

$$hc(E) = \frac{support(E')}{support((e_0, p, e_1) \Leftarrow (e_0, p, e_1))}. \tag{7.5}$$

Here, E is $(e_0, p, e_1) \Leftarrow a_1 \wedge \cdots \wedge a_n$, and E' is $(e_0, p, e_1) \Leftarrow a_1 \wedge \cdots \wedge a_n \wedge (e_0, p, e_1)$.

[2] `http://robokopkg.renci.org/browser/`

The head-coverage value both parallels and contrasts with the value of confidence, in that it is also defined as a conditional probability, but it is the conditional probability that the explanation pattern A can be grounded in the KG $\mathcal{G}$ given that the hypothesis (e_0, p, e_1) can be grounded in $\mathcal{G}$. In other words, hc measures how likely A is to be true based on (e_0, p, e_1) being known to be true. Our consideration of explanations as inference rules allows for head coverage to be interpreted as the proportion of KG triples with predicate p that could be inferred by the rule $(e_0, p, e_1) \Leftarrow A$. Therefore, head coverage serves as a measure of how often (e_0, p, e_1) is associated with A in $\mathcal{G}$ and, therefore, how relevant A is to the hypothesis (e_0, p, e_1).

For our running example with E, (e_0, p, e_1) is $(w, \textit{treats}, z)$. Then, in the ROBOKOP KG, $support(E') = 144$ and $support((e_0, p, e_1) \Leftarrow (e_0, p, e_1)) = 16135$, which means that $hc(E) = 0.009$. In other words, of all the groundings of $(w, \textit{treats}, z)$ in the ROBOKOP KG, 0.9% of them appeared with the explanation pattern A as well. Thus, we would expect A to explain only 0.9% of the $(w, \textit{treats}, z)$ triples.

7.2.2.3 F1-score

The confidence metric is analogous to precision, i.e., the ratio of correct inferences to total inferences. Likewise, the head coverage is analogous to recall, i.e., the ratio of correct inferences to correct hypotheses. This analogy leads us to a natural definition of our third metric, which is the harmonic mean, i.e., the F1-score, of confidence and head coverage:

$$F1\text{-}score(E) = 2 * \frac{conf(E) * hc(E)}{conf(E) + hc(E)}. \tag{7.6}$$

The standard F1-score metric helps balance the trade-off between precision and recall. In our case, we want to balance the trade-off between the confidence and head coverage of the derived explanations, to find explanations that are both highly associated with, and highly relevant to, the input hypotheses. In our running example, $F1\text{-}score(E) = 0.018$.

7.3 Experimental Results

We now present our experimental results on large-scale biomedical knowledge graphs (KGs). Section 7.3.1 gives details about the implementation and experimental settings, while the remaining sections each detail the setup and results of a single experiment. The results suggest that:

- The KGEG metrics introduced in Section 7.2.2 can accurately and reliably assess explanation quality, see Section 7.3.2;

- The KG-pattern explanations defined in Section 3.5 and generated by the KGEG approach can be reusable for future input hypotheses, see Section 7.3.3;
- The KGEG explanation-extraction approach of Section 7.2.1 can reliably classify hypotheses as *true* or *false* (see Section 7.3.4); also, by pruning a vast majority of the useless explanations, it can efficiently extract all explanations for the input hypothesis, see Section 7.3.5; and
- The KGEG approach can be used to explain and predict many types of relationships, see Section 7.3.6, and it can be tuned to avoid contraindications, see Section 7.3.7.

7.3.1 Implementation and Experimental Settings

We have implemented the KGEG explanation-extraction algorithm and all the code used for running the experiments in Python 3.9. We hosted the KGs used in the experiments in the Neo4j DBMS version 4.2.1 and used the `py2neo` package for the code-DBMS interactions.[3] The experiments were conducted on a computer with an Intel i7-1165G7 CPU processor running at 2.8 GHz with 12GB of RAM and Windows version 10.

7.3.1.1 Data sets and their preprocessing

We used four KGs in our experiments. First, we used a sample of the 2021 version of the KG called Reasoning Over Biomedical Objects linked in Knowledge Oriented Pathways[4] *(ROBOKOP)* (Bizon et al. 2019). The sampling, which was done due to the requirements of the software (Lajus et al. 2020) for generating inference rules, was executed using random walks of the original KG as suggested by Leskovec and Faloutsos (2006). We also used the 2022 version of ROBOKOP; this version, *ROBOKOP2*, is much larger and more cleanly curated than the 2021 version, and has more specific entity-types and predicates.[5] The other two KGs used in the experiments are the Drug Repurposing Knowledge Graph *(DRKG)* (Ioannidis et al. 2020) and *Hetionet* (Himmelstein et al. 2017).[6] Please see Table 7.1a for some details.

Negative Triple Sampling. In our experiments we did KG extraction of *supporting explanations* for triples with target predicate *treats*, as well as of *refuting explanations* for triples with

[3] Despite much effort to optimize the queries used in our code, we found that Neo4j would time out on queries for explanations with more than roughly 4 atoms. We eliminated any such explanations from the experimental results and calculations.

[4] `http://robokopkg.renci.org/browser/`

[5] We overcame the software limitations for ROBOKOP2.

[6] `https://neo4j.het.io/browser/`

Table 7.1: Summary statistics of the knowledge graphs and rule sets used in the experiments.

(a) Knowledge graphs

Knowledge graph	Entities ($\mathscr{N}$)	Entity types ($\mathscr{T}$)	Predicates ($\mathscr{P}$)	Triples ($\mathscr{L}$)
ROBOKOP	307,959	10	86	4,016,135
ROBOKOP2	1,055,251	20	95	21,782,649
DRKG	97,234	13	100	5,878,876
Hetionet	45,158	11	25	2,250,452

(b) Rule sets

Knowledge graph	Rules	Predicates
ROBOKOP	561	67
ROBOKOP2	840	85
DRKG	974	100
Hetionet	210	25

target predicate *not_treats*. To be able to do that, we had to use *negative triples* such as, e.g., *(cocaine, not_treats, chronic kidney disease)*, which are not stored in KGs. To generate *(i.e., to sample)* negative triples, we relied on the popular closed-world assumption *(CWA)* (Trouillon et al. 2017, 2016; Meilicke et al. 2018; Garcia-Duran et al. 2016; Borrego et al. 2020) stating that triples not present in a KG can be considered false for the purposes of negative-triple sampling. We recognize that CWA is not a perfect assumption, as KGs inherently operate under the open-world assumption. Our reasoning for the experiments was that the vast majority of missing KG triples are false; thus, random negative sampling would result in a negligible amount of false negatives.

To perform negative sampling, we adopted the procedure used in Borrego et al. (2020); Wang et al. (2016); Alspector et al. (1987); Mazumder and Liu (2017). Specifically, for each KG we considered each existing triple (s, treats, o) and randomly sampled either a new subject s' or a new object o' of the same entity-type m times, forming m new triples of the form $(s', \mathit{not\_treats}, o)$ or $(s, \mathit{not\_treats}, o')$, respectively. If any sampled triple was found in the KG as a true *treats* triple, we would repeat the process until this was not the case. Since there is no accepted value for the negative sampling rate, see Meilicke et al. (2018); Kazemi and Poole (2018); Bordes et al. (2013); Garcia-Duran et al. (2016); Yang et al. (2014); Borrego et al. (2020); Trouillon et al. (2016, 2017); Wang et al. (2016); Alspector et al. (1987); Mazumder and Liu (2017), we set $m = 1$ for each existing *treats* triple, in an effort to avoid class imbalance.

7.3.1.2 Rule sets

To ensure that we could perform our experiments at scale, we used rules mined by AMIE (Lajus et al. 2020), the state-of-the-art approach in KG rule mining. We mined rules with AMIE on each of the chosen data sets, generating the rule sets $\mathscr{A}^{ROBOKOP}$, $\mathscr{A}^{ROBOKOP2}$, $\mathscr{A}^{DRKG}$, and $\mathscr{A}^{Hetionet}$ used in our experiments on each KG. Details regarding the size of each rule set are presented in Table 7.1b. The rules in each set were obtained by running AMIE on the respective KG using the maximal rule length of two (to ensure AMIE termination), followed by thresholding the support value at 25 and keeping for each predicate just the top ten rules with the highest standard confidence. The filtering was done to drop weak rules that appear infrequently and are thus unlikely to produce explanations when used in the proposed KGEG approach.

7.3.2 Accuracy of Explanation-Quality Metrics

In this experiment we tested the accuracy and reliability of the scores assigned by our proposed metrics and, thus, their viability for evaluating explanations of hypotheses in KGs. In summary, our proposed metrics provided significantly higher accuracy than the state of the art (Raedt et al. 2007; Gad-Elrab et al. 2019) in most cases in this experiment, see Section 7.3.2.5. This suggests that our metrics could provide an improvement on the state of the art for determining the truth of drug-treats-disease hypotheses.

The experiment consisted of using the proposed KGEG approach to derive explanations for selected hypotheses, and then measuring the accuracy of the proposed explanation-quality metrics against the baselines adopted from the state of the art (Raedt et al. 2007; Gad-Elrab et al. 2019). We now provide the details.

7.3.2.1 Hypothesis selection

To form the sets $\mathscr{H}^{ROBOKOP}$ and $\mathscr{H}^{DRKG}$ of true hypotheses for this experiment, we randomly selected 250 triples with the *treats* predicate *(treats triples)* from ROBOKOP and DRKG, respectively. Then for each true hypothesis in $\mathscr{H}^{ROBOKOP}$ and in $\mathscr{H}^{DRKG}$, we generated five false alternative hypotheses using the procedure described in Section 7.3.1.1 with $m = 5$. As an example, for the true hypothesis (*benazepril, treats, chronic kidney disease*), possible false alternative hypotheses would be (*benazepril, treats, lung cancer*) and (*cocaine, treats, chronic kidney disease*).

Next, for each true hypothesis h in $\mathscr{H}^{ROBOKOP}$ and $\mathscr{H}^{DRKG}$ we formed a *hypothesis group* comprising both h itself and all its false alternative hypotheses obtained in this manner. We

use $G_{\mathscr{H}}^{ROBOKOP}$ and $G_{\mathscr{H}}^{DRKG}$ to denote the set of hypothesis groups extracted from ROBOKOP and DRKG, respectively. Finally, we removed from each KG the corresponding set $\mathscr{H}^{KG}$ of true hypotheses selected for this experiment, to avoid trivial explanations of the form $(e_0, p, e_1) \Leftarrow (e_0, p, e_1)$.

7.3.2.2 Explanation extraction

For each hypothesis h of the form (e_0, p, e_1), we obtained a set of explanations both for h and for the *negation* of h, i.e., for $(e_0$, not-$p, e_1)$. We call explanations for h *positive explanations*, and call explanations for the negation of h *negative explanations*. To extract the explanations, we ran the KGEG algorithm with the following inputs: (1) one of ROBOKOP and DRKG (referred to as KG) with the set $\mathscr{H}^{KG}$ removed; (2) the corresponding rule set $\mathscr{A}^{KG}$ generated by AMIE (see Section 7.3.1.2); (3) the set $G_{\mathscr{H}}^{KG}$ of 250 hypothesis groups, along with the negation of each hypothesis in $G_{\mathscr{H}}^{KG}$; and (4) maximal KGEG search depth of 3, to ensure Neo4j termination. We denote by $Out_{KGEG}(G_{\mathscr{H}}^{KG})$ the set of explanations output by this run of KGEG.

7.3.2.3 The baseline metrics

As baselines, we adapted to KG patterns the state-of-the-art explanation-evaluation metrics of Raedt et al. (2007); Gad-Elrab et al. (2019). For an explanation E of the form $(e_0, p, e_1) \Leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_n$, these metrics are defined as follows:

- The ProbLog metric of Raedt et al. (2007) is defined as

$$ProbLog\text{-}score(E) = \prod_{r \in R} prob(r)\,. \tag{7.7}$$

  Here, R is the set of rules used to produce the explanation E, and $prob(r)$ is the probability associated with rule r, measured as the standard confidence score of r obtained by AMIE (Lajus et al. 2020) during rule mining.

- The ExFaKT metric of Gad-Elrab et al. (2019) is defined as

$$ExFaKT\text{-}score(E) = \frac{1}{|A|} \sum_{a \in A} \frac{1}{a.depth}\,. \tag{7.8}$$

  Here, $A = \{a_1, a_2, \ldots, a_n\}$ is the explanation pattern of E, and $a.depth$ is the same as the value $a.depth$ used in the proposed KGEG approach.

7.3.2.4 Measuring the accuracy of explanation metrics

We now review the process borrowed from Nakashole and Mitchell (2014); Gad-Elrab et al. (2019) that we used in this experiment to rate the explanation-quality metrics with the help of the explanations $Out_{KGEG}(G^{KG}_{\mathscr{H}})$ (see above) obtained via the KGEG approach. Intuitively, a reliable metric for evaluating explanations would assign higher scores to explanations of true hypotheses than to explanations of false hypotheses. We have formalized this intuition with formulae for explanation-set quality, truth score of a hypothesis, and accuracy, as follows.

First, we define the *quality* of a set of explanations $\mathscr{O}$ as the average score of its explanations:

$$quality(\mathscr{E}) = \frac{1}{|\mathscr{E}|} \sum_{E \in \mathscr{E}} score(E)\,. \tag{7.9}$$

Here, $score(E)$ stands for the explanation-evaluation metric in use, e.g., $conf(E)$ or $hc(E)$. The *quality* metric allows us to incorporate several explanations into an evaluation of a single hypothesis. This reduces the impact of potential outlier explanations, thus resulting in a more reliable evaluation.

Second, each output $Out_{KGEG}(G^{KG}_{\mathscr{H}})$ of KGEG contains a set of *positive explanations*, $\mathscr{E}^+_q$, and of *negative explanations*, $\mathscr{E}^-_q$, for each hypothesis q of the form (e_0, p, e_1) in the set $G^{KG}_{\mathscr{H}}$ (see Section 7.3.2.2). The *truth score* $trSc$ assigned to the hypothesis q is then defined as

$$trSc(q) = quality(\mathscr{E}^+_q) - quality(\mathscr{E}^-_q)\,. \tag{7.10}$$

The score $trSc$ indicates whether we should consider the hypothesis (e_0, p, e_1) to be true, by determining which of $\mathscr{E}^+$ and $\mathscr{E}^-$ is stronger (has a higher quality score). A positive truth score indicates that the positive explanations $\mathscr{E}^+$ are stronger; a negative truth score indicates the opposite.

We can now define the *accuracy* metric. To this end, we evaluate each explanation-quality metric on each group $G_i \in G^{KG}_{\mathscr{H}}$, where $G_i = \{h_i, h'_{i1}, h'_{i2}, \ldots, h'_{im}\}$, and on the explanations for G_i in the outputs $Out_{KGEG}(G^{KG}_{\mathscr{H}})$ of KGEG, by computing the *accuracy score* of each G_i as follows.

$$accuracy(G_i) = \frac{1}{m} \sum_{h'_{ij} \in G_i} [trSc(h_i) \geq trSc(h'_{ij})]\,. \tag{7.11}$$

Here, $[x] = 1$ if x is true, otherwise $[x] = 0$. The *accuracy score* for a group G_i indicates the proportion of false hypotheses $h'_{ij} \in G_i$ that scored lower than their true hypothesis h_i. As we expect reliable metrics to score h_i higher than the false alternative hypotheses in G_i, the *accuracy score* is a measure of how reliable the scoring metric is for G_i.

Table 7.2: A comparison of the accuracies of the proposed metrics versus the baseline metrics on appropriately scoring the true hypothesis of each group in $G'^{KG}_{\mathscr{H}}$ above the false hypotheses in the group (see Section 7.3.2.5). The row labels specify the proposed metrics, the column labels specify the baseline metrics, and the values indicate the percentages of groups in $G'^{KG}_{\mathscr{H}}$ for which the accuracy of each proposed metric was higher than (strictly higher than) the accuracy of each baseline. Tables 7.2a and 7.2b show the results with respect to $G'^{ROBOKOP}_{\mathscr{H}}$ and $G'^{DRKG}_{\mathscr{H}}$, respectively.

(a) ROBOKOP

	ProbLog-score	ExFaKT-score
Confidence	89.51% (18.18%)	91.61% (37.06%)
Head Coverage	84.62% (19.58%)	89.51% (32.17%)
F1-score	90.21% (18.88%)	92.31% (37.06%)

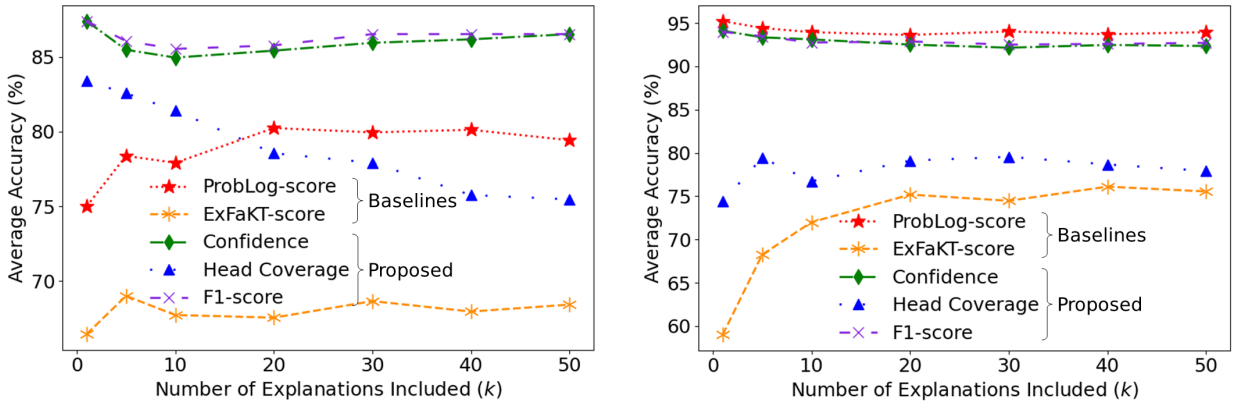
(b) DRKG

	ProbLog-score	ExFaKT-score
Confidence	94.74% (2.87%)	94.74% (43.06%)
Head Coverage	67.94% (2.87%)	80.38% (32.54%)
F1-score	95.69% (2.39%)	95.69% (42.58%)

7.3.2.5 Experimental results

We removed from the experimental data sets $G^{ROBOKOP}_{\mathscr{H}}$ and $G^{DRKG}_{\mathscr{H}}$ all the groups for which the run of KGEG of Section 7.3.2.2 could not produce an explanation for the true hypothesis and at least one false hypothesis, as the accuracy for these groups could not be computed. We also did not include in the computations false hypotheses that had no explanations. After this postprocessing, we denote the updated sets of hypothesis groups for each KG by $G'^{KG}_{\mathscr{H}}$. The sets $G'^{ROBOKOP}_{\mathscr{H}}$ and $G'^{DRKG}_{\mathscr{H}}$ have a total of 143 groups and 209 groups, respectively.

We first report our results considering only the top five positive and top five negative explanations for each hypothesis in $G'^{KG}_{\mathscr{H}}$; this setup is similar to that of Gad-Elrab et al. (2019). To compare the performance of our three proposed metrics of Equations 7.4–7.6 to the two baseline metrics of Equations 7.7–7.8, we computed the percentage of the groups in $G'^{KG}_{\mathscr{H}}$ for which our metrics achieved accuracy scores higher ($\geq$) and *strictly* higher ($>$) than the baselines, see Table 7.2. We can see that on both ROBOKOP (Table 7.2a) and DRKG (Table 7.2b), our proposed metrics achieved accuracy that is at least as high as that for the baselines a vast majority of the time, indicating that the scores assigned by our metrics to the true hypotheses are at least as accurate as those assigned by the baselines. Moreover, there is a substantial number of groups for which our proposed metrics strictly outperformed the baselines.



(a) ROBOKOP

(b) DRKG

Figure 7.1: The average accuracy of each metric over all the hypothesis groups in the set $G'^{KG}_{\mathscr{H}}$ (see Section 7.3.2.5) when scoring hypotheses according to the top k positive and the top k negative explanations derived by the proposed approach KGEG. The X-axis indicates k, the number of explanations included when scoring each hypothesis, and the Y-axis indicates the average accuracy. Figures 7.1a and 7.1b show the results with respect to $G'^{ROBOKOP}_{\mathscr{H}}$ and $G'^{DRKG}_{\mathscr{H}}$, respectively.

Next, we considered the accuracy of each metric while varying the cutoff value k used when computing scores to determine how this might impact performance. That is, we considered more cases than that of just the top five positive and top five negative explanations. Figure 7.1 shows the average accuracy of each metric for $k \in \{1,5,10,20,30,40,50\}$. We first notice that our proposed metrics confidence and F1-score consistently achieved very high accuracy on both ROBOKOP (Figure 7.1a) and DRKG (Figure 7.1b) for all values of k. Moreover, they significantly outperformed both baseline metrics on ROBOKOP and outperformed the ExFaKT-score on DRKG. While head coverage did not perform as well as our other proposed metrics, it still outperformed the ExFaKT-score in all cases and outperformed the ProbLog-score in some cases as well. We also notice that our confidence and F1-score metrics have comparable average accuracy for all values of k. These results suggest that our proposed metrics, especially confidence and F1-score, could provide more reliable, accurate scoring no matter how many explanations are allowed for each hypothesis.

7.3.3 Knowledge Graph Pattern Usefulness

Next, we considered the potential of our explanation-derivation approach to compose new inference rules that could be helpful in explaining previously unseen hypotheses. We found that a relatively small set of KG patterns previously derived as explanations was helpful in

explaining most of the selected hypotheses (see Section 7.3.3.4). This result suggests that KG patterns in our proposed format, once generated by our approach, can be used to explain multiple drug-treats-disease hypotheses over time. (Unlike our explanations, the explanations produced by other works, including Raedt et al. (2007); Gad-Elrab et al. (2019), cannot be generalized to new hypotheses. Thus we cannot use baselines in this experiment.)

7.3.3.1 Hypothesis selection

To form the sets $\mathscr{T}^{ROBOKOP}$ and $\mathscr{T}^{DRKG}$ of true hypotheses for this experiment, we randomly selected 100 true *treats* triples from ROBOKOP and from DRKG, respectively (disjoint from the data of Section 7.3.2). Then, for each KG, we formed the *experimental data set* $\mathscr{S}^{KG}$ from the set $\mathscr{T}^{KG}$ by also adding for each $t \in \mathscr{T}^{KG}$ one false alternative *not_treats* hypothesis, to keep the true and false classes balanced. (For the false-alternative hypothesis generation process, see Section 7.3.1.1.) Again, we removed from each KG the corresponding set $\mathscr{T}^{KG}$ of true hypotheses selected for this experiment, to avoid trivial explanations.

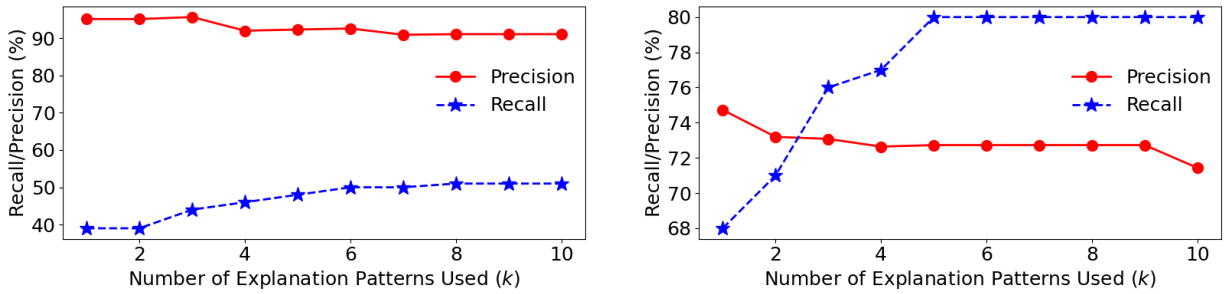
7.3.3.2 Explanation extraction

To study the potential reuse of explanations, we used as new inference rules the explanations generated by KGEG in the experiment discussed in Section 7.3.2. For each KG, we formed a set of these new inference rules $\mathscr{R}^{KG}$; $\mathscr{R}^{ROBOKOP}$ includes 614 unique explanations derived for the predicate *treats* and 579 unique explanations for the predicate *not_treats*, and $\mathscr{R}^{DRKG}$ includes 593 unique *treats* explanations and 307 unique *not_treats* explanations. For each rule $R \in \mathscr{R}^{KG}$ we calculated the *prevalence* of R as the proportion of true hypotheses $h \in \mathscr{H}^{KG}$ from the experiment of Section 7.3.2 whose set of explanations $\mathscr{E}_h$ output by KGEG included R:

$$prevalence(R) = \frac{|\{h \in \mathscr{H}^{KG} : R \in \mathscr{E}_h\}|}{|\mathscr{H}^{KG}|} . \tag{7.12}$$

Next, for each k in $\{1, 2, \ldots, 10\}$ we formed a new rule set $\mathscr{R}_k^{KG}$ from the top k most prevalent rules in the set $\mathscr{R}^{KG}$ for the predicate *treats* and, separately, for the predicate *not_treats*.

Then, for each k, we extracted explanations for all the hypotheses in $\mathscr{S}^{KG}$ and their negations, by running the KGEG algorithm with the following inputs: (1) one of the KGs (ROBOKOP or DRKG) with the set $\mathscr{T}^{KG}$ removed; (2) the corresponding rule set R_k^{KG}; (3) the set $\mathscr{S}^{KG}$ of true and false hypotheses and their negations; and (4) maximal KGEG search depth of one, to ensure that each explanation is derived from a rewriting of a triple in $\mathscr{S}^{KG}$ by a single rule in $\mathscr{R}_k^{KG}$.



(a) ROBOKOP (b) DRKG

Figure 7.2: The precision and recall achieved when explaining 100 true *treats* hypotheses and their false alternatives in the set $\mathscr{S}^{KG}$ (see Section 7.3.3.4) using as new inference rules the top-k most prevalent explanations derived in the experiment of Section 7.3.2. The X-axis indicates k; the Y-axis indicates precision and recall percentages. Figures 7.2a and 7.2b show the results with respect to $\mathscr{S}^{ROBOKOP}$ and $\mathscr{S}^{DRKG}$, respectively.

7.3.3.3 Measuring the usefulness of explanation patterns

To evaluate the explanation performance, we used precision and recall, defining *precision* as the proportion of hypotheses in $\mathscr{S}^{KG}$ with at least one explanation that are true *treats* triples from $\mathscr{T}^{KG}$, and *recall* as the proportion of true triples in $\mathscr{T}^{KG}$ with at least one explanation. In the following equations for precision and recall, we use $\mathscr{E}_t$ to denote the set of explanations extracted for the hypothesis t.

$$precision(S^{KG}) = \frac{|\{t \in \mathscr{S}^{KG} : \mathscr{E}_t \neq \emptyset \text{ and } t \in \mathscr{T}^{KG}\}|}{|\{t \in \mathscr{S}^{KG} : \mathscr{E}_t \neq \emptyset\}|}. \tag{7.13}$$

$$recall(S^{KG}) = \frac{|\{t \in \mathscr{T}^{KG} : \mathscr{E}_t \neq \emptyset\}|}{|\mathscr{T}^{KG}|}. \tag{7.14}$$

7.3.3.4 Experimental results

Figure 7.2 shows the precision and recall scores for all k values for both KGs. In the results on ROBOKOP (Figure 7.2a), we notice first that precision is above 90% for all the values of k. This indicates that the selected rules were very reliable at explaining only the true hypotheses and not false hypotheses. Next, we notice that it took only six rules to achieve a recall of 50%. This means that our rules were able to explain half of the true hypotheses, despite not having been derived *for* these hypotheses.

In the results on DRKG (Figure 7.2b), precision is lower than for ROBOKOP, but recall is substantially higher. Since precision and recall tend to be opposing forces, this behavior is expected. Still, precision scores above 70% indicate that the selected rules performed well, i.e.,

they explained mostly true hypotheses and not false ones. Moreover, the recall reached 80% with only five rules, so in this case our previously derived rules were still able to explain most of these new true hypotheses.

Overall, this evidence suggests that the KG patterns that we used as rules occur frequently in proximity with *treats* triples, and only a small set of them may be needed to explain most true hypotheses, even when using previously derived explanations for new hypotheses.

7.3.4 Explanation Extraction for Classification

In this experiment we tested the potential of the proposed KGEG explanation-extraction approach to serve as a classifier for hypotheses, classifying them as either true or false. This can be done by differentiating between hypotheses based on whether or not any explanations were found for them: If explanations for a hypothesis have been found, then the hypothesis is classified as true; otherwise, it is classified as false. We used the proposed KGEG approach as a classifier in this manner and compared its performance to a traditional baseline approach from rule mining. In summary, the proposed KGEG approach was more effective as a classifier than the baseline approach used, according to standard classification performance metrics (see Section 7.3.4.4). This suggests that the KGEG approach could be used to reliably identify strong drug-repurposing hypotheses.

7.3.4.1 The baseline approach

As an explanation-extraction baseline, we used a traditional approach involving Horn rules, which has been adopted by knowledge-graph rule-mining approaches (Ahmadi et al. 2020; Lajus et al. 2020; Kolthoff and Dutta 2015; Meilicke et al. 2018; Ortona et al. 2018; Sadeghian et al. 2019; Wang and Li 2015). The baseline approach applies the input Horn rules one at a time to the input hypothesis, by using the hypothesis as the rule head and checking to see if the rule body can be instantiated in the KG. If it can, then the body of the rule serves as the explanation for the hypothesis. While the KGEG approach finds explanations in a similar manner, it expands beyond this baseline by allowing Horn rules to be synthesized together before attempting to use them as an explanation, as opposed to only using each rule by itself.

7.3.4.2 Explanation extraction

For this experiment we used both ROBOKOP and DRKG, along with the same experimental data as in the experiment discussed in Section 7.3.2. For each KG, these data consist of a set $\mathscr{H}^{KG}$ of 250 true hypotheses, along with a set of 250 hypothesis groups $G^{KG}_{\mathscr{H}}$ formed from the

set $\mathscr{H}^{KG}$ by generating five false alternative hypotheses for each true hypothesis. (See Section 7.3.2.1 for the details.)

For the proposed KGEG approach, we analyzed the set $Out_{KGEG}(G^{KG}_{\mathscr{H}})$ of explanations extracted, which contains for each hypothesis $h \in G^{KG}_{\mathscr{H}}$ a set of explanations extracted for h. To emulate the baseline approach, we ran the KGEG algorithm with the maximal search depth of one. Using the maximal search depth of one rules out any rule synthesized during the explanation extraction; therefore, each input rule is used by itself, which aligns with the baseline approach. We denote by $Out_{Baseline}(G^{KG}_{\mathscr{H}})$ the set of explanations extracted by the baseline approach for the hypotheses in $G^{KG}_{\mathscr{H}}$. In this experiment we focus only on the positive *treats* explanations for each hypothesis.

7.3.4.3 Evaluating the classification performance

We treated an explanation-extraction approach A as a classifier, by classifying as true any hypothesis h that had at least one positive explanation in the set $Out_A(h)$ of the explanations output by A for h.

$$class(h,A) = \begin{cases} true & Out_A(h) \neq \emptyset \\ false & otherwise \end{cases} \tag{7.15}$$

We evaluated each approach with four standard classification-performance metrics: precision, recall, F1-score, and accuracy.[7] Below, we detail the meaning of each metric in the context of explanation extraction.

The *precision* of an approach A is the proportion of hypotheses in $G^{KG}_{\mathscr{H}}$ classified as *true* by A that are actually true hypotheses from $\mathscr{H}^{KG}$, and the *recall* of A is the proportion of true hypotheses from $\mathscr{H}^{KG}$ that were classified as *true* by A. The *F1-score* of an approach A is the harmonic mean of *precision* and *recall*, and the *accuracy* of A is the proportion of all the hypotheses that are classified correctly, i.e., true hypotheses classified as *true* or false hypotheses classified as *false*.

An ideal classifier would achieve a high score according to each of our metrics. However, there is a well understood trade-off between precision and recall. That is, increasing the precision of an approach tends to decrease its recall, and vice versa. Maximizing recall minimizes the number of false negatives, i.e., the number of true hypotheses that are classified as *false*, while maximizing precision minimizes the number of false positives, i.e., the number of false hypotheses that are classified as *true*. In the case of drug repurposing, biomedical experts value recall above precision, because they would rather have false positives than false negatives.

[7] Since $G^{KG}_{\mathscr{H}}$ contains five false alternative hypotheses for each true hypothesis, we upsampled the true hypotheses five-fold to avoid class imbalance in our calculations.

Table 7.3: A comparison of the performance of the proposed KGEG explanation-extraction approach versus the baseline approach as classifiers for true and false hypotheses (see Section 7.3.4.4). Hypotheses for which at least one explanation was extracted were classified as true hypotheses; otherwise, they were classified as false. The row labels specify the performance metric, and the column labels specify the approach. Tables 7.3a and 7.3b show the results of each approach on ROBOKOP and DRKG, respectively.

(a) ROBOKOP

	Baseline	KGEG
Recall	2.40%	65.60%
Precision	93.75%	79.15%
F1-score	4.68%	71.74%
Accuracy	51.12%	74.15%

(b) DRKG

	Baseline	KGEG
Recall	56.79%	90.00%
Precision	99.16%	69.57%
F1-score	72.23%	78.48%
Accuracy	78.15%	75.31%

Thus, while maximizing both precision and recall is ideal, recall is preferred to precision in drug repurposing.

7.3.4.4 Experimental results

Table 7.3 shows the precision, recall, F1, and accuracy scores achieved by the baseline approach and the proposed KGEG approach on each KG used. For both ROBOKOP (Table 7.3a) and DRKG (Table 7.3b), the proposed KGEG approach achieved significantly higher recall than the baseline approach, but achieved lower precision. This trade-off between precision and recall is expected, and, as discussed above, we prefer recall to precision here. Moreover, the recall improvement is much greater than the precision decline, so the overall performance of the proposed approach is greater. This is captured by the F1-score, which evaluates the classifier performance by balancing the trade-off between precision and recall. Finally, our approach is substantially more accurate than the baseline on ROBOKOP (Table 7.3a), and only slightly less accurate on DRKG (Table 7.3b). Overall, this evidence suggests that the proposed KGEG explanation-extraction approach performs better as a hypothesis classifier than the baseline approach traditionally adopted by rule-mining methods.

7.3.5 Explanation-Extraction Efficiency

In this experiment, we tested the efficiency of our proposed KGEG explanation-extraction approach by examining the number of explanations considered by the approach, compared to the total number of possible explanations. As discussed in Section 7.2.1, considering one atom per iteration of the main loop in Algorithm 8, instead of one explanation, allows us to prune

useless explanations early. In this experiment, we explored the significance of this pruning. In summary, we found that the proposed approach is able to substantially cut down on the number of explanations considered without sacrificing completion, i.e., the approach still produces all possible explanations within the depth limit (see Section 7.3.5.2).

7.3.5.1 Measuring the efficiency of explanation extraction

In this experiment, we used the same run of KGEG on the set $G^{KG}_{\mathscr{H}}$ of 250 hypothesis groups as described in Section 7.3.2. For each hypothesis h, we computed the number of distinct explanations e_h considered during the run of Algorithm 8. We also computed the total number of atoms a_h considered, i.e., the number of iterations of the main loop in Algorithm 8.

As a baseline, we used the total number of explanations and atoms, e_{total} and a_{total}, respectively, that can be formed by the rule set $\mathscr{A}^{KG}$. If the main loop of Algorithm 8 considered one explanation per iteration, as opposed to one atom, then it would have to iterate over all e_{total} possible explanations. Since the main loop instead considers one atom per iteration and prunes useless explanations early, our approach considers fewer explanations and atoms. We compared the average e_h and a_h values to e_{total} and a_{total}, respectively, to understand the magnitude and impact of this pruning.

7.3.5.2 Experimental results

Table 7.4 shows the total possible explanations and atoms, along with the average considered explanations and atoms, for both ROBOKOP and DRKG. The average number of extracted explanations is also included for each KG; this allows us to see how many of the considered explanations were valid explanations output by Algorithm 8. First, for both ROBOKOP and DRKG the average number of considered explanations and atoms is only a small portion of the total number of possible explanations and atoms (under 28% in all cases, and under 13% in most cases). Since the average number of extracted explanations is substantially lower than even the average number of considered explanations, it is notable that we are able to prune so many explanations. Next, Algorithm 8 considered a smaller portion of the possible explanations and atoms for ROBOKOP than for DRKG. We hypothesize that this is due to the possible number of explanations and atoms for ROBOKOP being much larger than DRKG. Therefore, there is significantly more room for pruning in ROBOKOP. Finally, recall that our Algorithm 8 is still complete, i.e., able to produce all valid explanations for a hypothesis without having to consider all possible explanations (see Section 7.2.1). Overall, these results suggest that our approach is substantially more efficient than the brute-force approach of considering all possible explanations due to its pruning mechanism.

Table 7.4: The average number (proportion) of explanations and atoms considered during a run of Algorithm 8, compared to the number of total explanations (exps.) and atoms possible, for ROBOKOP and DRKG, for the predicates *treats* and *not_treats*. This comparison shows the effect of explanation pruning (see Section 7.3.5.2).

Knowledge graph	Target predicate	Total possible		Average considered		Average extracted
		Exps.	Atoms	Exps.	Atoms	
ROBOKOP	*treats*	579,169	2,279,282	34,417 (5.94%)	41,863 (1.84%)	10
	not_treats	52,371	208,367	6,692 (12.78%)	10,597 (5.09%)	13
DRKG	*treats*	22,095	87,502	3,922 (17.75%)	6,087 (6.96%)	56
	not_treats	10,800	42,663	2,970 (27.5%)	4,400 (10.31%)	13

7.3.6 Explaining Other Predicates

In this experiment we sought to determine if the proposed approach could be used for tasks beyond treatment prediction. To this end, we used the approach to explain other predicates beyond just the *treats* predicate that we have used thus far. In summary, we found that the proposed approach can be used to reliably explain hypotheses of other predicates (see Section 7.3.6.4).

7.3.6.1 Hypothesis selection

For this experiment, we used ROBOKOP2 and selected eight different predicates of interest to biomedical experts: *affects*, *ameliorates*, *contraindicatedFor*, *contributesTo*, *geneAssociatedWithCondition*, *hasPhenotype*, *similarTo*, and *treats*. For each predicate p, we formed the experimental data set $\mathcal{O}_p$ consisting of 100 true hypotheses and 100 false alternative hypotheses (one false alternative for each true hypothesis), according to the procedure detailed in Section 7.3.1.1.

7.3.6.2 Explanation extraction

We extracted explanations for each experimental data set $\mathcal{O}_p$ by running the KGEG algorithm with the following inputs: (1) the ROBOKOP2 KG with the set $\mathcal{O}_p$ removed; (2) the rule set $\mathscr{A}^{ROBOKOP2}$; (3) the set $\mathcal{O}_p$ of hypotheses; and (4) the maximal search depth of 3. We denote by $Out_{KGEG}(\mathcal{O}_p)$ the set of explanations output by this run of KGEG on $\mathcal{O}_p$.

Table 7.5: The performance of the proposed KGEG explanation-extraction approach at classifying true and false hypotheses of different predicates from the ROBOKOP2 knowledge graph (see Section 7.3.6.4). Hypotheses for which at least one explanation was extracted were classified as true hypotheses; otherwise, they were classified as false.

Predicate	Recall	Precision	F1-score	Accuracy
affects	100.00%	96.15%	98.04%	98.00%
ameliorates	100.00%	93.46%	96.62%	96.50%
contraindicatedFor	100.00%	98.04%	99.01%	99.00%
contributesTo	100.00%	98.04%	99.01%	99.00%
geneAssociatedWithCondition	100.00%	93.46%	96.62%	96.50%
hasPhenotype	100.00%	86.96%	93.02%	92.50%
similarTo	100.00%	95.24%	97.56%	97.50%
treats	100.00%	89.29%	94.34%	94.00%

7.3.6.3 Measuring the performance on other predicates

For each predicate p, we used the corresponding experimental data set $\mathcal{O}_p$ and extracted explanations $Out_{KGEG}(\mathcal{O}_p)$ to compute the performance of our approach at classifying each hypothesis $h \in \mathcal{O}_p$ as true or false. We used the evaluation described in Section 7.3.4.3. In particular, we computed four standard classification-performance metrics: precision, recall, F1-score, and accuracy (see Section 7.3.4.3).

7.3.6.4 Experimental results

Table 7.5 shows the precision, recall, F1, and accuracy scores achieved by the proposed KGEG approach on each predicate used. For each predicate, we achieved 100% recall, indicating that our approach is particularly reliable at explaining true hypotheses. Moreover, the precision is above 85% for all predicates, indicating that there were very few false hypotheses for which explanations were found. Although all the recall and precision scores are high, remember that we prefer recall to precision when classifying hypotheses, as discussed in Section 7.3.4.3. Finally, all the F1-scores and accuracy scores are very high as well: above 92%. Overall, this evidence suggests that the proposed KGEG explanation-extraction approach could perform well on a variety of predicates, and is therefore applicable to tasks beyond treatment prediction.

7.3.7 Avoiding Contraindications

In this experiment we explored the balance between the *treats* predicate and its near antonym: *contraindicatedFor*. A drug predicted as a treatment for a disease of interest could end up having

adverse effects due to a variety of factors, e.g., dosage level, interactions with a comorbidity, or other patient-specific issues. Unfortunately, this detailed information is not present in KGs, so we cannot mine for it directly. In an attempt to overcome this deficiency, we sought to balance the support for a *treats* relationship and the support for a *contraindicatedFor* relationship. Our results suggest that our explanation-extraction approach performs well in the face of contraindications; it can be tuned to avoid contraindications and may also be robust against them, see Section 7.3.7.4.

7.3.7.1 Hypothesis selection

For this experiment we used ROBOKOP2, which has both *treats* and *contraindicatedFor* predicates. To generate hypotheses for this experiment, we randomly selected from ROBOKOP2 100 *treats* triples and 100 *contraindicatedFor* triples, and generated one false alternative hypothesis for each, according to the process described in Section 7.3.1.1.[8] Then, for each of the 200 false alternative hypotheses $h = (e_0, p, e_1)$, we added two hypotheses to the set $\mathscr{C}$ of hypotheses for this experiment: (1) $h^+ = (e_0, \textit{treats}, e_1)$ and (2) $h^- = (e_0, \textit{contraindicatedFor}, e_1)$.

7.3.7.2 Explanation extraction

For each pair of hypotheses $h^+, h^- \in \mathscr{C}$, we generated a set of explanations by running Algorithm 1 with the following inputs: (1) the ROBOKOP2 KG, (2) the rule set $\mathscr{A}^{ROBOKOP2}$, (3) the hypotheses h^+ and h^-, and (4) the maximal search depth of 3.

7.3.7.3 Balancing the treatment and contraindication potential

To identify strong drug-repurposing hypotheses, we aimed to identify drug-disease pairs with high *treats* likelihood but low *contraindicatedFor* likelihood. To this end, we used four different *contraindication-filtering criteria* to balance the support of *treats* and *contraindicatedFor* explanations for each pair of hypotheses $h^+, h^- \in \mathscr{C}$: (i) $|\mathscr{E}_{h^+}| > |\mathscr{E}_{h^-}|$, (ii) $\text{avg}_{E \in \mathscr{E}_{h^+}} score(E) > \text{avg}_{E \in \mathscr{E}_{h^-}} score(E)$, (iii) $\max_{E \in \mathscr{E}_{h^+}} score(E) > \max_{E \in \mathscr{E}_{h^-}} score(E)$, and (iv) $|\mathscr{E}_{h^+}| > 0$. Here, $\mathscr{E}_h$ denotes the set of explanations extracted for hypothesis h, and $score(E)$ stands for the explanation-evaluation metric in use. For brevity, we discuss $conf$ (Eq. 7.4) in this report; the results using hc (Eq. 7.5) and $F1\text{-}score$ (Eq. 7.6) are comparable.

After extracting explanations for the hypotheses in $\mathscr{C}$, we filtered the hypothesis pairs by each of the four criteria, generating the sets $\mathscr{C}_{(i)}, \mathscr{C}_{(ii)}, \mathscr{C}_{(iii)}, \mathscr{C}_{(iv)} \subseteq \mathscr{C}$. We aimed to identify

[8] Since ROBOKOP2 is much larger than the other KGs, randomly selecting entities during negative sampling pulls many hypotheses into sparse areas of the KG. To resolve this issue, we randomly selected entities from those in the true triples.

which criterion ought to be used to ensure identification of strong drug-repurposing hypotheses, while avoiding possible contraindications. To this end, we employed the open-source tool Abstract Sifter (Baker et al. 2017) to query PubMed,[9] a search engine encompassing a vast amount of biomedical literature. Abstract Sifter is designed to enable complex, efficient querying of PubMed by "sifting" abstracts for key terms and then presenting relevant articles to the user. For each pair of hypotheses $h^+, h^- \in \mathscr{C}$, we formulated the query "s AND o," where s and o are the subject and object, respectively, of h^+ and h^-. The result of such a query, denoted $Out_{Sifter}(h^+)$, is the list of PubMed articles mentioning both s and o. As recommended by the domain experts, we deemed the hypotheses with at least one co-mention to be *supported*, taking the PubMed articles as evidence that s treats o. To quantify this analysis, we define the *success* of a criterion X to be the proportion of hypotheses in $\mathscr{C}_X$ for which Abstract Sifter identified at least one supporting PubMed article.

$$success(X) = \frac{|\{h^+, h^- \in \mathscr{C}_X : Out_{Sifter}(h^+) \neq \emptyset\}|}{|\mathscr{C}_X|}. \tag{7.16}$$

7.3.7.4 Experimental results

To normalize the results, we removed from the experimental data set $\mathscr{C}$ any hypothesis (s, p, o) for which Abstract Sifter could not identify any PubMed articles that mentioned s alone and o alone. These were cases in which the entity names were not standard and therefore could not be recognized by PubMed. Table 7.6 shows the success without filtering the hypotheses, which is equivalent to random hypothesis selection, and the success of contraindication-filtering criteria (i)–(iv). All four criteria outperformed the random case, indicating that explanation extraction can improve upon random hypothesis selection. Since all four criteria performed well, experts can tune the extraction process to avoid contraindications by selecting the contraindication-filtering criteria that aligns with their problem and their preferences, without sacrificing much success. Criterion (iv) performed best without even considering *contraindicatedFor* explanations. This result suggests that filtering simply by the existence of *treats* explanations was sufficient for identifying strong drug-repurposing hypotheses. In other words, our explanation-extraction approach may be robust against contraindications.

7.3.8 Discussion of the Experimental Results

We now summarize the key takeaways of our experimental results. First, the results of Section 7.3.2 show that our proposed explanation-evaluation metrics are capable of achieving higher accuracy than the state-of-the-art metrics from Raedt et al. (2007); Gad-Elrab et al. (2019). Thus,

[9] `https://pubmed.ncbi.nlm.nih.gov/`

Table 7.6: The success of each contraindication-filtering criterion at identifying possible drug-repurposing hypotheses (see Section 7.3.7.4). Success is defined as the proportion of tested hypotheses in the set $\mathscr{C}_X$, where X is the criterion, that were supported by at least one PubMed article. The corresponding raw counts of tested hypotheses and supported hypotheses are also shown.

Hypothesis set	Hypothesis pairs	Supported	Success
$\mathscr{C}$	196	95	48.47%
$\mathscr{C}_{(i)}$	39	25	64.10%
$\mathscr{C}_{(ii)}$	33	22	66.67%
$\mathscr{C}_{(iii)}$	41	28	68.29%
$\mathscr{C}_{(iv)}$	75	53	70.67%

we posit that our metrics, due to their reliability, could serve as potential universal metrics for explanations to fill the gap identified in Kotonya and Toni (2020a). We also saw in Section 7.3.3 that the proposed KGEG approach produces explanation patterns that can serve as useful inference rules for downstream explanation tasks. That is, explanations in our novel format need not be derived anew for each hypothesis. Instead, time can be saved by applying previous, reliable explanations to new hypotheses. The results of additional experiments reported in this chapter indicate that the KGEG approach can also serve as a reliable hypothesis classifier, indicating which hypotheses are true and which are false, see Section 7.3.4. Moreover, the pruning mechanism of Algorithm 8 enables efficient explanation extraction, see Section 7.3.5. Results on a newer, much larger version of ROBOKOP suggest that our approach is extensible to other predicates and thus other tasks, see Section 7.3.6, as well as tunable and robust against contraindications, see Section 7.3.7.

7.4 Biomedical-Expert User Studies

We now present the results of two user studies conducted with biomedical experts. The user study of Section 7.4.1 tests the format of our explanations; the results suggest that the knowledge-graph (KG) pattern explanations defined in Section 3.5 and generated by the KGEG approach can be understandable and reasonable to experts working in the field of drug repurposing. The user study of Section 7.4.2 tests the value of the entity-types in our explanations; the results suggest that the entity-types in our explanations can be important for biomedical-expert understanding, as well as for the biomedical meaningfulness and reasonableness of the explanations.

7.4.1 Explanation Reasonableness

The aim of this user study was to determine whether our explanations, in their proposed novel format, are helpful in drug repurposing. In summary, analysis by biomedical experts indicated that our explanations provide reasonable justifications in a helpful form by presenting entity interactions relevant to drug-disease treatment relationships (see Section 7.4.1.2). This suggests that our explanation-derivation approach is capable of generating meaningful KG patterns that are helpful to experts working in the field of drug repurposing.

7.4.1.1 Measuring the reasonableness of explanations

To explore the quality of our explanations, we had biomedical experts on our team analyze several KG-pattern explanations output by the KGEG approach introduced in this chapter. We sorted all the explanations extracted from ROBOKOP in the experiment discussed in Section 7.3.2 according to each of our proposed metrics of Equations 7.4–7.6, and presented the five top-scoring explanations for each metric to the experts. (Note that since the expert analysis was manual and time consuming, we focus solely on the ROBOKOP KG in this section.) We instructed the experts to consider how reasonable each explanation pattern is as a justification that the hypothesis $(e_0, \textit{treats}, e_1)$ is true, i.e., how well the explanation explains the target triple $(e_0, \textit{treats}, e_1)$. They were asked to rate each explanation according to the scale (1) "reasonable," (2) "reasonable in some cases" (i.e., depending on other unknown factors), (3) "neutral," and (4) "unreasonable." We also asked the experts to provide short descriptions of their intuition and reasoning as a qualitative analysis. We deemed any explanation rated as a (1) or (2) to be *appropriate*, and define the *precision* of each metric as the proportion of appropriate explanations out of those presented to the experts.

7.4.1.2 Results of the user study

Table 7.7 shows the precision of each of our proposed metrics, which indicates the proportion of explanations deemed appropriate by the biomedical experts on our team. We found that for each metric, no more than one of the five top-scoring explanations was deemed inappropriate, and no explanations were given a rating of (4) "unreasonable." The results suggest that each of our proposed metrics is capable of assigning high scores to explanations that are biomedically reasonable.

We have two key takeaways from the qualitative feedback provided by the biomedical experts on our team. First, in some cases an explanation may enable experts working in the field of drug repurposing to not only identify a possible treatment relationship, but also understand the drug's mechanism of action; this is a direct benefit of our explanations being modeled

Table 7.7: A summary of the ratings assigned by the biomedical experts on our team to the explanations they analyzed, along with the precision of each of the proposed metrics, i.e., the proportion of the five top-scoring explanations that were deemed appropriate by the biomedical experts (see Section 7.4.1.2).

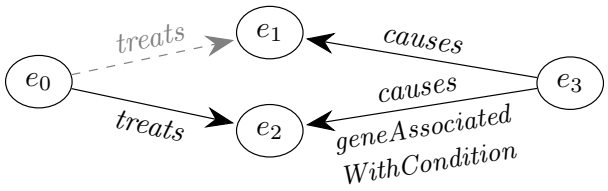
Metric	# 1s	# 2s	# 3s	# 4s	Precision
Confidence	-	4	1	-	80%
Head Coverage	2	3	-	-	100%
F1-score	3	1	1	-	80%

after COPs, see Section 1.4.4, and thereby justifies our design. Second, some explanations were rated lower because their entity-types were unknown and were not able to be inferred from the predicate-types. This methodological insight led us to add entity-typing functionality to the proposed approach; the impact of the addition is explored in Section 7.4.2.

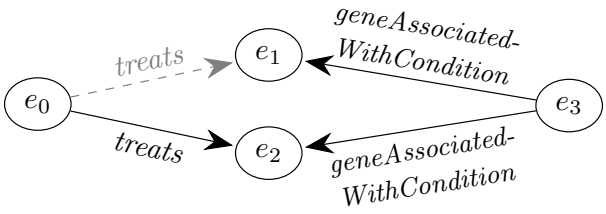
We next present a few explanations along with their ratings according to the biomedical experts on our team, and explore the qualitative feedback received. Figure 7.3 shows three example explanations that received different ratings. The reasoning provided by the biomedical experts for their rating of each of these explanations is discussed below. The target triple for each explanation is $(e_0, \textit{treats}, e_1)$.

First, Figure 7.3a shows an explanation that was rated (1) "reasonable" by the experts. Because diseases e_1 and e_2 share the common causative gene e_3, it is reasonable to the experts to deduce that the drug e_0, which treats disease e_2, would also treat disease e_1. This follows from the general understanding that similar diseases have similar treatments, especially diseases that have similar mechanistic causes, meaning that similar biological processes and interactions cause the disease. As this explanation pattern shows a strong relationship between drug e_0 and disease e_1 based on biological interactions, the experts have deemed it to be reasonable for the hypothesis $(e_0, \textit{treats}, e_1)$. Moreover, since gene e_3 causes diseases e_1 and e_2, the experts assume that drug e_0 likely acts on both diseases by inhibiting the function of e_3. In some cases, such as this one, an explanation allows us to identify not only a possible treatment, but also the drug's mechanism of action; this is a direct benefit of our explanations being modeled after COPs, see Section 1.4.4.

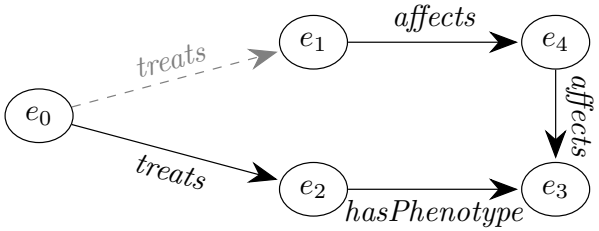
Next, Figure 7.3b shows an explanation that was rated (2) "reasonable in some cases." This explanation is very similar in structure to that of Figure 7.3a, with the only difference being the name of the relationship that gene e_3 has with diseases e_1 and e_2, i.e., *geneAssociatedWithCondition* instead of *causes*. Thus, this explanation also follows the intuition that similar diseases have similar treatments. However, it received a lower rating since the *geneAssociatedWithCondition* relationship is not as specific as the *causes* relationship, i.e., there is no indication of the



(a) An explanation that received expert rating (1) "reasonable."



(b) An explanation that received expert rating (2) "reasonable in some cases."



(c) An explanation that received expert rating (3) "neutral."

Figure 7.3: Explanations that received various ratings according to the biomedical experts on our team; Figures 7.3a, 7.3b, and 7.3c show explanations rated (1), (2), and (3), respectively. Each directed edge with label p connecting nodes x and y represents the triple (x, p, y). For each explanation, the solid triples comprise the explanation pattern, and the dashed edge represents the target predicate, which connects the endpoints e_0 and e_1. Based on the analysis of the biomedical experts, the explanations in Figure 7.3a and Figure 7.3b are reasonable, since they indicate that diseases e_1 and e_2 are similar, and thus they are likely to have a common treatment, i.e., the drug e_0. However, the explanation in Figure 7.3a is stronger, since the *causes* relationship between gene e_3 and diseases e_1 and e_2 is stronger and more specific than the corresponding *geneAssociatedWithCondition* relationship in Figure 7.3b. The explanation in Figure 7.3c is weaker than the other two due to the ambiguity of the *affects* relationship, i.e., there is no qualifier indicating the type of affect. Thus, there is not enough information to infer a strong relationship between drug e_0 and disease e_1.

type or level of association. A common gene association is still useful for determining disease similarity, but a common gene causation is stronger evidence that the diseases will behave similarly in the presence of the drug e_0.

Finally, Figure 7.3c shows an explanation that was rated (3) "neutral." This explanation is weaker than those previously mentioned, largely due to the fact that its pattern reveals a much weaker connection between the drug e_0 and the disease e_1. In general, patterns with more specific relationships are able to reveal more about the actual biological behavior of the entities involved, giving a clearer indication of how the entities will interact. This is not the case for relationships such as *affects*, which are useful but ambiguous, since they give no clarification on their nature or type. Moreover, one cannot deduce the entity-type of e_4, since *affects* can have many different subject and object entity-types.[10] This uncertainty creates a mechanistic knowledge gap in the explanation, making it less reasonable. Thus, while this pattern does identify a connection between drug e_0 and disease e_1, it is not strong enough or specific enough to fully explain the hypothesis (e_0, *treats*, e_1).

7.4.2 Entity-Typed Explanations

In this user study we considered the case of entity-typed explanations, see Section 3.5 for the details, and analyzed their value over regular explanations with respect to domain-expert understanding, domain meaningfulness, and domain reasonableness. Based on Linder et al. (2021) and on expert feedback presented in Section 7.4.1, we expected that the added detail would improve the quality of our explanations with respect to these three criteria. In summary, feedback from biomedical experts indicated that explanations with entity-types are preferred to explanations without entity-types (see Section 7.4.2.2). Thus, it is important that our approach supports entity-typing.

7.4.2.1 Measuring the importance of entity-types

To measure the importance of having entity-types in our explanations, we had biomedical experts on our team analyze several explanations with and without entity-types. In this user study, we considered the same 15 explanations considered in the user study of Section 7.4.1. However, we also imposed entity-types on all 15 explanations, see Section 7.2.1.1. Then, we presented these 15 explanations, along with their entity-typed versions, to the domain experts and asked them to consider entity-types with respect to the following three properties: (I) *understandability*: is the explanation more understandable with entity-types?, (II) *meaningfulness*:

[10] Due to this limitation, the proposed approach extracts entity-typed explanations; the impact is explored in Section 7.4.2.

is the explanation more (biomedically) meaningful with entity-types?, and (III) *reasonableness*: for each entity-typed version of the explanation, how reasonable is the explanation pattern as justification that the hypothesis $(e_0, treats, e_1)$ is true? For (III), we asked the experts to rate each entity-typed explanation according to the scale detailed in Section 7.4.1.1: (1) "reasonable," (2) "reasonable in some cases," (3) "neutral," and (4) "unreasonable." We also asked the experts to provide some reasoning for their responses, which we discussed with them afterwards.

Based on the expert feedback, we found that there were some explanations for which it was possible to infer the entity-types from the predicate names; e.g., for any triple of the form $(s, geneAssociatedWithCondition, o)$, one can infer that s is a gene and o is a condition (disease). On the other hand, there were many cases in which the entity-types could not be inferred, e.g., for any triple of the form $(s, causes, o)$, s and o can take on several different entity-types. In cases where the entity-types could be inferred, the experts confirmed that inferring the entity-types was important if they were not already provided. Thus, based on the expert responses to (I) and (II), we deemed entity-types to be *important for understanding* and *meaningfulness* if the experts either responded "yes," or if they responded "no" in those cases where the entity-types could be inferred. Based on the responses to (III), we deemed the entity-types to be *important for reasonableness* if there were at least two different entity-typed versions of the explanation with different ratings, i.e., the entity-types changed the reasonableness of the explanation. (We did not consider any explanations for which there was only one entity-typing, as these could not be evaluated.)

7.4.2.2 User-study results

Table 7.8 shows the proportion of explanations for which entity-types were deemed *important* for the three properties of interest: *understandability*, *meaningfulness*, and *reasonableness*. These results suggest that entity-types were almost always important for all three of these properties. Thus, we conclude that adding support for entity-types to the KGEG approach was valuable for the usefulness and applicability of the proposed approach.

7.4.3 Discussion of User-Study Results

We now summarize the key takeaways of our user study results. The first user study, see Section 7.4.1, provides evidence that KG-pattern explanations can be understandable and reasonable to experts working in the field of drug repurposing. Moreover, the user study provides evidence that our proposed KGEG metrics are capable of identifying strong, biomedically reasonable explanations. The results of the second user study, see Section 7.4.2, suggest that including entity-types in our explanations enables extraction of understandable, meaningful,

Table 7.8: The proportion of explanations for which the biomedical experts on our team indicated that entity-types were *important*, i.e., the presence of and changes in entity-types changed the expert's interpretation, for each of the properties studied: *understandability*, *meaningfulness*, and *reasonableness* (see Section 7.4.2.2).

Property	Importance
Understandability	100.00%
Meaningfulness	93.33%
Reasonableness	80.00%

and reasonable explanations.

7.5 Limitations

In this section we discuss some observed limitations of our work. Sections 7.5.1 and 7.5.2 discuss the impacts of knowledge-graph sparsity and knowledge-graph quality, respectively, on our approach.

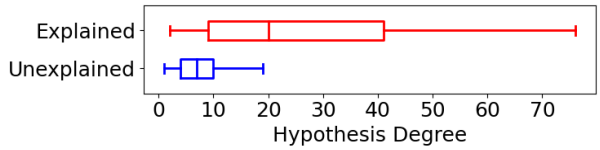
7.5.1 Knowledge-graph sparsity

In this section, we explore why the recall scores reported in Sections 7.3.3.4 and 7.3.4.4 were not higher, i.e., why the proposed approach was unable to explain some true hypotheses. To this end, we observed that some areas of knowledge graphs (KGs) are very dense, while others are sparse, and we hypothesized that many unexplained true hypotheses reside in sparse areas of the KGs. After all, tasks such as explanation extraction are much more difficult in such areas due to the lack of available data.

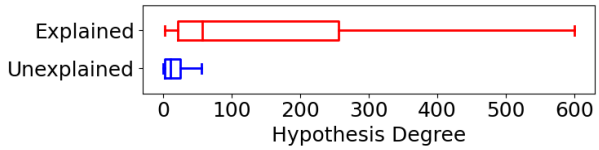
To estimate the impacts of KG density, we used node degree as a proxy for local KG density, and computed the *degree* of each true hypothesis in our experimental data sets $\mathscr{T}^{KG}$ and $\mathscr{H}^{KG}$ (see Sections 7.3.3 and 7.3.4). We define the *degree* of a hypothesis (s, p, o) to be the minimum degree of its subject s and object o:

$$degree(s, p, o) = \min\{degree(s), degree(o)\}. \tag{7.17}$$

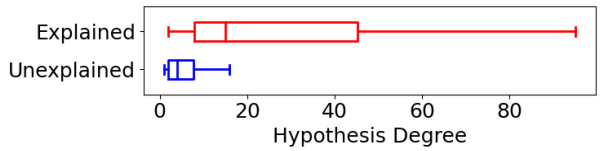
We classified each true hypothesis as (1) *explained:* at least one explanation was found, or (2) *unexplained:* no explanations found. We analyzed the degrees of the hypotheses in the two groups.



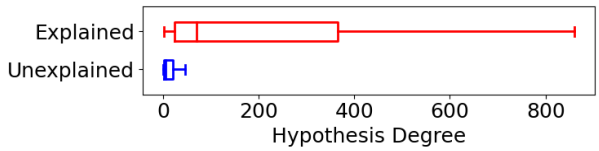
(a) Degree distributions for $\mathscr{T}^{ROBOKOP}$ (Section 7.3.3).



(b) Degree distributions for $\mathscr{T}^{DRKG}$ (Section 7.3.3).



(c) Degree distributions for $\mathscr{H}^{ROBOKOP}$ (Section 7.3.4).



(d) Degree distributions for $\mathscr{H}^{DRKG}$ (Section 7.3.4).

Figure 7.4: A comparison of the distributions of degrees of explained versus unexplained hypotheses (see Section 7.5.1). *Explained hypotheses* are those for which at least one explanation was extracted by the proposed KGEG approach; *unexplained hypotheses* are those for which no explanations were found. The X-axis indicates the hypothesis degree, and the Y-axis indicates the class of hypotheses (explained or unexplained). Each pair of boxplots shown is for a single knowledge graph and a single set of true hypotheses from one of our experimental settings, which is specified in the corresponding captions.

Figure 7.4 shows the distributions of the hypothesis degrees for the *explained* and *unexplained* hypotheses in each experimental data set of true hypotheses.[11] In each case, both the spread and center of the two sets are very different and, in particular, the degrees of the explained hypotheses tend to be much higher than those of the unexplained hypotheses. This indicates that many of the unexplained hypotheses came from sparse sections of the KG, as expected; as a result, it was significantly more difficult to find explanations for these hypotheses. To further confirm this analysis, we conducted Welch's t-tests (Welch 1947) between the two groups of the experimental data sets. In each case, the resulting p-values were significantly less than the standard cutoff of 0.05. This confirmed that the hypothesis degrees are statistically significantly different between the two groups in all cases. Overall, this evidence suggests that graph sparsity is a key reason why many true hypotheses were unexplained. Thus, graph sparsity, i.e., the lack of available data, is likely a limiting factor during explanation derivation.

7.5.2 Knowledge-graph quality

In this section we discuss our experience with the KG Hetionet (Himmelstein et al. 2017). We attempted to run our experiments on Hetionet, but found that the graph may not be well

[11] We have omitted outliers from Figure 7.4 for the sake of clarity.

Table 7.9: The proportions of true and false hypotheses that were supported by Abstract Sifter (Baker et al. 2017), and were therefore assumed to be verified true facts (see Section 7.5.2).

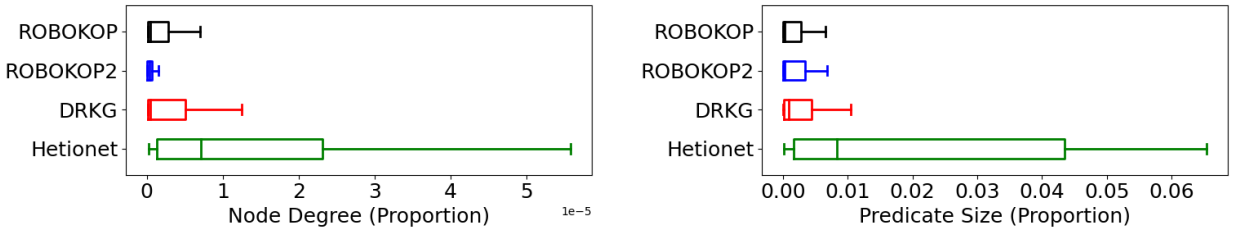
Knowledge graph	True hypotheses		False hypotheses	
	Hypotheses	Supported (%)	Hypotheses	Supported (%)
ROBOKOP	235	205 (87.23%)	607	112 (18.45%)
ROBOKOP2	236	207 (87.71%)	674	151 (22.40%)
DRKG	243	232 (95.47%)	1076	416 (38.66%)
Hetionet	250	247 (98.80%)	1249	751 (60.13%)

suited for the proposed approach. First, it appears that Hetionet is likely missing many true facts. KGs are expected to be incomplete, but some graphs are missing more information than others. Incomplete graphs negatively impact our ability to sample false hypotheses, which therefore affects the quality of the explanation extraction and evaluation results. To estimate the incompleteness of the *treats*[12] facts in each KG, we ran a sample of true and false hypotheses through Abstract Sifter (Baker et al. 2017) (see Section 7.3.7).[13] We assumed that any false hypotheses supported by Abstract Sifter are actually true facts missing from the graph. Table 7.9 shows the percentages of true and false hypotheses supported by Abstract Sifter. All these KGs have a high percentage of true hypotheses supported, as expected. However, the percentage of supported false hypotheses, which should be relatively low, is quite high for Hetionet. In particular, it is substantially higher than for the other KGs. This evidence suggests that Hetionet is more incomplete than the other KGs that we used in the experiments, thus potentially creating challenges for explanation extraction. While DRKG's percentage of supported false hypotheses is much lower than that of Hetionet, it is much higher than those of both ROBOKOP and ROBOKOP2. We hypothesize that this is due to the curation efforts for each graph. In particular, ROBOKOP and ROBOKOP2 are part of a federally funded project, with ongoing curation efforts. Thus, it is likely a high-quality KG, thus well suited for explanation extraction.

Next, we found that our experiments ran significantly slower on Hetionet than on other KGs. We hypothesize that, although Hetionet is much smaller than the other KGs used in the experiments, the distribution of edges and predicates was responsible for the increased query time. To this end, we analyzed two key properties that can impact query efficiency: (1) *node degree*, and (2) *predicate size*. *Node degree* represents the graph's local density; dense sections of the KG are more costly to query, because there are many paths that must be explored. *Predicate size* is the number of triples for a given predicate, indicating the prevalence of that predicate;

[12] In Hetionet, the predicate *CtD* is equivalent to *treats*.

[13] For this test, we reused the hypothesis groups $G_{\mathscr{H}}^{KG}$ for each KG, see Section 7.3.2.1.



(a) Node-degree distributions

(b) Predicate-size distributions

Figure 7.5: The distributions of node degrees and predicate sizes for the knowledge graphs ROBOKOP, ROBOKOP2, DRKG, and Hetionet (see Section 7.5.2). The node degrees and predicate sizes have been normalized via division by the total number of triples in the corresponding knowledge graph.

prevalent predicates are more costly to query, because there are many instances that must be explored. To normalize the results, we divided each value by the total number of triples in the corresponding KG, thereby giving proportions, not raw values. Figure 7.5 shows the node-degree (Figure 7.5a) and predicate-size (Figure 7.5b) distributions for ROBOKOP, ROBOKOP2, DRKG, and Hetionet. Hetionet has many nodes with degree much higher than those in the other KGs, and also has many predicates that are more prevalent. As mentioned, these features of Hetionet could certainly cause increased query times. While both KG completeness and triple/predicate distribution are out of our control, it is important to recognize that the proposed approach displays limitations with respect to these features.

7.5.3 Discussion of Limitations

Based on this analysis, we conclude that KGs must have sufficient hypothesis degrees for explanation extraction, see Section 7.5.1. Moreover, well-curated KGs with minimal missing information are likely to be more reliable for explanation extraction and evaluation, see Section 7.5.2.

7.6 Summary

Explainable fact-checking is a promising direction for knowledge discovery, especially over knowledge graphs. It enables rapid inference of new information, while also enabling domain experts to verify the findings afterwards, increasing the reliability of the inferred knowledge. Drug repurposing can be viewed as an impactful application of explainable fact-checking. Using this perspective, in this work we presented a novel view of explanations as knowledge-graph

patterns, and introduced a new scalable explanation-extraction strategy, as well as metrics for explanation evaluation that incorporate the existing data as evidence. Our proposed approach is distinct from previous fact-checking work because it was built with the biomedical domain in mind: We specifically addressed the challenges that arise when adopting existing tools for drug repurposing. Our user-study results indicate that our explanations and metrics are understandable and useful for biomedical experts, especially due to the addition of entity-types to our explanations and to our modeling them after COPs. We also obtained experimental outcomes suggesting that our proposed metrics can be reliable and accurate, and that our derived explanations can serve as useful inference rules for downstream explanation tasks. Moreover, we showed that our explanation-extraction approach is efficient, extensible to other tasks, reliable as a classifier, and tunable for avoiding contraindications. These results indicate that the proposed methods for explanation extraction and evaluation could be used as viable approaches for facilitating innovation in drug repurposing, and potentially in other knowledge-discovery tasks. We have provided biomedical experts with the predictions of our approach for their further consideration, and anticipate future use of our tools by domain experts in drug repurposing or other related tasks. In our ongoing work, we have been looking into expanding these approaches to other tasks in the biomedical domain and potentially beyond. Although our work was inspired by our collaboration with biomedical experts, the approach is entirely domain independent. Thus, we anticipate testing our work on other impactful knowledge-discovery applications. Other directions of our current work include formally analyzing the complexity of the proposed algorithm, as well as formulating metrics that refine the proposed ones for specific explanation instances, by potentially incorporating edge weights, node properties, and/or node promiscuity (Hou et al. 2022; Newman 2018; Kleinberg 1999; Mencher and Wang 2005).

CHAPTER

8

CONCLUSIONS

8.1 Summary of the Dissertation

This dissertation pertains to the use of domain knowledge graphs in all stages, from construction to completion, with aims to improve knowledge graph usability and domain scientists' experiences. To this end, *the four C's pipeline* of knowledge graphs is introduced, covering four key phases of domain scientists' use of knowledge graphs: construction, curation, compression, and completion. These phases are nontrivial, so a domain- and task-aware approach is proposed for each phase. While their acceptance of user inputs makes them domain- and task-aware, the approaches are also domain-independent, causing them to be broadly applicable in a variety of application domains. The pipeline and its component approaches suggest that storing heterogeneous data in domain knowledge graphs is feasible and can be advantageous for various domain applications. The work in this dissertation highlights the need for tools to support domain scientists in their use of knowledge graphs because of the impact that knowledge graphs can have on the richness of analytical results. When well-equipped, domain scientists can leverage knowledge graphs for knowledge discovery in their respective fields, leading to invaluable insights into their data. Additionally, this dissertation facilitates future discussion and development of domain knowledge graph tools for continued enhancement of the four C's pipeline.

8.2 Future Work

Future work in this area of research could follow multiple directions. First, domain data exist in many formats, and while the BUILD-KG workflow of Chapter 4 handles some of the most popular formats, it could be beneficial to expand this to more data formats that arise. This would increase the coverage of the workflow, thereby increasing its usability for other domains and tasks. Second, the need for knowledge graph curation can be ongoing as knowledge graphs evolve to include more data, or as they are used for new use cases. Therefore, the development of curation frameworks that would complement the KG-CURATE workflow of Chapter 5 could be used to address curation needs as they arise. Third, testing the GAME+ knowledge-graph abstraction approach of Chapter 6 on other domains, similarity functions, and analytical tasks could reveal additional use cases for the approach. Expanding the approach to allow for different similarity thresholds to be used for different entity-types is another direction of interest. Additionally, the KGEG knowledge-graph completion approach of Chapter 7 has been tested in the biomedical domain. As a domain-independent approach, KGEG could potentially be used for many impactful applications that would be an interesting direction of future research. Moreover, designing valuable metrics for evaluating knowledge graph paths and subgraphs is another problem of interest, e.g., incorporating into the metrics node-specific properties and edge weights. Finally, anticipating a continued rise in the use of knowledge graphs, it is foreseeable that additional knowledge graphs tasks could emerge. Thus, an important direction of future research is the development of additional domain- and task-sensitive frameworks that would be developed to address these emerging challenges faced by domain experts.

COPYRIGHT NOTICES

REFERENCES

Agrawal, R., Imielinski, T., Swami, A., Road, H., and Jose, S. (1993). Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216.

Ahmadi, N., Huynh, V.-P., Meduri, V., Ortona, S., and Papotti, P. (2020). Mining Expressive Rules in Knowledge Graphs. *Journal of Data and Information Quality*, 12(2):1–27.

Ahmadi, N., Lee, J., Papotti, P., and Saeed, M. (2019). Explainable Fact Checking with Probabilistic Answer Set Programming. *arXiv:1906.09198 [cs]*.

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019). FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59. Association for Computational Linguistics.

Alspector, J., Allen, R., Hu, V., and Satyanarayana, S. (1987). Stochastic learning networks and their electronic implementation. In *Neural Information Processing Systems*, volume 0.

Apache TinkerPop (2023). Gremlin - Apache TinkerPop.

Arzberger, P., Schroeder, P., Beaulieu, A., Bowker, G., Casey, K., Laaksonen, L., Moorman, D., Uhlir, P., and Wouters, P. (2004). Promoting Access to Public Research Data for Scientific, Economic, and Social Development. *Data Science Journal*, 3:135–152.

Asprino, L., Daga, E., Gangemi, A., and Mulholland, P. (2023). Knowledge Graph Construction with a *Façade*: A Unified Method to Access Heterogeneous Data Sources on the Web. *ACM Transactions on Internet Technology*, 23(1):1–31.

Atanasova, P., Simonsen, J. G., Lioma, C., and Augenstein, I. (2020). Generating Fact Checking Explanations. *arXiv:2004.05773 [cs]*.

Baker, N., Knudsen, T., and Williams, A. (2017). Abstract Sifter: a comprehensive front-end system to PubMed. *F1000Research*, 6.

Bizon, C., Cox, S., Balhoff, J., Kebede, Y., Wang, P., Morton, K., Fecho, K., and Tropsha, A. (2019). ROBOKOP KG and KGB: Integrated knowledge graphs from federated sources. *Journal of Chemical Information and Modeling*, 59(12):4968–4973.

Bonifati, A., Dumbrava, S., and Kondylakis, H. (2020). Graph Summarization. *arXiv:2004.14794*.

Bordes, A., Usunier, N., and Garcia-Duran, A. (2013). Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Borrego, A., Ayala, D., Hernández, I., Rivero, C. R., and Ruiz, D. (2020). CAFE: Fact checking in knowledge graphs using neighborhood-aware features. *The Semantic Web*.

Bosselut, A., Rashkin, H., Sap, M., Malaviya, C., Celikyilmaz, A., and Choi, Y. (2019). COMET: Commonsense Transformers for Automatic Knowledge Graph Construction. *arXiv preprint arXiv:1906.05317*.

Čebirić, Š., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G., and Zneika, M. (2019). Summarizing semantic graphs: a survey. *The VLDB Journal*, 28(3):295–327.

Chen, P., Lu, Y., Zheng, V. W., Chen, X., and Li, X. (2018). An automatic knowledge graph construction system for K-12 education. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, pages 1–4. ACM.

Chen, Z., Wang, Y., Zhao, B., Cheng, J., Zhao, X., and Duan, Z. (2020). Knowledge Graph Completion: A Review. *IEEE Access*, 8:192435–192456.

Ciampaglia, G. L., Shiralkar, P., Rocha, L. M., Bollen, J., Menczer, F., and Flammini, A. (2015). Computational Fact Checking from Knowledge Networks. *PLOS ONE*, 10(6):e0128193.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics.

Elhammadi, S., V.S. Lakshmanan, L., Ng, R., Simpson, M., Huai, B., Wang, Z., and Wang, L. (2020). A High Precision Pipeline for Financial Knowledge Graph Construction. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 967–977. International Committee on Computational Linguistics.

Fetro, C. and Scherman, D. (2020). Drug repurposing in rare diseases: Myths and reality. *Therapies*, 75(2):157–160.

Gad-Elrab, M. H., Stepanova, D., Urbani, J., and Weikum, G. (2019). ExFaKT: A Framework for Explaining Facts over Knowledge Graphs and Text. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*, pages 87–95. ACM.

Garcia-Duran, A., Bordes, A., Usunier, N., and Grandvalet, Y. (2016). Combining Two and Three-Way Embedding Models for Link Prediction in Knowledge Bases. *Journal of Artificial Intelligence Research*, 55:715–742.

Google LLC (2022). Google Cloud documentation.

Gupta, T., Zaki, M., Krishnan, N. M. A., and Mausam (2022). MatSciBERT: A materials domain language model for text mining and information extraction. *npj Computational Materials*, 8(1):102.

Hao, X., Ji, Z., Li, X., Yin, L., Liu, L., Sun, M., Liu, Q., and Yang, R. (2021). Construction and Application of a Knowledge Graph. *Remote Sensing*, 13(13):2511.

Harris, S., Seaborne, A., and Prud'hommeaux, E. (2013). SPARQL 1.1 Query Language. W3C Recommendation.

Heller, S., McNaught, A., Stein, S., Tchekhovskoi, D., and Pletnev, I. (2013). InChI - the worldwide chemical structure identifier standard. *Journal of Cheminformatics*, 5:1–9.

Himmelstein, D. S., Lizee, A., Hessler, C., Brueggeman, L., Chen, S. L., Hadley, D., Green, A., Khankhanian, P., and Baranzini, S. E. (2017). Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *eLife*, 6:e26726.

Hou, P.-Y., Korn, D. R., Melo-Filho, C. C., Wright, D. R., Tropsha, A., and Chirkova, R. (2022). Compact walks: Taming knowledge-graph embeddings with domain- and task-specific pathways. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, page 458–469.

Huaman, E. and Fensel, D. (2021). Knowledge Graph Curation: A Practical Framework. In *The 10th International Joint Conference on Knowledge Graphs*, pages 166–171. ACM.

Huang, C., Fang, Y., Lin, X., Cao, X., and Zhang, W. (2022). ABLE: Meta-Path Prediction in Heterogeneous Information Networks. *ACM Transactions on Knowledge Discovery from Data*, 16(4):1–21.

Huang, H., Hong, Z., Zhou, H., Wu, J., and Jin, N. (2020). Knowledge Graph Construction and Application of Power Grid Equipment. *Mathematical Problems in Engineering*, 2020:1–10.

Huang, X. and Lai, W. (2006). Clustering graphs for visualization via node similarities. *Journal of Visual Languages & Computing*, 17(3):225–253.

Hughes, J. P., Rees, S., Kalindjian, S. B., and Philpott, K. L. (2011). Principles of early drug discovery. *British Journal of Pharmacology*, 162(6):1239–1249.

Ioannidis, V. N., Song, X., Manchanda, S., Li, M., Pan, X., Zheng, D., Ning, X., Zeng, X., and Karypis, G. (2020). DRKG - Drug Repurposing Knowledge Graph for Covid-19. `https://github.com/gnn4dr/DRKG/`.

Jaccard, P. (1912). The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50.

Jin, D. and Koutra, D. (2017). Exploratory Analysis of Graph Data by Leveraging Domain Knowledge. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 187–196.

Jin, D., Rossi, R. A., Koh, E., Kim, S., Rao, A., and Koutra, D. (2019). Latent Network Summarization: Bridging Network Embedding and Summarization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 987–997. ACM.

Kazemi, S. M. and Poole, D. (2018). SimplE Embedding for Link Prediction in Knowledge Graphs. *arXiv:1802.04868 [cs, stat]*.

Kejriwal, M. (2019). *Domain-Specific Knowledge Graph Construction*. SpringerBriefs in Computer Science. Springer International Publishing.

Kleinberg, J. M. (1999). Hubs, authorities, and communities. *ACM Computing Surveys (CSUR)*, 31(4es):5–es.

Kolthoff, K. and Dutta, A. (2015). Semantic Relation Composition in Large Scale Knowledge Bases. In *CEUR Workshop Proceedings*, volume 1467, pages 34–47.

Korn, D., Thieme, A. J., Alves, V. M., Yeakey, M., Borba, J. V., Capuzzi, S. J., Fecho, K., Bizon, C., Edwards, S. W., Chirkova, R., Colvis, C. M., Southall, N. T., Austin, C. P., Muratov, E. N., and Tropsha, A. (2022). Defining clinical outcome pathways. *Drug Discovery Today*, 27(6):1671–1678.

Kotonya, N. and Toni, F. (2020a). Explainable Automated Fact-Checking: A Survey. *arXiv:2011.03870 [cs]*.

Kotonya, N. and Toni, F. (2020b). Explainable Automated Fact-Checking for Public Health Claims. *arXiv:2010.09926 [cs]*.

Koutra, D., Kang, U., Vreeken, J., and Faloutsos, C. (2014). VOG: Summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 91–99.

Lajus, J., Galárraga, L., and Suchanek, F. (2020). Fast and Exact Rule Mining with AMIE 3. In *The Semantic Web*, pages 36–52. Springer International Publishing.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., and Kang, J. (2020). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Leskovec, J. and Faloutsos, C. (2006). Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 631–636. ACM Press.

Li, F.-L., Chen, H., Xu, G., Qiu, T., Ji, F., Zhang, J., and Chen, H. (2020a). AliMeKG: Domain Knowledge Graph Construction and Application in E-commerce. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2581–2588. ACM.

Li, L., Wang, P., Yan, J., Wang, Y., Li, S., Jiang, J., Sun, Z., Tang, B., Chang, T.-H., Wang, S., and Liu, Y. (2020b). Real-world data medical knowledge graph: construction and applications. *Artificial Intelligence in Medicine*, 103:101817.

Lin, P., Song, Q., and Wu, Y. (2018). Fact Checking in Knowledge Graphs with Ontological Subgraph Patterns. *Data Science and Engineering*, 3(4):341–358.

Linder, R., Mohseni, S., Yang, F., Pentyala, S. K., Ragan, E. D., and Hu, X. B. (2021). How level of explanation detail affects human performance in interpretable intelligent systems: A study on explainable fact checking. *Applied AI Letters*, 2(4):e49.

Liu, Y., Safavi, T., Dighe, A., and Koutra, D. (2018). Graph summarization methods and applications: A survey. *ACM Computing Surveys*, 51(3).

Lu, Y.-J. and Li, C.-T. (2020). GCAN: Graph-aware Co-Attention Networks for Explainable Fake News Detection on Social Media. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 505–514. Association for Computational Linguistics.

Luan, Y., He, L., Ostendorf, M., and Hajishirzi, H. (2018). Multi-Task Identification of Entities, Relations, and Coreference for Scientific Knowledge Graph Construction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3219–3232. Association for Computational Linguistics.

Marrero, M., Urbano, J., Sánchez-Cuadrado, S., Morato, J., and Gómez-Berbís, J. M. (2013). Named Entity Recognition: Fallacies, challenges and opportunities. *Computer Standards & Interfaces*, 35(5):482–489.

Martinez-Rodriguez, J. L., Lopez-Arevalo, I., and Rios-Alvarado, A. B. (2018). OpenIE-based approach for Knowledge Graph construction from text. *Expert Systems with Applications*, 113:339–355.

Mazumder, S. and Liu, B. (2017). Context-aware Path Ranking for Knowledge Base Completion. *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1195–1201.

Meilicke, C., Fink, M., Wang, Y., Ruffinelli, D., Gemulla, R., and Stuckenschmidt, H. (2018). Fine-Grained Evaluation of Rule- and Embedding-Based Systems for Knowledge Graph Completion. In *The Semantic Web – ISWC 2018*, volume 11136, pages 3–20. Springer International Publishing.

Mencher, S. K. and Wang, L. G. (2005). Promiscuous drugs compared to selective drugs (promiscuity can be a virtue). *BMC Clinical Pharmacology*, 5(1):1–7.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Nakashole, N. and Mitchell, T. M. (2014). Language-Aware Truth Assessment of Fact Candidates. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1019. Association for Computational Linguistics.

Nakov, P., Corney, D., Hasanain, M., Alam, F., Elsayed, T., Barrón-Cedeño, A., Papotti, P., Shaar, S., and Da San Martino, G. (2021). Automated Fact-Checking for Assisting Human Fact-Checkers. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 4551–4558. International Joint Conferences on Artificial Intelligence Organization.

NebulaGraph (2023). NebulaGraph Query Language (nGQL).

Newman, M. (2018). *Networks*. Oxford University Press.

Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., and Taylor, J. (2019). Industry-scale knowledge graphs: lessons and challenges. *Communications of the ACM*, 62(8):36–43.

Ortona, S., Meduri, V. V., and Papotti, P. (2018). Robust Discovery of Positive and Negative Rules in Knowledge Bases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1168–1179.

Paulheim, H. (2017). Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508.

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics.

Popat, K., Mukherjee, S., Strötgen, J., and Weikum, G. (2017). Where the Truth Lies: Explaining the Credibility of Emerging Claims on the Web and Social Media. In *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion*, pages 1003–1012. ACM Press.

Popat, K., Mukherjee, S., Yates, A., and Weikum, G. (2018). DeClarE: Debunking Fake News and False Claims using Evidence-Aware Deep Learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 22–32. Association for Computational Linguistics.

Pushpakom, S., Iorio, F., Eyers, P. A., Escott, K. J., Hopper, S., Wells, A., Doig, A., Guilliams, T., Latimer, J., McNamee, C., Norris, A., Sanseau, P., Cavalla, D., and Pirmohamed, M. (2019). Drug repurposing: progress, challenges and recommendations. *Nature Reviews Drug Discovery*, 18(1):41–58.

Pyysalo, S., Kanerva, J., Virtanen, A., and Ginter, F. (2021). WikiBERT models: Deep transfer learning for many languages. In *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, pages 1–10. Linköping University Electronic Press, Sweden.

Raedt, L. D., Kimmig, A., and Toivonen, H. (2007). ProbLog: A Probabilistic Prolog and its Application in Link Discovery. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 7, pages 2462–2467.

Reitz, K. (2022). Requests: HTTP for humans.

Richardson, L. (2020). Beautiful Soup documentation.

Roessler, H. I., Knoers, N. V., Van Haelst, M. M., and Van Haaften, G. (2021). Drug Repurposing for Rare Diseases. *Trends in Pharmacological Sciences*, 42(4):255–267.

Russell, S. and Norvig, P. (2021). *Artificial Intelligence: A Modern Approach, Global Edition 4th*. Pearson Education, Inc.

Sacenti, J. A. P., Fileto, R., and Willrich, R. (2022). Knowledge graph summarization impacts on movie recommendations. *Journal of Intelligent Information Systems*, 58(1):43–66.

Sadeghian, A., Armandpour, M., Ding, P., and Wang, D. Z. (2019). DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. *arXiv:1911.00055 [cs, stat]*.

Safavi, T., Belth, C., Faber, L., Mottin, D., Müller, E., and Koutra, D. (2019). Personalized Knowledge Graph Summarization: From the Cloud to Your Pocket. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 528–537.

Samarinas, C., Hsu, W., and Lee, M. L. (2021). Improving Evidence Retrieval for Automated Explainable Fact-Checking. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations*, pages 84–91. Association for Computational Linguistics.

Schatz, K., Korn, D., Tropsha, A., and Chirkova, R. (2022). Workflow for Domain- and Task-Sensitive Curation of Knowledge Graphs, with Use Case of DRKG. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 3692–3701.

Schatz, K., Melo-Filho, C., Tropsha, A., and Chirkova, R. (2021). Explaining Drug-Discovery Hypotheses Using Knowledge-Graph Patterns. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 3709–3716.

Shen, T., Zhang, F., and Cheng, J. (2022). A comprehensive overview of knowledge graph completion. *Knowledge-Based Systems*, 255:109597.

Sheth, A., Padhee, S., and Gyrard, A. (2019). Knowledge Graphs and Knowledge Networks: The Story in Brief. *IEEE Internet Computing*, 23(4):67–75.

Shi, B. and Weninger, T. (2016). Discriminative predicate path mining for fact checking in knowledge graphs. *Knowledge-Based Systems*, 104:123–133.

Shiralkar, P., Flammini, A., Menczer, F., and Ciampaglia, G. L. (2017). Finding Streams in Knowledge Graphs to Support Fact Checking. *arXiv:1708.07239*.

Shu, K., Cui, L., Wang, S., Lee, D., and Liu, H. (2019). dEFEND: Explainable Fake News Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 395–405. ACM.

Small, N. (2019). The py2neo v4 handbook.

Song, Q., Wu, Y., and Dong, X. L. (2016). Mining Summaries for Knowledge Graph Search. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1215–1220.

Song, Q., Wu, Y., Lin, P., Dong, L. X., and Sun, H. (2018). Mining Summaries for Knowledge Graph Search. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1887–1900.

Sung, M., Jeong, M., Choi, Y., Kim, D., Lee, J., and Kang, J. (2022). BERN2: an advanced neural biomedical named entity recognition and normalization tool. *Bioinformatics*, 38(20):4837–4839.

The Neo4j Team (2022a). APOC user guide 4.1.

The Neo4j Team (2022b). The Neo4j Cypher manual v4.4.

The Neo4j Team (2022c). The Neo4j operations manual v4.4.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Tran, T.-K., Gad-Elrab, M. H., Stepanova, D., Kharlamov, E., and Strötgen, J. (2020). Fast Computation of Explanations for Inconsistency in Large-Scale Knowledge Graphs. In *Proceedings of The Web Conference 2020*, pages 2613–2619. ACM.

Trouillon, T., Dance, C. R., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2017). Knowledge Graph Completion via Complex Tensor Factorization. *arXiv:1702.06879*.

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *Proceedings of the International Conference on Machine Learning*, volume 48, pages 2071–2080.

Unni, D. R., Moxon, S. A., Bada, M., Brush, M., Bruskiewich, R., Caufield, J. H., Clemons, P. A., Dancik, V., Dumontier, M., Fecho, K., et al. (2022). Biolink model: A universal schema for knowledge graphs in clinical, biomedical, and translational science. *Clinical and Translational Science*.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Vedula, N. and Parthasarathy, S. (2021). FACE-KEG: Fact Checking Explained using KnowledgE Graphs. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 526–534. ACM.

Walker, N., Trewartha, A., Huo, H., Lee, S., Cruse, K., Dagdelen, J., Dunn, A., Persson, K., Ceder, G., and Jain, A. (2021). The impact of domain-specific pre-training on named entity recognition tasks in materials science. *Available at SSRN 3950755*.

Wang, C., Ma, X., Chen, J., and Chen, J. (2018). Information extraction and knowledge graph construction from geoscience literature. *Computers & Geosciences*, 112:112–120.

Wang, Q., Liu, J., Luo, Y., Wang, B., and Lin, C.-Y. (2016). Knowledge Base Completion via Coupled Path Ranking. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1308–1318. Association for Computational Linguistics.

Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

Wang, Z. and Li, J. (2015). RDF2Rules: Learning Rules from RDF Knowledge Bases by Mining Frequent Predicate Cycles. *arXiv:1512.07734 [cs]*.

Weikum, G. (2021). Knowledge graphs 2021: a data odyssey. *Proceedings of the VLDB Endowment*, 14(12):3233–3238.

Weininger, D. (1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36.

Welch, B. L. (1947). The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1-2):28–35.

Wu, L., Rao, Y., Zhao, Y., Liang, H., and Nazir, A. (2020). DTCA: Decision Tree-based Co-Attention Networks for Explainable Claim Verification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1024–1035. Association for Computational Linguistics.

Wu, X., Wu, J., Fu, X., Li, J., Zhou, P., and Jiang, X. (2019). Automatic Knowledge Graph Construction: A Report on the 2019 ICDM/ICBK Contest. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1540–1545.

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding Entities and Relations for Learning and Inference in Knowledge Bases. *arXiv:1412.6575 [cs]*.

Yang, F., Pentyala, S. K., Mohseni, S., Du, M., Yuan, H., Linder, R., Ragan, E. D., Ji, S., and Hu, X. B. (2019). XFake: Explainable Fake News Detector with Visualizations. In *The World Wide Web Conference - WWW '19*, pages 3600–3604. ACM Press.

Ye, H., Zhang, N., Chen, H., and Chen, H. (2022). Generative Knowledge Graph Construction: A Review. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–17. Association for Computational Linguistics.

Yu, Y., Huang, K., Zhang, C., Glass, L. M., Sun, J., and Xiao, C. (2021). SumGNN: multi-typed drug interaction prediction via efficient knowledge graph summarization. *Bioinformatics*, 37(18):2988–2995.

Zhang, H., Wang, X., Pan, J., and Wang, H. (2023). SAKA: an intelligent platform for semi-automated knowledge graph construction and application. *Service Oriented Computing and Applications*, 17(3):201–212.

Zou, X. (2020). A Survey on Application of Knowledge Graph. *Journal of Physics: Conference Series*, 1487(1):012016.