SORBONNE UNIVERSITÉ

EURECOM
Sophia Antipolis

A Framework for the Continuous Curation of a Knowledge Base System

Dissertation

submitted to

Sorbonne Université

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Author:

Naser AHMADI

Scheduled for defense on the 8th December, 2021, before a committee composed of:

Reviewers

Prof.	**Fabian SUCHANEK**	Telecom Paris, France
Prof.	**Serena VILLATA**	University of Nice Sophia Antipolis, France

Examiners

Prof.	**Ziawasch ABEDJAN**	Leibniz University Hannover, Germany
Prof.	**Raphael TRONCY**	EURECOM, France

Thesis Advisor

Prof.	**Paolo PAPOTTI**	EURECOM, France

SORBONNE UNIVERSITÉ

EURECOM
Sophia Antipolis

Un cadre pour la curation continue d'un système de base de connaissances

Thèse

soumise à

Sorbonne Université

pour l'obtention du Grade de Docteur

présentée par:

Naser AHMADI

Soutenance de thèse prévue le 08 Décembre 2021 devant le jury composé de:

Rapporteurs

Prof.	**Fabian SUCHANEK**	Telecom Paris, France
Prof.	**Serena VILLATA**	University of Nice Sophia Antipolis, France

Examinateurs

Prof.	**Ziawasch ABEDJAN**	Leibniz University Hannover, Germany
Prof.	**Raphael TRONCY**	EURECOM, France

Thesis Advisor

Prof.	**Paolo PAPOTTI**	EURECOM, France

To Niloufar

Abstract

Entity-centric knowledge graphs (KGs) are becoming increasingly popular for gathering information about entities. The schemas of KGs are semantically rich, with many different types and predicates to define the entities and their relationships. These KGs contain knowledge that requires understanding of the KG's structure and patterns to be exploited. Their rich data structure can express entities with semantic types and relationships, oftentimes domain-specific, that must be made explicit and understood to get the most out of the data. Although different applications can benefit from such rich structure, this comes at a price. A significant challenge with KGs is the quality of their data. Without high-quality data, the applications cannot use the KG. However, as a result of the automatic creation and update of KGs, there are a lot of noisy and inconsistent data in them and, because of the large number of triples in a KG, manual validation is impossible.

In fact, KG creation and maintenance is a never-ending process and semi-automatic curation techniques are needed for adding new facts to a KG and for removing noises and inconsistencies. Computational methods can be employed in the creation and curation of KGs. Deep learning techniques are one of such computational methods that can be exploited for finding new relationships by matching entities in a KG. Mining systems are another computational approach that can help the users improving the quality of KGs. In this line of work, logical rules are used to express dependencies between entities in KGs. They are useful in tasks such as query answering, data curation, and automatic reasoning, but they are not included with the KGs. These rules must be defined manually or discovered using rule mining techniques.

In this thesis, we present different tools that can be utilized in the process of continuous creation and curation of KGs. We first present an approach designed to create a KG in the accounting field by matching entities. The proposed approach starts by extracting entities from auditing documents and then finds the links between related entities. This is especially challenging because auditing entities can have different granularity, such as activities, taxonomies, and topics. We then introduce methods for the continuous curation of KGs. We present an algorithm for conditional rule mining and apply it on large graphs. Our results show that conditional rules can help human curators in finding more accurate rules for specific types of entities. Next, we describe RuleHub, an extensible corpus of rules for public KGs which provides functionalities for the archival and the retrieval of rules. RuleHub defines different measures for capturing the confidence and the quality of each rule. We also report methods for using logical rules in two different applications: teaching soft rules to pre-trained language models (RuleBert) and explainable fact checking (ExpClaim).

Abrégé

Les graphes de connaissances (KG) centrés sur les entités sont de plus en plus populaires pour recueillir des informations sur les entités. Les schémas des KG sont complexes, avec de nombreux types et prédicats différents pour définir les entités et leurs relations. Ces KG contiennent des connaissances spécifiques à un domaine, mais pour tirer le maximum de ces données, il faut comprendre la structure et les schémas du KG. Leurs données comprennent des entités et leurs types sémantiques pour un domaine spécifique. En outre, les propriétés des entités et les relations entre les entités sont stockées. En raison de l'émergence de nouveaux faits et entités et de l'existence de déclarations invalides, la création et la maintenance des KG est un processus sans fin. Les règles logiques sont l'un des outils qui peuvent être utilisés pour mettre à jour un KG avec de nouveaux faits et en supprimer les données invalides. Les règles logiques sont employées pour exprimer les dépendances entre les entités dans les KG. Elles sont utiles pour des tâches telles que les Systèmes de questions-réponses, la curation des données et le raisonnement automatique, mais elles ne sont pas inclues dans les KGs. Ces règles doivent être définies manuellement ou découvertes à l'aide des techniques de fouille des règles.

Dans cette thèse, nous présentons d'abord une approche destinée à créer un KG dans le domaine de l'audit en faisant correspondre des documents de différents niveaux. L'approche proposée extrait d'abord les entités des documents d'audit et trouve ensuite les liens entre les entités liées. Les entités d'audit peuvent être des mots, des termes, des paragraphes ou des documents. Nous introduisons ensuite des méthodes pour la curation continue des KGs. Nous présentons un algorithme pour la fouille des règles conditionnelles et l'appliquons sur de grands KGs. Nos résultats montrent que laes règles conditionnelles peuvent nous aider à trouver des règles plus précises pour un type spécifique d'entités. Ensuite, nous décrivons RuleHub, un corpus extensible de règles pour les KGs publiques qui fournit des fonctionnalités pour l'archivage et la récupération des règles. RuleHub définit différentes mesures pour capturer la confiance et la qualité de chaque règle. Nous proposons également des méthodes pour l'exploitation des règles logiques dans deux applications différentes : l'apprentissage de règles souples à des modèles de langage pré-entraînés (RuleBert) et la vérification explicable des faits (ExpClaim).

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Paolo Papotti for providing me with the opportunity to join his team and for his noble guidance and constant support of my research. Without his tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study. Besides my advisor, I would like to thank all the members of my thesis jury committee for generously offering their time and for their encouraging words and thoughtful comments and suggestions: Prof. Fabian Suchanek, Prof. Serena Villata, Prof. Ziawasch Abedjan, and Prof. Raphael Troncy.

I would like to thank the members of my research group for their feedback and cooperation, in particular Mohammed, for all the moments we were working together. An acknowledgement goes to my teammates in KPMG which it was a pleasure to work with: Hansjorg, Hendrik, and Florian.

During the years that I have spent at EURECOM, I had a chance to make amazing friends. I want to thank all of them for the time that we spent together, with a very special thank to Ismail for his support and all the fun moments we had.

Thank you to my Yaar, Niloufar, I am very lucky to have you in my life. Thank you for your encouragement, unconditional support, and endless patience. Without you, this thesis would not have been possible.

Last but not the least, I deeply thank my parents, for their great love, tremendous support, and hope they had given to me. I am especially grateful to my siblings Fahime, Zahra, Yaser, and Reza for always being there for me and for the strength they gave me.

Nice, December 2021 **Naser**

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

The acronyms and abbreviations used throughout the manuscript are specified in the following. They are presented here in their singular form, and their plural forms are constructed by adding and *s*, e.g. KG (Knowledge Graph) and KGs (Knowledge Graphs). The meaning of an acronym is also indicated the first time that it is used.

CRF	Conditional Random Fields
CWA	Closed World Assumption
ER	Entity Resolution
GATE	General Architecture for Text Engineering
ILP	Inductive Logic Programming
KB	Knowledge Base
KBS	Knowledge Base System
KG	Knowledge Graph
LCWA	Local-Closed World Assumption
LM	Language Model
ML	Machine Learning
NLP	Natural Language Processing
OWA	Open World Assumption
PLM	Pre-trained Language Model
RDF	Resource Description Framework
SHACL	SHApes Constraint Language
ShEx	Shape Expressions
TM	Text Matching
UIMA	Unstructured Information Management Architecture
VN	Value Normalization

Chapter 1

Introduction

A *Knowledge Graph* (KG) is a structured representation of information which stores real-world entities as nodes, and relationships between them as edges. The entities and relations in a KG have semantic descriptions in the form of types and properties associated with them. KGs represent data with large collections of interconnected entities. Usually, a rich set of types (classes) are available to describe the entities (e.g., entity *Paris* is a *city*, *France* is a *country*), while predicates describe their relationships (a city *isCapital* of a country) and their properties (France has a *population:62M*). RDF KGs organize information in the form of triples with a *predicate* expressing a binary relation between a *subject* and an *object*. KGs store large amounts of factual information, and KG triples, or *facts*, represent information about real-world entities, their properties, and their relationships, such as "Larry Page is the founder of Google". Research has been conducted on KG creation both in the research community (NELL [3], DBPedia [4], Yago [5], Wikidata [6], FreeBase [7], DeepDive [8]) and KGs have also attracted interest from the industry, as evident from the ongoing efforts in several companies such as Google [9] and Wal-Mart [10]. For example, the English version of DBpedia stores 850 million facts[1].

The syntactic and semantic structures of knowledge in KGs are useful in building applications, such as Question Answering [11, 12] and Semantic Search [13].

KGs are usually built around facts and relationships. Consider for example the Knowledge Graph at Google with millions of entities and relationships between them [14]. Google search engine exploits the semantic information in the KG to improve the search service [15–17]. For example, given a query term, they recognize it as an entity and show a summary of information about it, find a category for the entity, and identify related entities. The KG structure and schema enable to understand user intent and the meaning of the content. KGs also enable querying, since the data has a schema; Google shows this feature by querying for example "when was Tesla founded". Intelligent assistants rely on KGs to answer simple user queries [16, 18, 19]. More complex queries can be defined directly on the schema of the graph, for example with aggregates (e.g., average net profit for auditing companies in 2015). The KG can also be used for autocomplete (type-ahead) by using the alias predicate to rewrite queries with synonyms, or by using type/class hierarchies to translate specific entities with little support in the data into more general concepts [20].

[1] https://www.dbpedia.org/resources/snapshot-release

However, in an enterprise domain, the KG looks very different. While the Google KG is *encyclopedic*, covering objects and facts in the real world, some enterprises may have information which is mostly composed of abstract topics and rules, making it close to a *commonsense* KG. See examples that highlight the difference in Figure 1.1. The latter category is much harder to build automatically, and most efforts rely on humans, usually in a crowdsourcing fashion, such as ConceptNet [21] and ATOMIC [22]. The specific and technical domain of the enterprise content is one of the biggest challenges in creating an industrial KG.

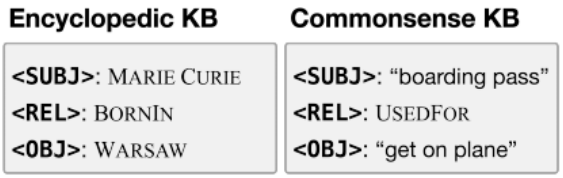


Figure 1.1 – Examples of knowledge triples from encyclopedic and commonsense KGs [1].

External commonsense resources, such as ConceptNet, are used in some of the methods that are introduced next, but they are not a direct solution to the KG construction problem. Many terms are domain-specific, so they are either missing from the existing resource or their modeling in the commonsense KGs does not match the level of details that is needed in the enterprise setting. For example, in an accounting dictionary *AIM* stands for *Alternative Investment Market* and *goodwill* is "a type of tangible assets that occurs when a buyer acquires an existing business", while these words have very different meanings in a general dictionary.

In this work, we propose tools for automating different parts of a framework for continuous creation and curation of KGs. Our tools help the users in two tasks: i) constructing a KG from unstructured data and ii) KG maintenance and curation.

We start this chapter with an example of a KG we created with KPMG and then explain the difficulties and challenges in finding entities and relationships in creating an auditing KG (Section 1.1). Next, we discuss the general problem of quality of data in KGs and introduce computational methods to address this challenge (Section 1.2).

1.1 A Knowledge Graph for Audit Information

We start by introducing a very high-level KG based on node entities and only two kinds of relationships between entities. This KG is different from traditional entity-centric knowledge graphs and is motivated by text data and taxonomies that are available in the KPMG corpus. This information must be modeled according to their target applications.

Figure 1.2 shows a sample with a few sentences for two documents (left) and a fragment of a taxonomy for accounts (right). In our corpus there are thousands of documents with variable size, from very short (only one sentence) to quite large documents with dozens of paragraphs. For the taxonomies, they can vary in size but are in the order of hundred nodes, each composed of a short sentence. These can be considered the starting point of the KG construction and from those several other nodes are derived as we discuss next.

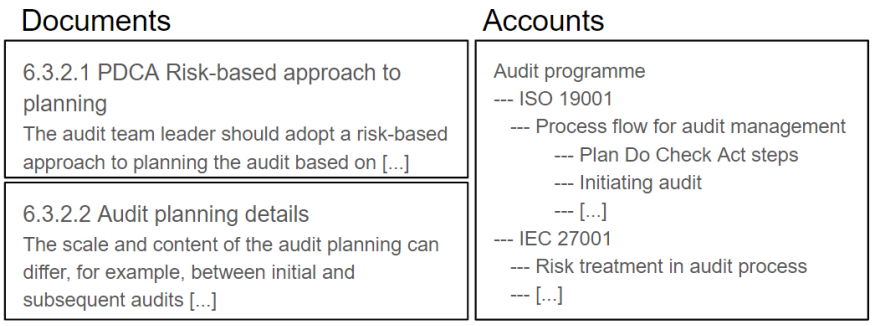


Figure 1.2 – An example of KPMG's documents (left) and an account taxonomy (right).

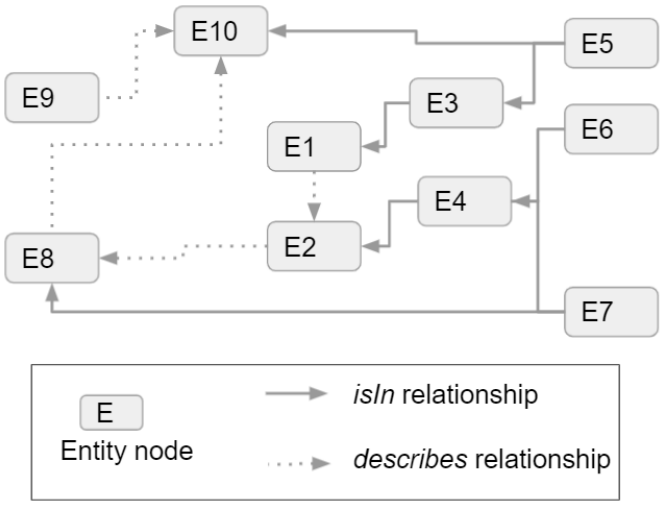


Figure 1.3 – Generic KG with one node type and two kinds of relationships.

In Figure 1.3, there is only one kind of node, representing entities. Those can be single words, paragraphs or long documents. The relationships across them are represented by directed edges and the nodes are connected in many to many relationships. We consider two kinds of relationships. The first one is the containment, in the example E6 *is contained in* E4. This could be a word contained in a document, for example, or a sub-element in a hierarchy (e.g. the relation between *IEC 27001* and *Audit programme* in accounts' hierarchy in Figure 1.2). Also, E2 could be a topic that *describes* document E8. We remark that all edges are assumed to have the same weight, i.e., 1, but in the KG edges can be weighted with a value between 0 and 1. This representation is very generic and simplified, we introduce it to give a feeling of the kind of graph that we are interested in.

We are now ready to discuss the KG for audit information. In Figure 1.4, we report a simplified small portion of the KPMG's KG. The nodes are of six different types and are represented with different colors:

- **Documents** (D) nodes are (possibly long) texts containing one to multiple paragraphs. For

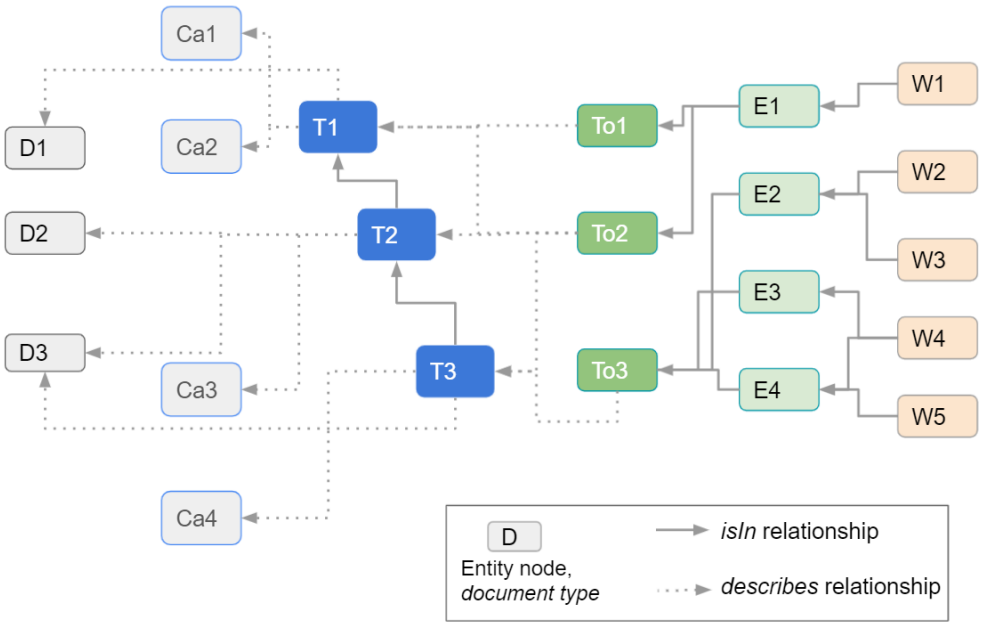


Figure 1.4 – KPMG's KG with six types of nodes.

example, in Figure 5.9 two activities are shown; those correspond to two D nodes.

- **Taxonomy** (T) nodes are auditing concepts following a hierarchical structure. For example, every account can be represented as a path from the root node to the leave, e.g., *Audit programme* → *ISO 19001* → *Initial audit*.
- **Caption** (Ca) nodes are short documents nodes that are described by taxonomy nodes.
- **Topics** (To) nodes are terms with one or multiple related entities; e.g., "risk treatment" and "audit process" are topics for the *Risk treatment in audit process* account. Entities are associated to a topic with a weight.
- **Entities** (E) nodes contain n-gram terms that are representative of relevant items, names and concepts in the audit domain. Every entity is the representative for a *family* of words, where a family includes (with isIn relationships) synonyms and abbreviations that can be used to express such entity in documents. Entities are contained in documents, but we do not represent such edges in the figure for clarity.
- **Word** (W) nodes are words in the topics or their synonyms or other variations. E.g., *auditing*, *adt* and *prc* are words for entity *audit process*. Word are contained in documents, but we do no represent such edges in the figure for clarity.

1.1.1 Building and curating the Audit Knowledge Graph

Given the very challenging nature of the auditing content and regulation, it is not very advantageous to rely on automatic methods for encyclopedic KG construction [19, 23, 24]. We

experimented largely with such methods, but with results that were far away from the required quality [25]. The most difficult problem is that auditing entities are not standard named entities, such as *France* and *IBM*. These challenging entities are contained in auditing documents and can be acronyms, abbreviations, topics, and terms, and hence existing tools do not obtain high-quality results. Another issue is that there is no training data in this domain, and general corpora miss the subtle differences in the audit domain [19, 26].

As the project moved forward, different components of the KG have been manually defined by the domain experts at KPMG. This process had identified some of the opportunities to introduce automatic methods to help in the KG construction.

In general, the first task is the automatic identification of nodes and a second task is the identification of relationships across the different nodes. We first tackle the task of generating the entity nodes. We generate families of words for each entity node. The goal is to find a group of semantically equivalent words and to identify the representative entity given only the documents (including the taxonomy nodes and the associated captions). We utilize mentions of entities to find families. A mention is a span of text that refers to an entity. Words and representative entities are related with weighted *isIn* relationships. We then propose a method to identify relationships of type *describes* between nodes, and we conduct experimental campaigns on the discovery of relations between documents (activities) and taxonomy nodes (accounts). In particular, we report results matching accounts to activities which is a challenging task for existing methods because of the quantity of text in our entities.

1.2 Quality of Data

A major challenge with KGs is the quality of their data, due to the processes involved in their creation and update. This problem also applies for the process we introduced in Section 1.1.1 for the auditing KG. The structured data in the KGs is usually extracted from multiple sources, possibly from the Web, without human validation. This raises two main issues. The first problem is factual mistakes. Incorrect or outdated data can be transmitted from the sources to the KGs, or noise can originate from the automatic extractors [9, 27]. The second issue is incompleteness. As a graph is rarely complete in practice, *closed world assumption* (CWA) does not hold in KGs [9, 28], i.e., it is not possible to conclude that a missing fact is false. We therefore follow the *open world assumption* (OWA) and consider it as *unknown*. Also, in many cases the KG schema is not fixed, i.e., the set of predicates changes over time and new facts can be inserted without integrity checks.

Because of these issues, the quantity of incompleteness and errors in KGs can be large, with up to 30% errors reported for data extracted from Web sources [29, 30]. KGs can have lots of data, e.g., WIKIDATA has over a billion facts and millions of entities, therefore it is not feasible to manually verify triples to find mistakes and add missing facts. Tools are needed to assist humans in the KG curation. A natural approach in this direction is to mine *declarative rules*. Once validated, these rules can be run at scale on the KG to increase the quality of its data [10, 28, 31–33].

$$r_{pos} = \texttt{spouse}(a,b) \wedge \texttt{child}(a,v_0) \rightarrow \texttt{child}(b,v_0)$$
$$r_{neg} = \texttt{parent}(a,b) \wedge \texttt{birthDate}(b,v_0) \wedge \texttt{birthDate}(a,v_1) \wedge v_0 > v_1 \rightarrow \bot$$

For example r_{pos} and r_{neg} are two logical rules that can be used to deduce additional statements about entities of a KG. Rule r_{pos} states that if two persons are married, each one of them is the parent of the child of the other one, and rule r_{neg} expresses that a person cannot have a bithDate smaller than her parents.

These rules must be manually crafted to be executed over KGs. This process can be difficult because the domain experts, who know the semantics for the dataset at hand, may miss the computer science background to formally express rules. Moreover, the task of writing a set of rules is time-consuming, as there can be thousands of them that are needed to achieve good quality results [34]. In this context, a system to mine rules is important to assist the users in data curation, as well as in any task involving reasoning over the KG [35, 36].

Rule mining is the process of automatically extracting logical rules from KGs. These rules can be exploited in KG curation by reducing inconsistencies or adding new facts to it. E.g. r_{pos} can help us to complete the KG by adding new edges between entities while r_{neg} will help us in removing nodes or edges that are disagreeing with it. Four main issues make the mining of rules from KGs challenging:

Quality of the Data. Most rule mining algorithms assume that input data has very small amounts of errors [37–40], but KGs are incomplete and can have high percentages of mistakes.

Open World Assumption. Some methods assume that positive and negative examples are available [41, 42]. However, KGs contain only positive statements, and it is not possible to assume CWA, as there is no obvious solution to derive negative facts acting as counter examples.

Data Volume. Several existing algorithms for rule mining exploit the premise that the input graph can fit entirely in the main central memory [28, 32, 33, 43]. As KGs can have a very large size, existing methods limit the size of the search space by restricting the mining to a simple rule language, which can miss some of the important patterns in the data.

Soft Rules. Most logical rules are not true in all the cases and for this reason a confidence score should be assigned to them. Defining these confidences in a way that represent the accuracy of rules is another challenge that should be solved in the automatic rule discovery task. For example, the pattern that expressed in r_{pos} certainly has exceptions, but it would still cover the majority of the cases.

We start this thesis by proposing a method for creating a KG for the auditing domain. We create this KG using auditing documents, taxonomies and the relationships between them. Next, we extend a rule mining system to enhance its performance by extracting conditional rules and rules over literal predicates. We then propose a confidence score for computing the accuracy of positive and negative rules. Finally, we employ logical rules in tasks such as fact checking and transferring common-sense knowledge to Pre-trained Language Models.

1.3 Outline of the Thesis

The main goal of this thesis is to study novel approaches for the creation and continuous curation of KGs. For this matter, we introduce an algorithm for finding entities and their relationships, and develop an efficient rule mining algorithm for finding positive and negative rules. We also propose measures for computing the confidence of positive and negative rules and employ logical

rules in a fact checking application.

The remainder of the thesis is organized as follows:

Chapter 2. This chapter defines the main notions of Knowledge Graphs and discusses their architecture. We then introduce the process of KG construction and describe each step and our work will be positioned w.r.t. existing KG curation techniques.

Chapter 3. In this chapter, we explain our proposal for finding entities and relationships in an audit KG. For identifying representative entities, we first extract important entities from documents, and then apply an entity matching algorithm to find representatives. For each representative we generate a family of words. In the second section of this chapter, we will present an unsupervised method for matching text to data [44]. This method was initially designed for identifying edges in KPMG's KG but our experiments show that it performs well in other tasks. It extends recent solutions designed for the **ER** (entity resolution) task [45–47]. Our first contribution is introducing a module for graph generation as it creates a rich representation which is then reflected in the *domain-specific* embeddings for metadata nodes. Our second contribution is the expansion and compression approach that, together with the matching generation module, exploits the benefits of embeddings metadata nodes in an *unsupervised solution*. We also provide two new datasets for the text to data matching task. This section is based on the following paper:

> Naser Ahmadi, Hansjorg Sand, and Paolo Papotti, **Unsupervised Matching of Data and Text**, International Conference on Data Engineering (ICDE), 2022.

Chapter 4. This chapter reports our work in extending an existing system for rule discovery in RDF KGs (RuDiK) [48]. The purpose of our work is to enable RuDiK to extract *more accurate rules*. Many rules are only valid in a given geographic area, at a specific time, or for specific categories of entities. We extend the core mining methods to capture conditional rules, which are rules that apply just to a subset of the data. Conditional rules are characterized by a selection with a constant over the values of an entity or a type. Our purpose is to find subsets of the data for which mining leads to new rules that are not identified by the general mining. The following paper forms the basis of this chapter:

> Naser Ahmadi, Phi Huynh, Vamsi Meduri, Stefano Ortona and Paolo Papotti, **Mining Expressive Rules in Knowledge Graphs**, ACM Journal of Data and Information Quality, 2019.

Chapter 5. This chapter discusses methods for collecting and annotating logical rules and proposes a technique for using them to transfer common-sense knowledge to Pre-trained Language Models. Chapter 5 is presented in three sections:

The first section introduces RuleHub [49] which is the first open corpus of (automatically generated) rules for public KGs. RuleHub annotates logical rules with different metrics and introduces a new confidence computation method for negative rules, i.e., rules that identify contradictions in the data. This section is based on our following paper:

> Naser Ahmadi, Duyen Truong, Mai Dao, Stefano Ortona, and Paolo Papotti, **RuleHub: a Public Corpus of Rules for Knowledge Graphs**, ACM Journal of Data and Information Quality, 2020.

In the second part of Chapter 5, we propose two methods for extracting logical rules for Wikidata KG [50]. This section is based on the following poster:

> Naser Ahmadi, and Paolo Papotti, **Wikidata logical rules and where to find them**, Wiki Workshop, 2021.

The third section of this chapter tackles the problem of transferring common-sense knowledge to Pre-trained Language Models. For this matter, we develop a model [51] to transfer the knowledge in logical rules extracted for KGs to a language model. In this section, we introduce the problem of teaching soft rules expressed in a synthetic language to PLMs through fine-tuning and release the first dataset for this task. We also introduce techniques to predict the correct probability of the reasoning output for the given soft rules and facts. This section is based on the paper below:

> Mohammed Saeed, Naser Ahmadi, Paolo Papotti, and Preslav Nakov, **Teaching Soft Rules to Pre-trained Language Models**, The 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2021.

Chapter 6. We introduce ExpClaim [52] in this chapter. ExpClaim is a fully *automated and interpretable fact checking system* that effectively exploits uncertain evidence. Experimental results on a real KG show that our method (i) obtains qualitative results that are comparable or better than existing black-box ML methods and (ii) outputs human-consumable explanations. The following paper is the basis for this chapter:

> Naser Ahmadi, Joohyung Lee, Paolo Papotti and Mohammed Saeed, **Explainable Fact Checking with Probabilistic Answer Set Programming**, Conference for Truth and Trust Online (TTO), 2019.

Chapter 7. This chapter concludes this thesis and states some further directions for future work.

Chapter 2

Knowledge Graphs

A knowledge graph (KG), which acts as the repository of data, is usually composed of triples in the form of (subject, predicate, object), where subjects are entities (e.g., The Mona Lisa painting, nodes in the graph), entities or lexical values (e.g. Leonardo da Vinci or 1503, nodes or properties in the graph) play the role of objects and their relationship is an instance of a generic predicate (createdBy, an edge in the graph). KG is usually coupled with a component which is the language used to express knowledge representation and reasoning. This component is made of a logical formalism for expressing facts and rules, and an optional and customizable reasoning engine that uses this language. KGs are used in problem-solving procedures and to support human learning, decision-making and actions. Due to the automatic creation and noisy data sources, there are inconsistencies and incompleteness in KGs and methods are needed to tackle these problems. In this chapter, we first discuss KGs and their architecture and explain three steps of the KG construction process (Section 2.1). We then introduce different methods that can be used in curating KGs (Section 2.2).

2.1 Knowledge Graphs

With the emergence of computers with "machine knowledge" that can power intelligent applications, methods for converting noisy Internet data into well-structured knowledge are needed. Major advances in the automatic construction of large-scale-high-quality knowledge graphs have made this vision a reality [53].

The creation of RDF knowledge graphs is an important activity in modern information systems and over the last 15 years, lots of large public KGs became available [3–6]. These KGs provide millions of entities (individuals, organizations, countries, books, etc.) and their properties and relationships (who is the spouse of a person, who wrote a book, where a company is located, etc.). Researchers also introduced a number of institutional KGs [7, 14, 54].

A *Knowledge Graph* (KG) is a structured representation of information storing real-world entities and lexical values as nodes, and relationships between them as edges. Most KGs organize information in the form of RDF triples. Each triple contains a *predicate* (edge in the graph) expressing a binary relationship between a *subject* and an *object* (nodes in the graph). The structured data not only enables "classic" analytics, such as querying with structured languages (say SQL, SPARQL, or any graph oriented declarative language), but also advanced analysis,

such as logical reasoning. As KGs are modelled as graphs, traditional mining algorithms can be applied on the data: clustering [55], link prediction [56], topological sort [57], shortest path [58], PageRank/HITS [59], or learning and using embeddings [60]. Knowledge Graphs can be used to enable or improve a wide range of applications including such as semantic search [11], question answering [13], data cleaning [61], and information retrieval [62].

2.1.1 Knowledge Graph Architecture

This section introduces the fundamental elements for representing information in a KG.

Entities

In a KG, subjects and objects are entities like individuals, countries, sport teams, companies, etc. An entity is any abstract or concrete object of fiction or reality [63]. For example *France*, *Thierry Henry*, *Arsenal F.C.* can be some entities of a KG. We define *identifiers* to denote entities unambiguously. An entity's identifier is a string of characters that uniquely identifies it [63]. Entities can have some properties. For an entity of type person, properties such as age, height, birthDate, and birthPlace can be defined. For example, *Thierry Henry* have these properties in *DBpedia*:

birthDate (Thierry Henry, 1977-08-17)

height (Thierry Henry, 188)

KGs model only entities that match their scope and domain. For example, KPMG's KG represents auditing activities (documents) and taxonomies (accounts) in Figure 1.2 as entities. Finding these entities is not easy and usually manual annotations from domain experts is needed.

Types

Each entity is a member of one or multiple semantic classes (types). A class, also known as a type, is an identified group of entities that have similar characteristics [63]. A member of such group is referred to as a class instance. For example:

type (Thierry Henry, Person),

type (Thierry Henry, Football Player),

type (France, Country),

type (Arsenal F.C., Football Club)

As it can be seen, an entity can belong to multiple types, and also types can relate to each other, e.g. A 'Company' is founded by a 'Person'. There can be hierarchical relationships between classes, e.g. a 'Football player' is a 'Person'. In creating a KG, we consider general concepts as a type if we are interested in specific instances of those types. E.g., in KPMG's KG concepts like topic, activity, and taxonomy are among types and each one of them can have multiple instances. In Figure 1.2, each account (e.g. *Process flow for audit management*) is an entity of type taxonomy and is composed of multiple topic entities (e.g. *process flow*, *audit management*).

Relationships

Besides their properties, entities can have relationships with other entities. These relationships express facts about entities and can be expressed with predicates.

For example in the following there are two triples which express facts about an entity (*Thierry Henry*).

playsFor (Thierry Henry, Arsenal F.C.),

bornIn (Thierry Henry, France)

In Figure 1.2, each document entity can have relationships with one or multiple accounts and account entities can have hierarchical relationships (e.g. *ISO 19001* and *Initiating audit*).

2.1.2 Knowledge Graph Construction

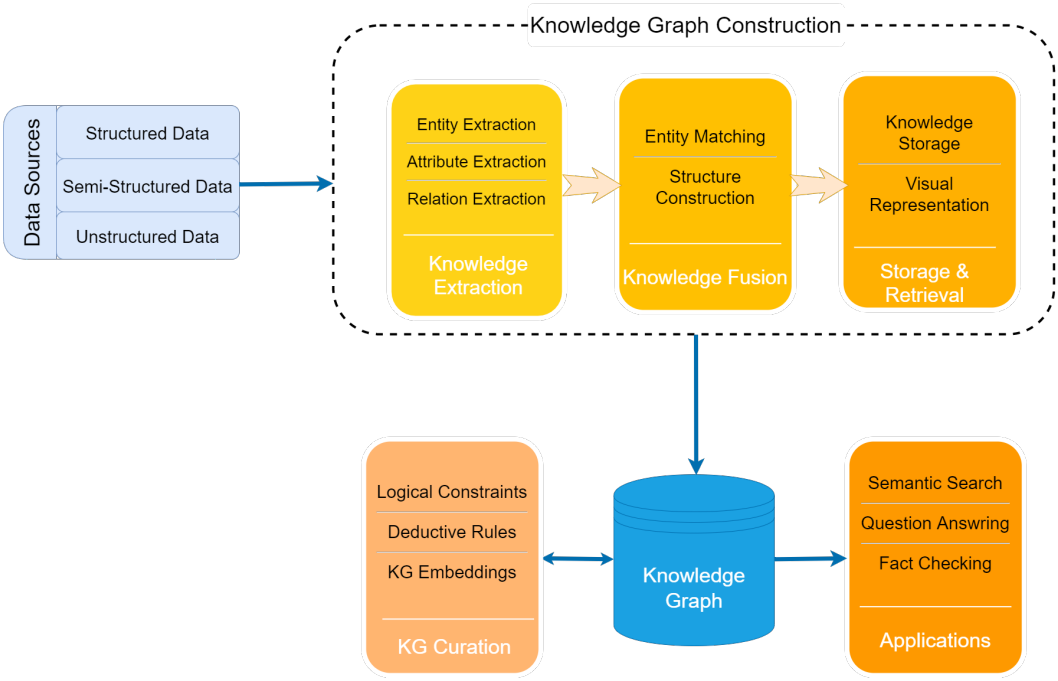


Figure 2.1 – The process of KG construction starts with data sources and contains three modules. After KG creation, because the applications cannot use the KG without high-quality data, continuous curation is required.

Constructing a KG in a bottom-up approach is an iterative update process, including three tasks: i) knowledge extraction, ii) knowledge fusion, iii) knowledge storage and retrieval [64]. This process starts with extracting knowledge from different types of data sources, then it applies methods for constructing KG's structure and matching entities. Finally, the created KG should be stored and represented in an appropriate scheme considering its structure. After creation, KG can be utilized in a wide range of applications such as semantic search and fact checking. Also, because information changes over time, the KG needs to be curated continuously in order to be updated and clean. Figure 2.1 shows the different modules in KG construction and curation tasks.

Knowledge extraction consists of extracting different KG's components from data sources. There are three kinds of data sources for extracting knowledge: structured data like relational data, semi-structured data such as HTML and JSON files, and unstructured data like text documents. There are four types of elements that should be defined for constructing a KG: (1) entities, (2) relationships, (3) mentions of entities, and (4) relationship mentions [25]. We use mentions to define unique identifiers for entities and relationships. Identifying synonyms and disambiguating mentions is a key step for creating a high-quality KG [63]. In the process of constructing KPMG's KG that we discuss in Chapter 3, we use *isIn* predicate to show the relationships between entities with semantically equivalents mentions.

Entity extraction is the task of discovering entities from a wide variety of knowledge resources and classifying them into pre-defined categories such as person, location, and organization [64]. Entity extraction is usually incomplete when the goal is to fully cover a certain domain such as auditing, and to associate all entities with their relevant classes [63].

Over the years, many tools for knowledge extraction have been released. Early knowledge extraction mainly employed dictionary-based and pattern-based entity spotting approaches [65,66]. Machine learning models have been exploited by more recent tools. They include supervised approaches that use learning algorithms to build models from manually annotated training datasets. Supervised models based on CRF [67] and LSTM [68] have been applied in knowledge extraction process. More recently, some semi-supervised algorithms have been suggested to avoid the annotation effort [25].

The next step is to find attributes and relationships for each entity [69]. The purpose of this process is to enrich the entities with information in the form of triples. Extracted information cover two types of relations: 1) attributes with literal values such as the birthdate of a person, the founding date of a company, or the weight of a laptop; and 2) relations with other entities such as spouse of a person, author of a book, composer of a song, or publisher of a game. A wide range of relationship extraction methods have been proposed. Early methods employed methods such as pattern-based extraction techniques [70] while more recent methods use models based on distant supervision [71] and transformer networks [72, 73].

Knowledge fusion is the second step in the KG construction and contains entity matching and constructing KG's structure. Discovering entities and type information in web pages and text documents may generate noisy output such as two distinct entities in the KG that refer to the same real-world entity. For this reason, we need to normalize mentions and make sure that there is only one entry for all of the entities' variants. Entity matching is the process to judge whether different entities refer to the same objects of the real world [74]. Several link discovery [75] and entity matching [76] methods have been proposed to tackle this problem. In the next chapter, we

explain an entity matching algorithm that we utilized to find representatives of families.

Knowledge storage and retrieval is the last step in the KG construction. Using methods from the previous steps, we can now assume that we have a KG that has a set of entities and each one of them has a set of attributes and relationships. The final step is to store the KG in a way that it is easy to use and update. There are two main storage types for KGs: RDF-based store [77], and graph database store [78]. RDF is a representation of knowledge graphs, which uses triples and unique identifiers of entities to describe the knowledge graph while graph database uses graph structures with nodes, edges, and properties to represent and store KGs [64]. The advantage of RDF-based stores is that the efficiency of query and merge-join of triple patterns is good while the advantage of graph databases is that they support a variety of graph mining algorithms [64].

2.2 Knowledge Graph Curation

Even though the construction methods attempt to make sure that a KG has a high quality, there will always be some errors and noise in the KGs. As a result, additional quality assurance and curation tasks should be done to maintain and enhance the quality of data. After creating a KG, it should be maintained, developed, and enhanced over a long period of time [63]. This demands a long-term approach of **KG curation** in order to keep the KG material clean and valid, as well as to ensure quality. The KG curation is a "never-ending" process that includes two tasks [63, 79, 80]: i) **Removing inconsistencies.** The KG content should be continuously updated in order to maintain consistency and avoid contradictory statements. For example, the *playFor* relationship between a player and a football club can be valid only for a specific period of time, while *bornIn* is always valid. ii) **Adding new facts.** At a rapid rate, new entities and facts arise and are covered by new web sources. As a result, KGs should be improved and updated with this new information. However, since a full coverage is usually impossible, KGs follow an open world assumption policy.

In fact, given the large number of entities and relationships in KGs, it is impossible to manually find all errors and add all missing facts. As a result, we need automatic curation tools for validating triples in KGs. Tools such as **Constraints**, **Rules**, and **KG Embeddings** can assist the curators in this process.

Constraints are invariants that the KG must satisfy in order to be logically consistent [63]. There is a wide range of invariants that a KG should consider. For example, a person can only be in the spouse relationship with an object of type person, the sibling relationship should be symmetric, or a person cannot have more than one birthdate.

Wikidata is one of the KGs that uses constraints to ensure the quality of data. Property constraints have been defined to specify how properties should be used and the relationships that should exist or not exist for the classes they apply to [2]. Figure 2.2 shows two examples of constraints violation in Wikidata. We can use constraints languages such as *SHACL* [81] and *ShEx* [82] to define such constraints.

Logical Rules are another method for curating a KG. Rules can infer statements to make a KG consistent and to enhance its coverage [63]. For example, a rule can state that a person is always a child of her parents or cannot have a birthdate smaller than her parents. The first rule can add some new facts to a KG while the second rule will remove inconsistencies. Logical rules

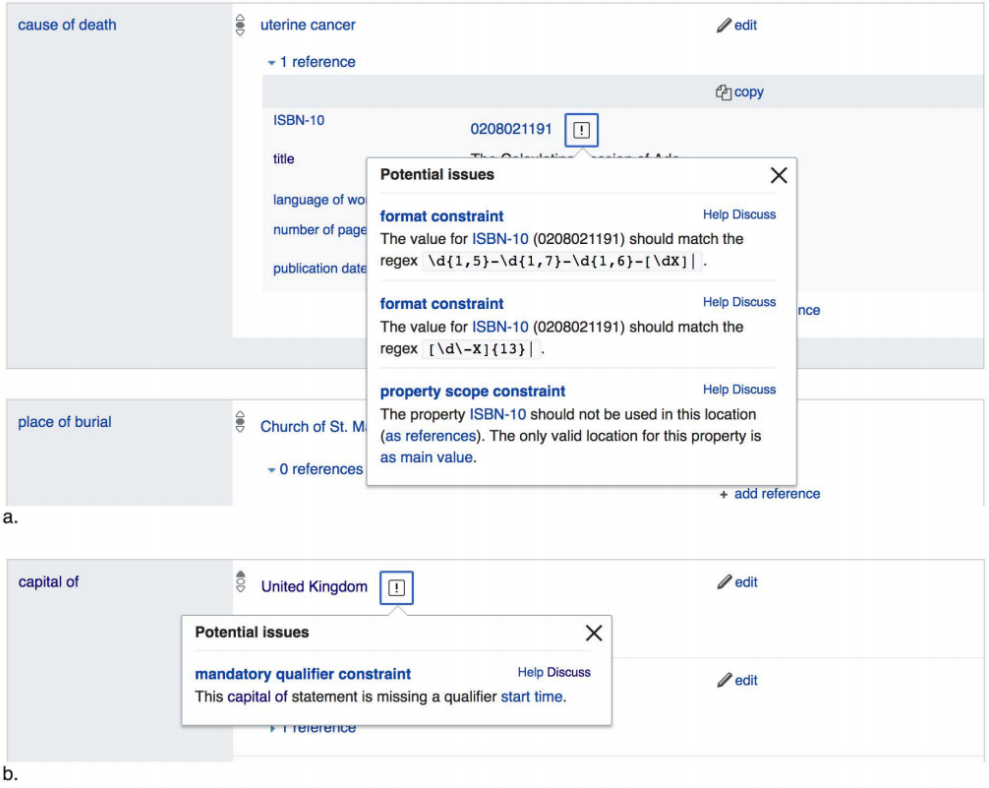


Figure 2.2 – Examples of constraints violation in Wikidata [2].

are usually restricted to **Horn Rules**. A Horn Rule is a disjunction of *atoms* with at most one unnegated atom. In the implication form, they have the following format:

$$A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow B$$

where $A_1 \wedge A_2 \wedge \cdots \wedge A_n$ is the *body* of the rule (a conjunction of atoms) and B is the *head* of the rule (a single atom). Rule mining algorithms such as *Ontological Pathfinding* [33], *AMIE* [83, 84], and *RuDiK* [79] can be applied on KGs in order to extract logical rules. In Chapter 4 we will cover logical rules in more details.

Knowledge Graph Embedding models are another tool that can help in the KG curation task. A KG embedding aims to map a KG into a dense, low-feature space, which is capable of preserving as much structure and property information of the graph as possible [85]. The basic idea behind KG embedding models is to embed KG components such as entities and relationships into continuous vector spaces. These embeddings can be used to help with tasks such as KG completion, relation extraction, entity classification, and entity resolution [60]. In recent years, a wide range of KG embedding models have been proposed. TransE [86] is a translation-based model which is inspired by Word2vec [87]. TransE models relationships by interpreting them as translations operating on the low-dimensional embeddings of the entities [86]. Translation-based extensions of TransE such as TransH [88] and TransR [56] have been proposed. Other popular KG embedding works can be divided into two groups: *Semantic Matching Models* such as

RESCAL [74] and *Neural Network-Based Models* like SME [76]. KG embeddings have been used in different KG completion tasks such as entity prediction [56, 89], link prediction [56, 90], relation extraction [91], and triplet classification [90].

2.3 Summary

In this chapter, we first introduced knowledge graphs and explained different elements in their architecture. Next, we presented a three steps process for constructing KGs. This process starts from data sources, extracts entities and relationships from them and after matching entities, stores constructed KG in a repository. Finally, we explained different techniques that can be used in the KG curation task. In the next chapters, we first describe our experience in constructing a KG in the auditing domain and then propose algorithms for KG curation.

Chapter 3

Nodes and Relations Identification

Entity and relation extraction are two important tasks in the KG construction. In this chapter, we propose tools for automatic knowledge extraction with the aim of building a KG in the auditing domain. We first present our method for automatic identification of entities for creating KPMG's KG in Section 3.1, and then discuss our solution for the task of identifying relationships, edges in the KG, across the different auditing entities in Section 3.2. As existing solutions could not solve the problem of text to structured text matching, we decided to develop a framework for identifying edges in KPMG's KG (text to structured text matching). Our experiments show that the proposed model performs well for other matching tasks as well.

3.1 Node Identification

The task of node identification covers the generation of two node types required for the KPMG applications: entities and words. It contains two sub-tasks: i) finding in the documents corpus the entities that can be the representative ones, and ii) finding the family members (a group of entities with semantically equivalent mentions) for each representative entity. The input of this task is a corpus of documents and the output will be a group of entity nodes (representatives) and words related to each one (family members). The representatives and their family members will be added as the nodes of the last layer of the KG depicted in Figure 1.4

3.1.1 Related Work

Knowledge extraction is the task of extracting the information from structured or unstructured text by identifying references to entities and defining relationships between them. Entities are often represented by nouns because they act as names of things [92]. In our work, we propose a dictionary-based method to find entities in unstructured auditing documents. Employing a rich name dictionary is a simple method for spotting entities and it has been used in multiple frameworks. A Name dictionary is a dictionary composed of all the named entities in the KG and it includes different forms of entities such as abbreviations, acronyms, and nicknames [63]. Tools like ConceptNet [21] and WordNet [93] can be utilized to enrich a name dictionary. Other systems, such as UIMA [94] and GATE [65], are among the NLP solutions that use name dictionaries.

In our work, these frameworks could not be applicable due to the domain-specific words. For example, acronyms like *AIM* or *PDCA* are not in the general dictionaries. Additionally, auditing entities are different from standard named entities and cannot be detected with standard name dictionaries. We employed an accounting dictionary to extract entities from KPMG corpus. This dictionary is generated by the field experts in KPMG and contains different forms of entities (acronyms, abbreviations, etc.).

Given our target application, there is a requirement of explicitly capturing all the variants for a given entity. A group of entities contains different variants of a representative entity. For example, *audit, auditing, audits, audited, aud* are a family of entities generated by experts in KPMG. We assume that in every family there is a natural representative that an expert can always identify. We use the representative of the group (*audit*) to make sure all the entities refer to the same entity in the KG. The problem of finding all strings that refer to the same representative is called value normalization (VN) [95]. Entity linking methods such as NERD [96] can be utilized in the family creation task. These methods can help in extracting named entities from text and linking them to their respective resources from a KG. Winston [95] is another approach which addresses value normalization problem by proposing an algorithm based on human operations and clustering.

Rule-based (pattern-based) systems are another category of knowledge extraction techniques which use hand-crafted patterns to find entities and their relationships. ReVerb [97] identifies and extracts binary relationships from English sentences using POS-based regular expressions. [98] proposes a list of patterns that can be employed to extract entities and their relationships from a text. For example, *X which is a (example|class|kind) of Y* or *Y such as* X_1, X_2 show the relationships between entities X and Y. EXEMPLAR [99] is another pattern-based approach which uses hand-crafted patterns based on dependency parsing trees in the task of extracting n-ary relationships.

In our case, we found that pattern-based approaches give acceptable results in detecting representatives and generating families. For example, it is easy to detect patterns for variations such as *audit* $\rightarrow$ *auditing* while some variants are more difficult (e.g. *audit* $\rightarrow$ *aud*). However, we could not apply existing patterns proposed by previous works on our documents as they are extracted by analyzing general texts. We applied rules (patterns) in two tasks: finding representatives of entity groups (Section 3.1.2) and generating different word forms for each representative 3.1.3. Inspired by Hearst patterns [98], we defined a group of patterns for these tasks.

3.1.2 Finding Representative Entities

For finding the candidate representatives, we assume that the relationships between taxonomy nodes and their captions have been computed as discussed in Chapter 1. These matches are available from a previous work done by auditors and have been manually validated as they are exploited in this step to identify related words.

In Figure 3.1, we report an example of KPMG accounts and captions from the *Balance Sheet* taxonomy. Here there are two accounts: *Property plant and equipment net* and *Property plant and equipment gross* and they are linked to three and two captions, respectively. An account can be denoted as a taxonomy path (*Assets* $\rightarrow$ *Property plant and equipment* $\rightarrow$ *Property plant and*

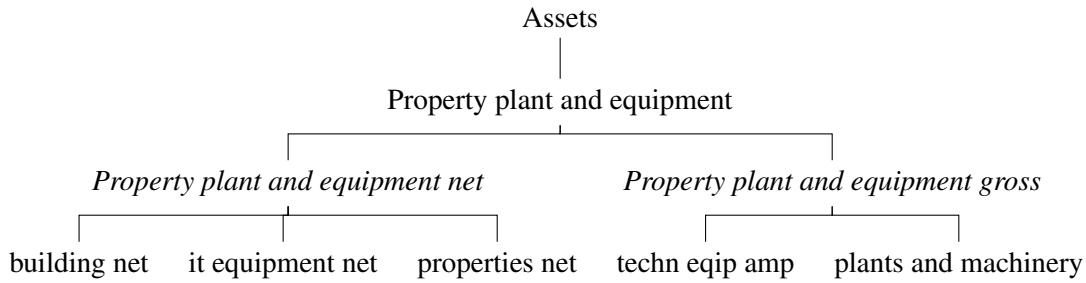


Figure 3.1 – An example of taxonomy nodes for accounts and associated captions (leaves, in Italic).

equipment net).

Assuming the captions have already been matched to the correct taxonomy nodes, we employ a two-step procedure in order to find representative entities. First, we group entities that are close to each other according to a metric, by comparing accounts and captions, and then detect the representative of each group.

In the first step, each account will be compared with all of its captions. The comparisons are at the token level and two metrics are used for matching:

1. **Distance.** We use a string similarity algorithm [100] to find the distance between two strings. This metric works better in finding tokens with the same root (e.g. *equipment* and *eqpm*).

2. **Closeness.** For capturing the cases that are not captured with the string distance algorithm, we use a pre-trained word embedding model [101]. This metric is equal to the cosine similarity between embeddings generated for two tokens. This metric helps us in finding more difficult cases like *expense* and *payment*

Algorithm 1 shows the process of grouping similar entities.

The output of this algorithm will be a list of groups. Each group contains one or more entities. Next, we detect a representative entity for each family. We define the representative of a group as "the shortest noun that is a valid word" and check the validity with an external (English[1]/German[2]) dictionary of words.

Example 1: Consider the taxonomy in Figure 3.1 as a corpus of nine documents (each node as a short document). By applying Algorithm 1 on this corpus, *(property, properties)*, *(equipment, eqip)*, and *(plant, plants)* will be among the extracted entity groups. Using the definition of the representative, *property*, *equipment*, and *plant* will be the representative entities of our groups.

3.1.3 Creating Entity Families

In Section 3.1.2, we generated a list of families for different variations of words, and then we selected *representatives*. The generated families, will be employed to find entities in unseen

[1] nltk.corpus

[2] https://sourceforge.net/projects/germandict/files

Algorithm 1: Grouping entities

```
Input. An account ac and its captions;
groups = set()
foreach caption in the captions set do
    foreach token1 in ac do
        foreach token2 in caption do
            dist = stringDistance(token1,token2)
            closeness = embeddingSimmilarity(token1,token2)
            if dist < β & closeness > γ then
                                                    ▷ Merging with existing groups
                flag = False
                foreach group in groups do
                    if intersect((token1,token2),group) then
                        flag = True
                        groups.add(union((token1,token2),group)
                        groups.remove(group)
                end
                                                    ▷ Adding as a new group
                else if ¬ flag then
                    groups.add((token1,token2))
            end
        end
    end
end
Output. List of groups
```

documents (client documents) and, because these new documents may have new variants of the words (e.g. new forms of abbreviations that are not in the family), we have to improve the families with new entities. The output of this step is the list of families expanded with new words.

property	ppty, prop, prope, proper, propert, properties, property, propt, propty
plant	pl, pla, plan, plant, plants, pln, plnt
equipment	eqipm, eqmt, eqp, eqpm, eqpmnt, eqpmt, equ m, equi, equip, equipm, equipme, equipmen, equipment, equipments, equipmnt, equipmt, equp, equpm, equpmnt

Table 3.1 – Examples of word families.

The members of a word family are a representative's different forms such as plural and acronyms. The family also contains different forms of abbreviations for the word. The different variations of a word are equivalent together, e.g., *properties, plants and equipment* is equivalent to *prop, plant, eq* or *propt, plnt, eqpm*. For example, the word families generated for the representatives in Example 1 are shown in Table 3.1.

Improving these entity families can help us in many tasks such as the process of mapping clients captions to KPMG accounts. For expanding these families, we start with the representative of each group and add its different variations to the group. We can employ the process explained

in Section 3.1.2 for finding the representatives.

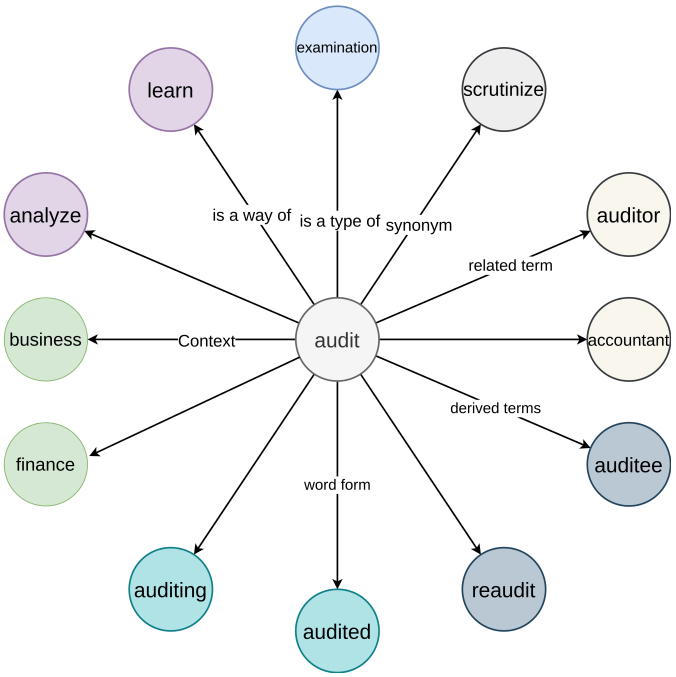


Figure 3.2 – Edges for entity 'audit' in ConceptNet.

For generating variations of a word, we employ four methods. There may be overlaps between outputs of these methods.

1. **Stemming.** It includes three different techniques for: i) finding the stem of a word; ii) finding the lemma of a word; and iii) finding singular and plural forms.

2. **Affixes.** We add affixes from a list of language-specific affixes (post-fix and pre-fix) to the word and keep the result if it is valid by checking it in an English/German dictionary of words.

3. **Abbreviations.** By looking at the example of captions from clients, we could infer some techniques they use for generating abbreviation forms for a word, such as take top 2 or top 3 consonants, take first character plus top 2 consonants, and so on. We applied this techniques to produce the abbreviation forms for a word.

4. **ConceptNet [21].** ConceptNet is a freely-available semantic network, designed to help computers understand the several meanings of common words[3]. For each entity, a wide range of different entities and their relationships to it can be extracted. Figure 3.2 shows some of the edges and nodes connected to the word 'audit'. In this technique, we extract different 'Word forms' for a word in the ConceptNet knowledge graph.

[3] https://conceptnet.io/

Algorithm 2: Generate variations for entity

Input: word w

$G = \{w\}$; // initialize

▷ Method 1

$G.add(stemmize(w))$; // stem of the word

$G.add(lemmzatize(w))$; // lemma of the word

// singular or plural

if *w is plural* **then** $G.add(singular(w))$;

else $G.add(plural(w))$;

▷ Method 2

// add all the affixes and keep the valid forms

foreach *affix in affixes* **do**

  $w' = w.add(affix)$

  if *w' in Dictionary* **then** $G.add(w')$;

▷ Method 3

// Generate abbreviations

foreach *w' in genAbr(w)* **do**

  $G.add(w')$

▷ Method 4

// word forms from conceptNet

foreach *w' in wordForms(w)* **do**

  $G.add(w')$

Output: Generated family G

Algorithm 2 shows the process for generating different variations.

	English				German			
Model	#Families	Precision	Recall	F-Score	#Families	Precision	Recall	F-Score
Proposal	1716	0.32	0.79	0.40	1590	0.30	0.75	0.38
Baseline	1716	0.06	0.53	0.11	911	0.11	0.37	0.17

Table 3.2 – Quality evaluation of generated entity families.

3.1.4 Evaluation

Table 3.2 shows the results for the proposed family generation methods. We conducted an experiment to evaluate our method for English and German. The results are compared with a ground truth extracted by KPMG experts. For each representative, we compared the family generated by our algorithm with the family in the ground truth. Table 3.2 reports the performance of our model in terms of Precision, Recall, and F-score averaged over all the families. For both languages, we could generate more than 1500 families and the generated families have high quality in terms of recall measure with respect to the ground truth. We also compared our results with families generated by utilizing external sources. For the English dataset, we report the union

of different forms of the representative words form *WordNet* and *ConceptNet* as the baseline and for the German dataset, as *WordNet* does not support the language, we use *ConceptNet*. The results show that our proposal outperforms the baseline in both of the languages.

Results show that our method covers a part of the variations in the ground truth, but some variants are not covered as we lack the patterns to generate them. We could increase the precision by removing some patterns or making the patterns more precise, but, since the new patterns are not as common as the current patterns, it will decrease the recall. In our case, we decided to choose the high recall scenario and this was motivated by KPMG's preference of having a large pool and select among them, instead of having few and then write variations. The generated families will be evaluated by the experts and then added as nodes to the knowledge graph.

3.2 Matching Text and Data

In data integration, matching records referring to the same real world object is an important task, usually referred to as *entity resolution* (**ER**) [45, 47, 102]. In other communities, such as in Natural Language Processing (NLP), *text matching* (**TM**) is also a widespread task in many applications, such as question answering [103] and information retrieval [104]. However, in many scenarios the borders between the two tasks are not clearly defined. Several datasets have long textual cell values, such as product descriptions. Text documents have structural properties and content organized in hierarchies. Finally, and at the center of our attention, some applications *match textual content to structured data*, such as relational tuples [105, 106]. In an unsupervised setting, where no training data is available, it is not clear what is the right solution to tackle this more generic matching problem. Consider the following examples.

Example 2: Text and relational data. A corpus of product reviews is gathered from the Web. A company must link tuples in a relation to these reviews in planning a promotional campaign. However, the product reviews have no identifier (Figure 3.3).

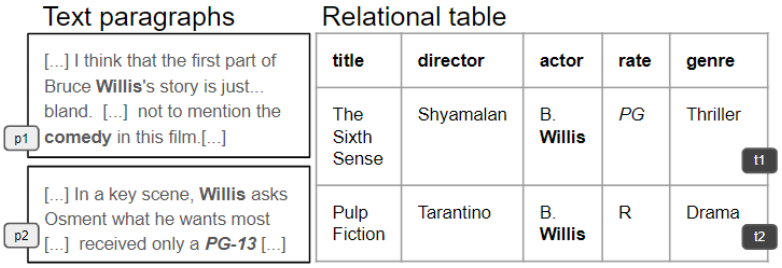


Figure 3.3 – Text and data: paragraph $p1$ matches tuple $t2$.

Example 3: Structured texts. An enterprise manual describing auditing processes is hard to navigate for the final users. To support search, the paragraphs in the manual must be matched to a large taxonomy of concepts (Figure 3.4).

This problem is difficult because matching is based on overlapping (or similar) content in the input objects, but this signal can be missing or ambiguous in this new setting.

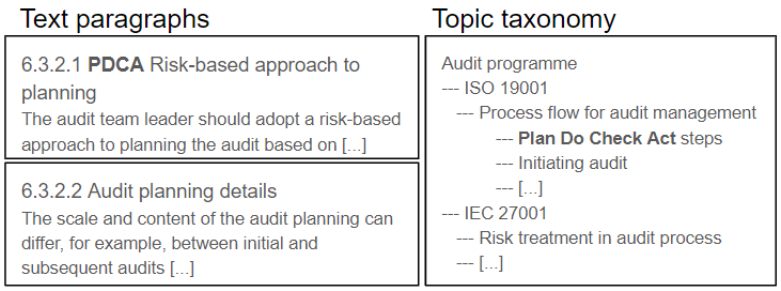


Figure 3.4 – Structured texts: 1^{st} paragraph matches 4^{th} node.

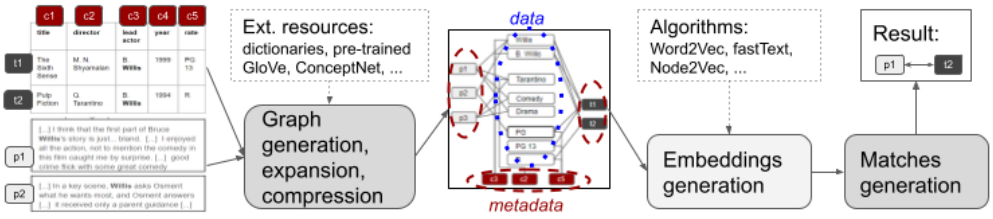


Figure 3.5 – The proposed framework: (1) text and structured data documents are jointly modeled in a graph, (ii) embeddings are produced for data and metadata nodes (representing texts, taxonomy nodes, tuples), (iii) metadata nodes are matched in an unsupervised approach.

Missing matches. In Example 2, movie *Pulp Fiction* is reported as *Drama* in the table but *comedy* is mentioned in the review. This problem can be partially tackled with the pre-trained embedding of *Tarantino*, which models that he is reported as director for both comedies and dramas [107]. But in Example 3, modeling the connection between *PDCA* and its full spelling is crucial to match the paragraph to the right taxonomy node. While pre-trained embeddings can be used to identify synonyms for common words and popular entities, they fail for *domain-specific* terms as in this example. **Challenge 1**: as specific vocabularies are not well modeled by pre-trained resources, we need to learn embeddings *across the heterogeneous corpora* at hand to discover similarities in their content.

Ambiguous matches. In Example 2, an actor named *Willis* appears in different paragraphs and tuples, but only one tuple in the relation is the correct match in this case. Similarly, the term *audit* appears in most paragraphs and taxonomy nodes in the second example. This suggests the need of a weighting mechanism for combining the matching tokens across two candidate objects. **Challenge 2**: there is the need to learn how to combine matching signals, but the lack of training data rules out solutions based on fine tuning of existing pre-trained models.

Previous methods lack the flexibility to cover such challenges. Existing approaches handle well traditional **ER** and **TM**, but they simply fail in terms of accuracy in the new use cases above.

The best **ER** results are obtained by methods exploiting deep learning techniques [45, 47, 108, 109], but they rely on the presence of a schema (missing in text) and therefore support only relational data to data matching. On the other hand, the advent of language models has enabled important improvements in **TM**, with transformer-based as the current state of the art

solutions [110, 111]. However, transformers are designed to capture the (hidden) relationships in the language and cannot be applied directly to relational data.

What is missing is a unified representation that is at the same time modeling the *relationships in the structured content* (for learning a good representation and identify similarities) and the *importance of every matching word* when comparing heterogeneous objects. The last point highlights the need of learning a comparable representation for sets and sequences of tokens, such as tuples and text documents.

To overcome these issues, we propose a framework for learning representations of data and text that (i) is tailored at the domain at hand with a joint modeling of heterogeneous corpora and (ii) exploits structured information whether available to improve the quality of the generated embeddings and of the matching process.

Figure 3.5 shows our framework, which solves the challenges above. First, it represents text documents and tables as nodes and edges in an undirected graph. This graph contains two main types of nodes. *Data* nodes represent tokens (words) in the corpora, either in text paragraphs or in table cells. *Metadata* nodes represent IDs for tuples, attributes and paragraphs. Graph edges represent the relationship between data and metadata, e.g., a tuple/attribute/paragraph contains the token in a data node. As our goal is to match metadata node, we aim at creating more paths between related nodes and at removing spurious connections. The first goal is achieved in an expansion step that exploits external resources, such as ConceptNet [21]. The second goal is obtained by pruning edges and nodes with a graph compression techniques designed for our matching task.

Next, we generate an embedding for every graph node. We rely on existing solutions for this step and the algorithm at hand can be replaced as the community makes progress in this task. Finally, we use the embeddings for the metadata nodes in an unsupervised algorithm to identify the matching ones, such as the paragraph and the tuple in the first example.

The framework enables users to improve the solution according to the requirements and the resources at hand. If relevant external resources exist, such as word dictionaries, they can be plugged in graph construction to merge data nodes. Knowledge graphs and ontologies can be plugged in the expansion step to find more relationships across metadata nodes. If a new embedding generation algorithm is available, it can be plugged in the second step to improve the quality of the embeddings.

Our proposal outperforms state-of-the-art methods by increasing quality performance up to 45% in absolute terms while taking a fraction of their time in matching. Finally, while we focus on unsupervised applications, any downstream classifier can be trained using the labeled data and the embeddings from our solution.

In the following, we first describe the proposed framework and discuss how we generate and refine the graph at the core of the proposal. Then we introduce algorithms to expand the original graph with external resources and compress it to keep its size manageable. We also present the methods for producing embeddings and matching metadata nodes. Finally we evaluate our work with datasets from real applications.

3.2.1 Related Work

Entity resolution for relational data [102] has been recently studied with deep learning solutions [45, 47, 108]. As they rely on the presence of a schema, one way to use them in this setting is to treat the text paragraphs as tuples within a single column, but this leads to poor results (more details in the supplement). In this work, we extend previous proposals with a graph and corresponding optimizations that model texts and tables in a unified representation for learning embeddings, with clear benefit in the matching tasks even with very lightweight algorithms. We model text matching both as a binary classification task and as a multi-label classification task. This enables us to use SOTA baselines based on fine tuning language models [112, 113]. These methods outperform traditional IR approaches, such as BM25 but do not focus on the problem of matching text and relational datasets. This latter problem has been studied in settings that do not cover our use cases, either because they assume supervision or because they are no domain-specific [105, 106, 114]. Other approaches for text and data matching assume a very expensive training over large document corpora and millions of tables [115, 116]. We do not compare against these methods as we target a setting where one (possibly small) corpus and one dataset are given. We show that good results, outperforming supervised methods based on language models, are obtained in seconds on a basic laptop. Our setting is also different from the problem of entity linking, as we are matching text to tuples in relational db and not to a knowledge graph [117].

Our default method to generate data and metadata representations is *Word2Vec* [87]. We confirm previous studies showing that it is powerful in discovering relationships in the corpus as well as similarity between tokens [118]. While we generate embeddings, other baselines in our study use pre-trained ones [107]. We report also methods that are based on (pre-trained) contextualized word embedding methods based on transformers, such as BERT [110]. Document embedding methods extend this line of work to longer text sequences by using functions to aggregate the vectors of words in the given sentence or paragraph [119, 120] or by learning the document vector with special tokens [113, 121].

Given our graph, it is also possible to generate embeddings directly for its nodes [122–124]. It has been shown that most of these methods lead to comparable results w.r.t. the random walks followed by word embedding generation for a data to data matching task [45]. Our study confirms that they do not bring clear benefit in our setting, but are more resources intensive than *Word2Vec*. Finally, our work can be seen as a new instance of the recent general approaches of using deep learning for data integration [46, 125, 126] and of improving pre-trained embeddings w.r.t. relational data [127].

3.2.2 A Graph for Heterogeneous Corpora

In this section, we describe the unsupervised algorithms for the generation of a graph across heterogeneous corpora.

We discuss the most general use case with text documents and tables as corpora, but the same algorithms apply for the case with documents (tables) only. We remark that external resources, such as pre-trained embeddings, are not needed to run the pipeline, but they can be naturally exploited as we discuss next.

The input of the graph creation are two corpora. A *corpus*, based on the task, is a table, some

structured text, or simple text. The *document* to match is a tuple, for tables, while the granularity of the text is user-defined and can span from a single sentence to a paragraph. These corpora are matched in three possible combinations, or *tasks*: text to structured text matching, text to data matching, or text to text matching. The purpose of each task is to find the top-k closest documents in the second corpus for all the documents in the first corpus.

We jointly represent the document corpora and tables in a graph, from which we then generate embeddings. We first perform some pre-processing steps for every corpus. This includes stop-words removal and stemming on the tokens coming from the texts and the cell values of the tables. We call *terms* these processed values and a term can be composed of one or multiple tokens. For example, “The Sixth Sense" is a term composed of three tokens.

We define two types of nodes. *Data nodes* represent the terms after the pre-processing. If a term is contained in multiple documents across the corpora, it still appears as a single node in the graph. *Metadata nodes* represent a group of tokens, such as a sentence, a tuple, or an attribute. Undirected and unweighted edges connect metadata with the respective data nodes, i.e., a tuple node is connected to the tokens that it contains.

The graph creation is presented in Algorithm 3. The algorithm takes as input two corpora of pre-processed documents. It creates a metadata node for each document in the first corpus (lines 3-4). For example, Figure 3.6 shows metadata nodes *t1* and *t2* for the tuples. If the document is a table, it also creates a metadata node for every attribute (lines 5-10), such as nodes *c2*, *c3*, *c4*, and *c5*. If the document is a structured text, its nodes are modeled as metadata nodes and edges are added to represent relations (lines 12-15). For each term associated to the document (metadata) node, the algorithm then creates term nodes (lines 18-20) and connects each data node to its respective metadata node (lines 21-24). For example, edges are created to connect *t1* to *Shyamalan*, *Willis*, *B._Willis*, *PG*, and *Thriller*. Next, metadata nodes for the documents in the second corpus are created (lines 27-28), thus adding *p1* and *p2* in Figure 3.6. Term nodes for these documents are connected to their metadata nodes (lines 29-34). For example, metadata node *p1* is connected to data nodes *Willis* and *Comedy*.

Algorithm 3 creates different metadata nodes based on the input documents. In this example, it outputs metadata nodes to represent tuple, columns, and text. For the other two tasks (text to text and text to structured text), only text metadata nodes are produced. Figure 3.7 shows the generated graph after applying Algorithm 3 on two paragraphs and two taxonomy concepts (*Process flow for audit management* $\rightarrow$ *Plan Do Check Act steps* and *Risk treatment in audit process*) of our example from auditing domain (Figure 3.4). Here, by starting the process from the taxonomy set we were able to filter a big portion of the tokens in the paragraphs. The graph shows that the first paragraph ($p1$) is equally close to both of the taxonomies while the second paragraph ($p2$) is closer to the first taxonomy concept.

Connecting metadata nodes

The graph creation algorithm never connects metadata nodes from different corpora, as we assume that these connections are hard to infer and are indeed the results of the downstream task in our system. However, metadata text nodes from the same structured document can be connected. For example, for the taxonomy in Figure 3.4, the corresponding graph connects metadata text nodes for *Audit programme* and *ISO 19001*. This edge represents the hierarchical relation between the

Algorithm 3: Graph Creation

```
1  Input. Two sets of documents;
2  G = An un-directed graph;
3  foreach document doc_i in the first set do
4      G.addNode(doc_i) ;
5      if get_type(first set)=table then
6          foreach column col_j in the document do
7              if ¬ G.hasNode(col_j) then
8                  G.addNode(col_j)
9              end
10         end
11     end
12     if get_type(first set)=structured then
13         parent ← get parent of document i ;
14         if G.hasNode(parent) then
15             G.addEdge(doc_i,parent)
16         end
17     end
18     terms ← get list of terms in document i ;
19     foreach term tm_k in terms do
20         G.addNode(tm_k)
21         G.addEdge(doc_i,tm_k)
22         if get_type(set)=table then
23             G.addEdge(col_j,tm_k)
24         end
25     end
26 end
27 foreach document doc_i in the second set do
28     G.addNode(doc_i);
29     terms ← get list of terms in the document;
30     foreach term tm_j in terms do
31         if G.hasNode(tm_j) then
32             G.addEdge(doc_i,tm_j)
33         end
34     end
35 end
36 Output. Graph G
```

concepts.

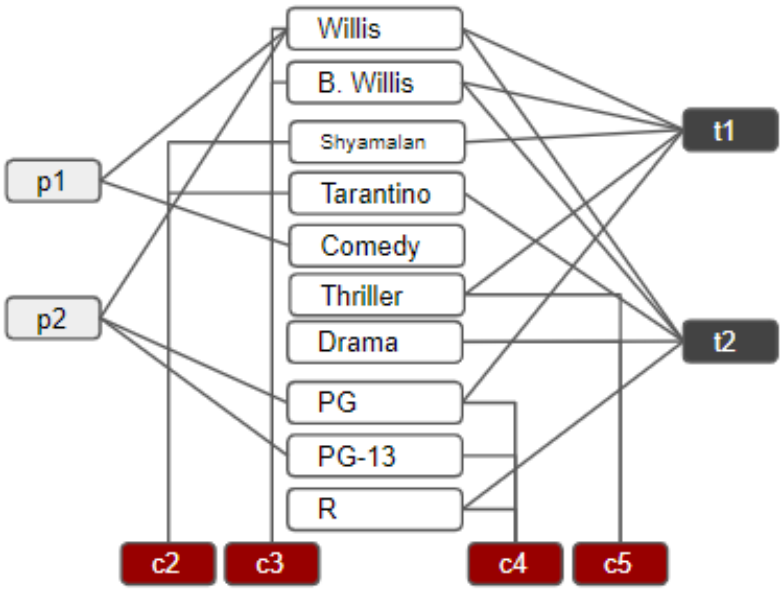


Figure 3.6 – Graph with a sample of the nodes for Example 2

Filtering nodes

The graph can become extremely large with real text corpora. This is a problem in terms of performance both for the execution time and for the quality, as it leads to the modeling of a lot of terms that do not contribute to the final matching tasks. To address this problem, we filter out irrelevant terms in the graph creation. Algorithm 3 does not create data nodes for all the terms in both corpora. It starts by creating term nodes for documents in the corpus with a smaller number of distinct tokens and filters out terms from the second corpus that are not in the graph.

As our goal is to model the connections across the two corpora, we compromise the loss of some words (and possibly relationships) in the second corpus to focus the learning in the next step on the terms that create bridges between metadata nodes. To limit the loss of possibly relevant words and to create a more compact graph, we present next some techniques to merge token nodes across corpora.

Merging nodes

Intuitively, (correctly) merging data nodes increases the connectivity between related metadata nodes across corpora and ultimately improves the matching tasks. For example, merging data nodes *Bruce Willis* and *B. Willis* in Figure 3.3 decreases the distance in many paths connecting metadata nodes *P1* and *t1*. Merging data nodes is easier that solving the metadata matching task and there are several resources available for this operation. For this matter, we use different techniques to merge data nodes:

- *Stemming* merges different forms of a word. For example, in Figure 3.4, stemming merges *planning* from the first paragraph with node *Plan* from *Plan Do Check Act Steps*.

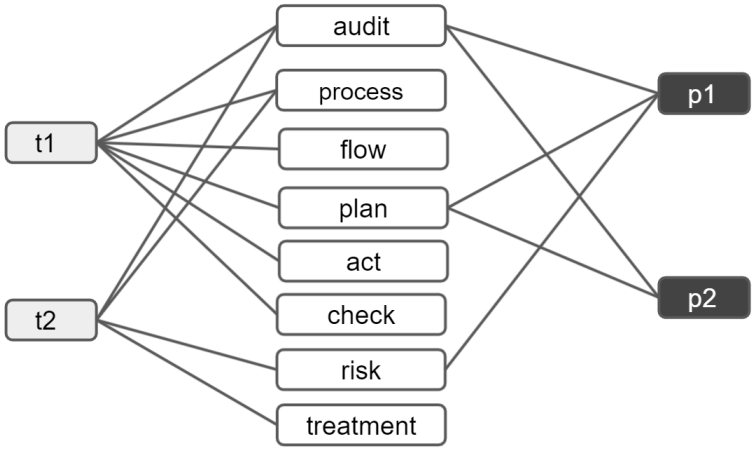


Figure 3.7 – Graph with a sample of the nodes for Example 3

- *Bucketing* (or binning) is a pre-processing technique to improve the performance in ML. We merge data nodes with numeric values by using equal width binning and the *Freedman–Diaconis* rule [128] to compute buckets' width.

- For merging synonyms, acronyms, and typos we use external resources such as pre-trained word embedding models.

For the last case, we find the nodes that can be merged by computing the cosine similarity between their embedding vectors. If the similarity between two nodes is higher than a threshold γ. For calculating γ, we use a list of 17K synonym terms from *WordNet* [129] and define γ as the average cosine similarity between their vectors in the pre-trained model that we use for merging. Specifically, for Wikipedia2Vec [130] we identify and set $\gamma = 0.57$. This approach is widely used in tasks such as entity linking [117] and retrieval [131].

Tokens and terms

One important aspect in graph creation is handling multi-tokens data nodes. There are a lot of meaningful multi-tokens words in document (e.g., movie names) and information is lost if they are split over different single-word data nodes in the graph, e.g., *The Sixth Sense* split over data nodes *Sixth* and *Sense* with *The* filtered out as stop word. A possible solution for tables is to represent the whole cell value as a single data node (*The_Sixth_Sense*). But this granularity has also drawbacks as it may lead to graphs that miss important connections across corpora. For example, if *B. Willis* in the review and node *Willis* are not merged, a strong connection between the correct metadata nodes is lost.

We use a combination of the two approaches that solves both problems. For each text in a corpus, we generate possible n-gram tokens for $n = 1, \ldots, n$. For example, for $n = 3$, the graph represents *The Sixth Sense* using five data nodes: *Six*, *Sense*, *The_Six*, *Six_Sense*, and *The_Six_Sense*. This increases the chance of connecting terms of the second corpus with nodes generated from the first corpus. We identify the value of n for every scenario. To set n, we

profile a dump file of titles of *Wikipedia* articles. About 99% of the titles have at most three tokens. This value is supported by our experimental analysis and by the literature. Experiments in Appendix A.1.1 show that increasing n up to three improves the quality performance but there are diminishing with higher values. Other results also show that by increasing n in character tokenization, the lexicon size grows rapidly and precision diminishes for most languages [132].

3.2.3 Graph Expansion and Compression

By generating the graph with Algorithm 3, we model the relationships between metadata nodes. The edges and paths in the graph represent such relations which are present in the documents/relations. The network of connection leads to embeddings that ultimately guide the metadata matching process. However, the data relationships are not all the existing relations between two real objects represented in the graph. Real entities and concepts are connected by other relationships that are missing from the corpus at hand.

For example, an actor and a director may have worked together in a movie that is not contained in the movie table or in any of the reviews. Being able to add such connection to the graph may lead to embeddings that are closer to the reality.

Such external relationships can be very valuable if represented in our graph as they lead to better embeddings and ultimately enable better matching. However, while expanding the graph is a valid solution to include external information, we should be careful in trying to remove useless or even misleading new nodes and edges in order to keep the graph as little as possible in terms of size. We present solutions to address these two tasks in this section.

Expanding the graph with external information

A natural approach to expand the generated graph is to employ external resources, such as existing ontologies and knowledge bases. By exploiting external resources, we can find new information about the nodes in the graph. This information can be added as new nodes and new edges in order to enrich the generated graph.

For example, in Figure 3.6, *p1* is the review related to tuple *t2*. Even though there are seven paths between these two metadata, only one of them has three or less nodes: p1 $\rightarrow$ Willis $\rightarrow$ t2. By expanding this graph with new nodes and edges, we can add new meaningful paths between these nodes, improve their embeddings, and increase their chance of being matched. Consider as a resource of external information the knowledge graph DBpedia [4]. Among the relations for entity *Tarantino* in DBpedia, there is the following triple: *style(Tarantino, Comedy)*. Adding this new edge to the graph creates nine new path between *p1* and *t2* including one with less than three nodes: p1 $\rightarrow$ Comedy $\rightarrow$ Tarantino $\rightarrow$ t2.

Different external resources can be exploited in order to expand a graph. In graphs which contain named entities, we are interested in finding new information about those named entities, such as data about their spouse, country, university, workplace, etc. This information can be extracted from existing entity-centric knowledge bases such as *DBpedia* and *Wikidata*. For example, the graph presented in Figure 3.6 contains information about movies and their casts, those are entities for which we can use a knowledge base for expansion. By expanding this graph with DBpedia, we enrich it with new edges such as *starringOf(Willis, Pulp Fiction)*, *spouse(Shyamalan, Bhavna Vaswani)*. Some of this information are shown in Figure 3.8.

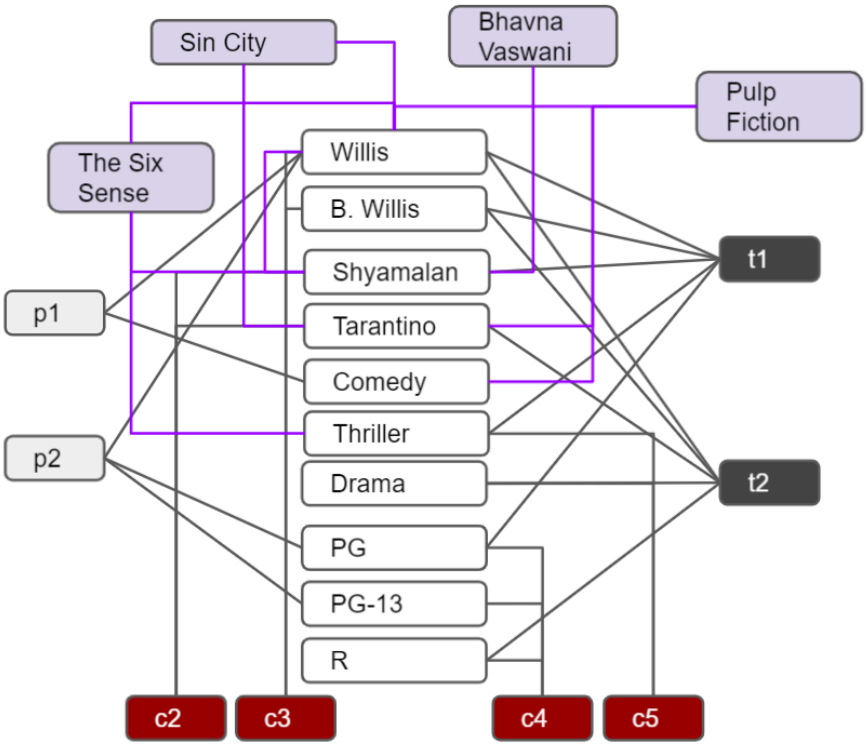


Figure 3.8 – Expanded graph for Example 1.

Textual corpora do not contain only named entities, but also concepts, generic nouns and verbs. For example, by expanding the word *management* in Figure 3.4, we connect it with the relevant words in the correct text paragraph, such as *planning*. For expanding the graph in these cases, other external resources can be exploited. A wide range of external resources such as *ConceptNet* [21] and *Wordnet* can be employed for text nodes expansion.

Algorithm 4 shows how we exploit any external resource to fetch all connections for every data node in the graph. We also remove any sink node, i.e., nodes that are not connected to more than one other node. E.g., in Figure 3.8 node *Bhavna Vaswani* is only connected to *Shyamalan* and can be removed.

Using this expansion technique, we introduce a lot of new paths in the graph and these paths affect the matching process between metadata nodes. For example, in the graph in Figure 3.6, there is only one path with less than five nodes between metadata nodes *p1* and *t2*, but the shortest path between such nodes is only of size two after expansion.

Pruning nodes and edges for compression

Expansion introduces correct connections between related nodes, but it also increases its size by adding new nodes and edges that are not helpful needed for our tasks. For example, there are more than 800 relations for entity *Quentin Tarantino* in DBpedia but only a few of them increase the chance of matching *p1* and *t2* (e.g., *directorOf(Quentin Tarantino, Pulp Fiction)*, *redirectsOf(Quentin Tarantino, Samuel Jackson)*).

Algorithm 4: Graph Expansion Algorithm

```
   Input: Un-directed graph G
1          External resource E
   // Expanding by fetching connections from E
2  foreach node in G do
3      if node is not a metadata node then
4          relations ← all connections of node in E;
5          foreach (node,m) in relations do
6              if ¬ G.hasNode(m) then
7                  G.addNode(m)
8              end
9              G.addEdge(node,m)
10         end
11     end
12 end
   // Cleaning the graph
13 foreach node in G do
14     if degree(node) == 1 then
15         G.removeNode(node)
16     end
17 end
18 Output. Expanded graph G
```

Moreover, as adding new nodes and relationships increases the execution time for random walks and embedding generation, we should avoid keeping nodes and edges that do not contribute to the connections among metadata nodes.

For these matters, we introduce a graph compression techniques to reduce the size of our graph after the expansion phase. Compression for static graphs has been studied for long time due to its benefits in terms of reduction of data volume and storage, which in turn enable speedup of algorithms and queries [133]. Noise elimination has also been reported as an important effect of the compression, with the removal of erroneous nodes and labels [134]. Compression methods can be based on node sampling [133, 135, 136], edge sampling [137, 138], or exploration based sampling [139–141]. Most methods are configurable w.r.t. the desired compression ratio, i.e., the desired size of the output graph compared to the input graph. .

As these methods are very general, they are not application specific and cannot make use of the node types in our graph. Our experiments in Section 3.2.6 show that these techniques can help in reducing the size of the graph by filtering nodes, but do not guarantee pruning and good performance in the matching task at the same time.

Our key observation is that the goal of our graph and our embeddings it to match metadata nodes. A crucial component in determining the distance between the embeddings for two metadata nodes in their distance in the graph. We therefore start the design of our compression algorithm from the idea that it should preserve the shortest path across all the metadata nodes in the two

corpora. This is a quadratic number of paths w.r.t. the number of metadata nodes, but this is unavoidable as we do not know at compression time what are the metadata to match.

Inspired by an existing graph compression technique that exploits shortest paths, namely *SSP* [140], we introduce an algorithm tailored at our graph and matching application. The original *SSP* is an exploration based sampling method which takes a sampling size as input. It randomly picks a pair of nodes in each iteration, computes their shortest path, and adds nodes and edges of the shortest path to the output graph. In our setting, the idea is to use metadata nodes in distinct corpora for the selection of node pairs. This guarantees that metadata nodes are connected and keeps in the graph the data nodes that are modeling their relationship concisely.

Algorithm 5: Graph Compression MSP

```
   Input: Un-directed graph G
1            Compression ratio β  CG = empty un-directed graph
2  i = 0
3  L = β * size(G.nodes())
4  while i < L do
       // Select two random metadata nodes
5      first ← a random node from the first corpus
6      second ← a random node from the second corpus
7      shortest_paths ← find all shortest paths between first and second in G
       // Add nodes and edges of the paths to CG
8      foreach path in shortest_paths do
9          CG.add(path)
10     end
11     i += 1
12 end
13 Output. Compressed graph CG
```

Algorithm 5 shows our graph compression based on the idea of using Metadata Shortest Path (*MSP*). It takes an un-directed graph and a compression ratio β as input and returns a compressed graph. We define the number of iterations of the Algorithm by multiplying β and the number of nodes in the graph. We also make sure that all metadata nodes, even if not sampled at lines 6-7, are connected to graph with at least one shortest path.

There is however an orthogonal challenge in such an aggressive compression. Keeping only shortest paths among metadata nodes may lead to a graph with similar embeddings for all nodes. Assume that all shortest paths are of length three and all pivot on a single, very popular data node. This extreme situation leads to all metadata nodes ending up with the same embedding, therefore making the matching process impossible. This observation highlights that compression is not suitable in all cases. We discuss in the experiments how we can recognize setting that are less likely to benefit from the compression.

3.2.4 Matching Text and Structured Data

In this section, we first describe present how to generate the embeddings for the unsupervised matching of the objects across the corpora.

3.2.5 Embeddings Generation

We generate embeddings for the graph nodes. Multiple methods can be employed to generate embeddings directly from the graph [123, 142]. A less resource intensive solution is to use word embedding models on walks over the graph *Node2Vec* [124]. In our default setting, we use the second approach as we found the results with different methods comparable in quality, but the latter is faster and less demanding in terms of computing resources.

In our default method, a random walk starts from every graph node and at each step it randomly chooses the next node among the current node's neighbors. A sentence is derived with the concatenation of nodes traversed by the walk. The union of the sentences is then processed with *Word2Vec* or similar method. Multiple parameters of the random walks affect the quality of the embeddings, including whether random walk should be limited to some nodes or not, the length of a random walk, and how many random walks should be generated for every graph node. In Appendix A.1.1, we report the impact of these parameters on the quality of the proposed model.

We stress the importance of the unified graph and of the walks in our context. While any data structure can be serialized as a sequence of sentences (e.g., row by row), this is not effective in practice for learning embeddings for two reasons. First, the resulting sentences do not follow meaningful patterns as in real languages; the relationships that are nicely captured in real text are missing. This is especially true for existing transformed based solutions based on attention. Second, a simple serialization misses the structural dependencies in a relation; existing features in the data are not exploited. Given text documents and a relation, our graph and walks enable the joint representation for the given corpora.

Matching Metadata Nodes

In the final step, the input of the matching module is a metadata node for a document in the first corpus and the metadata nodes representing all documents in the second corpus. The embedding vectors are used to match such nodes and ultimately the documents they represent (text paragraphs, text sentences, or tuples). The distance between two nodes' vectors is used for matching them. Given the embeddings, we use cosine similarity to identify the top-k neighbours in the second corpus for the metadata node from the first corpus.

Differently from the graph creation, in the metadata matching we found more effective to start the process from the larger corpus and this is our default configuration. However, this decision can be changed according to the specific task. For example, in text matching we do it claim by claim (from the smaller corpus) as this is the natural setting for the final application.

3.2.6 Experiments

We first introduce our execution setting and the baseline methods. We then report the results in three matching tasks: i) text to data, ii) text to structured text, iii) text to text. We do not

report performance for the data to data task since it has already been studied in [45]. Finally, we report execution times and discuss the impact of the different parameters and optimizations in our solution[4].

Execution setting. Experiments have been conducted on a laptop with CPU Intel i5-7300U, 4x2.6GHz cores and 8GB RAM. For fine tuning the baselines, we used a *Google Colaboratory* instance with CPU Intel 2x2.20GHz, 13GB RAM, and NVIDIA Tesla T4 GPU (16GB memory). Algorithms are written in *Python* with the *Numba* compiler.

Baselines. We compare our approach to 8 baselines. While our method handles well the three tasks, we report only the best performing baselines for every task. Three baselines rely on *training* over the given documents and data (as our approach), while the rest use *pre-trained* resources. We further distinguish unsupervised solution and supervised ones. All unsupervised methods match objects with the algorithm in Section 3.2.5.

For unsupervised methods based on *training*, we test *Word2Vec* (W2VEC) for word embedding, plus *Doc2Vec* (D2VEC) for document embedding. We obtain embeddings from the documents at hand and then use such embeddings to identify matches for every document. As common in the literature [120], we generate embeddings for longer texts with the mean of the vectors of their tokens. We use vectors of size 300, Skip-Gram for *Word2Vec* and DBOW for *Doc2Vec*. For unsupervised approaches using *pre-trained* embeddings, we report on *SentenceBERT* (S-BE), as it does better than other popular embeddings, such as GloVe [107].

We denote supervised methods using pre-trained models with * for clarity and always report results for 5-fold cross validation. The first approach is based on fine-tuning for a multi-label classification task on *BERT* large. However, for text to data the number of possible labels is too large to successfully train a classifier, so we also fine tune a binary classifier (S-BE*). We use S-BE to encode sentences, compute the cosine similarity score between them, and use such scores to train a binary classifier to predict whether a match is true or false. We also report the results for two supervised state-of-the-art entity matching methods for the text to data task: *Ditto* [47] and *DeepMatcher* [109]. These methods take two tables as input and compute the matching probability for tuples from different tables. We represent text documents as tuples of a table with one attribute. We use 60% of the annotated data to train these models. Finally, we report for *Reranking* (RANK*), a supervised algorithm that learns to rank using a pairwise loss [112].

For our unsupervised approach, we used *Word2vec* (W-RW) on the random walks (RW) generated on the graph. In the default configuration, we generate 100 random walks of length 30 for every node. For the text to data task, we use *Skip-gram* with a window of size three as in the data to data match [45], while for text oriented tasks we use *CBOW* with a window of size 15. We report for our method with ((W-RW-EX)) and without ((W-RW)) applying the expansion technique in Section 3.2.3. We use ConceptNet as our default external resource for expanding graphs, except for *IMDB* where we employed DBpedia as this relation contains mostly entities.

[4]Code and all datasets, except the KPMG one, are available online at https://github.com/naserahmadi/TDmatch.

Table 3.3 – Quality of match results for *IMDb* scenario.

	Method	MRR	MAP@k			HasPositive@k		
			1	5	20	1	5	20
WT	S-BE	.254	.088	.142	.159	.171	.339	.510
	W-RW	.853	.400	.678	.682	.802	.919	.942
	W-RW-EX	**.868**	**.410**	**.691**	**.706**	**.820**	**.926**	**.955**
	S-BE*	.287	.103	.164	.183	.205	.363	.555
	RANK*	.535	.218	.351	.376	.438	.645	.797
	DITTO*	.759	.349	.549	.553	.699	.839	.877
NT	S-BE	.218	.067	.118	.139	.136	.301	.454
	W-RW	.780	.362	.574	.589	.727	.841	.906
	W-RW-EX	**.792**	**.371**	**.587**	**.598**	**.749**	**.854**	**.911**
	S-BE*	.233	.073	.125	.142	.146	.318	.489
	RANK*	.404	.156	.236	.260	.312	.494	.688
	DITTO*	.560	.265	.386	.410	.428	.689	.814

Text to Data

For the *text to data* matching we use two datasets. We created a first scenario from the Internet Movie Database (*IMDb*) website with a corpus of movies reviews and a database of movies. We also report results for the *CoronaCheck* scenario, which matches COVID-19 claims to the official datasets [106]. For both scenarios, the task is to find tuples related to each sentence. For example, a sentence "Number of cases in US is higher than China" required to match two rows of a table to verify the claim.

Datasets. As the task is novel, we release two new scenarios:

1. IMDb. We created the dataset by manually matching two reviews for every movie in "top 1K of all times" to a sample of 50k tuples from the official IMDb dataset. The 2k reviews contain one to 207 sentences, sixteen on average. We created two versions of the target relation: an easier one with 13 attributes, including the title information (**WT**) and a more challenging one without title (**NT**).

2. CoronaCheck. This scenario contains a corpus of sentences about COVID-19 spread and effects, such as daily total death cases and new confirmed monthly cases, annotated w.r.t. the corresponding tuples in a dataset with 1.2k tuples about daily cases for all countries. We report for a dataset with 7k sentences created from the data (**Gen**) and a more challenging dataset with 50 sentences submitted by users on the website `https://coronacheck.eurecom.fr` (**Usr**) [106].

Evaluation Measures. Mean Reciprocal Rank (*MRR*) is the average of reciprocal ranks of queries, i.e., the multiplicative inverse of the rank of the first correct answer. Mean Average Precision (MAP) is the mean of the precision scores after each relevant document is retrieved and we report MAP truncated at rank k (*MAP@k*). We also report HasPositive@k for determining whether there is a true positive among the top-k results.

Table 3.4 – Quality of match results for *CoronaCheck* scenario.

	Method	MRR	MAP@k			HasPositive@k		
			1	5	20	1	5	20
Gen	S-BE	.486	.294	.463	.483	.295	.752	.916
	W-RW	.728	.575	.718	.725	.578	.945	.995
	W-RW-EX	**.755**	**.601**	**.746**	**.752**	**.611**	**.959**	**.996**
	S-BE*	.550	.372	.531	.545	.363	.811	.947
	RANK*	.460	.287	.438	.455	.289	.703	.845
	DEEP-M*	.376	.347	.368	.374	.349	.395	.439
	DITTO*	.160	.030	.161	.203	.066	.283	.518
Usr	S-BE	.354	.177	.284	.320	.200	.620	.860
	W-RW	.518	.296	.427	.472	.306	.755	.979
	W-RW-EX	**.538**	**.329**	**.451**	**.496**	**.371**	**.771**	**1**
	S-BE*	.397	.263	.342	.366	.260	.640	.820
	RANK*	.332	.137	.256	.303	.160	.600	.880
	DEEP-M*	.321	.200	.200	.248	.280	.280	.600
	DITTO*	.153	.020	.100	.123	.040	.s281	.407

Matching results. As the training-based methods (W2VEC, D2VEC) do not take tables as input, we serialize every tuple to a sentence using two special tokens (*[COL]* and *[VAL]*) [47]. For example, the first row in Figure 3.3 starts with "[COL] title [VAL] The Sixth Sense [COL] director [VAL] Shyamalan". We then generate an embedding vector for every resulting sentence and match vectors for tuple and text metadata nodes. As results are poor for these baselines, we do not report them. On the other hand, these lightweight word embeddings methods on our walks lead to good results with our walks. For the pre-trained models, we report for S-BE, S-BE*, RANK*, DITTO*, and DEEP-M*.

Table 3.3 and Table 3.4 show the results on the **IMDb** and **CoronaCheck** scenarios, respectively. Our method outperforms unsupervised S-BE in all scenarios and applying techniques presented in Section 3.2.3 has a positive effect in both datasets. In **IMDB**, we used DBpedia as external resource as it is better in extracting new relations for named entities; for **CoronaCheck** we used ConceptNet. For **IMDb**, we observe an absolute increase of 0.45 for MRR with W-RW in both datasets and at least 410x relative improvement for Positive@1. In **CoronaCheck**, the increase for **Gen** sentences is an absolute 0.24 for MRR and up to 0.30 for MAP@k and Positive@k. For the **Usr** sentences, increases are up to 0.2 for MRR and MAP and up to 0.18 for Positive@k. Our model clearly outperforms also supervised methods. Results show that pre-trained models fail short in this task and that the joint modeling enabled by our graph is needed to achieve good matches. We do not report *DeepMatcher* on *IMDB* because it failed due to the limited amount of memory in our machine.

Text to Structured Text

In this task, we match taxonomy elements to a text document in a real enterprise scenario from an auditing company.

Dataset. This scenario contains 1622 audit text documents (containing one to 17 sentences, three on average) and a taxonomy containing 747 auditing concepts. Each path spans multiple nodes, e.g., $r_1 : a \rightarrow b \rightarrow c \rightarrow d$, where each variable is a concept. Right arrows show the hierarchical relations between terms, e.g., in r_1 c is a child of b and b is a child of a. The length of taxonomy paths are between two and five nodes (four on average). The final graph has 5.9k nodes and 164k edges. Text documents are manually matched to concept nodes by domain experts. About 40% of documents are annotated with one concept, 10% are matched to two concepts, and the rest are matched with three to 27 concepts (four on average).

Evaluation Measures. For this task, we change quality measures as we show results at different granularity. We report Precision, Recall and F-score for concepts (in the taxonomy) assigned to every document w.r.t. the ground truth. As different taxonomy nodes can contain the same text, we compare the root to node path in the measures. With **Exact** matches, we consider a match in the top-k valid only if it is *equal* to the path in the ground truth. As two paths can be partially overlapping, we consider also partial matches with the **Node** score, which measures the intersection between the matched path(s) and the closest path(s) in the ground truth. For an accurate calculation, we exclude two most general levels of the taxonomy (root and first level under it) in the intersection and denote the new path with p'. We then use formula (3.1) below to calculate the **Node** score for two paths p_1 and p_2.

$$Node(p_1, p_2) = \frac{intersection((nodes(p'_1), nodes(p'_2))}{maximum((nodes(p'_1), nodes(p'_2))} \tag{3.1}$$

Consider r_1 and $r_2 : a \rightarrow b \rightarrow c$. After excluding the general nodes, we obtain $r_1 : c$ and $r_2 : c \rightarrow d$, thus Node(r_1,r_2) = 0.5.

Matching results. Table 3.5 reports for both measures the precision, recall and F-score for matching top-k paths to every document for different k values. Results show that the task is very difficult. Indeed, different auditors have different opinions about the right matches for a given taxonomy node and the ground truth is constructed after a discussion to reach consensus. In this hard task, our methods outperform unsupervised methods with a large margin. This scenario contains some domain-specific terms that are not covered by pre-trained models as we can observe by D2VEC (trained on the audit data) outperforming unsupervised S-BE. Only for the top-1 case the supervised *BERT_large* shows small margins for both measures. Supervised classifiers are effective for documents matched against one concept but do not have enough training data for the other cases. In Appendix A.1.2, we show how our model combined with S-BE outperform supervised *BERT_large* solution also for k=1.

Text to Text

We evaluate our framework in matching documents between two text corpora. While our solution is tailored towards structured data and text, we report its results as they are better than unsupervised state of the art baselines for this task and close to supervised ones. We use datasets for the task of detecting previously fact-checked claims [112]. In this context, the goal is, given a check-worthy input claim and a set of verified claims, to rank the verified claims that help check the input claim, or a sub-claim in it, above other claims.

Datasets. The *Snopes* dataset contains a set of 1k claims (tweets) and 11k verified claims (facts),

Table 3.5 – Exact and Node scores for structured text matches.

		Exact Scores			Node Scores		
	Method	*P*	*R*	*F*	*P*	*R*	*F*
K=1	D2VEC	.254	.217	.234	.554	.503	.527
	S-BE	.094	.071	.081	.379	.358	.368
	W-RW	.346	.265	.300	.593	.530	.560
	W-RW-EX	**.367**	**.282**	**.319**	**.601**	**.545**	**.572**
K=1	RANK*	.162	.125	.138	.425	.392	.408
	L-BE*	.381	.304	.338	.626	.567	.595
K=3	D2VEC	.176	.386	.242	.485	.564	.521
	S-BE	.065	.014	.088	.362	.431	.393
	W-RW	201	.434	.275	.521	.652	.579
	W-RW-EX	**.214**	**.475**	**.295**	**.528**	**.670**	**.594**
K=3	RANK*	.162	.125	.138	.425	.392	.408
	L-BE*	.183	.417	.254	.487	.678	.566
K=5	D2VEC	.132	.470	.206	.457	.679	.546
	S-BE	.052	.179	.080	.356	.473	.406
	W-RW	.145	.508	.222	.478	.699	.568
	W-RW-EX	**.151**	**.533**	**.236**	**.485**	.719	**.580**
K=5	RANK*	.072	.242	.110	.365	.522	.429
	L-BE*	.135	.508	.213	.446	.740	.556
K=10	D2VEC	.087	.587	.152	.42	.758	.541
	S-BE	.038	.253	.066	.347	.541	.423
	W-RW	.092	.613	.160	.437	.768	.557
	W-RW-EX	**.094**	**.629**	**.164**	**.438**	**.783**	**.562**
K=10	RANK*	.051	.324	.088	.350	.592	.440
	L-BE*	.081	.584	.141	.393	.797	.526

while the *Politifact* dataset contains 768 claims (made by politicians) and 16.6k verified claims (facts). Text documents contain from one to nine sentences in *Snopes* and from one to 11 in *Politifact*. On average they have less than two sentences. We match top$-k$ verified claims (facts) for every claim.

Evaluation Measures. We use Mean Reciprocal Rank (*MRR*), Mean Average Precision at k (*MAP@k*), and HasPositive@k.

Baselines. We use the methods with the best results reported by previous work on these datasets [112], including unsupervised S-BE and supervised RANK.

Matching results. Results in Table 3.6 and Table 3.7

show that our method is the best unsupervised solution, outperforming the best baseline (S-BE) in all measures and scenarios. In this task, our approach sits between the best unsupervised

Table 3.6 – Quality of match results for *Politifact* scenario.

Method	MRR	MAP@k			HasPositive@k		
		1	5	20	1	5	20
S-BE	.395	.354	.372	.382	.362	.417	.496
W-RW	.489	.346	.396	.401	.409	.579	.702
W-RW-EX	**.507**	**.358**	**.406**	**.418**	**.429**	**.600**	**.726**
RANK*	.608	.531	.588	.599	.535	.688	.787

Table 3.7 – Quality of match results for *Snopes* scenario.

Method	MRR	MAP@k			HasPositive@k		
		1	5	20	1	5	20
S-BE	.543	.457	.527	.535	.457	.648	.724
W-RW	.695	.586	.688	.693	.587	.820	.886
W-RW-EX	**.708**	**.613**	**.698**	**.706**	**.614**	**.843**	**.898**
RANK*	.788	.691	.782	.784	.693	.894	.925

baseline and the best supervised method. One explanation is that these datasets contain generic textual claims with common terms, which is the best scenario for pre-trained models trained on very large corpora. Also, long natural language English sentences are nicely modelled by the attention mechanism in transformers. As we discuss in Appendix A.1.2, by combining our embeddings with pre-trained language models, we can improve our performance and get closer to supervised results for *Politifact* and we can even do better for *Snopes*.

Compression Results

We report the performance of the compressing technique introduced in Section 3.2.3. As a baseline technique, we report also for *SSuM*, a state of the art method that employs node merging and edge sparsifying to generate a super-graph as output [143]. Table 3.9 compares the performance of compression methods in terms of *size* (number of nodes and edges) of the compressed graph and of *quality* in the matching task (MRR).

For *MSP*, we report results for iterations equal to half (*MSP (0.5)*) and a quarter (*MSP (0.25)*) of the expanded graph's nodes. *SSuM (0.1)* is set with a compression ratio of 0.9 as this is the value generating the best quality results (MRR) in our experiments.

In terms of size reduction, *MSP (0.25)* is the compression method with the best results in four cases and it is second to *SSuM* only for IMDB. However, it shows an higher decrease in match quality results w.r.t. *MSP (0.5)*, which is the compression method with the best results in all cases. For Corona it even does better than the expanded graph. This dataset contains many numerical values (about 25% of its data nodes in the expanded graph), which are misleading in some cases, as they are more likely to raise spurious connections in the graph. In general, *MSP* performs better than *SSuM*. For higher compression, *MSP (0.25)* produces smaller graphs with better match accuracy in most cases. For *MSP (0.5)*, we observe better quality in the matches in all cases except Audit, and comparable size in the compressed graphs. The results show the

Table 3.8 – Train and test execution times (sec).

Method	Text to data		Structured text		Text to text	
	Train	*Test*	*Train*	*Test*	*Train*	*Test*
W2VEC	13.9	239	3.5	11.82	5.0	107
D2VEC	47.7	17.2	8.5	1.30	14.9	21.95
S-BE	-	2.6	-	1.16	-	7.5
W-RW	152	0.07	207	0.05	189	0.41
RANK*	3206	0.09	3916	0.05	6918	1.2
S-BE*	2863	3.55	3734	2.3	6855	7.4
L-BE*	3616	0.25	3280	0.69	251	2.6
DEEP-M*	3492	0.68	-	-	-	-
DITTO*	34528	2.28	-	-	-	-

Dataset	Original Graph			Expanded Graph			MSP (0.5)			MSP (0.25)			SSuM (0.1)		
	#N	#E	MRR	#N	#E	MRR	#N	#E	MRR	#N	#E	MRR	#N	#E	MRR
IMDB	107k	1m	.780	237k	1.5m	**.792**	82k	887k	.779	75k	840k	.755	**27k**	**540k**	.601
Corona	10k	43k	.728	15k	56k	.755	10k	40k	**.769**	**8.5k**	**32.5k**	.757	10k	33k	.610
Snopes	35k	129k	.695	142k	622k	**.708**	84k	479k	.647	**49k**	**292k**	.586	83k	470k	.590
Politi	24k	168k	.489	62k	317k	**.507**	37k	242k	.500	**33k**	**225k**	.484	52k	257k	.397
Audit	6k	164k	.421	17k	202k	**.452**	7k	161k	.389	**5.5k**	**144k**	.362	14k	150k	.392

Table 3.9 – Compression performance: number of graph nodes (#N) and edges (#E) compared with matching quality MRR.

benefit of considering shortest paths among metadata nodes. For *MSP*, the graph size and the match accuracy follow the expected behavior w.r.t. the compression ratio.

MSP, in both executions, shows its best results for scenarios with at least one relational table. In these cases, the compressed graph is smaller than the original one (and much smaller than the expanded one), with better or very close matching quality. For text-only scenarios, the size reduction is remarkable, and better than *SSuM*, but a significant drop in matching quality can be observed. The conclusion is that the use graph compression depends on the kind of data and the requirements for the target application at hand.

Execution Times.

Table 3.8 reports execution times for all methods averaged over the experiments for every task. For training time, embeddings methods (W2VEC, D2VEC) and transformer-based methods (RANK*, S-BE*, L-BE*) are trained (fine tuned) on a smaller corpus than our method (W-RW). This is because we create 100 walks for each node in the graph, which leads to bigger corpora in general. Due to this difference, *Word2Vec* and *Doc2Vec* are faster than other methods in training, while our method has execution times smaller than those taken to fine tune transformers. S-BE has no training.

We report the average execution time for a single match (test). Our solution is the fastest. Document based methods, like *Doc2Vec*, are faster than word based embedding solutions. This is because for the latter methods we generate vectors for *all* tokens in the document and aggregate

them. Classifiers are faster than document embeddings based matching, but slower than our method. In the training step of our method, expansion and compression take less than 3k seconds in all cases, with the exception of IMDB (by far the largest) with 79k seconds for expansion (with DBpedia) and 51k seconds for compression with *MSP (0.5)*.

3.3 Summary

In this chapter, we first proposed a method for automatic identification of entities and their families. The proposed method first finds representative entities in KPMG taxonomies and then applies four techniques to generate families for each one of them. Our experiments show that proposed method can generate families close to the ground truth in both English and German documents.

We then presented a new generic matching task that allows both structured text documents and relational data. Results show that even lightweight embeddings effectively model the similarity between heterogeneous corpora. Our proposal outperforms in quality and test time all unsupervised baselines and it is competitive to supervised solutions. Our graph expansion always leads to the best matching quality, while compression is effective in reducing the graph size, but it comes with a trade off in the performance of the matching for text-only corpora.

Chapter 4

Mining Expressive Rules in Knowledge Graphs

In this chapter, we study the mining and the applications for two kinds of rules: *(i) positive rules*, which can be executed to add new facts to the KG with the aim of increasing its coverage of the reality; *(ii) negative rules*, which can be used to identify logical inconsistencies in the KG, ultimately detecting incorrect triples. This chapter extends the original RuDiK system [79] with the discovery of type conditional rules holding only for a subset of the elements in the KG, such as politicians or artists; and a more extensive experimental evaluation of the solution.

RuDiK is a system that exploits disk based solutions to enable the mining of a larger search space over KGs. The ability to mine with a more expressive language leads to the discovery of rules with comparisons across elements beyond equality and the mining for graph predicates involving literal values (e.g., birthDate). More patterns enable the identification of a bigger number of both errors and new facts with high accuracy. Such improvements are obtained by building system around three main contributions.

(i) Approximate Rule Mining. RuDiK specifies the problem of robust rule mining over incomplete and erroneous KGs. The input of the mining is a KG predicate and positive and negative examples for it. In terms of output, in contrast to the long lists of rules ranked on a measure of support [28, 41, 144], Section 4.3 gives a definition that identifies a subset of *approximate* rules, i.e., patterns that may not hold over all examples because of missing and incorrect triples in KGs. The solution is then the smallest set of rules covering most of the positive examples and few negative examples.

(ii) Generation of the Examples. The input examples for a predicate strongly affect the quality of the output rules. While positive example are available from the graph, manually creating negative examples is an expensive task. Section 4.4 introduces example generation techniques that are aware of the issues due to inconsistencies and missing data in the KG.

(iii) A Greedy Algorithm for Rule Mining. RuDiK employs a $\log(k)$-approximation algorithm for the rule mining problem, where k is the maximum number of positive examples covered by a rule. The disk-based algorithm, described in Section 4.5, incrementally materializes the KG in the mining process by traversing only paths that can generate promising rules. The resulting low memory footprint allows the mining with a more expressive rule language.

In this Chapter, we propose a method to enable RuDiK to extract more accurate rules. In many cases, rules are correct when defined for a specific geographical area, in a certain time, or for specific types of entities. To capture these more subtle patterns, in Section 4.6 we extend the core mining algorithms to discover *conditional rules*, i.e., rules that apply only for a subset of the data, identified by a selection with a constant over the values of an entity or of a type. By exploiting clustering of the entities, we are able to identify subsets of the data for which the mining leads to new rules that are not identified with the general algorithm.

We first describe RuDiK example generation (Section 4.4) and rule mining (Section 4.5) modules and then introduce our proposed improvement in Section 4.6. We also conducted experiments to show the quality of generated conditional rules in Section 4.7.2.

In Section 4.8 we conclude with open directions for research in this topic. We open sourced the code of the system online, together with the experimental results and discovered rules at `https://github.com/ppapotti/Rudik`.

4.1 Related Work

For *relational data*, dependencies are discovered over the attributes of a given schema and encoded into formalisms, such as Functional Dependencies [37, 39, 40] and Denial Constraints [38, 145].

However, these techniques cannot be applied to KGs for three main reasons: *(i)* the schema-less nature of RDF data and the open world assumption; *(ii)* traditional approaches rely on the assumption that data is either clean or has a small amount of errors, which is not the case with KGs; *(iii)* even when the algorithms are designed to support more errors [30, 146], there are scalability issues on large RDF datasets: a direct application of relational database techniques on the graph requires the materialization of all possible predicate combinations into relational tables.

Fan et. al. [147] laid the theoretical foundations of Functional Dependencies on Graphs (GFDs) and have introduced an algorithm for their discovery [148]. However, their language does not include general literal comparisons, which is useful when detecting errors in KGs.

RuDiK is the first approach that is generic enough to mine both positive and negative rules in RDF KGs. Rule mining approaches designed for positive rule discovery in KGs, such as AMIE [28] and OP algorithm [33], load the entire graph into memory prior to the traversal step. This is a constraint for their applicability over large KGs, and neither of these two approaches can afford value comparison. In contrast to them, by generating the graph on-demand, RuDiK discovers rules on a small fraction of the graph. This makes it scalable and the low memory footprint enables a bigger search space with rules that can have literal comparisons. Finally, [32] recommends new facts by using association rule mining techniques. Their rules are made only of constants and are therefore less general than the rules generated by RuDiK.

ILP systems such as WARMR [41] work under the CWA and require the definition of error-free positive and negative examples. These assumptions do not hold in KGs and AMIE outperforms these two systems [28]. Sherlock [144] is an ILP system that extracts first-order Horn Rules from Web text. While extending RuDiK to free text is an interesting future work, the statistical significance estimate needs a threshold to discover meaningful rules.

Error detection in KGs has also been studied by using ILP methods to discover axioms concerning properties' domain and range restrictions that identify contradictions [149]. Another approach identifies outliers after grouping subjects by type [150]. For example, for "population"

it groups by "city" and "state" and then detects anomalies. Both methods are orthogonal and complementary to our negative rules. Finally, the generation of new facts in a graph is related to the task of *link prediction* [86].

4.2 Rule Mining

Logical rules can assist us in two tasks related to the KG curation: i) they can be employed to find inconsistencies in KG and ii) they can be exploited to add new facts to KG. Although logical rules can increase the quality of KGs, it is not feasible to manually generate a lot of them and as a results, we need methods for automatic extraction of rules. Rule mining is the process of automatically extracting logical rules from KGs. In this section, we first introduce positive and negative logical rules and then introduce the converge of a rule which is a metric that can be used in order to improve the quality of logical rules.

4.2.1 Logical Rules

A Horn Rule is a disjunction of *atoms* with at most one unnegated atom. In this work we focus on Horn rules of the form:

$$\vec{B} \rightarrow r(x,y) \tag{4.1}$$

where $r(x,y)$ is a single atom (head of the rule) and $\vec{B}$ (body of the rule) is a conjunction of atoms $B_1(z_1,z_2) \wedge B_2(z_3,z_4) \wedge \cdots \wedge B_n(z_{n-1},z_n)$. An atom is a predicate connecting two variables, two entities, an entity and a variable, or a variable and a lexical value. We distinguish two kinds of rules:

- **positive rules**, which can be executed to add new facts to the KG with the aim of increasing its coverage of the reality. e.g., "if two persons have a child in common, they are in the spouse relation".

$$r_1 : \texttt{parent}(v_0,a) \wedge \texttt{parent}(v_0,b) \Rightarrow \texttt{spouse}(a,b)$$

- **negative rules**, which can be used to identify logical inconsistencies in the KG, ultimately detecting incorrect triples. This rules have a negated atom in the head identify false facts from the ones in the KG, e.g., " parents birth date cannot be biger than their children birth date" or "A person cannot be the founder of a company which has been founded before her birth date".

$$r_2 : \texttt{parent}(a,b) \wedge \texttt{birthDate}(b,v_0) \wedge \texttt{birthDate}(a,v_1) \wedge v_0 > v_1 \Rightarrow \bot$$

$$r_3 : \texttt{foundingYear}(a,v_0) \wedge \texttt{birthYear}(b,v_1) \wedge >(v_1,v_0) \wedge \texttt{foundedBy}(a,b) \Rightarrow \bot$$

Positive rules define relationships between KG elements that identify missing triples, e.g., "if two persons have a child in common, they are in the spouse relation". *Negative rules* have a negated atom in the head identify false facts from the ones in the KG, e.g., "parents cannot marry

their children". As an example of using negative rules for error detection, this negative rule $child(x,y) \rightarrow \neg spouse(x,y)$ can be rewritten as $child(x,y) \wedge spouse(x,y) \rightarrow \bot$ and the body is run as a query on the KG.

Given a KG, a set of positive example facts, and a set of negative example facts, a rule mining algorithm aims at identifying a set of rules that, when executed on the KG, have all the positive examples facts in their output and none of the negative facts. In other words, the algorithms are after rules that are general, as they cover all true facts, and do not make mistakes, as they do not produce false facts. As these requirements are quite strict in the presence of noisy and incomplete KGs, they are relaxed in some mining algorithms with a notion of weight (or score) defined on the number of positive and negative examples in the output of the set of mined rules.

We study how to mine declarative rules from RDF KGs. An RDF KG is a database representing information with triples (or *facts*) $\langle s,p,o \rangle$, where a *predicate* p connects a *subject* s and an *object* o. For example, the fact that Hillary Clinton has a child named Chelsea Clinton is expressed with a triple $\langle Hillary\_Clinton, hasChild, Chelsea\_Clinton \rangle$. In a triple, the subject is an *entity*, i.e., a real world concept; the object is either an entity or a *literal*, i.e., primitive type such as number, date, and string; and the triple predicate specifies a relationship between subject and object.

4.2.2 Rule Coverage

Given a pair of elements (x,y) from a KG kg and a Horn Rule r, we say that r_{body} *covers* (x,y) if $(x,y) \models r_{body}$. More specifically, given a rule $r : r_{body} \Rightarrow \texttt{r}\,(a,b)$, r_{body} covers a pair of elements $(x,y) \in kg$ iff we can substitute a with x, b with y, and the rest of the body can be instantiated over kg. Given a set of pair of elements $E = \{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$ and a rule r, we denote by $C_r(E)$ the *coverage* of r_{body} over E as the set of elements in E covered by r_{body}: $C_r(E) = \{(x,y) \in E | (x,y) \models r_{body}\}$.

Given the body r_{body} of a rule r, we denote by r^*_{body} the *unbounded body* of r. The unbounded body of a rule is obtained by keeping only atoms that contain a target variable and substituting such atoms with new atoms where the target variable is paired with a new unique variable. As an example, given $r_{body} = \texttt{rel}_1\,(a,v_0) \wedge \texttt{rel}_2\,(v_0,b)$ where a and b are the target variables, $r^*_{body} = \texttt{rel}_1\,(a,v_i) \wedge \texttt{rel}_2\,(v_{ii},b)$. While in r_{body} the target variables are bounded to be connected by variable v_0, in r^*_{body} they are unbounded. Given a set of pair of elements $E = \{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$ and a rule r, we denote by $U_r(E)$ the *unbounded coverage* of r^*_{body} over E as the set of elements in E covered by r^*_{body}: $U_r(E) = \{(x,y) \in E | (x,y) \models r^*_{body}\}$. Given a set E, $C_r(E) \subseteq U_r(E)$.

Example 4: Let E be the set of all possible element pairs in kg. The coverage of r_2 over E ($C_r(E)$) is the set of all pairs of entities $(x,y) \in kg$ s.t. both x and y are the subject for a birth year triple and the year for x is higher. The unbounded coverage of r_2 over E ($U_r(E)$) is the set of all pairs of entities (x,y) s.t. both x and y are the subject for a birth year triple without any condition on the relationships between such year values.

Unbounded coverage plays a crucial role in distinguishing inconsistent and missing data. Given entities (x,y) in a pair, if the birth year is missing for at least one of them, it is not possible to state who was born first. Only if both entities have their birth years *and* x is born before y, we conclude that r_2 does not cover (x,y). Given that KGs are largely incomplete [151], the ability to

discriminate missing and conflicting information is of paramount importance. We extend next the definitions for coverage and unbounded coverage to a set of rules $R = \{r_1, r_2, \cdots, r_n\}$ as the union of individual coverages:

$$C_R(E) = \bigcup_{r \in R} C_r(E) \qquad U_R(E) = \bigcup_{r \in R} U_r(E)$$

4.3 Rule Discovery for Noisy Knowledge Graphs

For the sake of simplicity, we define the discovery problem for a given *target predicate*. To obtain all rules for a KG, we compute rules for every predicate in it. We characterize a predicate with two sets of pairs of elements, where an element is either an entity or a literal value. The *generation set* G contains examples for the predicate, while the *validation set* V contains counter examples for the same. Consider the discovery of positive rules for the `hasChild` predicate between entities; G contains true pairs of parents and children and V contains pairs of people who *are not* in such relation. If we want to identify errors (negative rules), the sets of examples are the same, but they switch role. To discover negative rules for `hasChild`, G contains pairs of people not in a child relation and V contains pairs of entities in a child relation. As another example, consider predicate `birthDate`, which has a literal value as object. For mining positive rules, G contains true pairs with subject entities and their birth dates and V contains pairs with subjects and literal values that are not their birth dates. Again, for identifying negative rules, we switch the role of the sets.

We formalize next the *exact discovery problem*. In the following definitions, we assume for the sake of simplicity that all possible valid rules and the sets of examples have been already generated, we detail in the rest of the article how they are efficiently obtained from the KG.

Definition 1: Given a KG kg, two sets of pairs of elements G and V from kg with $G \cap V = \emptyset$, and all the valid Horn Rules R for kg, a solution for the *exact discovery problem* is a subset R' of R s.t.:

$$\underset{R'}{\operatorname{argmin}}(size(R') | (C_{R'}(G) = G) \wedge (C_{R'}(V) = \emptyset))$$

The minimal set of rules covering all pairs in G and none of the pairs in V forms the exact solution. It minimizes the number of rules in the output ($size(R')$) to favour generic rules that can affect large portions of the graph. In fact, given a pair of elements (x, y), there is always an overfitting rule whose body covers only pair (x, y) by assigning target variables to literal values x and y.

Example 5: Assume we are mining positive rules for the predicate `couple` by using examples from the people forming the Obama family. We are given only two couple examples. The positive one is the pair (Barack,Michelle) and the negative one contains their daughters (Natasha,Malia). Consider now the three rules below.

$$r_3 : \texttt{livesIn}(a, v_0) \wedge \texttt{livesIn}(b, v_0) \Rightarrow \texttt{couple}(a, b)$$

$$r_4 : \texttt{hasChild}(a, v_i) \wedge \texttt{hasChild}(b, v_i) \Rightarrow \texttt{couple}(a, b)$$

$$r_5 : \texttt{hasChild}(\mathit{Michelle}, \mathit{Natasha}) \wedge \texttt{hasChild}(\mathit{Barack}, \mathit{Natasha}) \Rightarrow \texttt{couple}(\mathit{Barack}, \mathit{Michelle})$$

Positive rule r_3 states that two individuals are likely to be a couple if they live in the same location, while rule r_4 states that this is the case when they have at least one child in common. Assuming triples for predicates `livesIn` and `hasChild` are in the KG, both r_3 and r_4 cover the positive example. While r_4 is an exact solution (not covering the negative example), r_3 is not, as the daughters also reside in the same location. Rule r_5 explicitly mentions entity values (constants) in its head and body. It is also an exact solution, but, in contrast to r_3, it only applies for the given positive example.

If any `hasChild` triple is missing in G, the exact discovery finds only r_5 as a solution. This example demonstrates why the exact discovery is not robust to data quality issues in KGs. Even in cases in which a valid rule exists, missing or incorrect triples associated to the examples in G and V lead to misleading coverage. In the worst case, the exact solution may collapse to a set of rules where every one of them covers only one example in G, effectively discovering rules with no coverage outside of the examples when executed on the graph.

4.3.1 Weight Function

Being aware of errors and missing data in the graph, we remove the requirement of (not) covering (V) G exactly with the rules. We instead identify rules that hold for most of the data (soft-constraints) to be robust w.r.t. incompleteness and noise. Coverage still guides our assessment of the rule quality: good rules should cover as many example pairs in G as possible, while covering very few to zero pairs in V. We implement these ideas with a *weight* computed for every rule.

Definition 2: Given a KG kg, two sets of element pairs G and V from kg with $G \cap V = \emptyset$, and a Horn Rule r, the *weight of* r is defined as follow:

$$w(r) = \alpha \cdot (1 - \frac{|\, C_r(G)\,|}{|\, G\,|}) + \beta \cdot (\frac{|\, C_r(V)\,|}{|\, U_r(V)\,|}) \tag{4.2}$$

with $\alpha, \beta \in [0,1]$ and $\alpha + \beta = 1$, thus $w(r) \in [0,1]$.

The weight models rule quality w.r.t. G and V – a rule covering all generation elements of G and none of the V validation elements has a weight of 0. Therefore, the better the rule, the smaller its weight. The weight has two parts normalized by parameters α and β. The first part quantifies the coverage over generation set G – the ratio between the coverage of r over G and the size of G. If r covers all elements in G, this component becomes 0. The second part measures the coverage of r over the unbounded coverage of the rule over V, instead of the total elements in V. Since some elements in V might not satisfy the predicates in r_{body}, we restrict V with unbounded coverage to validate on "qualifying" example pairs that have the information tested by the rule's body.

Parameters α and β set the relevance of each part. A high value for β guides the mining towards rules with high precision by penalizing the ones that cover V pairs, while a high value for α favours the recall by championing rules that cover more G pairs.

Example 6: We use rule r_2 from Section 4.2. Assume two sets of element pairs G and V from a KG kg. The first part of $w(r_2)$ corresponds to 1 minus the number of examples $(x,y) \in G$ where x is born after y divided by the size of G. The second part is the number of examples $(x,y) \in V$ with x born after y divided by the number of examples $(x,y) \in V$ having birth date values for both x and y in kg, i.e., $U_{r_2}(V)$ does not contain pairs without birth date values.

Definition 3: Given a KG kg, two sets of element pairs G and V from kg with $G \cap V = \emptyset$, and a set of rules R, the *weight for R* is:

$$w(R) = \alpha \cdot (1 - \frac{|\, C_R(G)\,|}{|\, G\,|}) + \beta \cdot (\frac{|\, C_R(V)\,|}{|\, U_R(V)\,|})$$

Weights model the presence of errors in KGs. Consider the case of negative rule discovery, where V contains positive examples from the graph. We report in the experimental evaluation several negative rules with significant coverage over V, which corresponds to errors in the KG.

4.3.2 Problem Definition

We can now state the approximate version of the problem.

Definition 4: Given a KG kg, two sets of element pairs G and V from kg with $G \cap V = \emptyset$, all the valid Horn Rules R for kg, and a weight function w for R, a solution for the *robust discovery problem* is a subset R' of R such that:

$$\underset{R'}{\operatorname{argmin}}(w(R') | C_{R'}(G) = G)$$

The *robust* version of the discovery problem aims at identifying rules covering all elements in G and as few elements as possible in V. As we want to avoid overfitting rules in R, we do not allow in the solution rules with constants in the target variables.

Our problem definition can be mapped to the NP–complete *weighted set cover problem* [152]. The reduction follows from the following mapping: the set of elements (universe) corresponds to the generation pairs in G, the input sets are identified by the rules in R (where each rule covers a subset of G), the non-negative weight function $w : r \rightarrow \mathbb{R}$ is $w(r)$ (Definition 2), and the cost of R is defined to be its total weight (Definition 3).

4.4 Generation of Rules and Examples

We start by discussing the generation of the universe of all possible rules. We first assume that examples in G and V are available, and then discuss methods to obtain them automatically. Our solution does not depend on how examples are obtained. We explicitly generate them in our approach, but the same rule mining algorithms apply if humans provide pairs for G and V.

We describe the mining of *positive* rules with correct facts in G and incorrect ones in V. When mining *negative* rules, our rule generation and mining algorithms are the same: it is enough to swap the role of generation set G (therefore containing false facts) and validation set V (in this case containing true facts). The example generation algorithm also applies in both scenarios.

4.4.1 Rule Generation

We represent a KG kg as a directed graph: elements (entities and literals) form the nodes with a directed edge from node a to node b for each fact $\langle a, rel, b\rangle \in kg$. The relation rel that connects subject to object is used as label for the corresponding edge. We report four facts in Figure 4.1.

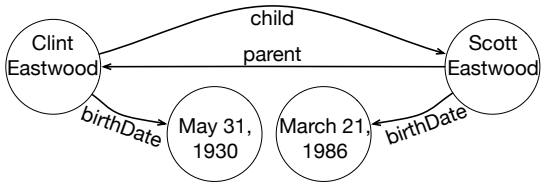


Figure 4.1 – Four DBpedia facts in the graph representation.

If we allow navigation of edges independently of the edge direction, we can navigate kg as an *undirected graph*. The body of a rule can be seen as a path in the undirected graph. In Figure 4.1, the body $\texttt{child}(a,b) \wedge \texttt{parent}(b,a)$ corresponds to the path *Clint Eastwood* − *Scott Eastwood* − *Clint Eastwood*. A valid body contains target variables a and b at least once, every other variable at least twice, and atoms are transitively connected. Given a pair of elements (x,y), a *valid body* is a valid path p on the undirected graph s.t.: (i) p starts at the node x; (ii) p covers y at least once; (iii) p ends in x, in y, or in a different node that has been already visited.

Given the body of a rule r_{body}, r_{body} covers a pair of elements (x,y) iff there exists a valid path on the graph that corresponds to r_{body}. This implies that for a pair of elements (x,y), we can generate bodies of all possible valid rules by computing all valid paths that start from x with a breadth-first search. The ability to navigate each edge in any direction by turning the original directed graph into an undirected one is key for the generation task, but we must preserve information about the original direction of the edges. This is used when translating paths to rule bodies. In fact, an edge directed from a to b produces the atom $\texttt{rel}(a,b)$, while b to a produces $\texttt{rel}(b,a)$, according to the direction in the original directed graph.

Since every node can be traversed multiple times, for two elements x and y there might exist a large number of valid paths starting from x. This number is constrained with a *maxPathLen* parameter that limits the search space by determining the maximum number of edges in the path, i.e., the maximum number of atoms in the body of the corresponding rule.

We are now ready to describe the generation of all possible rules. Every rule in R (universe of all possible rules) must cover at least one example in the generation set G. The universe of all possible rules is therefore created by analyzing G elements.

(i) Path Generation. Given an example with elements (x,y), we retrieve all nodes from x or y at distance $maxPathLen-1$ or less. We collect also the edges in this navigation. The retrieval is a recursive exploration with a queue of elements. The queue is initialized with x and y. For each node e in the queue we run a SPARQL query against the graph to get nodes (and edges) at distance 1 from e (*single hop queries*). At the n-th step, we add the new nodes to the queue iff they are at a distance less than $(maxPathLen-n)$ from x or y.

Given the graph for every (x,y), we compute all valid paths that start from every x with a simple DFS exploration.

(ii) Path Evaluation. Computing paths for every example in G is related to the computation of the rule coverage. The *coverage* of a rule r corresponds to the number of G pairs for which there exists a path corresponding to r_{body}.

Given that the universe of rules and their coverage over G have been computed, the unbounded coverage is computed over V with a simple execution for each rule of two SPARQL queries over the KG.

Example 7: Consider a scenario where we mine the `sibling` predicate for positive rules. Starting from a positive example with a pair (Natasha, Malia), one path can traverse three nodes such as: Natasha− Barack−Malia. If several pairs in G have this path (siblings have one common parent), the corresponding rule would be $\texttt{hasParent}(b,a) \wedge \texttt{hasParent}(c,a) \Rightarrow \texttt{sibling}(b,c)$.

In the example, the shared variable a between two `hasParent` atoms corresponds to a join, therefore a comparison based on value equality. Since one of our goals is to mine rich rules, we make use of more atom types during the path generation, as discussed next.

Literal comparison.

To discover rich rules with comparisons beyond equalities, the graph should have edges connecting literals with a symbol from $\{<, \leq, \neq, >, \geq\}$, e.g., Figure 4.1 would contain an edge '$<$' from node "*March 31, 1930*" to node "*March 21, 1986*". Unfortunately, the original KG does not have this information explicitly, and materializing such edges among all literals is infeasible.

However, in our algorithm we discover paths for a pair of elements from G in isolation. The graph for a pair of elements has a size that is orders of magnitude smaller than the KG. The very small number of nodes for a single example graph enables us to represent all the literal pairwise comparisons across them.

Besides equality comparisons expressed in joins, we add explicit relationships with comparisons '$>$','$\geq$','$<$','$\leq$' between numbers and dates, and $\neq$ between all literals. These comparisons are added to the graph as normal edges (atoms in the formula): $x \geq y$ leads to $\texttt{rel}(x,y)$, where `rel` is $\geq$. Once we add comparison edges for every input example graph, the discovery of literal comparisons is equivalent to the discovery with predicate atoms.

Not equal entities. We introduce "not equal" comparison for literal values, but this comparison is useful also for entities. For example, a negative rule may state that if a person a was born in a location *different* from b, then such person cannot be the president of b ($\texttt{bornIn}(a,x) \wedge x \neq b \wedge \texttt{president}(a,b) \Rightarrow \bot$). One way to introduce inequalities among entities in the graph is to add edges among all entity pairs. It is clear that this strategy is inefficient and may actually introduce obvious edges, e.g., person are different from locations. To limit the search space while aiming at meaningful rules, we therefore exploit the `rdf:type` triples associated to elements, which is usually available in KGs. We add a new inequality edge in the input example graph only between pairs of elements of the same type. In the example, x and b are compared as they are both locations.

4.4.2 Negative Examples Generation

Given a KG kg and a predicate $rel \in kg$, generation set G and validation set V are generated in our approach. For positive rule mining, we follow the traditional silver standard creation and set G equals to the facts for predicate rel [15], i.e., all pairs of elements (x,y) such that $\langle x, rel, y \rangle \in kg$. These are going to be mostly correct, depending on the quality of kg. Set V contains counter examples for rel. These must be generated because of the open world assumption in KGs. Differently from mining in relational databases, it is not possible to make the assumption that anything not in a KG is false (closed world assumption), thus in our graphs missing information is *unknown*. However, it is unlikely that two randomly selected elements are a true positive example for any predicate. This intuition leads to a simple way of creating counter examples for any predicate by randomly selecting pairs from the cross-product of the elements [42].

While the simple generation "at random" can generate negative examples with high accuracy, a very small percentage of these element pairs will be *semantically related*. In other terms, the elements will be unrelated in time, space, and topic. This data aspect is of crucial importance when mining negative rules. In fact, two unrelated elements have a smaller number of paths between them than semantically related ones; this is reflected by a lower number of rules that can be mined when counter examples are in G. As we show experimentally, unrelated elements are less likely to lead to meaningful patterns in the KG, and therefore the rules are of lower quality.

It is important to notice that there is no such issue when G contains correct examples, i.e., mining positive rules. In fact, at least one semantic connection is present for any example by generation, since pairs in G are obtained from the triples of the predicate of interest. To generate negative examples that are likely to be correct (truly false facts) and that are semantically related, we generate queries over the graph that identify elements that satisfy these requirements.

First, we exploit the notion of *Local-Closed World Assumption* (*LCWA*) [9, 28] to identify the elements that are likely to be completely described in the KG. The assumption states that if for a given subject and predicate the KG contains at least one object value, then it contains all possible object values; e.g., if a graph has one spouse for Barack Obama, then we assume that all his spouses are in the graph. While this is always the case for *functional* predicates (e.g., `capital`), it might not hold for non-functional ones (e.g., `hasChild`).

Under this assumption, we identify elements that are likely to be complete and create negative examples by taking the union of elements satisfying the LCWA. For a predicate *rel*, a negative example is a pair (x,y) where either x is the subject of at least one triple $\langle x, rel, y' \rangle$ with $y \neq y'$, or y is the object of one or more triples $\langle x', rel, y \rangle$ with $x \neq x'$. For example, if $rel =$ `hasChild`, a query would identify as negative example any pair (x,y) s.t. x has some children in the graph who are not y, or y is the child of someone who is not x.

Second, for a candidate negative example over elements (x,y), x must be connected to y via a predicate that is different from the target predicate. In other words, given a KG kg and a target predicate *rel*, (x,y) is a negative example if $\langle x, rel', y \rangle \in kg$, with $rel' \neq rel$. These restrictions make the size of V of the same order of magnitude as G and guarantee that, for every $(x,y) \in V$, x and y are semantically related by at least one predicate.

Example 8: A negative example (x,y) for the target predicate `hasChild` has the following characteristics: *(i)* x and y are not connected by the `hasChild` predicate; *(ii)* either x has one or more children (different from y) or y has one or more parents (different from x); *(iii)* x and y are connected by a predicate that is different from `hasChild` (e.g., `colleague`).

Similarly, for a negative example (x,y) for predicate `birthDate`: *(i)* x and y are not in a `birthDate` relationship; *(ii)* x has a birth date (different from y) or y appears in a birth date relation with a subject (different from x); *(iii)* x and y are connected by a predicate different from `birthDate`.

To enhance the quality of the input examples and avoid cases of mixed types, we require that for every example pair (x,y), either in G or V, all the x (y) occurrences have the same *type*.

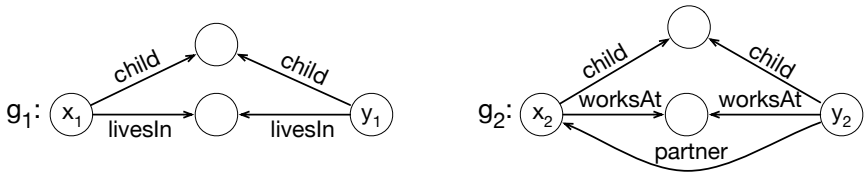


Figure 4.2 – Two positive examples.

4.5 Discovery Algorithm

This sections introduces a greedy approach to solve the approximate discovery problem (Section 4.3.2). Since the number of possible rules can be very large, RuDiK introduces an algorithm that generates only promising rules from the KG, while preserving the quality guaranteed by the exhaustive generation.

4.5.1 A Greedy Algorithm Based on Marginal Weight

Our goal is to discover a set of rules to produce a weighted set cover for the given examples. We therefore follow the intuition behind the greedy algorithm for weighted set cover by defining a *marginal weight* for rules that are not yet included in the solution [152].

Definition 5: Given a set of rules R and a rule r such that $r \notin R$, the *marginal weight* of r w.r.t. R is defined as:

$$w_m(r) = w(R \cup \{r\}) - w(R)$$

The marginal weight quantifies the weight increase by adding r to R. It measures the contribution of r to R in terms of new elements covered in G and V. Since we aim at minimizing the total weight, a rule is not added to the solution if its marginal weight is greater than or equal to 0.

When all rules have been generated, the algorithm for greedy rule selection is simple: given G, V, and the universe of rules R, select the rule r with minimum marginal weight, add it to solution R', and iterate over $R \setminus r$. Algorithm stops when at least one of the three termination conditions is met: *1)* all R rules are in the solution; *2)* R' covers all elements of G; *3)* among the remaining rules in R, none of them has a negative marginal weight.

The greedy algorithm has quadratic complexity over the number of rules and guarantees a $\log(k)$ approximation to the optimal solution [152], where k is the largest number of elements covered in G by a rule in R. If the optimal solution is made of rules that cover disjoint sets over G, then the greedy solution coincides with the optimal one.

4.5.2 Graph Traversal with A^* Search

The greedy algorithm for weighted set cover is based on the fact that the universe of rules R is available. We could generate R by traversing all valid paths from a node x to a node y, for each pair $(x,y) \in G$. However, generating all these paths is not actually needed for every example.

Example 9: Suppose we discover positive rules for predicate `spouse`. The generation set G includes entities g_1 and g_2 shown as graphs in Figure 4.2. Assume that all rules in R have the same coverage and unbounded coverage over the validation set V. One possible rule is

$r : \texttt{child}(x, v_0) \wedge \texttt{child}(y, v_0) \Rightarrow \texttt{spouse}(x, y)$, stating that entities x and y with a child in common are also married. In the graph, r covers both g_1 and g_2. Since all rules have the same coverage and unbounded coverage over V, any other rule will not bring any benefit to the current solution. In fact, any other candidate will not cover new elements in G, therefore their marginal weights will be negative. Therefore, the exploration of edges `livesIn` in g_1, `worksAt` in g_2, and `partner` in g_2 is superfluous.

Based on the above observation, we avoid generating the entire universe R, but rather follow at each iteration the most promising path on the graph.

For each example $(x, y) \in G$, we start the exploration from x. Inspired by the A^* graph traversal algorithm [153], we use a queue of invalid paths (which would lead to invalid rules), and at each iteration we pick the path with the minimum marginal weight so far, which corresponds to several paths in the graphs. We expand the path by following the edges and add new paths to the back of the queue. Unlike A^*, we do not stop when a path reaches the target node y, i.e., becomes valid. The algorithm adds the valid path (which would turn into a valid rule) to the solution and keeps looking for other valid paths until one of the termination conditions of the greedy set cover algorithm is met.

To guarantee the optimality of the solution in A^*, the estimation function must be *admissible* [153], i.e., the estimated cost must be less than or equal to the actual cost. We define an admissible estimation of the marginal weight for an invalid path that can still be expanded.

Definition 6: Given a rule $r : A_1 \wedge A_2 \cdots A_n \Rightarrow B$, we say that a rule r' is an *expansion* of r iff r' has the form $A_1 \wedge A_2 \cdots A_n \wedge A_{n+1} \Rightarrow B$.

In the graph exploration, expanding r means traversing one further edge on the path constructed by r_{body}. To guarantee optimality, the estimated marginal weight for a rule r that is invalid must be less than or equal to the actual weight of any valid rule that is generated by means of expanding r. Given a rule and some expansions of it, we can derive the following.

Lemma 1: *Given a rule r and a set of pair of elements E, then for each r' expansion of r, $C_{r'}(E) \subseteq C_r(E)$ and $U_{r'}(E) \subseteq U_r(E)$.*

The above Lemma states that the coverage and unbounded coverage of an expansion r' of r are contained in the coverage and unbounded coverage of r, respectively, and directly derives from the augmentation inference rule for functional dependencies [37], i.e., expanding a rule with a new atom to its body makes the rule more selective.

The only positive contribution to marginal weights is given by $|C_{R\cup\{r\}}(V)|$, which is equivalent to $|C_R(V)| + |C_r(V) \setminus C_R(V)|$. If we set $|C_r(V) \setminus C_R(V)| = 0$ for any invalid r, we guarantee an admissible estimation of the marginal weight. We estimate the coverage over the validation set to be 0 for any rule that can be further expanded, since expanding it may bring the coverage to 0.

Definition 7: Given an *invalid* rule r and a set of rules R, we define the *estimated marginal weight* of r as:

$$w_m^*(r) = -\alpha \cdot \frac{|C_r(G) \setminus C_R(G)|}{|G|} + \beta \cdot \left(\frac{|C_R(V)|}{|U_{R\cup\{r\}}(V)|} - \frac{|C_R(V)|}{|U_R(V)|}\right)$$

The estimated marginal weight for a valid rule is equal to the actual marginal weight from Definition 5. Given Lemma 1, we can easily see that $w_m^*(r) \leq w_m^*(r')$, for any r' expansion of r. Thus our marginal weight estimation is admissible.

Algorithm 6: RuDiK Rule Discovery.

input : G (generation set), V (validation set), *maxPathLen* (maximum rule body length)
output : R_{opt} – union of rules in the solution

1 $R_{opt} \leftarrow \emptyset$;
2 $N_f \leftarrow \{x | (x,y) \in G\}$;
3 $Q_r \leftarrow$ `expandFrontiers`(N_f);
4 $r \leftarrow \underset{r \in Q_r}{\text{argmin}}(w_m^*(r))$;
5 **repeat**
6 $\quad Q_r \leftarrow Q_r \setminus \{r\}$;
7 $\quad$ **if** `isValid`(r) **then**
8 $\quad\quad R_{opt} \leftarrow R_{opt} \cup \{r\}$;
9 $\quad$ **else**
$\quad\quad$ `// rules expansion`
10 $\quad\quad$ **if** `length`$(r_{body}) < maxPathLen$ **then**
11 $\quad\quad\quad N_f \leftarrow$ `frontiers`(r);
12 $\quad\quad\quad Q_r \leftarrow Q_r \cup$ `expandFrontiers`(N_f);
13 $\quad r \leftarrow \underset{r \in Q_r}{\text{argmin}}(w_m^*(r))$;
14 **until** $Q_r = \emptyset \vee C_{R_{opt}}(G) = G \vee w_m^*(r) \geq 0$;
15 **if** $C_{R_{opt}}(G) \neq G$ **then**
16 $\quad R_{opt} \leftarrow R_{opt} \cup$ `singleInstanceRule`$(G \setminus C_{R_{opt}}(G))$;
17 **return** R_{opt}

We now introduce Algorithm 6, a modified set cover procedure including the A^*-like rule expansion. For simplicity, we blur the difference between paths and rules in the description, however the search is on the undirected graph. For a rule r, we call *frontier nodes*, $N_f(r)$, the last visited nodes in the paths that correspond to r_{body} from every example graph covered by r. Expanding r means navigating a single edge from any of the frontier nodes. The set of frontier nodes is initialized with starting nodes x, for every $(x,y) \in G$ (Line 2). The algorithm keeps a queue of rules Q_r, from which it picks at each iteration the rule with minimum estimated weight. The function `expandFrontiers` selects all nodes (along with edges) at distance 1 from frontier nodes and returns the set of all rules constructed with this expansion. Q_r is thus initialized with all rules of length 1 starting at x (Line 3). In the main loop, the algorithm checks if the current best rule r is valid or not. If r is valid, it is added to the output and it is not further expanded (Line 8). If r is invalid, it is expanded iff the length of its body is less than *maxPathLen* (Line 10). The termination conditions and the last part of the algorithm are the same of the greedy set-cover algorithm.

The combined generation and selection of the rules has two main advantages. First, the algorithm does not materialize the entire graph for every G pair.

The algorithm gradually materializes parts of the graph whenever they are needed for navigation (Lines 3 and 12). Second, the weight estimation enables the pruning of unpromising rules that do not cover neither new G or new V pairs.

4.5.3 Algorithm Analysis

Complexity. Each iteration of the algorithm picks the next best rule r according to the marginal weight, and then expands r by (possibly) generating new rules. The procedure of expanding a rule is the following: for each example e in G covered by r, it takes the frontier node of e and navigates the outgoing and incoming edges of the node in the graph. Each navigated edge might lead to a new rule to be added to the queue Q_r, and for each new discovered rule we compute its marginal weight by issuing a single query against the KG in order to compute its coverage over the validation set. If we consider the cost of the query as constant, the asymptotic complexity of the expansion step is $\mathcal{O}(2 \cdot p \cdot |G|)$, where p is the total number of different predicates in the KG, and the multiplier 2 is given by the fact that each predicate can generate two new rules, one for each navigation direction. The expansion step is repeated at most k times, where k is the number of all possible rules we can generate from G. Because each atom in the body of the rule can be either a predicate from the KG or one of the six literal comparisons (Section 4.4.1), the total number of rules is at most $(2p+6)^1$ (rule with one atom) $+(2p+6)^2$ (rule with two atoms) $+\ldots+(2p+6)^{maxPathLen}$. Therefore the asymptotic runtime complexity of Algorithm 6 is $\mathcal{O}\left(p^{maxPathLen} \cdot |G|\right)$. As we will outline in the experimental section, the *maxPathLen* parameter plays a crucial role in the runtime of the algorithm.

Optimality. A^* algorithm is guaranteed to return the cheapest path from start to goal if the heuristic function is *admissible*, meaning the heuristic function will never overestimates the actual cost [153]. In our settings, our heuristic is admissible iff the estimated marginal weight for a rule r is always less or equal to the actual marginal weight of r. Because in the estimated weight we set the coverage of r over the validation set to 0 (Definition 7), and since the coverage over the validation set is the only positive contribution to the actual marginal weight, then for every possible rule the estimated marginal weight will always be less or equal to the actual marginal weight. This guarantees that the output of Algorithm 6 will always be equal to the output of the greedy set cover algorithm applied on the universe of all possible rules (Section 4.5.1).

4.6 Conditional Rules

In the previous section, we described original RuDiK algorithm that discovers rules with only universal variables in the head of the rule. Let us denote them as *generic* rules. These rules are the most applicable, as they can be satisfied by a large number of facts in the KGs. However, an important problem with generic rules is that several of them do not hold semantically in all cases.

This problem is reflected by low accuracy for some of the tasks involving the generic rules, regardless of the mining algorithm. Stating that two persons in a child relationship cannot be in a spouse relationship is correct in most of the cases ($\texttt{child}(x,y) \wedge \texttt{spouse}(x,y) \Rightarrow \perp$), but there are triples in KGs for which this generic rule does not hold, such as facts whose entities are fictional characters. Indeed, the rule becomes more accurate if we restrict its scope by constraining the type of the entities, such as $\texttt{child}(x,y) \wedge \texttt{spouse}(x,y) \wedge \texttt{type}(x,\textit{Royalty}) \wedge \texttt{type}(y,\textit{Royalty}) \Rightarrow \perp$. This observation leads to the idea of improving the quality of the discovered rules by extending them with type selection.

A similar reasoning applies for rules that apply only for a certain entity. For example, the rule we discussed above for presidents and place of birth does not apply for all countries.

The correct rule should be discovered for specific countries, such as the one stating that only someone born in U.S.A. can become an American president ($\texttt{bornIn}(a,b) \wedge b \neq \textit{U.S.A.} \Rightarrow \neg\texttt{president}(a,\textit{U.S.A.})$).

There is an important trade off with conditional rules. On the one hand, a more specific context decreases the rule applicability, but, on the other hand, the rule becomes immediately more accurate.

We therefore extend the rule mining algorithm to discover *conditional rules*.

4.6.1 Type condition

We start by using constraints on the element types, such as *Date* for literals and *Organization* for entities. The idea is to group the subject and the object instances by their type and run the mining on each combination of groups. As every element can have one or more types associated, the same entity or literal can belong to different groups.

We rely on the Algorithm 7 to identify the subject and object types for the triples involved in a predicate and then group them according to the different subject-object type combinations:

1. Extract the types for subject and object elements in the triples for a predicate;
2. Create subject and object type groups;
3. Create all combinations between type groups, each combination has a type assigned for subject and a (possibly equal) type for object;
4. For each combination, generate positive and negative examples (G and V sets), drop the combination if the number of generated examples is smaller than a threshold;
5. Run generic rule mining for the examples in every remaining combination.

The grouping process leads to discovery of rules that do not apply in general, such as:

$$\texttt{party}(x,v0) \wedge \texttt{party}(y,v1) \wedge v0 \neq v1 \wedge \texttt{type}(x,\textit{Politician}) \wedge \texttt{type}(y,\textit{OfficeHolder}) \wedge \texttt{spouse}(x,y) \Rightarrow \bot$$

It is much rarer to have politicians from different parties who are married, while it is more common for people (typed only with a generic *Person*) who are just registered with different parties.

While the method above is intuitive and already leads to accurate rules with type selection, there are pre-processing steps that better characterize the group types for the triples in the KG. One approach is to use *entity embeddings* [154]. Embeddings encode entities in the KG into a low-dimensional vector space while preserving structural information about the graph [86]. A relationship in the graph can then be interpreted as a translation from subject entity to object entity in such space.

Embeddings characterize entities with hundreds of (learned) features and lead to clusters that are of high quality. Specifically for our application, the clusters can be used to obtain groups that are characterized beyond type information. In clusters from embeddings, entities with popular types, such as *Person* and *Agent* can be spread across multiple clusters, but types with finer

Algorithm 7: Conditional Rule Mining for Top-K types.

```
   Input: Predicate p
1          Number of types K
2          Examples threshold E
3  conRules = []
4  STypes ← Top-K most common types for subject entities
5  OTypes ← Top-K most common types for object entities
6  foreach st in STypes do
7      foreach ot in OTypes do
8          G ← positive examples for (st,p,ot)
9          V ← negative examples for (st,p,ot)
10         if G < E or V < E then
11             continue
12         end
13         rules ← RuDiK (G,V)
14         foreach rule in rules do
15             conRules.add(rule)
16         end
17     end
18 end
19 Output. Conditional Rules conRules
```

granularity, such as *Politician* and *OfficeHolder* are grouped together. At the same time, clusters from embeddings are more uniform in other properties that are not captured by type information, such as the structural properties of the entities when expressed as nodes in the graph.

Embeddings can therefore be used to improve the group creation steps (i) and (ii) above: we start by clustering triples with the embeddings and then find subject and object types for each cluster.

4.6.2 Entity condition

We allow the discovery of rules with constant selections over the entities. Suppose that for a rule r that states that a person cannot be president of a given country if she was not born in it, all examples of r are people born in "*U.S.A.*", and there is at least one country for which this rule is not valid. According to our problem statement, the right rule is therefore:

$$\texttt{bornIn}(a,b) \wedge b \neq \textit{U.S.A.} \wedge \texttt{president}(a,\textit{U.S.A.}) \Rightarrow \bot$$

To discover such atoms, we introduce a refinement of the rule generation. For a given rule r, we promote a variable v in a given rule r to an element e iff for every $(x,y) \in G$ covered by r, v can always be instantiated with the same value e. We allow the substitution of every variable indiscriminately, including variables in the head of the rule. This could potentially lead to rules having only constants, but we do not accept them in our solutions.

Table 4.1 – Dataset characteristics.

KG	*Version*	*Size*	*#Triples*	*#Predicates*
DBPEDIA	3.7	10.06GB	68,364,605	1,424
YAGO 3	3.0.2	7.82GB	88,360,244	74
WIKIDATA	20160229	12.32GB	272,129,814	4,108

4.7 Experiments

The discussed techniques have been implemented in RuDiK [79], our rule mining system (https://github.com/ppapotti/Rudik). We group the results for our system evaluation into five parts: *(i)* showing the accuracy of our discovered rules; *(ii)* comparing RuDiK with related systems; *(iii)* discussing examples and quality of conditional rules;

In our research, we used general KGs (*DBpedia*, *Wikidata*) instead of the KPMG KG. This decision was made for two reasons: (i) their KG is not mature enough for applying these techniques, it does not have many predicates and in most of the cases there are not alternative paths between nodes of the KG (ii) because of confidentiality agreement, we cannot expose their KG.

Setup. We run our evaluation on a desktop with a quad-core i5 CPU at 2.80GHz and 16GB RAM. We deployed on the same machine a SPARQL endpoint through OpenLink Virtuoso, optimized for 8GB RAM of available memory. Whenever not reported explicitly, we use weight parameters $\alpha = 0.3$ ($\beta = 0.7$) for positive rules, $\alpha = 0.4$ ($\beta = 0.6$) for negative rules, and allow at most 3 atoms in the body of a rule ($maxPathLen = 3$).

Metrics. We evaluated the quality of the discovered rules as the main metric to judge the effectiveness of the mining. For each KG, we started by sorting predicates w.r.t. their popularity, i.e., the number of facts for the predicate. We then chose the top 3 predicates for which we knew there existed at least one significant rule, and other 2 top predicates for which we were not aware of the existence of significant rules.

We carried out the evaluation of the discovered rules according to the best practice for rule evaluation [28]. If for a rule it was possible to declare it semantically correct in all cases, we marked all the triples coming from the application of the rule over the KG as true. Similarly for rules that were clearly incorrect, all its output triples would be marked as false. If a rule correctness was not clear just by reading its formula, we randomly sampled 30 triples from the application of the rule, which would be either new facts (positive rules) or detected errors (negative rules), and manually checked each of the facts. The *precision* of a rule is then computed as the ratio of correct triples to all of its output triples.

4.7.1 Quality of Generic Rules Discovered by RuDiK

The first set of experiments computed the accuracy of output rules over three widely used KGs: DBPEDIA, YAGO, and WIKIDATA. Table 4.1 shows the characteristics of these KGs.

The size of a KG matters, as loading all the triples in memory requires to either use powerful machines [33, 43], or to drastically shrink it by eliminating literal values [28]. Given the small memory footprint of our algorithm, we are able to discover rules with commodity HW resources without dropping the literals, which are crucial for obtaining meaningful rules. While RuDiK works with a target predicate at a time, we can use it to discover rules on the entire KG by using

each predicate as input. Next results are discussed for subsets of predicates since the manual annotation of the computed new facts and errors is an expensive process. However, when RuDiK is executed on all the predicates of a KG, results are consistent in terms of size of the output and execution times. For example, for roughly 600 predicates in DBPEDIA, we mined about 3000 positive rules, with at most 26 rules per predicate, and 4000 negative rules, with at most 32 rules per predicate.

Table 4.2 – RuDiK Rule Precision.

	Positive Rules			Negative Rules			
KG	*Avg Exec. Time*	*Avg Precision Predicates with Rules (All)*	*# Labels*	*Avg Exec. Time*	*# Detected Errors*	*Avg Precision All Predicates*	*# Labels*
DBPEDIA	35min	**97.14**% (63.99%)	139	19min	499	92.38%	84
YAGO 3	59min	**84.44**% (62.86%)	150	10min	2,237	90.61%	90
WIKIDATA	141min	**98.95**% (73.33%)	180	65min	1,776	73.99%	105

Positive Rules. We evaluate the precision for the discovered positive rules on the top 5 predicates for each KG. The number of new triples varies significantly across rules. To avoid the overall precision to be dominated by rules with big output, we first compute the precision for each rule, and then average values over all discovered rules. Table 4.2 shows precision values, along with average running time (per predicate), and the number of manually annotated facts. We distinguish predicates for which there existed at least one meaningful rule (in bold), and all predicates.

For predicates such as `academicAdvisor`, `child`, and `spouse`, the output rules show a precision above 95% in all KGs. However, when we consider other predicates, average values are brought down by few cases, such as `founder`, where valid positive rules probably do not exist at all. When a valid rule existed, the system was able to find it, but it did not recognize all cases where no positive rules existed. In our experience, it was sufficient to read the rules to identify semantically incorrect rules. Also, results show that the more accurate is the KG, the better is the quality of the positive rules. WIKIDATA contains few errors, since it is manually curated, while DBPEDIA and YAGO are automatically generated by information extractors, hence their quality is lower.

The size of the KG and the target predicate at hand have impact on the run time. The more the connections of a node (KG element), the more paths we traverse in the graph. Some elements have a large number of incoming edges, e.g., entity "*China*" in WIKIDATA has more than 600K. When the generation set includes such popular elements, the traversal of the graph becomes slower. Parameter *maxPathLen* also has a big impact on the run time

Negative Rules. Negative rules are evaluated as the percentage of actual incorrect triples over all discovered triples. Table 4.2 reports, for every KG, the total number of potential erroneous facts discovered with the output rules, whereas the precision is given as the percentage of actual errors among all the triples identified as possibly incorrect.

Generic negative rules have better accuracy than positive ones when averaging all predicates, as there are more in reality. While the quality of the rules is good, especially on the noisy KGs, we also discover rules that are supported by the vast majority of the data but do not hold semantically.

As an example, RuDiK identifies the rule that two people cannot be married if they have the same gender both in YAGO and WIKIDATA. Such rule has a precision of 94% in YAGO and of

57% in WIKIDATA.

Literals' role is bigger in negative rules, compared to the positive case. In fact, several valid negative rules rely on temporal aspects where something cannot take place before/after something else. Temporal data is usually encoded through dates and years literal values.

Mining negative rules is usually faster than mining positive ones because of the different nature of the examples covered by validation queries. Whenever we identify a potentially valid rule, we translate its body to a SPARQL query against the KG in order to obtain its coverage over V. Such queries are faster for the negative case because V has only entities connected by one predicate, whereas this is not the case in the mining of positive rules.

Table 4.3 – Sample of Discovered Rules from DBPEDIA.

Generic Rules
e_1 : `foundedBy` $(v0, y) \wedge$ `foundedBy` $(v0, v1) \wedge$ `foundedBy` $(x, v1) \Rightarrow$ **foundedBy** (x, y) e_2 : `birthDate` $(y, v0) \wedge v0 > v1 \wedge$ `foundingYear` $(x, v1) \wedge$ **foundedBy** $(x, y) \Rightarrow \bot$
e_3 : `parent` $(v0, x) \wedge$ `parent` $(v0, y) \Rightarrow$ **spouse** (x, y) e_4 : `spouse` $(y, v0) \wedge$ `parent` $(x, v0) \wedge$ **spouse** $(x, y) \Rightarrow \bot$

Conditional Rules from Type Grouping
e_5 : `foundedBy` $(x, v0) \wedge$ `associatedBand` $(v0, v1) \wedge$ `associatedBand` $(y, v1) \wedge$ `type` $(x, Company) \wedge$ `type` $(y, Artist) \Rightarrow$ **foundedBy**(x, y) e_6 : `parentCompany` $(y, v0) \wedge$ `owningCompany` $(v0, v1) \wedge$ `owningCompany` $(x, v1) \wedge$ `type` $(x, Company) \wedge$ `type` $(y, Organisation) \wedge$ **foundedBy**$(x, y) \Rightarrow \bot$
e_7 : `parent` $(v0, x) \wedge$ `parent` $(v0, y) \wedge$ `type` $(x, BritishRoyalty) \wedge$ `type` $(y, Royalty) \Rightarrow$ **spouse**(x, y) e_8 : `party` $(y, v0) \wedge$ `party` $(x, v1) \wedge v0 \neq v1 \wedge$ `type` $(x, Politician) \wedge$ `type` $(y, OfficeHolder) \wedge$ **spouse**$(x, y) \Rightarrow \bot$

Conditional Rules from Entity Clustering and Type Grouping
e_9 : `occupation` $(y, v0) \wedge$ `occupation` $(v1, v0) \wedge$ `foundedBy` $(x, v1) \wedge$ `type` $(x, Organisation) \wedge$ `type` $(y, Artist) \Rightarrow$ **foundedBy**(x, y) e_{10} : `foundingYear` $(x, v0) \wedge v0 < v1 \wedge$ `foundingYear` $(y, v1) \wedge$ `type` $(y, Organisation) \wedge$ `type` $(x, Organisation) \wedge$ **foundedBy**$(x, y) \Rightarrow \bot$
e_{11} : `parent` $(v0, x) \wedge$ `parent` $(v0, y) \wedge$ `type` $(x, Royalty) \wedge$ `type` $(y, Royalty) \Rightarrow$ **spouse**(x, y) e_{12} : `activeYearStartYear` $(x, v0) \wedge v0 > v1 \wedge$ `deathDate` $(y, v1) \wedge$ `type` $(x, Artist) \wedge$ `type` $(y, Person) \wedge$ **spouse**$(x, y) \Rightarrow \bot$

4.7.2 Conditional Rules

We now compare the results for the discovery of generic rules with the RuDiK, the conditional rules coming from the type group analysis, and the conditional rules coming from clustering with embeddings followed by type grouping. The experiment has been executed on the full DBPEDIA for the 6 largest clusters obtained with *TransE* [86], we then kept the type groups with at least 20 subject (object) elements.

From the rules reported in Table 4.3, it is easy to see that mining with grouping leads to meaningful rules for a subset of the entities. For example, a positive one stating that two artists in the same band are likely to be its co-founders (e_5), or the negative rule stating that two politicians

in different parties are unlikely to be married (e_8). Similarly for the rules coming from clustering and grouping, where rules for artists are identified both for `spouse` and `foundedBy` (e_9, e_{12}). It is interesting to see that different pre-processing algorithms lead to different sets of rules. As there is no clear winner, we argue that all methods should be executed to get the richest possible set of rules. It also can be seen that as in conditional rules we use selected predicates to choose the candidate types for subjects and objects, the generated rules are dependent on the selected predicates. E.g. for *foundedBy* predicate, *Organisation* and *Company* are among the chosen types while for *spouse*, entities have types such as *Artist* and *Royalty*.

Evaluating the Quality of Conditional Rules

We evaluate the quality of conditional rules versus generic rules by comparing the outcome of the rules for two DBPEDIA predicates: `spouse` and `foundedBy`. For each predicate, we consider both positive and negative rules. Evaluation of the quality of rules is based on three different metrics: number of rules extracted, average of manually estimated precision for the rules, and average number of triples in the output of the rules. For computing the estimated precision of a given rule r, we randomly choose 20 triples from DBPEDIA that satisfy r and then check to see how many of those triples are true (or errors in the case r is a negative rule). The estimated precision is the number of correct instances divided by 20.

Table 4.4 – Comparison between conditional and generic rules.

	Conditional Rules			**Generic Rules**		
Predicate	*#Rules*	*Avg Precision*	*Avg #Triples*	*#Rules*	*Avg Precision*	*Avg #Triples*
`spouse`	25	**81.79%**	2,785	4	77.52%	57,314
`negSpouse`	78	**99.70%**	4,977	13	98.51%	13,602
`foundedBy`	22	**57.00%**	902	5	38.01%	15,071
`negFoundedBy`	31	**95.83%**	786	6	95.77%	13,146

From the results reported in Table 4.4, it is clear that we identify a larger number of rules with our proposed extension for a given predicate. For grouping and clustering rules, the precision is higher than generic ones, but their scope is obviously lower. For all predicates, except `negFoundedBy`, the precision of conditional rules is higher. As expected, an absolute average precision improvement of 6.1% comes at the cost of a lower number of triples in the output, e.g., 70K vs 229K for `spouse`.

4.8 Summary

In this chapter, we first described RuDiK, a rule mining system that discovers generic rules from noisy and incomplete KGs, and then extended it with a conditional declarative rule mining component. The system discovers both positive rules, which suggest new potential facts for the KG, and negative rules, which identify inconsistent triples. Negative rules not only identify potential errors in KGs, but also generate representative training data for ML algorithms. We also presented a method for extracting conditional rules in KGs and our experiments showed that this technique is able to generate more rules in compare to the generic technique. It is also

showed that conditional rules are more precise than generic rules and they cover a smaller subset of triples.

Chapter 5

A Public Corpus of Rules

In this chapter, we introduce Rulehub, an open corpus for collecting and annotating rules extracted from different knowledge graphs. Our focus is to manage a large number of rules, by computing an estimate of their quality and enabling their review, storing, and querying. In building the corpus, we collected rules from many mining algorithms, but most of the rules come from AMIE [28] and RuDiK [79]. In Section 5.2, we utilize two methods for extracting logical rules from Wikidata. First, we collect a group of logical rules using the method presented in Chapter 4, and then propose translation, as an alternative to direct mining, in order to map rules mined from DBpedia into Wikidata. In Section 5.3, we use soft logical rules to teach PLMs to reason over natural language. Finally, we conclude the chapter in Section 5.4.

5.1 Rulehub

Entities across several KGs have been aligned to create the public web of *linked open data*, contributing to the creation of a larger graph [155]. Public and institutional KGs fuel several applications, including semantic search, personal assistants, and question answering in general [54, 156].

One of the great benefits of a structured dataset is that it is possible to define *dependencies* over it. KG dependencies (or *logical rules*) are used for identifying errors [79], adding new facts [28], executing queries faster [157], and reasoning for many tasks, such as explaining decisions in fact-checking [52]. Unfortunately, KGs do not come with a set of logical rules. In fact, manually crafting such rules is an expensive, human-intensive task. To support the definition of rules, several methods have been proposed. The literature for rule mining goes back to classic work on inductive logic programming and several methods have been recently proposed to mine large KGs [28, 79, 144, 148, 158]. This stream of works enables the mining of rules for any KG, thus limiting the user effort to the selection and the refinement of such rules. Users select valid rules from the many (possible thousands) discovered by a system and manually refine the ones that need changes in their conditions to be more general or more precise. Once a good set of semantically valid rules has been identified, they are usually annotated with a measure of their quality, such as a *confidence* of the rule applicability. This is necessary, as there are very few rules that are true for each and every case. As an example, consider a rule stating that "a country has always one capital". This is true for most countries, but there are 15 countries that have two

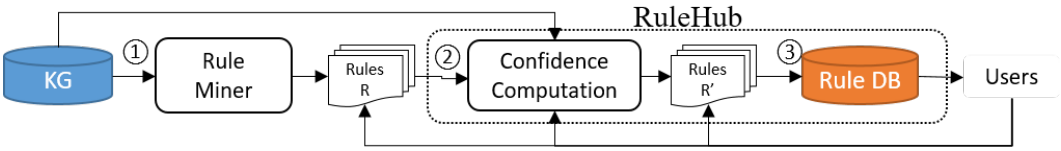


Figure 5.1 – Architecture of RuleHub.

or more capitals. Therefore the rule would have a very high confidence, but we cannot state that it is certain.

Collecting a set of high quality and well described rules is therefore an essential exercise conducted by users of KGs to improve the performance of their applications. While mining tools help in this task, there is still a lot of manual work in selecting good rules, refining them, and annotating them with their metadata. Given that several KGs are public and are used by thousands of researchers and practitioners around the world, a lot of human work on rule discovery is redundant and is not taking any benefit from this critical mass of users. We argue that rule discovery and managing should be a collective task, where we can capitalize our understanding of the data and avoid redundant work in collecting, refining, and annotating rules.

To support a collective rule discovery effort, we introduce *RuleHub*, a system that exposes in a website an extensible corpus of rules for public KGs (`http://rudik.eurecom.fr`). RuleHub is designed with rule mining tools as the principal source of rules in mind, as depicted in Figure 5.1. Users can query or browse the repository of rules, based on the KG and the predicate of interest. Moreover, they can manually specify and add new rules or update existing ones by providing more metadata, such as the confidence of a rule. We believe confidence is crucial for rules, as very few rules are completely correct or wrong in general. For this reason, we also provide a module that computes the confidence for a given rule over a KG. Rules for any KG and of any kind can be handled by the system. In this chapter, we report our experience in building RuleHub and populating it with rules discovered by existing rule mining systems. We believe RuleHub can be an enabler for a much needed collaborative work in defining metadata for public KGs.

5.1.1 Rule Confidence

We now discuss the computation of the confidence for positive and negative rules. We start with the notion of confidence of a positive rule from the literature [28] and extend it to negative rules.

The *support* of a rule is defined as the number of distinct pair of subjects and objects in the head of all instantiations of the rule that appear in the KG:

$$supp(\vec{B} \rightarrow r(x,y)) := \#(x,y) : \exists z_1,...,z_n : \vec{B} \wedge r(x,y) \tag{5.1}$$

where $z_1,...,z_n$ are the non target variables.

We remark that the support makes use of the non target variables, but count only the number of distinct pairs for the head values. Consider the rule "if two persons have a child in common, they are in the spouse relation" ($hasChild(x,v0) \wedge hasChild(y,v0) \rightarrow spouse(x,y)$) and the Obama family. Assume Barack Obama, Michelle Obama, and their two children are in the KG with the correct Spouse and Child triples to represent their relationships. The support would count

this family as one occurrence. While it uses the non target variables (referring to the child), the measure does not count this family twice, despite the rule can be instantiated twice (once for child). This design choice takes care of possible skew in the data, making sure that a rule does not get assigned a very high support when it applies for only one (distinct) pair of head entities.

The *counter-support* of a rule quantifies the number of false predictions over the existing KG. A challenge to compute this number is that KGs do not provide negative evidence, but we want to claim a mistake only if we have some evidence to support the case. To address this issue, we rely on Local Closed World Assumption. This assumption states that if we know one y (resp. x) for a given x (resp. y) and r, then we know all y (resp. x) for that x (resp. y) and r. This is widely used in practice and has proven to be an effective heuristic to overcome incompleteness of KGs [14, 28, 79]. Assume a rule concludes that $P(x_1, y_1)$ should exist. If this is the only triple involving P, x_1, and y_1, then we do not count it neither as supporting or not supporting. But if there is already a triple such as $P(x_1, y_2)$ in the KG, then we count this a counter support. This allows us to exploit counter-evidences less restrictively than the assumption 'all facts that are not in KG are false'.

$$counter\_supp(\vec{B} \rightarrow r(x,y)) := \#(x,y) : \exists z_1, ..., z_n, r(x,y') \vee r(x',y) : \vec{B} \wedge \neg r(x,y) \tag{5.2}$$

For example, a rule predicts that "Luke" and "Mary" are married, this triple is not in the KG, but "Luke" is already reported as married to someone else in the KG. To take into account both true and false predictions for a rule, we introduce *confidence scores* for positive and negative rules.

Positive Rules. Considering a positive rule $\vec{B} \rightarrow r(x,y)$ for relation $r(x,y)$. We define its confidence score as following:

$$conf(\vec{B} \rightarrow r(x,y)) := \frac{supp(\vec{B} \rightarrow r(x,y))}{supp(\vec{B} \rightarrow r(x,y)) + counter\_supp(\vec{B} \rightarrow r(x,y))} \tag{5.3}$$

This formula normalizes the support by number of pairs (x,y) satisfying the condition that there exists $r(x',y)$ or $r(x,y')$. As an example, consider the rule $hasDependant(a,b) \rightarrow hasChild(a,b)$ with the following triples $hasDependant(Laure, Mark)$, $hasDependant(Laure, Mary)$, $hasDependant(Laure, Anne)$, $hasChild(Laure, Mary)$, $hasChild(Laure, Anne)$, in other words, one person has three dependants and two of them are her children. Assume we also have two more triples for another person: $hasDependant(Gary, Rose), hasChild(Gary, Mark)$. The confidence for this rule would be 0.5. In fact, support would be 2, coming from (Laure,Mary) and (Laure,Anne), and counter support would also be 2, coming from (Laure,Mark) and (Gary,Rose). The confidence value reflects the reality: being a dependant of someone does not imply that there is a child relationship, but it can happen. How often it happens in the given KG determines the support for the rule in that context, e.g., it may be different in Europe and USA because of legal or cultural reasons.

Negative Rules. Consider a negative rule $\vec{B'} \rightarrow \neg r(x,y)$ for relation $r(x,y)$. We define its confidence score as following:

$$conf(\vec{B'} \rightarrow \neg r(x,y)) := \frac{counter\_supp(\vec{B'} \rightarrow r(x,y))}{counter\_supp(\vec{B'} \rightarrow r(x,y)) + supp(\vec{B'} \rightarrow r(x,y))} \tag{5.4}$$

Intuitively, by definition, the support (resp. counter-support) of negative rule $\vec{B'} \rightarrow \neg r(x,y)$ is indeed the counter-support (resp. support) of corresponding positive rule $\vec{B'} \rightarrow r(x,y)$. As an example, consider the rule $spouse(a,b) \rightarrow \neg hasChild(a,b)$ and triples $spouse(Laure,Mark)$, $hasChild(Laure,Anne)$, $spouse(Gary,Rose)$, and $hasChild(Gary,Micheal)$. The confidence for this rule would be 1. In fact, counter support would be 2, coming from (Laure,Mark) and (Gary,Rose), and support would be 0. If we add one case such as $spouse(Paul,Ron)$ without children, the confidence does not change. If we add $spouse(Mary,George), hasChild(Mary,George)$, then the support would become 1 and the confidence would drop to 0.67.

Issues in Confidence Measures. In negative rules, support evidences are dominating, but counter-evidences are more important. For example, consider the negative rule:

$$r_1 : birthPlace(x,v0) \wedge country(v1,v0) \wedge child(y,v1) \rightarrow \neg spouse(x,y)$$

Due to the presence of the intermediate atom $country(v1,v0)$ and the common atom $birthPlace(x,v0)$, we can generate a large number of support examples for *spouse*. Meanwhile, the counter-support of this rule is always lower than the total number of facts *spouse* in KG. If formula (5.4) is used to evaluate the rule, we obtain a very high confidence score (≈ 1.0), which is misleading because the number of supporting examples is way higher then negative ones.

With the same computation, a rule $child(x,y) \rightarrow \neg spouse(x,y)$ gets a confidence score of 0.96. From these scores, the two rules seem to be of comparable quality, but intuitively rule $child(x,y) \rightarrow \neg spouse(x,y)$ can cover a much larger number of facts and its confidence score should be higher. Here, considering the body of the rule, $spouse(x,y)$ is false. Therefore, its score should be much higher than rule r_1.

The explanation is that the support $\#(x,y)$ for rule r_1 is very large, but because of the Cartesian product, $\#x$ can dominate $\#y$ or vice versa. This undesired property overestimates the support and thereby underestimates the counter-support of a rule. To avoid this issues we define a new support for negative rules by using the minimum number of facts satisfying the head predicate, denoted as *min_supp*:

$$min\_supp(\vec{B} \rightarrow \neg r(x,y)) := min(\#x,\#y) : \exists z_1,...,z_n, r(x,y') \vee r(x',y) : \vec{B} \wedge \neg r(x,y) \tag{5.5}$$

Which returns the smaller across the numbers of occurrences for variables x and y in the KG that already satisfy the predicate. Now, (5.4) becomes:

$$conf(\vec{B'} \rightarrow \neg r(x,y)) := \frac{min\_supp(\vec{B'} \rightarrow \neg r(x,y))}{min\_supp(\vec{B'} \rightarrow \neg r(x,y)) + counter\_supp(\vec{B'} \rightarrow \neg r(x,y))} \tag{5.6}$$

Here we re-write the support of the corresponding positive rule (second term of the denominator) as *counter_supp* for better clarification, but essentially $counter\_supp(\vec{B'} \rightarrow \neg r(x,y)) = supp(\vec{B'} \rightarrow r(x,y))$.

As an example, consider three more rules: $r_2 : parent(x,y) \rightarrow \neg spouse(x,y); r_3 : relative(x,y) \rightarrow \neg spouse(x,y); r_4 : occupation(x,v0) \wedge occupation(y,v0) \rightarrow \neg spouse(x,y)$.

Consider in Table. 5.1 the computed values for *min_supp*, *counter_support* and confidence score computed from Equation (5.6) for r_2, r_3, r_4. Although there are significant differences in

Rule	*min_supp*	*counter_supp*	Conf. Score Equation (5.6)	Conf. Score Equation (5.7)
r_2	8174	43	0.995	0.881
r_3	1859	123	0.938	0.375
r_4	11792	1757	0.870	0.211

Table 5.1 – Support, Counter_Support, Confidence Scores for r_2, r_3, r_4

term of *min_support* and *counter support* among the rules, they still get very high confidence score. In common sense, r_2 is better than r_3, and r_4 is not a useful rule. But their *min_support* values are still large enough to hide the differences in the *counter support* and lead to an overestimate of the confidence. In other words, the ratio of $counter\_supp : min\_supp$ of the three rules are 0.005, 0.066, 0.150, respectively, but is not reflected in the values for (5.6). We argue that counter-evidence is more important in measuring the quality of negative rules, so it is critical to make it contribute more to the confidence computation. A reasonable way to achieve this is to make the counter-evidence comparable in values to the support evidences. We achieve this by multiplying the *counter_supp* by a factor κ, as follows:

$$conf(\vec{B'} \rightarrow \neg r(x,y)) := \frac{min\_supp(\vec{B'} \rightarrow \neg r(x,y))}{min\_supp(\vec{B'} \rightarrow \neg r(x,y)) + \kappa * counter\_supp(\vec{B'} \rightarrow \neg r(x,y))} \quad (5.7)$$

How to set κ depends on how rare are errors. This is modeled by the actual percentage of errors in the KG at hand. We assume that we are given (or can estimate) the ε error rate of the KG, i.e., the ratio of erroneous triples over all triples. We set κ to $\frac{1}{\varepsilon}$. KGs with very low error rate will get higher values for κ to preserve the $counter\_supp : min\_supp$ ratio in the original confidence computation formula. Yago reports an estimated accuracy of 95% [5] and other papers report higher estimated values [79]; we experimentally found that an estimate accuracy of 96% works better for our confidence computation. Given accuracy of 96%, an ε equals to 0.04 implies $\kappa = 25$.

Using Equation (5.7), new confidence scores of r_2, r_3, r_4 are re-calculated in the last column of Tab. 5.1. The new confidence values match our intuitions about the rules.

5.1.2 A Corpus of Rules

There are different algorithms to discover rules over KGs, as a consequence there are many rules that can be generated from different experiments over time. With the aim to store and share discovered rules as well as to expose them with metadata, such as the confidence discussed in Section 5.1.1, we have built a website with an initial corpus of more than 8000 mined rules (`http://rudik.eurecom.fr`). Besides basic information about the rule itself, we expose different kinds of confidence score: *computed confidence*, which is calculated automatically, while *human evaluation* and *human confidence* are evaluated manually. Our web portal allows users to easily search for rules, add new rules, and export them in different formats. Our long term vision is to have users contributing by opening the database to users for editing as in Wikidata. For now the system has an admin validation panel, where rules submitted by users get reviewed before being added to the corpus. In this section, we describe the information in the corpus and how to use it.

Human measures

We start presenting our two measures of quality for the rules based on human judgment.
Quality evaluation. The first one is the *quality evaluation*, a subjective assessment about the rule semantic correctness, i.e., a score of their logical meaning expressed by a human. In this measure, a user expresses a subjective assessment of the quality of a rule by reading the rule in its logical form. For example, the rule $spouse(a,b) \rightarrow spouse(b,a)$ can be judged as clearly correct. We define five levels of accurateness as following:

Level 1: Good rules, the precision is more than 80 percent.

Level 2: Acceptable rules, the precision is around 60 - 80 percent.

Level 3: Neutral rules, the precision is around 40 - 60 percent.

Level 4: Rules make sense only in certain contexts, the precision is around 20 - 40 percent.

Level 5: Illogical rules, not recommended for use.

Because each individual has her own evaluation for this score, we allow different users to express their assessment in our system.

Human confidence. The second score is the *human confidence*. This score is calculated manually based on the examination of the triples resulting from the application of a given rule. For each rule, we apply it on the KG and randomly pick 20 instances from such output. Every instance in this sample is manually validated according to external resources, such as the Web, and human assessment.

Evaluation process:

1. Given a rule, we apply it over the KG and randomly select from the results 20 instances as a sample for human confidence computation. Even though increasing the number of instances could enhance the accuracy of the evaluation, our experiments in Section 5.1.3 show that, even with 20 instances, there is a high correlation between the manually computed confidence and the confidence computed by the proposed measures.

2. Three different annotators manually check instances and conflicts are resolved with a majority voting strategy. Instances are generated randomly and will be different for each annotator. For each positive rule, an instance is labeled as 1 if it is true, 0 otherwise. For each negative rule, an instance is labeled as 1 if it is erroneous, 0 otherwise.

3. From the result of the labeling process, the human confidence of a rule is then computed as the ratio of the number of labels 1 out of all items in the sample.

Even if computed on a sample, the second score is more objective than the human assessment and we use it as our reference to evaluate the quality of the confidence computed with the measures discussed in Section 5.1.1.

Rule information. Each rule is stored as a JSON (JavaScript Object Notation [159]) file in a MongoDB instance with metadata about its provenance, support, and manual evaluation results.

- *id*: the rule internal identifier.
- *knowledge_graph*: the knowledge graph in which the rule is valid.
- *rule_type*: true or false corresponds to positive or negative rule.
- *predicate*: the target predicate.
- *premise*: the rule body.
- *hashcode*: it is computed from the predicate and rule premise, it is used to check uniqueness of rules in the system.
- *human_confidence*: indicates the human confidence.
- *computed_confidence*: indicates the support score.
- *source*: indicates the rule's origin. Rule can be added by users or obtained from a discovery system.
- *configuration*: configuration of the mining system that found the rule.
- *quality_evaluation*: subjective human evaluation.

The last three fields can have multiple occurrences for the same rule.

Rule forms. Besides exploring rules in our web portal directly, we also provide rules in different formats which can be easily consumed by other systems.

- *JSON* is a popular light-weight format designed for easy parsing.
- *SPARQL* is an RDF query language. SPARQL queries return rule output from the KG [160].

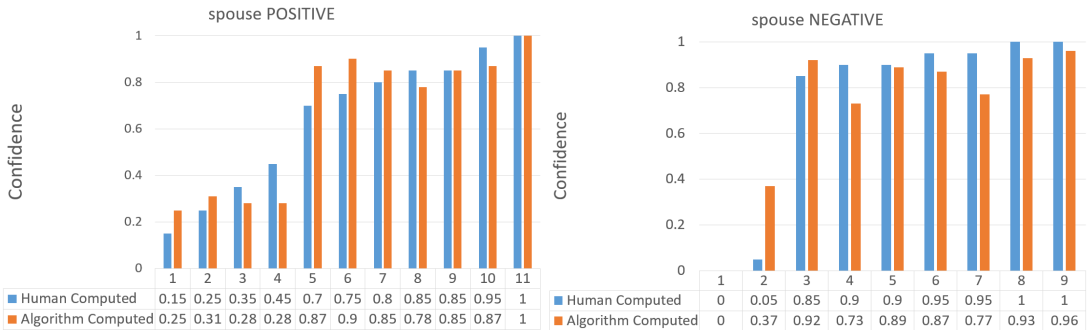
5.1.3 Experiments

In this section, we evaluate our confidence measures and report on the lessons learned in annotating rules. We group our evaluations into four parts: *(i)* showing the accuracy of our confidence measures by comparing them with the human computed confidences (see Section 5.1.2). *(ii)* discussing the effect of the κ parameter on the results. *(iii)* evaluating the performance of the proposed measure by reporting its execution time. *(iv)* measuring the impact of missing values and errors in the computation of confidence.

For this experiment, we focus on rules for two popular KGs: DBpedia [4] and Yago [5]. DBpedia is a KG derived from Wikipedia, we used version "2016-04". Yago is an open source KG with information from Wikipedia, Wordnet, and GeoNames[1]; we used Yago3.

We mined rules with rule learning systems and then chose a subset of predicates for the evaluation. Predicates were chosen based on two considerations: *(1)* it had at least five positive and negative rules in the corpus; *(2)* the computed confidences for its rules cover a wide range of values. Considering these constraints, we selected seven DBpedia predicates (*spouse*, *foundedBy*,

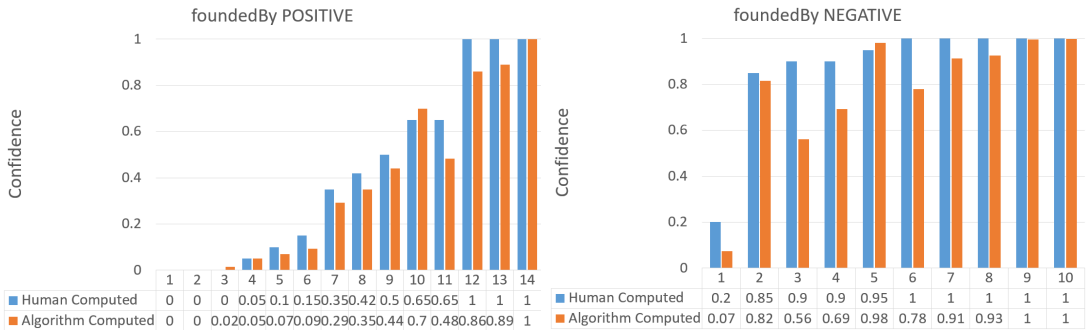
[1] https://www.geonames.org



	1	2	3	4	5	6	7	8	9	10	11
Human Computed	0.15	0.25	0.35	0.45	0.7	0.75	0.8	0.85	0.85	0.95	1
Algorithm Computed	0.25	0.31	0.28	0.28	0.87	0.9	0.85	0.78	0.85	0.87	1

	1	2	3	4	5	6	7	8	9
Human Computed	0	0.05	0.85	0.9	0.9	0.95	0.95	1	1
Algorithm Computed	0	0.37	0.92	0.73	0.89	0.87	0.77	0.93	0.96

Figure 5.2 – Confidence results for DBpedia predicate *spouse*.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Human Computed	0	0	0	0.05	0.1	0.15	0.35	0.42	0.5	0.65	0.65	1	1	1
Algorithm Computed	0	0	0.02	0.05	0.07	0.09	0.29	0.35	0.44	0.7	0.48	0.86	0.89	1

	1	2	3	4	5	6	7	8	9	10
Human Computed	0.2	0.85	0.9	0.9	0.95	1	1	1	1	1
Algorithm Computed	0.07	0.82	0.56	0.69	0.98	0.78	0.91	0.93	1	1

Figure 5.3 – Confidence results for DBpedia predicate *foundedBy*.

relative, *founder*, *publisher, employer* and *influencedBy*) and two Yago predicates (*isMarriedTo* and *hasChild*) for manual annotation. For both KGs, we use their online endpoint to have the latest version available. The annotators of the triples are authors of [49], therefore familiar with the process and considered experts. They checked triples independently and, given that we conducted the evaluation over general purpose KGs, they have been able to use internet resources (e.g., Wikipedia) to verify the validity of the randomly selected claims.

Accuracy of Confidence Measures

Figure 5.2 reports the confidence values for 20 rules for the *spouse* predicate in DBpedia. We divided positive and negative rules in two plots and report for every rule both the confidence computed by our method and the *human confidence*. Each point on the x-axis represents an individual rule with its id, and the two bar charts associated with a rule show the confidence measure computed either manually and automatically. The mean error is 8.4% for positive rules and 10.4% for negative. Similarly, we report results for 24 rules for the predicate *foundedBy* in Figure 5.3, with 5.4% mean error for positive and 11.2% for negative rules. Both figures show a highly correlation between the manually computed confidence and the confidence computed by the proposed measures. What is most important for us is that the quality of the computed confidence for negative rules is close to the traditional computed confidence for positive ones, despite computing the quality of negative rules from data is harder.

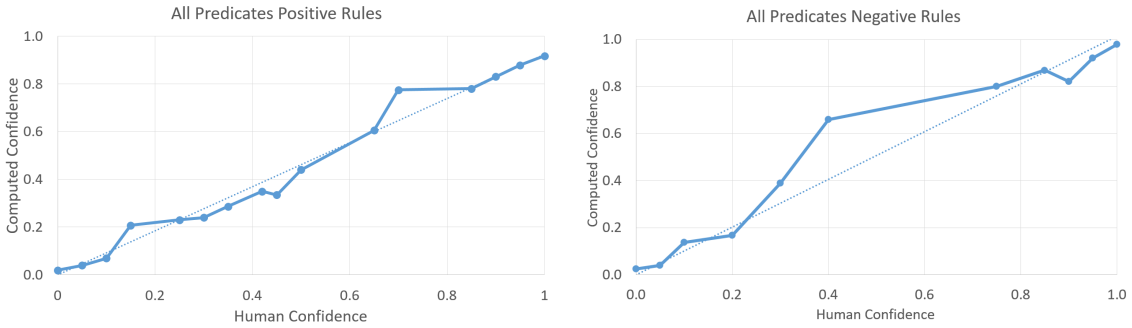
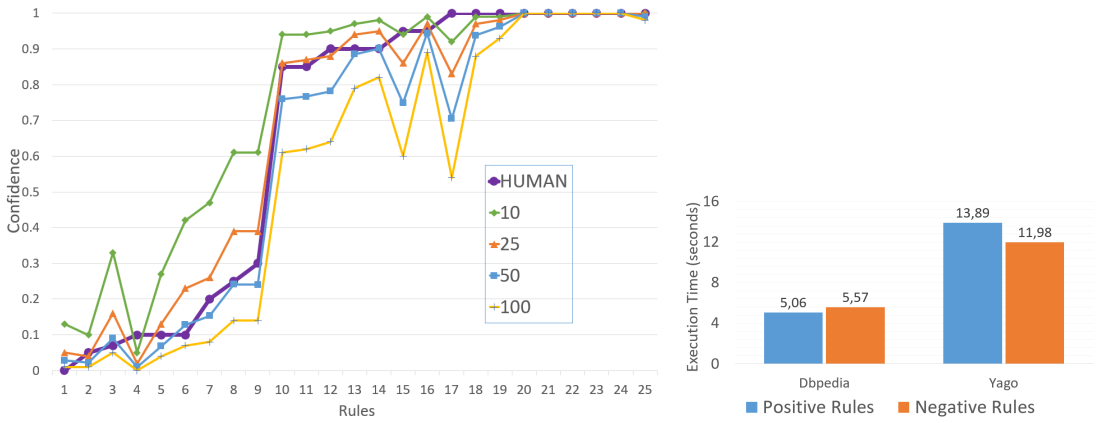


Figure 5.4 – Average computed and human confidence over rules for all predicates.

Figure 5.4 reports the computed and human confidences for all the rules over the nine predicates (DBpedia *spouse*, *foundedBy*, *relative*, *founder*, *publisher*, *employer*, *influencedBy*; Yago *isMarriedTo* and *hasChild*). We grouped rules by their human confidence and for each group we report a point. The horizontal (x) axis is the human confidence and the vertical (y) axis represents the *average* of the computed confidence for that group of rules. The trend-lines show the similar trends for computed and human confidences, with only 7.8% mean errors for positive rules and 9.0% for negatives. The aggregates results confirm the effectiveness of our measure for computing confidence. More experimental results are reported in Appendix B.



(a) (b)

Figure 5.5 – (a) Confidence for different values of κ. (b) Avg execution times for computing confidence.

Effect of the κ Parameter

In previous experiments, we used $\kappa = 25$. To show the effect of this parameter, we report results for executions with different κ values $(10, 25, 50, 100)$ for 25 negative rules over 7 predicates (We report the rules in Table B.1 in the Appendix). Figure 5.5a reports the results with the rules in increasing order of human confidence. The plot shows that the computed confidence for $\kappa = 25$ is the closest to the human confidence. By looking in more detail at rules 4, 15, and 17, we can

explain the significant difference between human confidence and computed confidence. Rule 4 is a clearly incorrect rule ($almaMater(object, v0) \wedge country(v0, v1) \wedge birthPlace(subject, v1) \wedge relative(subject, object) \rightarrow \perp$) and this is correctly reflected by its computed confidence (0.02). However, due to the sampling process on 20 triples, two spurious correct triples lead to an estimated human confidence of 0.1. Although rules 15 and 17 seem to be true, they have 30 and 11 *counter_support* triples in DBpedia, respectively. Some of these *counter_supports* are exceptions for the rule and the rest are errors. For example, rule 15 states that a person cannot be in a spouse relation with his parent's spouse. There are 30 *counter_supports* for this rule in DBpedia and by checking them we saw that 22 of them are errors and other 8 *counter_support* triples are exceptions for the rule, such as ancient monarchs and fictional characters. Considering the importance of *counter_support* in formula 5.7, these numbers reduce the confidence for these rules.

Computed Confidence Execution Time

We report the execution times of our method in computing the confidence measure. We computed confidence values for 199 positive and 307 negative rules for 4 DBpedia predicates and 36 positive and 40 negative rules for 6 Yago predicates. As reported in Figure 5.5b, average execution times are within seconds in both KGs despite we are using web APIs and more than 90% of the time goes in collecting responses. Execution times between the two KGs are not comparable as their endpoints are on different servers, they have different sizes in terms of predicates and triples, and different numbers of rules.

Impact of Quality Issues on Computing Confidence

Two main issues affect the quality of data in KGs: i) factual mistakes, such as incorrect or outdated data, and ii) incompleteness. In this experiment, we study the impact of data issues on the performance of the proposed confidence measure for negative rules. For this task, for rules we used in Section 5.1.3 (rules are reported in Table B.1), we manually identified *factual mistakes* and *missing* facts in their counter support. We then computed the confidence and observed the changes in the error rate w.r.t. the human confidence.

Factual mistakes are triples that are wrongly considered as counter support, as they are incorrect or outdated. These triples should be removed to have a correct counter support set. For example, some of the triples in the counter support of a rule (# 15) are entities in a *child* relation with themselves. As another example, another rule (#19) states that a person cannot be in a *spouse* relation with someone who died before she was born. This rule is logically correct but there are 13 real errors in the counter support for it in DBpedia.

Missing facts are triples that, based on KG relations, should be considered as counter support but because of incompleteness are not present in the KG. For example, for rule (# 10: $spouse(v0, object) \wedge parent(subject, v0) \wedge spouse(subject, object) \rightarrow \perp$), we observe in DBpedia: $spouse(Kaumualii, Deborah\_Kapule)$, with Kaumualii and Deborah_Kapule assigned to variables v0 and object, respectively; $parent(Kealiiahonui, Kaumualii)$ (subject,v0); and $spouse(Deborah\_Kapule, Kealiiahonui)$ (object,subject). The triple $spouse(Kealiiahonui, Deborah\_Kapule)$ would be counter support to this rule, but it is not in the KG, despite the pres-

ence of $parent(Kealiiahonui, Kaumualii)$ and the fact that the rule $spouse(a,b) \rightarrow spouse(b,a)$ is always true. We therefore count $spouse(Kealiiahonui, Deborah\_Kapule)$ as a missing fact for this rule.

Rule	Hum. Conf.	*C_S* 1	MF	IF	*C_S* 2	Conf. 1	Conf. 2	Conf. MF	Conf. IF
1	0	56	15	0	71	0.05	0.04	0.4	0.05
2	0.05	477	89	0	566	0.04	0.03	0.03	0.04
3	0.07	15	4	0	19	0.16	0.13	0.13	0.16
4	0.1	107	59	0	166	0.02	0.01	0.01	0.02
5	0.1	44	0	0	44	0.13	0.13	0.13	0.13
6	0.1	51	90	0	141	0.23	0.10	0.10	0.23
7	0.2	1	1	0	2	0.26	0.15	0.15	0.26
8	0.25	20	9	0	29	0.20	0.15	0.15	0.20
9	0.3	306	220	0	526	0.39	0.27	0.27	0.39
10	0.85	32	9	0	41	0.86	0.83	0.83	0.86
11	0.85	1	0	0	1	0.87	0.87	0.87	0.87
12	0.9	25	0	0	25	0.88	0.88	0.88	0.88
13	0.9	1	0	0	1	0.94	0.94	0.94	0.94
14	0.9	15	6	0	21	0.95	0.93	0.93	0.95
15	0.95	30	0	22	8	0.88	0.96	0.88	0.96
16	0.95	4	5	0	9	0.97	0.94	0.94	0.97
17	1	11	0	0	11	0.80	0.80	0.80	0.80
18	1	7	1	0	8	0.97	0.96	0.96	0.97
19	1	13	1	13	0	0.98	1	0.98	1
20	1	0	0	0	0	1	1	1	1
21	1	0	0	0	0	1	1	1	1
22	1	0	0	0	0	1	1	1	1
23	1	0	0	0	0	1	1	1	1
24	1	0	0	0	0	1	1	1	1
25	1	1	0	1	0	1	1	1	1

Table 5.2 – Negative rules information: rule #, human confidence obtained from triple annotation (Hum. Conf.), number of missing facts (MF) and incorrect facts (IF), original (*C_S* 1) and updated counter support (*C_S* 2), computed confidence before (*Conf. 1*) and after refining counter support (*Conf. 2*), computed confidence by only adding missing facts (*Conf. MF*) and by only removing incorrect facts (*Conf. IF*).

Adding missing facts and removing incorrect facts affect the counter supports of rules and therefore changes the confidence measure value. For measuring this impact, for every negative rule in this experiment (listed in Table B.1), we report in Table 5.2 its confidence before and after refining their counter supports. We manually checked every rule to compute the new counter support (*newC_S*) after adding missing facts (MF) and removing incorrect facts (IF). The original confidence measure (*Conf. 1*) and the one obtained with the new counter support (*Conf. 2*) are also reported. Identifying incorrect and missed counter supports for negative rules decreases the average error rate of the computed confidence measure w.r.t. human confidence (*Hum. Conf.*, obtained by annotating triples) from 4.3% to 3.3%.

We also report the updated confidence of every rule by only adding missing facts (*Conf. MF*) and by only removing incorrect facts (*Conf. IF*). We added 509 missing facts, which affected 13 rules (every rule has 20.4 missed facts by average), and removed 36 mistakes, which affected 3

rules (1.4 for each rule by average). By adding missing facts only to counter supports (Conf. MF), we decrease the average error rate of the computed confidence measure w.r.t. human confidence from 4.3% to 3.6%, while the average error rate *Conf. IF* is 4.0% by only removing mistakes. We observe that adding missing facts has a bigger positive impact on the confidence computation.

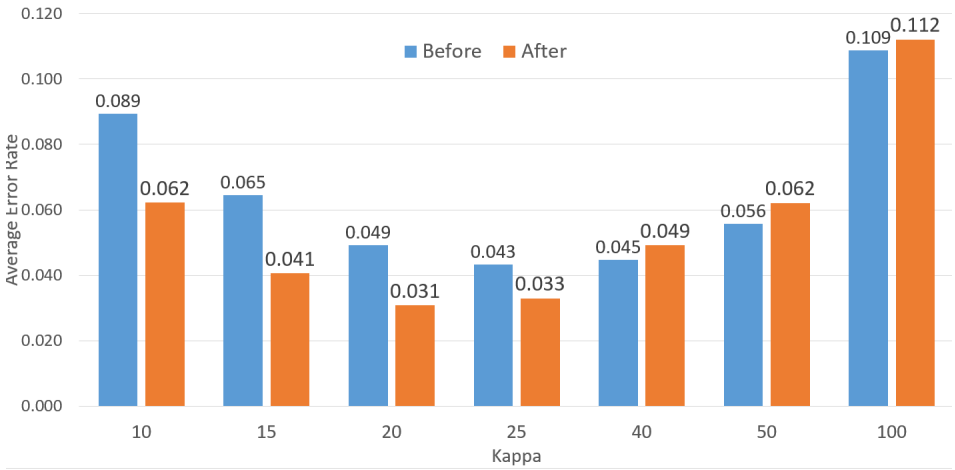


Figure 5.6 – Computed confidence error rate w.r.t. human quality with and without manual cleaning of the triples for different κ values.

The Effect of KG Cleaning on κ. As discussed in Section 5.1.1, the correct value of κ depends on the accuracy of a KG. We report the effect on the confidence computation quality when removing missing and incorrect facts with different κ values.

Results in Figure 5.6 show that cleaning counter supports has a positive impact on lower κ values while it can increase error rate for high values. Even a small amount of cleaning effectively decreases errors and incompleteness in KGs. As κ depends on KG accuracy, the results consistently show that by increasing the accuracy, a lower value for κ (20) has the minimum average error rate. Improvement can be observed for all values smaller than 25, while larger values report worse results, thus confirming that the estimate of the error rate is too big and a smaller κ value should be used.

Lessons Learned in Annotating Rules

One of the observations from our work is that evaluating rules is a non trivial task. In fact, we advocate that, when possible, rules should be evaluated by looking and annotating the output triples as true or false. Even with a small number of randomly sampled triples, the estimated confidence is usually more reasonable than a human quality evaluation.

On the other hand, it is much faster to come up with a subjective measure of the quality of a rule by looking at its logical form. This can be effective for some rules, but more difficult for others. We conducted experiments on quality evaluation to find more evidence of the gap between the two measures and to better understand what are the more complicated cases for annotators.

We recruited three graduate students, not involved in this work and not familiar with KGs. We gave them the task to assign a *quality evaluation* score to 20 (positive and negative) rules.

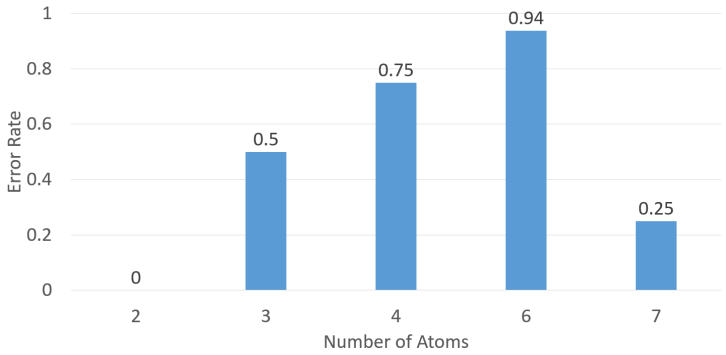


Figure 5.7 – Impact of number of rules atoms on quality evaluation annotations.

Each annotator assigned individually a score from 1 to 5 to every rule, where 5 is a completely incorrect rule and 1 is a correct one (see Section 5.1.2).

Experimental results show a good correlation between the average of the quality evaluation score from the non-expert annotators and the human confidence values based on triples annotations. After a more detailed analysis of the results, the triple annotation based method is clearly closer to the correct evaluation. In fact, quality evaluation scores are not always in agreement and the Kendall's tau-b correlation [161] score over the 20 rules is 0.49 as annotators had different understanding of some of the non-trivial rules.

There are multiple factors that affect the understanding of a rule. An important feature of rules is their type. In our experiments, quality evaluation annotators had a lower error rate in evaluating positive rules (0.50) in comparison to negative ones (0.72). Another factor is the number of atoms in the rule. Figure 5.7 shows that rules with a higher number of atoms seem to be more difficult to evaluate. The low error rate in Figure 5.7 is explained by the fact that only two rules have 7 atoms and both of them are positive.

5.2 Extracting Logical Rules from Wikidata

Wikidata is a knowledge base (KB) representing data with a large collections of interconnected entities. Wikidata supports all kinds of applications and it is the structured data storage for its Wikimedia sister projects.

A benefit of a KB is the ability to define *constraints* over it. Unfortunately, despite the positive impact of users contributing to the coverage and the quality of Wikidata information, rich logical rules are not part of this effort. Property constraints[2] have been defined for over 8K items[3], but these are mostly syntactic checks defined over the value of a property, such as the fact that an IMDb ID should follow a particular regular expression.

We are interested in soft (approximate) constraints expressed as dependencies (or *logical rules*), such as the constraint that "a person cannot be born after one of her children". Such rules have proven to be useful for error detection [79], adding missing facts [84], executing queries faster, and reasoning [52].

[2] https://www.wikidata.org/wiki/Wikidata:WikiProject_property_constraints
[3] https://w.wiki/YLS

Table 5.3 – Examples of rules mined on Wikidata. We report between square brackets the label (e.g., spouse) for the Wikipedia item IDs (e.g., P26) to favor readability.

	Rule	*C*	*H*	*Q*
Positive	$P185[doctoralStudent](o,s) \rightarrow P184[doctoralAdvisor](s,o)$	1	1	1
	$P1196[mannerOfDeath](s, Q171558[accident]) \rightarrow P509[causeOfDeath](s, Q171558)$	1	1	1
	$P22[father](o,s) \rightarrow P40[child](s,o)$	.99	1	1
	$P25[mother](o,s) \rightarrow P40(s,o)$	.99	1	1
	$P26[spouse](o,s) \rightarrow P26(s,o)$	.99	1	1
	$P40[child](s,v) \wedge P40(o,v) \rightarrow P26(s,o)$	.88	.9	1
	$P800[notableWork](o,s) \rightarrow P170[creator](s,o)$	.72	.7	1
Negative	$P40(o,v) \wedge P25[mother](v,s) \wedge P40(s,o) \rightarrow \bot$	.94	1	1
	$P1038[relative](s,o) \wedge P40(s,o) \rightarrow \bot$	.93	1	1
	$P25(o,v_0) \wedge P26(v_1,v_0) \wedge P112[foundedBy](s,v_1) \wedge P112(s,o) \rightarrow \bot$	.89	1	3
	$P180[depicts](s,o) \wedge P170[creator](s,o) \rightarrow \bot$	.32	.2	3

Not only these rules are not stated in Wikidata, but, to the best of our understanding, a way to express them as constraints is still to be defined in the repository. Moreover, even if primitives get exposed to the users for this task, manually crafting such rules is difficult as it requires both domain and technical expertise. Finally, once a set of semantically valid rules has been identified, these are usually annotated with a measure of their quality, such as a *confidence* of the rule applicability. This is necessary, as there are very few rules that are exact, i.e., true for each and every case. As an example, consider a rule stating that "a country has always one capital". This is true for most countries, but there are 15 countries that have two or more capitals. Therefore, the rule has a very high confidence, but it is not exact. Confidence not only is key to rank rules for user validation and refinement, but it is also used in applications that rely on reasoning with soft constraints.

Collecting a set of high quality rules is an essential but challenging task to curate Wikidata and to improve the performance of the applications built on it. The goal of this Section is to create a large collection of rules for Wikidata with their confidence measure. In this abstract, we report on two directions we have been exploring to obtain such rules, our results, and how we believe the Wikimedia community could benefit from this effort.

5.2.1 Searching Logical Rules

We first introduce logical rules and then describe two methods that we are evaluating for collecting Wikidata rules. The first one is a data mining approach, while the second one is based on the idea of translating rules from an existing corpus of DBpedia rules.

Logical Rules

We consider two kinds of rules. The first kind are *positive rules*, such as the first rule in Table 5.3, which identify relationships between entities, e.g., "if someone is the doctoral student of a second

Table 5.4 – Examples of DBpedia rules translated to Wikidata.

	Rule	*C*	*H*	*Q*
Positive	$P40[child](o,v) \wedge P25[mother](v,s) \rightarrow P40(s,o)$	.94	1	1
	$P1038[relative](o,s) \rightarrow P1038(s,o)$	.6	1	1
	$P144[basedOn](v_0,o) \wedge P144(v_0,v_1)$ $\wedge P50[author](s,v_1) \rightarrow P144(s,o)$	.7	.75	2
	$P287[designedBy](s,v_0) \wedge P287(v_1,v_0)$ $\wedge P408[softwareEngine](v_1,o) \rightarrow P408(s,o)$	.4	.5	2
	$P166[awardReceived](s,v_0)$ $\wedge P118[league](v_1,v_0) \wedge P166(v_1,o) \rightarrow P166(s,o)$	.25	.3	4
Negative	$P569[dateBirth](s,v_0) \wedge P570[dateDeath](o,v_1)$ $\wedge > (v_0,v_1) \wedge P26(s,o) \rightarrow \bot$	1	1	1
	$P26[spouse](o,v) \wedge P26(s,v) \wedge P26(s,o) \rightarrow \bot$	.99	1	1
	$P185[doctoralStudent](s,o) \wedge P185(o,s) \rightarrow \bot$	.95	1	1
	$P144(s,o) \wedge P86[composer](s,o) \rightarrow \bot$	.95	1	2

person, then the second person is her advisor“, or “if two persons have a child in common, they are in the spouse relation”. The second kind are *negative rules*, with $\bot$ in the conclusion, which identify data contradictions, e.g., “if two persons are in the relative relation, one cannot be the spouse of the other”. A fact, or a contradiction, is derived from a rule if all the variables in the premise of the rule can be replaced with constants from the KB.

Rule Mining

Several mining methods have been proposed to identify rules in large KBs [79, 84]. These approaches are effective, but computationally expensive and leave to the user the selection and the refinement of the mined rules. For this approach, we use a state of the art method for mining declarative rules over RDF KBs [79].

We use a RDF dump of Wikidata, which has been stripped of metadata such as qualifiers and references to other KBs.

We mined 80 positive and negative rules and report a sample in Table 5.3.

Translating DBpedia Rules

In this method, we convert the rules that have been mined over the DBpedia KB and are stored in Rulehub. For every DBpedia rule, we translate its predicates into the equivalent Wikidata properties. For example, property *P*184 in Wikidata corresponds to predicate *DoctoralAdvisor* in DBpedia. We use the *owl:equivalentProperty* information to generate a mapping between the two KBs for *59* properties. With this method, we obtained *241* Wikidata rules. A sample of these rules is shown in Table 5.4.

5.2.2 Experiments

Our experiments show how we (i) verify that we obtain rules of good quality and (ii) estimate effectively the confidence of the rules. For measuring rule confidence and quality, we use three

Table 5.5 – Examples of rules with the # of missing (top) and incorrect (bottom) statements detected by every rule in Wikidata.

	Rule	*# stms*
Posit.	$P1038[relative](s,o) \rightarrow P1038(o,s)$	13,690
	$P40[child](o,v_0) \wedge P25[mother](v_0,s) \rightarrow P40(s,o)$	226
	$P185[doctoralStud.](o,s) \rightarrow P184[doctoralAdv.](s,o)$	25
Negat.	$P569(s,v_0) \wedge P570(o,v_1) \wedge > (v_0,v_1) \wedge P26(s,o) \rightarrow \bot$	689
	$P22[father](o,s) \wedge P40(s,o) \rightarrow \bot$	41
	$P185(o,s) \wedge P185(s,o) \rightarrow \bot$	17
	$P25(o,s) \wedge P40(s,o) \rightarrow \bot$	6

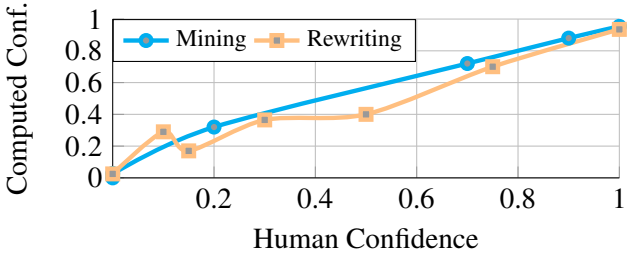


Figure 5.8 – Human and computed confidence comparison.

metrics presented in 5.1.2: *Computed Confidence*, *Human Confidence* and *Quality Evaluation* (**Q**).

Results

As reported in the examples in Tables 5.3 and 5.4, both methods identify rules with high confidence. We report also examples of rules with lower confidence values to show that the computed confidence correlates nicely with measures based on human annotation. We also report in Figure 5.8 the computed and human confidences for 40 rules over 18 properties. We grouped rules by their human confidence (H) and for each group we report a point. The horizontal (x) axis is the human confidence and the vertical (y) axis represents the *average* of the computed confidence (C) for that group of rules. The plot shows similar correlations between computed and human confidences for rules obtained with both methods.

Finally, we report in Table 5.5 the number of missing and incorrect statements (stms) identified by a sample of exact positive and negative rules, respectively. For example, the first positive rule identifies almost 14k statements when the rule is instantiated but the conclusion is not in the KB. Similarly, the first negative identifies 689 statements in which the subject is in the spouse relationship with an object who died before the subject was born.

5.3 Teaching soft rules to language models

Pre-trained language models (PLMs) based on transformers [110, 162] are established tools for capturing both linguistic and factual knowledge [163, 164]. However, even the largest models fail on basic reasoning tasks. If we consider common relations between entities, we see that

Input facts:
Mike is the parent of **Anne**. **Anne** lives with Mark. **Anne** is the child of Laure. **Anne** lives with *Mike*.
Input rules:
(r_1, .1) Two persons living together are married.
(r_2, .7) Persons with a common child are married.
(r_3, .9) Someone cannot be married to his/her child.
(r_4, 1) Every person is the parent of his/her child.

Test 1: Laure and *Mike* are married.
Answer: *True* with probability 0.7 [r_4, r_2]

Test 2: **Anne** and Mark are married.
Answer: *False* with probability 0.9 [r_1]

Test 3: **Anne** and *Mike* are married.
Answer: *False* with probability 0.9 [r_1, r_3, r_4]

Figure 5.9 – Examples of hypotheses that require reasoning using facts and possibly conflicting soft rules (rule id and confidence shown in brackets).

such models are not aware of negation, inversion (e.g., *parent-child*), symmetry (e.g., *spouse*), implication, and composition. While such relation properties are obvious to a human, they are challenging to learn from text corpora as they go beyond linguistic and factual knowledge [165, 166].

We claim that such reasoning primitives can be transferred to the language models by leveraging logical rules, such as those shown in Figure 5.9. In this section, we show how to reason over soft logical rules with PLMs. We provide facts and rules expressed in natural language, and we ask the PLM to come up with a logical conclusion for a hypothesis, together with the probability for it being true. Our model can even reason over settings with conflicting evidence, as shown in Test 3 in Figure 5.9. In the example, as Anne and Mike live together, they have a 0.1 probability of being married because of soft rule r_1. However, we can derive from exact rule r_4 that Anne is the child of Mike and therefore they cannot be married, according to soft rule r_3.

To model uncertainty, we pick one flavor of probabilistic logic programming languages, LP^{MLN}, for reasoning with soft rules [167]. It assigns weights to stable models, similarly to how Markov Logic assigns weights to models. However, our method is independent of the logic programming approach at hand, and different models can be fine-tuned with different programming solutions. Our proposal makes use of synthetic examples that "teach" the desired formal behavior through fine-tuning. In particular, we express the uncertainty in the loss function used for fine-tuning by explicitly mimicking the results for the same problem modeled with LP^{MLN}.

The data and the code for our experiments, as well as the resulting fine-tuned models are available online at `http://github.com/MhmdSaiid/RuleBert`

5.3.1 Dataset Generation

Each example is a triple *(context, hypothesis, confidence)*. *Context* is a combination of rule(s) and generated facts, such as "*If the first person is living together with the second person, then the first person is the spouse of the second person.*" and "*Anne lives with Mike.*" *Hypothesis* is the statement to be assessed based on the context, e.g., "*Laure is the spouse of Mike.*" *Confidence* is the probability that the hypothesis is valid given by the reasoner, e.g., 0.7. As we generate the examples, we know the confidence for each hypothesis.

Given a rule, we generate examples of different hypotheses to expose the model to a variety of contexts. Each example contains the context c and a hypothesis h with its probability of being true as obtained for the (c,h) pair from the LP$^{\text{MLN}}$reasoner. The intuition is that the examples show the expected behavior of a formal reasoner for every combination of possible facts for a given rule. This process is not about teaching the model specific facts to recall later, but teaching it reasoning patterns.

Unlike previous work [168], our rules allow for multiple variables. This introduces additional complexity as the examples must show how to deal with the symmetry of the predicate. For example, the facts *child(Alice,Bob)* and *child(Bob,Alice)* are not equivalent, since *child* is a non-symmetric predicate, while *spouse(Alice,Bob)* and *spouse(Bob,Alice)* are equivalent, as *spouse* is symmetric. We assume that metadata about the symmetry and the types is available from the KB for predicates in the rules. Given as input (*i*) a rule r, (*ii*) a desired number n of examples, (*iii*) an integer m to indicate the maximum number of facts given as a context, and (*iv*) a pool of values for each type involved in r's predicate *pools*, Algorithm 8 outputs a dataset D of generated examples.

We start at line 3 by generating facts, such as *child(Eve,Bob)*, using the function *GenFacts* (lines 15–18), which takes as input r, m, and the *pools*. A random integer less than m sets the number of facts in the context. The generated facts F have predicates from the body of r, their polarity (true or negated atom) is assigned randomly, and variables are instantiated with values sampled from the pool (line 16). Facts are created randomly, as we are not interested in teaching the model specific facts to recall later, but we are rather teaching it how to reason with different combinations of rules and facts. We then ensure that the rule is triggered in every context, eventually adding more facts to F with the function *GetRuleFacts* in line 17. After obtaining F, we feed rule r along with facts F to the LP$^{\text{MLN}}$reasoner, and we obtain a set O containing all satisfied facts and rule conclusions (line 4).

We generate different hypotheses, where each one leads to an example in dataset D. For each context, we add an example with different facts according to three dimensions. A fact can be (*i*) for a predicate in the premise or in the conclusion of a rule, could be (*ii*) satisfied or unsatisfied given the rule, and could have (*iii*) positive or negative polarity. This results in the generation of eight different hypotheses.

The first hypothesis h_1 is obtained by sampling a fact from the set F (line 5). We then produce the counter hypothesis h_2 by altering the fact (line 6) with the function *Alter* (lines 19-22). Given a hypothesis $p(s,o)$ (line 19), we return its negated form if p is symmetric (line 20). Otherwise, if p is not symmetric, we produce a counter hypothesis either by negation (line 21), or by switching the subject and the object in the triple as the predicate is not symmetric (line 22). We rely on a dictionary to check whether a predicate is symmetric or not.

Algorithm 8: Generate Synthetic Data

Input: rule r ; // child(a,b)→parent(b,a)
n ; // # of examples
m ; // max # of facts
pools ; // pools of names
Output: Generated Dataset D
1 $D = \{\}, i = 1$; // initialize
2 **while** $i \leq ceiling(n/8)$ **do**
3 $F = GenFacts(r, m, pools)$; // child(Eve,Bob),parent(Amy,Sam)
4 $O = LPMLN(r, F)$; // reasoner output
5 $h_1 = f \in F$; // child(Eve,Bob)
6 $h_2 = Alter(f)$; // negchild(Eve,Bob)
7 $h_3 = r(F)$; // parent(Bob,Eve)
8 $h_4 = Alter(r(F))$; // parent(Eve,Bob)
9 $h_5 = \text{pos}.f_l \notin F$; // child(Joe,Garry)
10 $h_6 = \neg h_5$; // negchild(Joe,Garry)
11 $h_7 = f_r \notin O$; // parent(Alice,Joe)
12 $h_8 = \neg h_7$; // negparent(Alice,Joe)
13 $D.add(h_{1-8})$;
14 $i \leftarrow i + 1$;
15 **Function** GenFacts(*r,m,pools*)**:**
16 $F = GetRandomFacts(r, pools, m)$;
17 $F.add(GetRuleFacts(r, pools))$;
18 **return** F
19 **Function** Alter($p(s,o)$)**:**
20 **if** *p is symmetric* **then return** $\neg p(s,o)$;
21 **if** *random()>0.5* **then return** $\neg p(s,o)$;
22 **else return** $p(o,s)$;

We then produce hypothesis h_3 (line 7), which is the outcome of triggering rule r with the facts added in line 17. The counter hypothesis h_4 is generated by altering h_3 (line 8). Moreover, we generate hypothesis h_5 by considering any unsatisfied positive fact outside F. Following a closed-world assumption (CWA), we assume that positive triples are false if they cannot be proven, meaning that their negation is true. We sample a fact f_l from the set of all possible positive facts that do not have the same predicate of the rule head (line 9). Thus, h_5 will never be in the output O of the reasoner, as it cannot be derived. We then produce h_6 by negating h_5 in line 10. We further derive h_7 by sampling a fact f_r that has the same predicate as that of the rule head, but does not belong to the output of the reasoner O (line 11). For a positive (negative) rule, such a fact is labelled as False (True). h_7 is then negated to get the counter hypothesis h_8 (line 12). All are added to D (line 13), and the process is repeated until we reach n examples.

Finally, we automatically convert the examples to natural language using predefined templates for the facts and the rules. A basic template for atom predicate p (type t_1, type t_2) is: "If the 1^{st} t_1 is p of the 2^{nd} t_2." ("If the first person is spouse of ...").

For the single-rule scenario, we release a dataset generated for 161 rules with a total of 3.2M examples, together with a partition of 80%:10%:10% split for training, validation, and testing.

5.3.2 Teaching PLMs to Reason

Uncertainty stems from the rule confidence. One approach to teach how to estimate the probability of a prediction is to treat each confidence value (or bucket of confidence values) as a class and to model the problem as a k-way classification instance (or regression), but this is intractable when multiple rules are considered. Instead, we keep the problem as a two-class one by altering how the information is propagated in the model to incorporate uncertainty from the rule confidence.

Let $D = \{(x_i, y_i)\}_{i=1}^{m}$ be our generated dataset, where x_i is one example of the form *(context,hypothesis,confidence)* and y_i is a label indicating whether the hypothesis is validated or not by the context (facts and rules in English), and m is the size of the training set. A classifier f is a function that maps the input to one of the labels in the label space. Let $h(x, y)$ be any classification loss function. The empirical risk of the classifier f is defined as

$$R_h(f) = \mathbb{E}_D(h(x,y)) = -\frac{1}{m}\sum_{i=1}^{m} h(x_i, y_i)$$

We want to introduce uncertainty in our loss function, using the weights computed by the LP$^{\text{MLN}}$solver as a proxy to represent the probability of predicting the hypothesis as being true. To do so, we apply a revised empirical risk:

$$R'_h(f) = \mathbb{E}_D(h(x,y)) = -\frac{1}{m}\sum_{i=1}^{m} (w(x_i) * h(x_i, 1) + (1 - w(x_i)) * h(x_i, 0))$$

where $w(x_i)$ is the probability of an example x_i being True. We now state that each example is considered as a combination both of a weighted positive example with a weight $w(x_i)$ provided by the LP$^{\text{MLN}}$solver and a weighted negative example with a weight $1 - w(x_i)$.

When trained to minimize this risk, the model learns to assign the weights to each output class, thus predicting the confidence for the true class when given the satisfied rule head as hypothesis.

5.3.3 Experiments

We first introduce the experimental setup (Section 5.3.3). We then evaluate the model on single rules (Section 5.3.3). We also show that a PLM fine-tuned on soft rules, namely RULEBERT, makes accurate prediction for unseen rules (Section 5.3.3).

Experimental Setup

Rules. We use a corpus of 161 soft rules mined from DBpedia. We chose a pool of distinct rules with varying number of variables, number of predicates, rule conclusions, and confidences.
Reasoner. We use the official implementation[4] of the LP$^{\text{MLN}}$reasoner. We set the reasoner to compute the exact probabilities for the triples.
PLM. We use the HuggingFace pre-trained *RoBERTa$_{LARGE}$* [162] model as our base model, as it is trained on more data compared to *BERT* [110], and is better at learning positional embeddings [169]. We fine-tune the PLM[5] with the weighted binary cross-entropy (wBCE) loss from Section 5.3.2. More details can be found in Appendix D.2.

[4] http://github.com/azreasoners/lpmln
[5] Prompt has form: *<s>context</s></s>hypothesis</s>*.

Rule	Conf.	*RoBERTa-wBCE* Acc.	CA@k .10	CA@k .01	*RoBERTa* Acc.	CA@k .10	CA@k .01
birthYear(a,c) ∧ deathYear(b,d) ∧ >(c,d)→negspouse(a,b)	.990	.995	.993	.993	.970	.490	.486
birthYear(b,d) ∧ foundYear(a,c) ∧ <(c,d)→negfounder(a,b)	.990	.928	.927	.927	.908	.486	.456
spouse(c,a) ∧ parent(b,c) → negspouse(a,b)	.923	.974	.963	.747	.875	.491	.279
relative(a,c) ∧ spouse(b,c) ∧ child(b,a) → relative(a,b)	.860	.922	.844	.801	.866	.342	.146
parent(c,a) ∧ child(b,c) → spouse(a,b)	.825	.944	.828	.444	.842	.342	.146
publisher(c,b) ∧ subsequentWork(c,a) → publisher(a,b)	.721	.909	.834	.765	.905	.358	.219
successor(b,a) → negspouse(a,b)	.718	.972	.896	.693	.949	.369	.313
child(c,b) ∧ relative(c,a) → negchild(a,b)	.644	.935	.880	.693	.905	.310	.303
child(c,b) ∧ spouse(a,c) → negrelative(a,b)	.562	.920	.907	.608	.915	.255	.250
relation(a,b) → negchild(a,b)	.549	.904	.886	.737	.902	.371	.366
child(c,b) ∧ spouse(c,a) → child(a,b)	.492	.901	.827	.422	.658	.223	.107
knownFor(b,a) → founder(a,b)	.387	.882	.601	.477	.839	.372	.215
founder(c,b) ∧ publisher(c,a) → negfounder(a,b)	.246	.886	.795	.665	.802	.311	.297
publisher(a,c) ∧ parentCompany(b,c) → negpublisher(a,b)	.235	.812	.748	.643	.811	.313	.271
successor(c,a) ∧ spouse(c,d) ∧ successor(d,b)→spouse(a,b)	.221	.927	.738	.628	.761	.248	.215
relative(a,c) ∧ parent(c,b) → child(a,b)	.135	.841	.704	.552	.727	.227	.182

Table 5.6 – Evaluation results for single-rule models.

Evaluations Measures. For the examples in the test set, we use accuracy (Acc) and F1-score (F1) for balanced and unbalanced settings, respectively. As these measures do not take into account the uncertainty of the prediction *probability*, we further introduce Confidence Accuracy@k (CA@k), which measures the proportion of examples whose absolute error between the predicted and the actual probabilities is less than a threshold k:

$$CA@k = \frac{\#\{x_i, |w_i - \hat{w}_i| < k\}}{\#\{x_i\}}$$

where x_i is the ith example of dataset, w_i is the actual confidence of the associated hypothesis given by the LP$^{\text{MLN}}$reasoner, $\hat{w}_i$ is the predicted confidence by the model, and k is a chosen threshold.

The measure can be seen as the ordinary accuracy measure, but true positives and negatives are counted only if the condition is satisfied, where lower values for k indicate stricter evaluation.

Single Soft Rule

We fine-tuned 16 models for 16 different positive and negative rules (one model per rule) using 16k training samples per rule. We compare the accuracy of each model (*i*) without teaching uncertainty using binary cross-entropy (RoBERTa), and (*ii*) with teaching soft rules using wBCE.
Results. Every row in Table 5.6 contains a rule with its confidence, followed by accuracy and CA@k for both loss functions. The results show that models fine-tuned using *RoBERTa-wBCE* perform better in term of CA@k. In terms of *accuracy*, both models perform well, with *RoBERTa-wBCE* performing better for all rules. Interestingly, the best performing rules are two rules that involve comparison of numerical values (birth years against death and founding years), which suggests the our method also handles comparison predicates.

	Rule	*FT-PLM*	RULEBERT$_{20}$	*FT-*RULEBERT$_{20}$
Known preds	child(a,b) → parent(b,a)	.719	.869	.989
	relative(a,b) → negspouse(b,a)	.885	.885	.963
	child(a,b) ∧ child(b,c) → negchild(a,c)	.835	.888	.918
	parent(a,b) ∧ parent(a,c) → spouse(b,c)	.754	.757	.814
	parent(a,b) → negchild(a,b)	.923	.933	.963
Unknown preds	knownFor(b,a) → founder(a,b)	.817	.795	.971
	worksFor(b,a) → negfounder(a,b)	.951	.915	.952
	occupation(a, b) → negalmaMater(a, b)	.939	.917	.972
	author(c,b) ∧ series(a,c) → author(a,b)	.965	.937	.989
	city(a,b) → negstate(a,b)	.923	.912	.971

Table 5.7 – Accuracy results for unseen rules. The first group contains rules with predicates seen by RULEBERT in the 20 rules used in fine-tuning, while the second group has rules with unseen predicates.

Testing RULEBERT on Unseen Rules

We have seen that a PLM can be successfully fine-tuned with rules. We now study the performance on the PLM after it has been fine-tuned on 161 (single) rules. We call this fine-tuned model, RULEBERT.

We start by showing the performance of RULEBERT on rules that have not been seen in the fine-tuning. We fine-tune our model with only twenty randomly selected rules and call it RULEBERT$_{20}$. We then select ten new rules divided into two groups: (*i*) five rules that contain predicates that are in the rules used in fine-tuning RULEBERT$_{20}$, and (*ii*) five rules completely unseen by the model (they do not share predicates with the rules of the model). For each rule in the test sets, we run a model fine-tuned (with 4k examples) only for that rule (FT-PLM), the model fine-tuned on the twenty original rules (RULEBERT$_{20}$), and the same model fine-tuned again for the rule at hand (FT-RULEBERT$_{20}$).

Results. Table 5.7 shows that RULEBERT$_{20}$ outperforms the fine-tuned model (FT-PLM) on the first group. The model fine-tuned on 20 rules has enough information about (*i*) symmetrical/transitive predicates and (*ii*) rule confidence to predict correctly, even better than rule-specific models. For the second rule group, the accuracy of RULEBERT$_{20}$ is high, but FT-PLM performs better. Applying the same fine-tuning on RULEBERT$_{20}$ yields the best results in all scenarios.

5.4 Summary

In Section 5.1, we introduced *RuleHub*, a system managing an open corpus of more than 8,000 rules collected from different systems for three KGs. Our system exposes a web portal to the users to let them retrieve rules for their tasks, add new rules, and refine and annotate rules in the corpus. We believe that metadata play a key role to make rules really effective. Towards this goal, we introduce a new confidence measurement to evaluate the quality of negative rules. An experimental comparison against human computed confidences show that our measure gives valid quality estimations. With the continuous development of KGs as well as the evolution of rule

mining techniques, we believe that *RuleHub* will play an important role in collective storing and managing of KG rules. In Section 5.2, we described two methods for collecting logical rules for Wikidata, and our experiments showed that both of them can generate high quality rules. While our effort shows that good rules can be gathered for Wikidata, we still need to make progress to be able to fully transfer this knowledge into the KG.

In Section 5.3, we have shown that PLMs can be taught to reason with soft rules over natural language, and conducted multiple experiments to evaluate the ability of proposed model in tasks such as negation and unseen rules.

Chapter 6

Explainable Fact Checking using Logical Rules

Due to the increase of sources spreading false information, computational *fact checking* has been proposed to support journalists and social media platforms with automatic verification of textual content [170]. We focus on *claims* that contain factual statements, such as "William Durant was the founder of Chevrolet," and their verification against reference data, i.e., *Knowledge Graphs* (KGs). Assuming entities and relations involved in "worth-checking" claims have been identified [171, 172], KGs are exploited to compute the *veracity* of claims expressed as structured data.

A KG is a structured representation of information which stores real-world entities as nodes, and relationships between them as edges. Entities and relations have semantic descriptions in the form of types and properties associated with them. KGs store large amounts of factual information and several of them are publicly available [5]. For example, the English version of DBpedia stores 6M entities and 9B relation triples.

Given a KG K and a claim f, several approaches have been developed to estimate if f is a valid claim in K. In some of these methods, facts in the KG are leveraged to create features, such as paths [173, 174] or embeddings [86, 175], which are then used by classifiers to label as true or false a given test claim. Other methods rely on searching for occurrences of the given claim on Web pages [14, 176]. However, such models are based on Machine Learning (ML) classifiers that in the best case can report the source of evidence for a decision but *lack the ability to provide comprehensible descriptions* of how a decision has been taken for a given claim.

To address this problem and effectively support transparent content moderation, we use existing KGs as sources of evidence, together with logical reasoning to make fact checking

```
0.75: foundedBy(a,b) ← keyPerson(a,b), foundedBy(c,b), product(c,d), product (a,d).
0.76: foundedBy(a,b) ← distributor(c,b),distributor(c,d),foundedBy(a,d).
0.97: negfoundedBy(a,b) ← foundingYear(a,c), birthYear(b,d), >(d,c).
0.56: negfoundedBy(a,b) ← foundedBy(a,c),relative(d,c),occupation(d,b).
0.67: negfoundedBy(a,b) ← parentCompany(b,c), subsidiary(c,d), parentCompany(d,a).
```

Table 6.1 – Example of discovered rules with their support for predicate *foundedBy* in DBpedia.

decisions. The key idea is to assess as true or false a given claim and to provide *human-interpretable explanations* for such decision in the form of supporting and contradicting evidence. Declarative Horn rules defined over the KG, such as those in Table 6.1, guide the decision process and provide semantic arguments for the conclusion. For example, "William Durant was the founder of Chevrolet" is marked as true and justified by the facts that Durant is a *key person* for Chevrolet and he *founded* another car company. This explanation comes from the first rule in the table[1]. On the other hand, "Elon Musk was the founder of Chevrolet" is marked as false with the explanation that Musk was born after the company foundation year (third rule in the table).

Unfortunately, two issues make the generation of such explanations hard. First, in general, KGs do not come with the rich rules we need in our task. To address this issue, we exploit rule mining approaches [28, 79], which automatically learn logical rules for a given KG (e.g., Table 6.1). Second, KGs have data quality issues due to the automatic methods that are used to build them at scale. Information stored in KGs is inevitably incomplete (Open World Assumption - OWA) and noisy, because of errors coming from the sources and the automatic extractors [14]. For these reasons, in many cases, rules cannot be triggered. We identify these cases and resort to mining Web pages to get evidence for missing facts that are crucial to reach a decision for a claim [176].

Discovering rules and mining facts enable a fully automatic system, but a new challenge arises from these approaches. Both rules and mined facts are *uncertain*, i.e., they come with a (possibly low) measure of the probability of being correct. To address this third challenge, we use probabilistic answer set programming [167]. The reasoner is the enabler of the inference that combines the evidence in producing a fact checking decision with its explanation.

6.1 Preliminaries

We employ knowledge graphs (see Chapter 2) and logical rules extracted from them (see Chapters 4 and 5) as the main elements of our framework. In this section, we describe other building blocks of our framework and define our problem.

Assessment of Claims on the Web. As KGs are usually incomplete, we also exploit textual documents for our analysis. Text mining systems get as input a claim c expressed in natural language and analyze c's credibility w.r.t. relevant Web documents. The systems exploit the joint interaction among language style of documents, their stance towards a claim, and source trustworthiness.

For example, consider the claim *foundedBy*(Chevrolet, W. Durant), which is not in the KG, and positive rule from Table 6.1: *foundedBy(a,b)* ← *keyPerson(a,b), foundedBy(c,b), product(c,d), product(a,d)*. Assume the KG contains the facts *keyPerson*(Chevrolet, W. Durant), *foundedBy*(GM, W. Durant), and *product*(GM, Automobile), but it misses the product information for Chevrolet. It can be a false fact or a true one missing from the KG (OWA). We therefore test *product*(Chevrolet, Automobile) with the text mining system and obtain that, according to Web documents, the fact is true with confidence 0.57.

In our framework, we adopt CREDEYE, a state of the art system for the automatic credibility assessment of textual claims [177]. To extract Web articles relevant to the input claim, it uses

[1] *Product* models the pairs (company **c**, product **p** of **c**).

a commercial search engine (i.e., Bing). Each document is divided into a set of overlapping snippets, and snippets that are strongly related to the claim in terms of unigram and bigram are extracted. Snippets are then used to compute *support* and *refute* scores with logistic regression classifiers trained on claims and evidence documents from the Snopes fact checking repository. The scores are fed as features into a classifier with L1-regularization, distantly trained on Snopes.

(Probabilistic) Answer Set Programming. Given a claim, we collect the rules, the evidence (from the KG and the Web sites), and cast fact checking as a reasoning problem. For this task, we adopt LP$^{\text{MLN}}$ [167], a probabilistic extension of answer set programs with the concept of weighted rules from Markov Logic. In ASP, search problems are reduced to computing stable models (a.k.a. answer sets), a set of beliefs that are described by the program. In the case of a Horn program, the stable models coincide with the minimal models, but they differ as soon as the program allows more expressive language constructs, such as negation, defaults, and aggregates. We refer to [178–180] for the definitions of stable models.

LP$^{\text{MLN}}$extends the (deterministic) stable model semantics by embracing the concept of weighted rules. In LP$^{\text{MLN}}$, a weight is assigned to each rule so that the more rules a stable model satisfies, the larger weight it gets, and the probability of the stable model is computed by normalizing its weight among all stable models. In our setting, given a set of rules and evidence facts, we want to see if the given claim belongs to the stable model.

More precisely, let σ be a signature as in first-order logic. An LP$^{\text{MLN}}$program Π is a finite set of weighted rules of the form:

$$w : A \leftarrow B \tag{6.1}$$

where A is a disjunction of atoms of σ, B is a conjunction of literals (atoms and negated atoms) of σ, and w is a real number or the symbol α. When A is $\bot$ (the empty disjunction), the rule asserts that B should be false in the stable model. An LP$^{\text{MLN}}$rule (6.1) is called soft if w is a real number or hard if w is α. An LP$^{\text{MLN}}$program is *ground* if its rules contain no variables. An LP$^{\text{MLN}}$program Π that contains variables is identified with a ground LP$^{\text{MLN}}$program $gr_\sigma[\Pi]$ which is obtained from Π by replacing every variable with every ground term of σ. The weight of a ground rule in $gr_\sigma[\Pi]$ is the same as the weight of the corresponding rule in Π. By $\overline{\Pi}$ we denote the unweighted logic program obtained from Π, i.e., $\overline{\Pi} = \{R \mid w : R \in \Pi\}$.

For a ground LP$^{\text{MLN}}$program Π, Π_I denotes the set of rules $w : R$ in Π such that I satisfies R (denoted $I \models R$) and SM$[\Pi]$ denotes the set $\{I \mid I$ is a (deterministic) stable model of $\overline{\Pi_I}\}$. The (unnormalized) weight of I under Π is defined as:

$$W_\Pi(I) = \begin{cases} exp(\sum\limits_{w:R\in\Pi_I} w) & \text{if } I \in \text{SM}[\Pi]; \\ 0 & \text{otherwise.} \end{cases}$$

The probability of I under Π is the normalized weight defined as: $P_\Pi(I) = \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_{J\in\text{SM}[\Pi]} W_\Pi(J)}$.

LPMLN2ASP [181] is an implementation of LP$^{\text{MLN}}$using ASP solver CLINGO. The system returns the most probable stable models. In our problem formulation, given a claim p(x,y), we identify the rules that have predicate p or negp in the conclusion and the evidence facts for the bodies of such rules. We then run LPMLN2ASP and check if p or negp are in the stable model.

Problem Statement. Given an input claim to be verified and a KG, our goal is to compute an

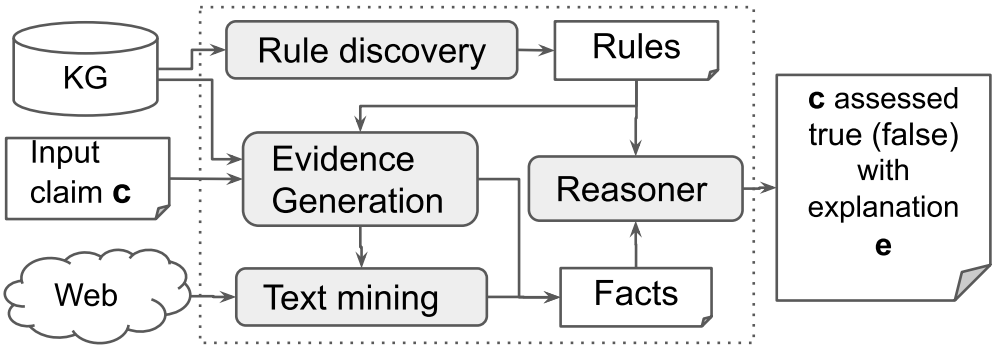


Figure 6.1 – Our Fact checking framework ExpClaim.

assessment of the veracity of the claim and the explanations for such decision, expressed as the union of substitutions for the body of the rules that have triggered the inference in the reasoner.

The uncertainty in the discovered rules and in the facts extracted from the Web make the problem challenging and the role of the reasoner important.

Limits of Existing Solutions. Both the text mining and the rule generation system can be used individually as fact checking tools according to our problem definition. However, they both have strong limitations. The uncertain rules alone cannot make a clear assessment decision in many cases because of (i) conflicting rules both supporting and refusing a fact at the same time, and (ii) lack of evidence in the KG. The Web mining cannot provide semantic explanations and also suffers from the cases where there is no enough evidence to obtain an answer. These limitations also apply for other ML fact checking systems [86, 173–175] and motivate our choice to use a unified framework to combine both sources of signals with a probabilistic reasoner.

6.2 Related Work

There are two main tasks in computational fact checking: (1) monitor and spot claims [171, 172], (2) check claims and explain outcomes. We focus on the second task and on factual facts, specifically. Related approaches try to align the fact to trusted data resources, such as KGs [182, 183], Web documents [184], and databases [185, 186]. These approaches create features for binary classifiers from the data in the KG. Features exploit the structure of the training examples, in the form of paths [173, 174] or geometric properties in a multi-dimensional space with embeddings [86, 175]. We distinguish from these works by providing semantically rich rules and evidence facts as explanations for a fact checking outcome.

Markov Logic combines first-order logic and Markov networks [187]. In principle, learning in Markov Logic could learn the uncertain rules and inference can be applied to the learned rules. We tested alchemy to learn rules for spouse relation with 10 positive examples and it was not able to produce results after 2 hours of execution. This illustrates that rule learning in Markov Logic has scalability issues with large KGs. ILP systems for rule discovery, such as ALEPH [188], make assumptions on the input data that do not hold in KGs and RuDiK outperforms this kind of systems [79].

Other proposals have studied the problem of explainable fact checking with rules, but they

focus on manually crafted constraints [35, 189], while our system relies on discovered rules only. Experimental results on the same DBpedia predicates reported in previous work [35] show that our solution performs better despite being fully automatic. Our proposal also does better than similar attempts that rely on KG only [190].

6.3 Framework

Figure 6.1 shows our framework, ExpClaim. The *Rule discovery* module takes as input the KG K to generate the rules. We then convert the discovered rules Σ into the input language of the reasoner, where the weight of a rule is its support. For the given claim $c : p(x,y)$, $p \in K$ and rules Σ, the *Evidence Generation* module collects relevant evidence facts (triples satisfying the body of the rules) from the KG and from Web with the *Text mining* module. We then feed rules and evidence to the *Reasoner* module, where different modes of computation can be used to infer if $p(x,y)$ or $negp(x,y)$ is in the answer set. The reasoner output includes a human-interpretable explanation for the decision. The details of the main steps are given next.

6.3.1 Rule Generation

Consider a claim $c : p(x,y)$ with $p \in K$, our first step is to obtain the set of rules Σ.

Rule Discovery: The rule discovery module starts by generating M positive and M negative examples for p. Positive examples are (x,y) entity pairs s.t. $p(x,y) \in K$, and negative examples are (x,y) pairs that satisfy the following conditions [79]:

- $p(x,y) \notin K$;
- there is either some $y' \neq y$ s.t. $p(x,y') \in K$ or some $x' \neq x$ s.t. $p(x',y) \in K$;
- there is some $p' \neq p$ s.t. $p'(x,y) \in K$.

RuDik uses the examples and the KG to mine positive and negative rules (Σ) for p.

Consider the mining of positive rules for predicate *spouse*. Positive examples are pairs of married people, and negative examples are pairs of people who are not married to each other, according to the three conditions above. Given the examples, the algorithm outputs *approximate* rules that (i) maximize the coverage of the positive examples and (ii) minimize the coverage of the negative ones. The example sets switch role for the discovery of negative rules, i.e., not married people play the role of the positive examples.

As in association rule mining, the support s of each rule is computed as the support value of the rule divided by the number of examples used in the rule discovery step [28].

Convert Rules into LP^{MLN}: Rules in Σ are rewritten into the input language of lpmln2asp with their weights. For instance, for the spouse predicate, a positive rule

is rewritten into LP^{MLN}as

$$w : \ spouse(a,b) \leftarrow child(a,c), parent(c,b). \tag{6.2}$$

An original support s equal to 0 corresponds to a weight w of $-\infty$ and a support of 1 to a weight of $+\infty$. We convert the rule support into a weight for a program with the equation: $w = \ln \frac{s}{1-s}$.

Generic Rules: We add two rules to the set associated to each predicate. These rules are generic and model natural constraints that play an important role in our fact checking system.

The first rule ensures that p(x,y) and negp(x,y) cannot be true at the same time, i.e., a claim should not be assessed as false and true. This is a hard rule, which is always valid.

$$\alpha: \ \bot \leftarrow p(x,y), negp(x,y) \tag{6.3}$$

The second rule enforces the *functionality* of a predicate. If a predicate is functional, such as the predicate expressing the capital of a country, then there is only one value that can be in the solution. However, this is not true for all predicates, e.g., a person can be the author of several books. The support of the rule models the functionality of the predicate.

We express this constraint stating that a claim cannot have two different object values.

$$w: \ \bot \leftarrow p(x,y), p(x,z), y \neq z \tag{6.4}$$

These generic rules steer the reasoner in the computation of the truthfulness/falseness probability for the input claim.

6.3.2 Evidence Generation

For a given claim, we execute the following steps to gather the evidence for a fact checking decision.

Generate Evidence Triples from KG: For each rule in Σ, we substitute the head variables with the values of the claim and collect all the triples in the KG that have a valid substitution to its body. More precisely, the head variables in the body of a rule are constrained to the value of the subject and object of the claim. Then, the evidence triples are identified by querying the KG with the rewritten body of the rule. For example, given the *spouse* rule above and claim *spouse*(Mike,Laure), the body is rewritten as a query: `child(Mike,c), parent(c,Laure)`, where c is a universal variable.

Generate Evidence Triples from Web: Our reasoner models also the uncertainty for the evidence facts. The KG is considered trustworthy, so the weights for the evidence from the KG are set at infinite. However, because of the OWA, we cannot find every true fact in the KG. For claims for which no rule can be executed, we resort to a Web text mining system [177]. For each rule, we substitute the subject and the object according to the input claim. If a single atom is non-replaceable with KG facts in the body of a rule, then we use the Web module to validate the missing fact. Notice that only grounded facts can be verified with the Web module, such as *child*(Mike,Marie). If the rewritten body contains a fact with a variable, such as *child*(Mike,c) above, we discard the claim. If the Web module returns a probability p of a fact being correct greater than 0.5, than we add it to our evidence.

As an example, consider the positive rule: *locatedIn(x,y)* $\leftarrow$ *hasCapital(z,x), locatedIn(x,y)*, the claim *locatedIn*(Sacramento, USA), and a KG with fact *hasCapital*(CA, Sacramento). Assuming that the fact for CA located in USA is missing from the KG, we query the Web module for *locatedIn*(CA, USA).

Similarly to the conversion of the rule support into the weight of an LP^{MLN} program (Section 6.3.1), we convert the probability p of a fact of being true into a weight w for the fact when we use it as evidence for the reasoner.

6.3.3 Inference for Fact Checking

We discuss two inference methods that enable us to expose the rules and the evidence triples involved in a decision for a claim $p(x,y)$.

- **Pure ASP** checks if $p(x,y)$ or $negp(x,y)$ is in the stable model of the rules without including the rule weights. This method only states if the positive or negative triple for the claim can be derived. Since we rely on Horn rules, there is only one stable model for them. If the stable model contains both $p(x,y)$ and $negp(x,y)$, it violates constraint (6.3), so we conclude neither $p(x,y)$ nor $negp(x,y)$. A similar case happens when the stable model violates the functionality of a predicate.
- LP$^{\text{MLN}}$**MAP inference with weighted rules** checks if $p(x,y)$ or $negp(x,y)$ is in the most probable stable model of the weighted rules using LPMLN2ASP. This method utilizes the weighted rules and the evidence facts to find a more likely answer at the cost of violating constraints (6.3) and (6.4).

Example. We want to check if Glen Cook is the author of the book Cold Copper Tears. The following weighted rules are mined from the KG[2]:

```
1  0.04: author(A,B) ← runtime(A,C), activeYearsStartYear(B,D), C<D.
2  0.04: author(A,B) ← birthYear(B,C),    runtime(A,D), C>D.
3  0.13: author(A,B) ← author(C,B), subsequentWork(A,C).
4  0.02: author(A,B) ← previousWork(A,C), literaryGenre(C,D),genre(B,D).
5  0.02: negauthor(A,B) ← writer(C,B),      format(C,D), format(A,D).
6  0.38: negauthor(A,B) ← runtime(A,C), activeYearsStartYear(B,D), C<D.
7  0.31: negauthor(A,B) ← birthYear(B,C), runtime(A,D), C>D.
8  0.02: negauthor(A,B) ← writer(C,B), previousWork(C,A).
9  0.02: negauthor(A,B) ← writer(C,B), previousWork(C,D), subsequentWork(A,D).
10 0.08: negauthor(A,B) ← writer(C,B),     genre(C,D), genre(A,D).
11 0.02: negauthor(A,B) ← writer(C,B), subsequentWork(C,A).
12 0.02: negauthor(A,B) ← previousWork(A,C), subsequentWork(D,C), writer(D,B).
13    α: ⊥← negauthor(A,B), author(A,B).
14 0.04: ⊥← author(A,B), author(A,C), B≠C.
```

Notice that not all rules are semantically correct: rule 1 is not valid (and has low support), while rule 3 is correct in most cases (in fact it has a higher support). Notice also rule 13, which is the hard constraint stating that a fact cannot be true and false at the same time and rule 14 reflecting the low functionality for the *author* predicate. The evidence generator module collects the following triples from the KG (facts with confidence 1) and the Web mining module (all other facts):

```
0.55: literaryGenre('Cold_Copper_Tears','Fantasy').
0.52: literaryGenre('Cold_Copper_Tears','Mystery_fiction').
1: previousWork('Cold Copper Tears','Bitter Gold Hearts').
0.69: subsequentWork('Cold Copper Tears','Old Tin Sorrows').
0.56: activeYearsStartYear('Glen_Cook','1970').
0.59: author('Bitter_Gold_Hearts','Glen_Cook').
1: author('Old Tin Sorrows','Glen Cook').
1: genre('Glen Cook','Fantasy').
1: genre('Glen_Cook','Science_fiction').
1: literaryGenre('Bitter_Gold_Hearts','Mystery_fiction').
```

[2]For readability, we report normalized support (confidence) for rules (evidence triples), instead of weights.

```
1: literaryGenre('Bitter Gold Hearts','Fantasy').
1: literaryGenre('Old_Tin_Sorrows','Mystery_fiction').
1: literaryGenre('Old_Tin_Sorrows','Fantasy').
1: previousWork('Bitter_Gold_Hearts','Sweet_Silver_Blues').
1: previousWork('Old_Tin_Sorrows','Cold_Copper_Tears').
1: subsequentWork('Bitter_Gold_Heart','Cold_Copper_Tears').
1: subsequentWork('Old_Tin_Sorrows','Dread_Brass_Shadows').
1: author('The_Black_Company','Glen_Cook').
1: genre('The_Black_Company','Dark_fantasy').
1: genre('The_Black_Company','Epic_fantasy').
1: genre('The_Black_Company','Fantasy_novel').
```

The LP^{MLN}inference outputs that the input fact is true because of rules 3 and 4 together with the facts in bold in the evidence set. Here, Old Tin Sorrows is the subsequentWork of Cold Copper Tears whose author is Glen Cook. These two facts satisfy the body of rule 3 to derive the author relation between Cold Copper Tears and Glen Cook. Similarly, for rule 4, Fantasy is the genre of Glen Cook, which is also the literaryGenre of book Bitter Gold Hearts. Further, Bitter Gold Hearts is the previousWork of Cold Copper Tears. This sequence of three facts in the evidence set satisfies the body of rule 4 to derive the author relation between the test entities. By using the MAP inference, we can find in the answer set:

```
author(Cold_Copper_Tears,Glen_Cook)
```

6.4 Experiments

We test our proposal against baseline methods over claims from a real KG. Code and datasets are available online[3].

	spouse	*deathPl.*	*vicePres.*	*almaMater*
Positive	22	25	65	27
Negative	72	33	27	21

Table 6.2 – Number of discovered rules for each predicate.

Datasets. From the latest (online) DBpedia, we selected four predicates P = *spouse*, *deathPlace*, *vicePresident*, *almaMater*. In the following, all rules have been mined from 2K positive and 2K negative examples. Statistics for the discovered rules are reported in Table 6.2.

We create 3 datasets with each containing 100 true and 100 false facts for every predicate, for a total of 2400 claims. True facts are randomly taken from the KG, false ones are created according to the procedure described in the previous section. True facts are then removed from the graph.

Metrics. For each claim in the 4 predicates, we count *correctly* labelled claims (T) and *incorrectly* labelled ones (F). We also count *Undecided* (U) claims, when a method cannot make a decision, e.g., neither *p(x,y)* or *negp(x,y)* are in the stable model. We use precision, defined as (T)/(T+F), recall, defined as (T)/(T+F+U), and their combination in the F-score (harmonic mean).

[3] https://github.com/ppapotti/expclaim

Methods. We run three baseline methods. The first is the Web text miner CREDE [177]. Although CREDE was not designed to check arbitrary claims, we show that it also handles KG facts reasonably well. The second is the state of the art link prediction method for KGs KGM [173], which uses the graph facts as training data and an ML classifier. The third baseline is the application of the discovered rules, without considering their weights (ASP). The first two approaches (CREDE and KGM) cannot provide explanations, while the third (ASP) does not exploit the reasoning. We identified 0.5 as the threshold value for both CREDE and KGM to maximize their F-score.

We consider two variants of our solution. The first is the LP$^{\text{MLN}}$MAP inference with weighted rules over the KG data only (MAP). The second is MAP with evidence collected from the KG and Web documents (MAP+W). For these methods, we check if the claim is in the stable model.

	almaMat.	*deathPl.*	*spouse*	*vicePres.*
CREDE	.41(.03)	.59(.06)	.44(.07)	.36(.15)
KGM	.73(.08)	.68(.01)	.86(.01)	**.81**(.03)
ASP	.70(.06)	.01(.01)	.31(.08)	.18(.16)
MAP	**.88**(.14)	.75(.15)	**.87**(.11)	.66(.22)
MAP+W	**.88**(.09)	**.83**(.11)	.86(.10)	.68(.18)

Table 6.3 – Average F-score results (SD) for four predicates with all methods over 3 datasets.

Results. Table 6.3 reports F-score results and standard deviation (SD) for true and false claims averaged over the 3 datasets. For two predicates, MAP+W is the best method in terms of F-score, with an average over all predicates of 0.81, followed by MAP with .79 and KGM with .77. For all predicates, method ASP has very poor performance because of a large number of claims with no rule to be grounded with the KG evidence. Several of these claims are solved with the reasoner in MAP with high precision (1 in most cases) but not perfect recall. Web evidence in MAP+W enables the triggering of more rules, but at the cost of a lower precision because the text miner is not always reliable, as shown in the results for CREDE.

The issue of undecided claims affects the results heavily for predicate *vicePresident* in all methods based on rules. In general, there is no clear correlation between the number of rules and the quality of the results for the rule-based methods. This suggests that the quality of the rules (and of the evidence facts) is more important than their number. Also, more functional predicates, such as *spouse*, are easier to fact check for most methods.

Table 6.6 reports a detailed analysis for predicate *deathPlace*. The first evidence is that KGM has the best performance for true claims but falls behind MAP methods for false ones. Neither CREDE performs well with false claims. We emphasize that in fact checking false claims are more important.

Results for the rule-based methods show that reasoning is key for our approach. For true claims, ASP correctly labels only 1% of the test facts, while the MAP labels 58% of them without mistakes on average. ASP cannot handle facts for which there is a contradiction among the positive and the negative rules, while MAP inference makes the right decision by exploiting the weights of the rules. However, for 42 true claims on average, none of the rules are triggered in MAP. The coverage is increased by adding more evidence with the Web mining module

```
FALSE : almaMater(Michael White, UT Austin)
     ← employer(Michael White, UT Austin)
     ← occupation(Michael White, UT Austin)
     ← almaMater(Michael White, Abilene Christian Univ.), almaMater(Michael White,
         Yale Divinity School)
```

Table 6.4 – Example of MAP+W output for claim *almaMater*(Michael White, UT Austin).

(MAP+W), at the cost of a lower precision but better overall F-score. The benefit of rules and Web evidence is more apparent with false claims. While in this setting CREDE and KGM show poor results, MAP+W reports high precision (94% on average) and an average recall of 83%, with a very significant increase in all metrics compared to MAP. From a manual verification, we explain the better results for false claims with the better quality of the negative rules w.r.t. positive ones for *deathPlace*, i.e., it is easier to find a negative rule than a positive rule for this predicate. This is consistent with previous rule quality assessments [79].

In all the cases for which an answer is produced, rule-based methods explain their decision by showing involved rules and corresponding evidence sets. This makes it relatively easy to identify what is the cause for a conclusion, as for the example reported in Table 6.4. The given claim is labeled as false because of the three rules that apply with evidence coming both from the KG and the Web.

	spouse	*deathPl.*	*vicePres.*	*almaMat.*
CREDE	6435	7377	7210	7355
KGM	16	15	12	13
ASP	7	8	9	8
MAP	475	822	1880	408
MAP+W	485	1897	3448	409

Table 6.5 – Average execution times (secs) for 200 claims.

Finally, we report on the execution times in Table 6.5. Methods KGM and ASP are the fastest, with a single claim checked in less than 0.1 seconds. Although we are not counting the time to gather Web pages, CREDE is the slowest method, with up to 37 seconds on average to check a claim. MAP and MAP+W are in the middle, taking from 2 to 17 seconds to check a claim on average. The time differences depend on the predicate at hand, as checking predicates with less evidence in KG requires more calls to the text mining module.

6.5 Summary

We presented a fully automated fact checking framework based on KGs and Web documents as reference information. Given a fact expressed as a triple over entities in the KG, our method validates its veracity with high accuracy and provides an explanation of the decision by exposing facts that support or contradict the given claim according to a set of rules. The system does not rely on a human configuration, as rules are automatically discovered and additional information

	True claims					False claims				
	CREDE	KGM	ASP	MAP	MAP+W	CREDE	KGM	ASP	MAP	MAP+W
Correct(/100)	50(8)	96(1)	1(1)	58(27)	62(31)	23(8)	7(2)	0	62(14)	78(8)
Incorrect(/100)	19(3)	4(1)	0	0	21(18)	55(1)	93(2)	0	11(4)	5(4)
Undecided(/100)	31(9)	0	99(1)	42(27)	17(13)	22(7)	0	100(1)	28(18)	17(9)
Precision	.72	.96	1	1	.75	.29	.07	1	.85	.94
Recall	.69	1	.01	.58	.83	.78	1	0	.72	.83
F-score	.70	.98	.01	.74	.79	.43	.13	.01	.78	.88

Table 6.6 – Average results (SD) for *deathPlace* predicate with all methods over 3 datasets.

to complement the KG is mined from the Web. As demonstrated by the experiments, the solution is comparable in terms of precision and recall with state of the art fact checking methods.

Chapter 7

Conclusion and Future Work

In this thesis, we first introduced our approach in creating a knowledge graph in the auditing domain and then presented novel approaches and systems designed to improve the performance of the rule mining on large Knowledge Graphs. We also presented models that are developed in order to exploit rules in different scenarios.

In Chapter 2, we presented some basic information in the areas of knowledge graph construction and introduced different tools for curating a KG.

In Chapter 3, we first developed a model for identifying nodes of the KPMG's KG. We first identified representative nodes, and for each one of them generated a family of words. In the second section of this chapter, we proposed a model for **Text to Data Matching** which can be exploited for creating edges between nodes in the KG (Text to Structured-Text Matching). We also tested our model in different scenarios (Text to Data and Text to Text), and showed that our unsupervised approach can even compete with supervised models.

Chapter 4 reported techniques and improvements that we used to enhance the performance of **RuDiK**, a previously developed rule extraction system. We extended RuDiK in order to extract conditional rules. Conditional rules can be applied on subjects and objects with a specific type.

RuleHub is an open corpus for collecting and annotating logical rules extracted from different Knowledge graphs, as introduced in Chapter 5. In Rulehub we aimed to manage a large number of rules, by computing an estimate of their quality in terms of computed confidence and human annotated quality score. In this chapter, we also proposed two methods which can be utilized in order to extract logical rules from Wikidata. We also introduced **RuleBERT** in this chapter. In RuleBERT, we used logical rules to transfer common-sense knowledge such as inversion, symmetry, and negation to Pre-trained Language Models. For this matter, we developed a method which converts soft rules (taken from RuleHub) into a format readable to PLMs and used them to generate our model. RuleBert achieved a very high performance in our experiments, even on external datasets.

In Chapter 6, we used logical rules to design **ExpClaim**, a framework for explainable fact checking. In ExpClaim, we considered logical rules as reference information in the KGs and used them to assess factual claims and provide explanations for the final true/false decisions.

7.1 Future Work

We describe here an outlook of broad research directions in continuation of the work done in this thesis.

Audit KG

For dealing with the problems specific to handling information in auditing firms, we envision a few promising research directions:

- **Discovering more relationships in text**: we focused on two kinds of relationships, but there are many more that can be expressed across entities, documents and words. For example, a certain document may be "defining" or "regulating" another document. However, the problem with these relationships is that they are not binary, i.e., they involve multiple nodes at the same time. Some relations between a subject and object are valid only for a specific domain and under a specific source and they cannot be represented as a triple. For example, in the KPMG corpus of *UK* and based on *ALEXCMS*, *Changes in accounting policies and errors* is a subtype of *Changes in equity*. This relationship cannot be expressed as a triple and two more conditions (*Domain: UK* and *Source: ALEXCMS*) must be included. Discovering n-ary relationships is an open problem in literature and improvements in this direction can potentially be very useful for the KPMG's KG.

- **Rule mining**: a very rich form of relationship is expressed by patterns and the most powerful patterns are those that express rules. Given the small number of relationships types in the current KG, existing methods that assume a large number of different predicate types, such as those for rule mining [79], cannot be used in this setting, yet. Moreover, it is not clear if existing methods for binary relationships could be useful in this setting, where rules are very complex and specific, likely involving the n-ary relationships in the previous point. This may motivate research for mining rules in the presence of n-ary relationships.

- **Language independence**: Most of our methods are designed to be independent of a specific target language. However, more experiments are needed to measure the performance of the proposed methods in documents beyond English. The experiments with German for family generation shows that the effort to configure the methods to handle a language different from English is reasonable and we have highlighted in the description the steps that rely on language-specific resources.

- **Human in the loop matching**: We plan to extend our framework presented in Section 3.2 to have the users involved in actively solving matches to quickly improve the underlying representation. To enable interactive response, blocking techniques should be designed for this setting. Also, the graph has clear opportunities to be extended with more external information, such as typed or weighted edges. Random walk generator is another module of the framework that can be improved by including more information in the process of the edge selection.

Rule Discovery and Management

Given the recent efforts in the interactive discovery of the rules for relational data [191, 192], open questions are related to similar tasks in KGs. The size of G has a direct impact on the search space and hence on the running time. Since we generate all valid rules for each example in G, the search space grows roughly linearly with its size. If we could identify a subset of examples that lead to the generation of all valid rules, then we could use only those few examples. However, it is not clear how to sample examples while not compromising on the quality of the mined rules.

A second promising research direction is the expressiveness of the language. We aim at mining even richer rules that better exploit spatial and temporal information through a smart analysis of literal values' distributions and correlations [30], e.g., "if two person have age difference greater than 100 years, then they cannot be married".

For **RuleHub**, we plan to keep enriching the corpus by supporting more kinds of rules, such as graph functional dependencies [148]. Indeed, these dependencies have a syntax that is not modelled by the Horn rules supported in our current system. Another possible future work is to apply *RuleHub* architecture for collecting and annotating rules in other domains (e.g. business process compliance [193]). We will improve the interface to make it more convenient for users and better allow them to contribute to the hub. We also believe that, in terms of metadata, there is a great opportunity in going beyond confidence and extend our system to compute more statistical measures, such as unexpectedness [194]. We also plan to design a novel rule mining algorithm that makes use of the proposed method to estimate the confidence for negative rules.

Finally we aim to develop a method for adding the building blocks to define logical rules to the Wikidata infrastructure. The implementation is not obvious because of the uncertain nature of most rules. One way to go from the logical form of a rule with high confidence to a fully implemented constraint is to identify and define the exceptions to the rule. Going back to the example about capitals and countries in Section 5.2, it should be implemented as an exact rules with 15 exceptions. We are studying how to automatically go from non-exact rules to exact rules with either *more conditions* (that narrow their scope) or with *exceptions*.

Logical Rules Applications

An interesting direction for extending the ExpClaim framework is to include a module for claim detection and explore the opportunities of an end-to-end system [195]. We also will increase the number of the rules and predicates covered by our system.

A second direction is to exploit the information from the reasoner to steer the quality management of the KG [14], e.g., inspect undecided claims to identify parts of the KG that need data curation. Also, we aim at integrating natural language generation techniques to produce explanations that are easier to read for the target users [196].

Finally, one direction is the development of explainable models for fact-checking by extending RuleBert for this application. Promising approaches to make RuleBert more transparent include an occlusion-based method that removes parts of the input and checks the impact on the output [168] and building proofs iteratively through 1-hop inference [197].

Appendices

Appendix A

Text to Data Matching: More experimental results

A.1 Ablation Study

We first report on the impact of different parameters on the performance of our method W-RW. We then evaluate the impact of the improvements proposed in Chapter 3.2.2.

A.1.1 Impact of parameters

We examine the impact of length and number of random walks, followed by number of tokens in data nodes. For *CoronaCheck*, we report results for the union of the *Generated* and *User* sentences.

Length of random walks. Figure A.1 shows the mean average precision results for all scenarios when increasing the length of the random walks. Increasing the walk length increases the performance for all scenarios up to size 20. The increase is higher at lower values and then stabilizes or gradually decreases for most scenarios. We explain the different behavior for *IMDb* and *Audit* with the fact that they have the biggest and most dense graphs. *IMDb* graph is the biggest both in terms of nodes (107k nodes vs 35k node for *Snopes*) and edges (1m vs 168k edges for *Politifact*). Because of their size, larger graphs benefit of walks longer than 20.

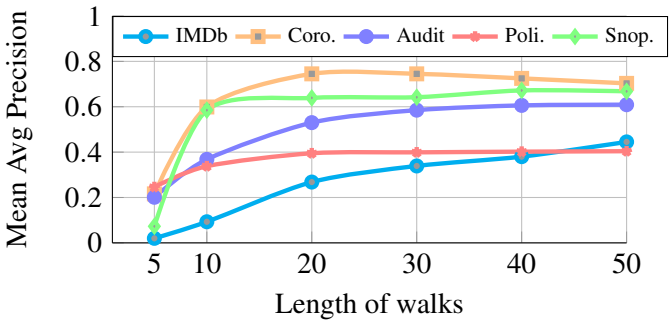


Figure A.1 – Match quality with increasing walk length.

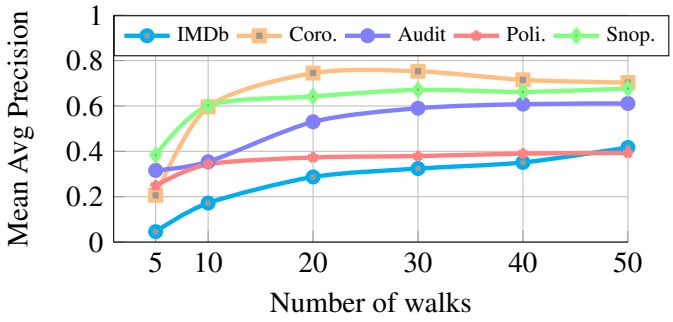


Figure A.2 – Increasing number of random walks per node.

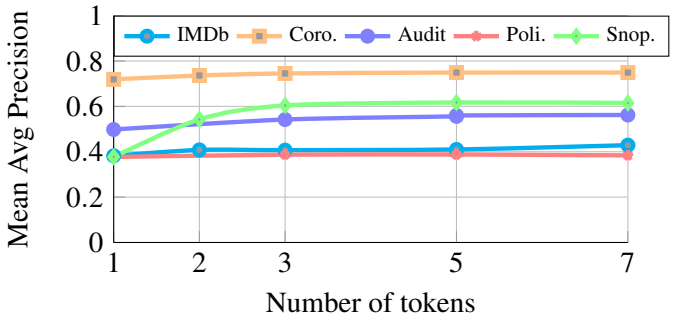


Figure A.3 – Average precision w.r.t. number of tokens in a term.

Number of walks. Figure A.2 shows the mean average precision when increasing the number of walks. The performance for all datasets improve with more walks, but with diminishing results. Results also confirm that graphs with more edges per node need more walks to obtain the best results. After 20 walks per node, results for *IMDb* keep improving, while for *CoronaCheck*, which is the most sparse graph with an average of four edges per node, there is no improvement.

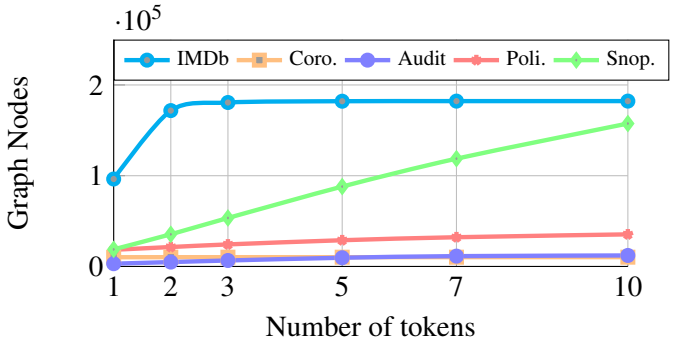


Figure A.4 – Graph size w.r.t. number of tokens in a term.

Number of tokens in terms. Results in Figure A.3 show that allowing more tokens in data nodes (terms) increases the mean average precision in all scenarios. There is a significant increase in quality going from one to two tokens and the impact is smaller with higher values. The highest increase is for *Snopes* and *Audit* datasets with an increase up to 0.24 and 0.07, respectively. For text to data scenarios, *IMDb* has an increase up to 0.05 and *CoronaCheck* up to 0.03. The amount of increase for a scenario is related to the number of new nodes added to the graph. For *Snopes*,

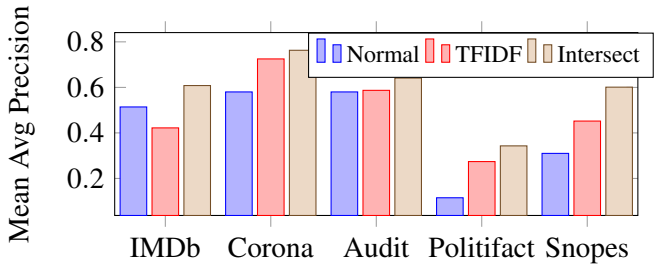


Figure A.5 – Impact of data node filtering.

an increase in the number of tokens in a term adds an average of 24k new nodes to the graph, which is close to the initial number of nodes (35K). For *Audit*, the increase is 1.7k, which is 62% of the initial graph's token nodes. In *IMDb*, 23K new nodes are generated (22% of initial graph) at every increase.

Figure A.4 shows the impact of an increasing number of tokens allowed in terms of total number of nodes in the generated graph. For all scenarios, except *Snopes*, the number of nodes does not increase drastically after allowing three tokens in a term. The behaviour of a dataset is determined by the length of documents in its first corpus as tokens in the documents of the second corpus get filtered. *Snopes* has the biggest documents in its first corpus in comparison to the other datasets: it has claims of 43 tokens on average while *PolitiFact* has 18.

A.1.2 Improving graph generation

We discuss here the impact of the techniques introduced in Section 3.2.2 to improve embeddings and matching quality.

Connecting metadata nodes. For measuring the impact of edges between related metadata nodes in a structured text, we run the same experiments in Section 3.2.6 for W-RW without edges among metadata nodes. The quality of the matches is negatively affected, with the Node F-scores for increasing values of K (1, 3, 5, 10) dropping by .08, 0.04, 0.02, and 0.01 in absolute values.

Filtering data nodes. Merging tokens create shorter paths between related metadata nodes. In graph creation, we filter tokens of the second corpus based on the nodes from the first one. In this experiment, we compare the performance of our technique (Intersect) against a solution based on *TF-IDF* [47], which retains non-stopword tokens with high *TF-IDF* scores. It has been shown that in a data to data matching task this technique increases the quality performance on a text-heavy dataset from 41% to 93% [47]. For implementing this method, for each document we keep k tokens with highest TF-IDF scores for different k values. For each scenario, we run k =3, 5, 10 and 20 and report the best result.

Figure A.5 shows the performance of these two techniques in terms of mean average precision for all scenarios. The results show that, except for *IMDb* dataset and *TF-IDF*, both summarizing techniques improve the mean average precision of matching. It also shows that our technique works better than *TF-IDF* in all scenarios.

Combining matching scores. Figure A.6 depicts the results of averaging the cosine similarity scores from our solution with those from the pre-trained S-BE. Our solution already outperforms

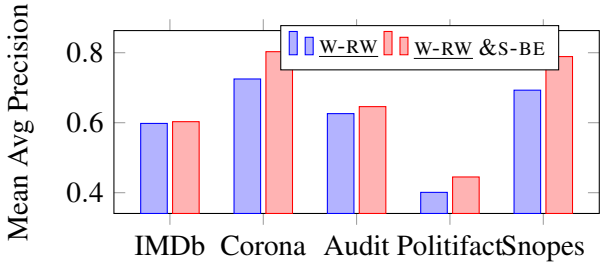


Figure A.6 – Our method combined with *SentenceBERT*.

S-BE in the original experiments, but averaging the two methods improves the matching quality in all scenarios even above the results from our methods alone. The biggest improvements are for *Snopes* and *CoronaCheck* with 0.9 and 0.8 increase, respectively. This simple combination show the benefit of exploiting domain specific embeddings and pre-trained ones from large, generic corpora.

Merging nodes. For all scenarios, we employ different techniques and resources to improve their quality results (Section 36). In *CoronaCheck*, 17% of the nodes are numeric values and merging with equal-width buckets decreases the number of graph nodes. We had the best results with equal-width buckets of size 7, which increased the mean average precision from 0.72 to 0.76. In *Audit*, *Politifact*, and *Snopes* datasets, there are few numeric values and bucketing had no effects. In *IMDB*, we observe a small loss because numeric values are release dates, for which is better to avoid merging.

We also use *Wikipedia2Vec* to merge similar nodes. *IMDb* scenario contains variations for the same entity (e.g., director names) and merging them with a threshold $\gamma = 0.57$ increases the performance by 2.5%. For *Snopes* and *Politifact*, using the same value, the increase is by 1.7% and 1.5%, respectively. As the *CoronaCheck* scenario contains typos in user sentences (e.g., country names), merging such typos leads to a 3.4% increase. With *Auditing*, we do not observe improvements by merging nodes with pre-trained resources. The problem is the difference between the general meaning of a term and its meaning in such a specific domain. Since pre-trained models are trained on general corpora, they do not help much in a domain specific scenario. For example, based on *Wikipedia2Vec*, the similarity is extremely high between *Financial statement* and *Financial reporting*, same for *Auditor* and *Risk control*. However, these terms have different meanings in auditing documents.

Appendix B

Rulehub: More experimental results

B.1 Other Predicates Confidence Results

In this section, we report more results about our experiments. Figure B.1 shows a comparison between *human confidence* and *computed confidence* for the rules involving DBpedia:spouse. Figure B.2 shows the results for the rules involving the DBpedia:foundedBy predicate. Figure B.3 reports the results for the union of the rules for two DBpedia predicates: *relative* and *publisher*. The mean error for these predicates is 8.36% for positive rules and 8.71% for negative rules. The correlation between *human confidence* and *computed confidence* for both positive and negative rules for these predicates is evident from the plots.

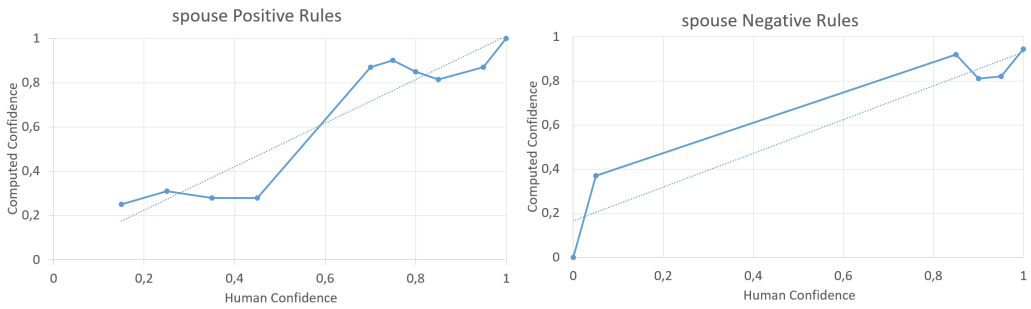


Figure B.1 – Confidence measures of the rules for DBpedia:spouse.

B.2 A Sample of Rules Used for Evaluations

We report in Table B.1 the negative rules that we used for evaluating our confidence measures. Figure 5.5a shows the computed confidence of these rules for different κ values.

In Table B.2 we report the positive rules that we used for evaluating our confidence measures.

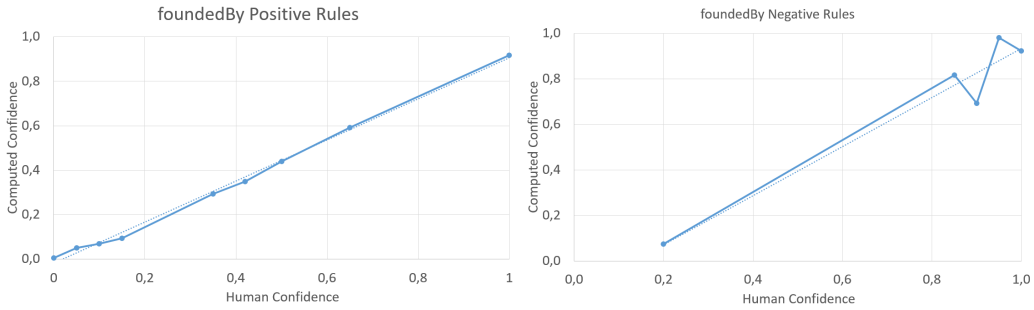


Figure B.2 – Confidence measures of the rules for DBpedia:foundedBy.

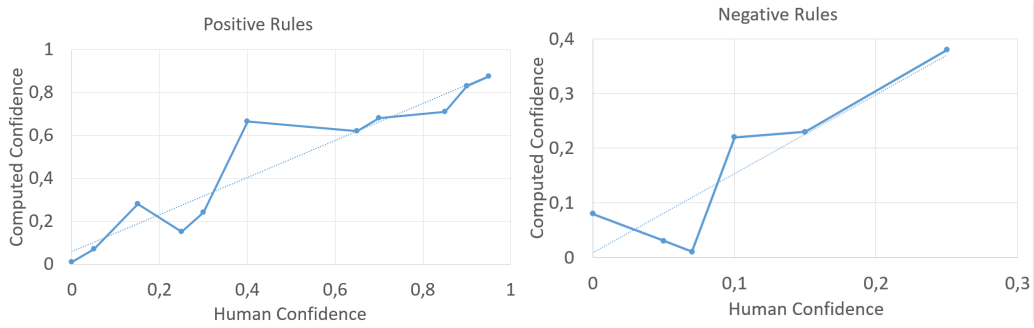


Figure B.3 – Confidence measures of the rules for DBpedia:relative and DBpedia:publisher (union of the two rule sets).

B.3 Quality Evaluation Measure Experiment

In this experiment, we assess the *quality evaluation* measure (Section 5.1.2) by comparing it with the human computed confidence. We randomly chose 20 annotated positive and negative rules from the RuleHub database and asked three non-experts to assign quality evaluation score to each one of them. We provided three annotators with some general guidelines about logical rules and KGs, as well as with a brief description in English for every rule to help them understand the semantic meaning of the rule. Each annotator assigned individually a score from 1 to 5 to every rule, where 5 is a completely incorrect rule and 1 is a correct one.

In Figure B.4, we report on the horizontal axis the average of annotators quality evaluation score for the rules (between 1 and 5) and in the vertical axis their human confidences (based on triple annotation). The results show a clear correlation between the two measures, with the exception of the two rules marked with red points. The first exception rule (red point on the right) is $predecessor(object, v0) \wedge spouse(v0, v1) \wedge predecessor(subject, v1) \rightarrow spouse(subject, object)$, which has 0.35 human confidence and the average of annotators score is 4.7. This rule is a positive rule which states that if the job predecessors of two given persons are married then these two persons are married. Even though it was difficult for annotators to find supports for this rule, there are 547 support triples for this rule in DBpedia.

In fact, this is true in most of the cases when kings or presidents are replaced. For example, given that $predecessor(Donald\_Trump, Barak\_Obama)$

	Rule
1	$owningCompany(object,v0) \wedge manufacturer(v1,v0) \wedge computingPlatform(subject,v1) \wedge publisher(subject,object) \rightarrow \bot$
2	$relative(v0,object) \wedge child(v0,subject) \wedge parent(subject,v0) \wedge relative(subject,object) \rightarrow \bot$
3	$country(subject,v0) \wedge location(v1,v0) \wedge successor(v1,object) \wedge publisher(subject,object) \rightarrow \bot$
4	$almaMater(object,v0) \wedge country(v0,v1) \wedge birthPlace(subject,v1) \wedge relative(subject,object) \rightarrow \bot$
5	$employer(v0,object) \wedge birthYear(v0,v1) \wedge deathYear(subject,v1) \wedge employer(subject,object) \rightarrow \bot$
6	$publisher(subject,v0) \wedge product(object,v0) \wedge publisher(subject,object) \rightarrow \bot$
7	$owningCompany(subject,object) \wedge owner(v0,object) \wedge author(v0,object) \wedge type(object,Organisation) \wedge type(subject,Organisation) \wedge foundedBy(subject,object) \rightarrow \bot$
8	$publisher(subject,v0) \wedge parentCompany(v1,v0) \wedge successor(object,v1) \wedge publisher(subject,object) \rightarrow \bot$
9	$birthYear(object,v0) \wedge birthYear(subject,v1) \wedge > (v0,v1) \wedge influencedBy(subject,object) \rightarrow \bot$
10	$spouse(v0,object) \wedge parent(subject,v0) \wedge spouse(subject,object) \rightarrow \bot$
11	$writer(v0,object) \wedge artist(v0,subject) \wedge foundedBy(subject,object) \rightarrow \bot$
12	$predecessor(v0,object) \wedge predecessor(subject,v0) \wedge spouse(subject,object) \rightarrow \bot$
13	$associatedMusicalArtist(object,subject) \wedge associatedBand(object,v0) \wedge associatedMusicalArtist(v0,subject) \wedge foundedBy(subject,object) \rightarrow \bot$
14	$deathYear(subject,v0) \wedge activeYearsStartYear(object,v1) \wedge < (v0,v1) \wedge spouse(subject,object) \rightarrow \bot$
15	$spouse(v0,object) \wedge child(v0,subject) \wedge spouse(subject,object) \rightarrow \bot$
16	$deathPlace(object,v0) \wedge region(v0,v1) \wedge birthPlace(subject,v1) \wedge type(subject,Royalty) \wedge type(object,Royalty) \wedge spouse(subject,object) \rightarrow \bot$
17	$birthPlace(object,v0) \wedge deathPlace(object,v0) \wedge predecessor(object,subject) \wedge spouse(subject,object) \rightarrow \bot$
18	$spouse(subject,v0) \wedge parent(object,v0) \wedge parent(object,subject) \wedge spouse(subject,object) \rightarrow \bot$
19	$deathDate(object,v1) \wedge birthYear(subject,v0) \wedge > (v0,v1) \wedge spouse(subject,object) \rightarrow \bot$
20	$spouse(v0,object) \wedge parent(subject,v0) \wedge predecessor(subject,object) \wedge spouse(subject,object) \rightarrow \bot$
21	$successor(object,subject) \wedge parent(subject,v0) \wedge spouse(object,v0) \wedge spouse(subject,object) \rightarrow \bot$
22	$parent(object,v0) \wedge spouse(subject,v0) \wedge successor(subject,object) \wedge spouse(subject,object) \rightarrow \bot$
23	$artist(v0,subject) \wedge recordedIn(v0,v1) \wedge birthPlace(object,v1) \wedge foundedBy(subject,object) \rightarrow \bot$
24	$foundedBy(subject,v0) \wedge deathPlace(v0,v1) \wedge locationCountry(object,v1) \wedge type(object,Company) \wedge type(subject,Company) \wedge foundedBy(subject,object) \rightarrow \bot$
25	$foundingYear(subject,v0) \wedge activeYearsStartYear(object,v1) \wedge < (v0,v1) \wedge founder(subject,object) \rightarrow \bot$

Table B.1 – Examples of negative rules.

and *predecessor*(*Melanie_Trump*, *Michelle_Obama*), since *spouse*(*Barak_Obama*, *Michelle_Obama*) then *spouse*(*Donald_Trump*, *Melanie_Trump*). The second rule deathDate(subject,v0) $\wedge$ <(v0,v1) $\wedge$ activeYearsStartYear(object,v1) $\wedge$ spouse(subject,object) $\rightarrow \bot$ has 0.9 human confidence but a lower evaluation score of 2.7. This rule states that a person (object) cannot be in spouse relationship with a person (subject) who died before object has started his career. In both cases, the triple annotation based method seem to be closer to the correct evaluation. We observe that in these two cases there was always one non-expert making the evaluation very different from the one based on triple annotation, thus skewing the results. In fact, the Kendall's tau-b correlation [161] score for annotating the 20 rules is 0.49, which is a fair agreement but also shows that the annotators had different understandings of some of the non-trivial rules. These results confirm that the triple based annotation is more time-consuming but also more reliable than the qualitative analysis of the rules.

	Rule
1	$recordLabel(v0, subject) \wedge foundedBy(object, v0) \rightarrow foundedBy(subject, object)$
2	$parentCompany(subject, v0) \wedge successor(v1, v0) \wedge foundedBy(v1, object) \rightarrow foundedBy(subject, object)$
3	$producer(v0, object) \wedge producer(v0, subject) \rightarrow relative(subject, object)$
4	$predecessor(v0, object) \wedge spouse(v1, v0) \wedge successor(subject, v1) \rightarrow spouse(subject, object)$
5	$author(subject, v0) \wedge author(v1, v0) \wedge publisher(v1, object) \rightarrow publisher(subject, object)$
6	$relative(v0, object) \wedge spouse(v0, subject) \rightarrow relative(subject, object)$
7	$child(object, v0) \wedge spouse(v0, subject) \rightarrow relative(subject, object)$
8	$successor(subject, v0) \wedge parent(v0, object) \rightarrow spouse(subject, object)$
9	$relative(subject, v0) \wedge relative(v0, object) \wedge child(object, v0) \rightarrow relative(subject, object)$
10	$spouse(subject, v0) \wedge spouse(v1, v0) \wedge spouse(v1, object) \rightarrow spouse(subject, object)$
11	$developer(subject, object) \wedge developer(v0, object) \wedge composer(v0, v0) \rightarrow publisher(subject, object)$
12	$child(subject, v0) \wedge child(object, v0) \rightarrow spouse(subject, object)$
13	$relative(object, subject) \wedge birthPlace(object, v0) \wedge birthPlace(subject, v0) \rightarrow relative(subject, object)$
14	$spouse(object, subject) \rightarrow spouse(subject, object)$
15	$parent(v0, object) \wedge parent(v0, subject) \rightarrow spouse(subject, object)$

Table B.2 – Examples of positive rules.

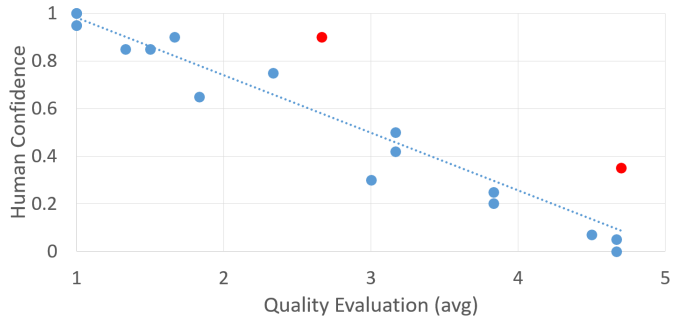


Figure B.4 – Quality Evaluation (subjective value between 1 and 5) vs Human Confidence (derived from triple annotation).

B.4 Important factors in Computing Rules Confidence

We report some observations from comparing computed rule confidence with human confidence. There are 105 manually annotated rules in the RuleHub corpus: 56 positive and 49 negative ones. Comparing computed confidence values with human confidence values shows that there is a lower error rate for positive rules (12.9%) in comparison to negative rules (16.5%). The number of atoms in a rule also affects the computed confidence. Figure B.5 shows that the computed confidence has a better accuracy for rules with a higher number of atoms. Intuitively, the more a rule is specific, the less likely it is that it is satisfied with incorrect triples. In fact, assuming most of the errors are independent from each other (i.e., they are no systematic), it is very unlikely that multiple atoms show a pattern by chance.

Conditional rules are rules that apply only for a subset of data and therefore come with at least two extra atoms compared to the same rules without type constraints [48]. For example, party(object,v0) $\wedge$ party(subject,v1) $\wedge$ type(subject,Politician) $\wedge$ type(object,OfficeHolder) $\wedge$ spouse(subject,object) $\rightarrow \perp$ is a conditional rule that applies where subject has type *Politician*

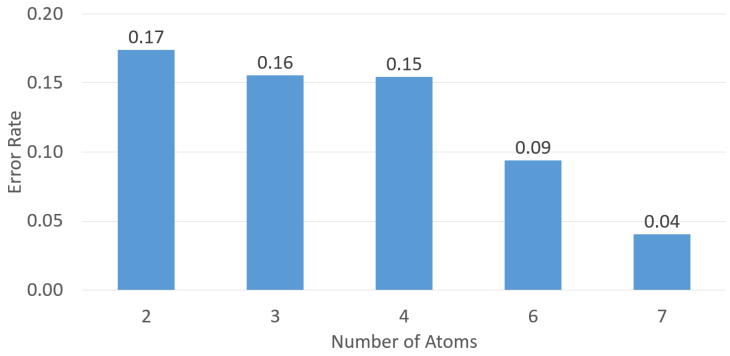


Figure B.5 – Impact of number of atoms on the error rate for the computed confidence w.r.t. human confidence.

and object has type *OfficeHolder*. In accordance with our observation above, conditional rules show on average a lower error rate in the computation of their confidence (11.6%) in comparison to the average for rules that are not conditional (15.1%).

B.5 RuleHub Web Page

In this section, we report more details about the web interface of RuleHub. A step-by-step user manual is available at `http://rudik.eurecom.fr/user-manual`.

B.5.1 Add new rules

To enrich the corpus, RuleHub allows users to easily add more rules into the system through a simple form with the following mandatory fields:

- Knowledge Base
- Predicate
- Rule Type
- Premise
- Human Confidence
- Quality Evaluation

When a rule is submitted, the system computes the hash code to check if the rule already exists in the corpus. If it is a new rule, it will automatically calculate the support score based on the number of matching cases and not-matching cases over the corresponding KG (see Section 5.1.1), then store it into the database as a rule to be validated. These rules will not be published until they are approved by administrators of the site. Figure B.6 reports a screenshot of the main page of the *RuleHub* portal. Figure B.7 reports a screenshot of the page to add rules. Figure B.8 reports a screenshot of the rule management page.

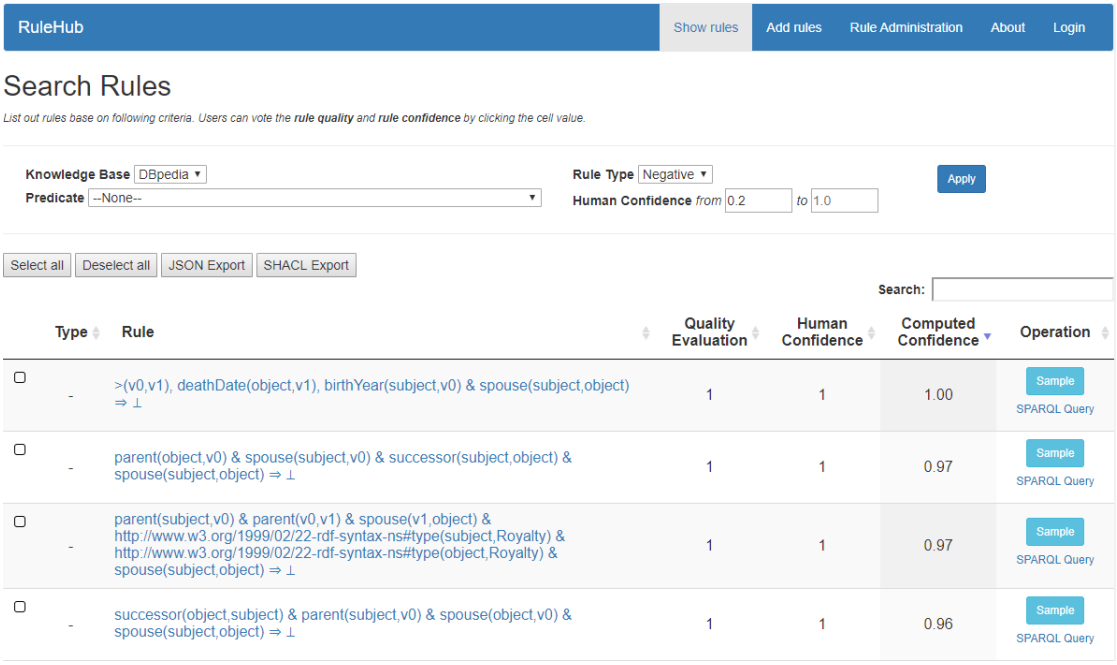


Figure B.6 – Screenshot of RuleHub web portal - Search for rules.

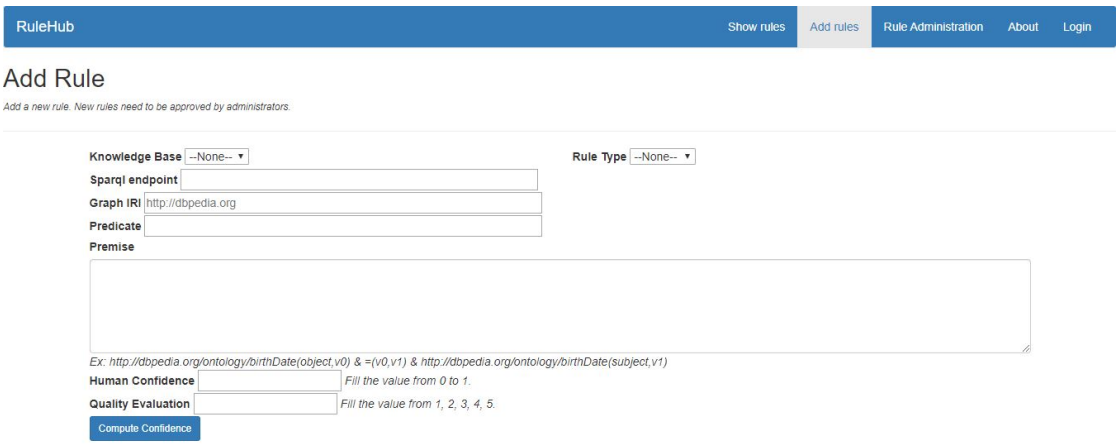


Figure B.7 – Screenshot of RuleHub web portal - Add rule.

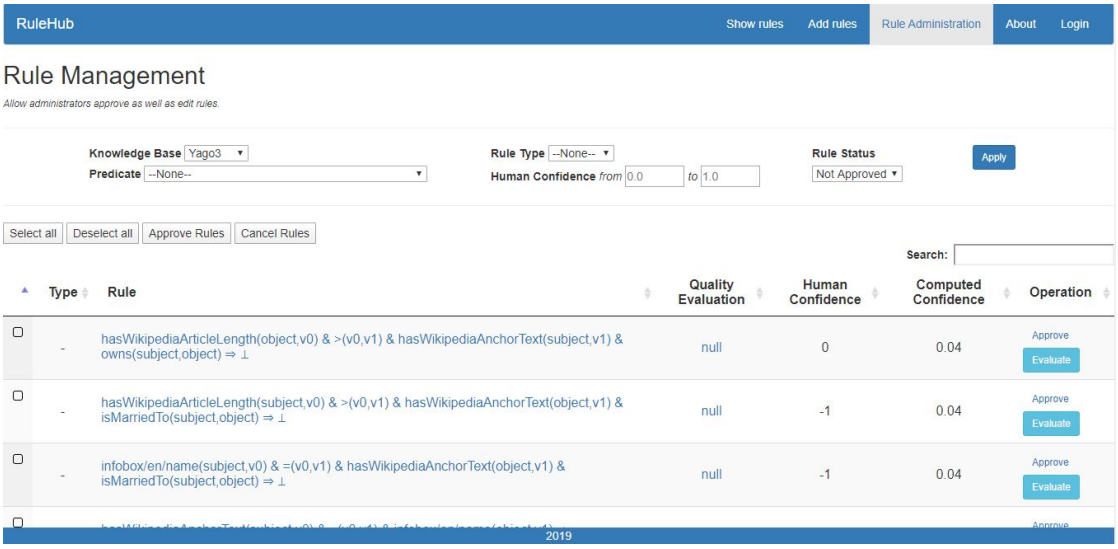


Figure B.8 – Screenshot of RuleHub web portal - Rule Management.

B.5.2 Evaluate rules

Administrators can get samples of the output of a rule execution and label them through RuleHub. Figure B.9 reports a screenshot of the evaluation page.

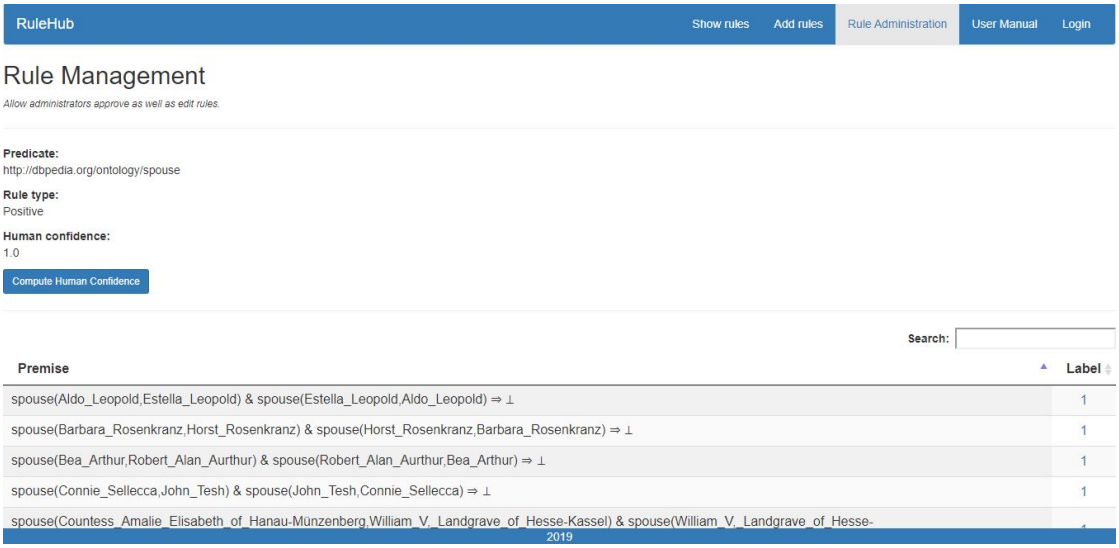


Figure B.9 – Screenshot of RuleHub web portal - Rule Evaluation.

Appendix C

ExpClaim: More Experimental Results

C.1 REASONER

In this section, we will discuss how to compute the confidence of a claim given the learned rules and the evidence from the knowledge base and Wiki Corpus. In the first part of the section, we will discuss about Answer Set Programming (ASP), which is a declarative programming paradigm oriented towards difficult search problem. After that, we will tackle LP^{MLN} which is a probabilistic extension of *ASP*. Then, we will show the results of some experiments done.

C.1.1 Answer Set Programming *ASP*

ASP [198] is a declarative programming paradigm based on stable model semantics Gelfond and Lifschitz [178]. The goal is to represent a problem by a set of rules and evidence facts and find a solution among a large but finite number of possibilities. It is suitable for solving knowledge intensive combinatorial search problems.

A rule is of the form

$$A \leftarrow B \wedge N \tag{C.1}$$

where, A is the disjunction of atoms, B is the conjunction of atoms, N is a negative forumla constructed from atoms with a conjunction, disjunction, and negation.

A logic program is a finite set of rules and it is called ground if all the variables in program are replaced by constants. The answer set is the set of atoms that can be generated by applying the rules in the program in any order. An Herbrand interpretation I is a model of a ground program Π if I satisfies all implications in Π. The reduct of Π relative to an interpretation I, denoted by Π^I consists of $A \leftarrow B$ for all rules in Π such that $I \models N$. An Herbrand interpretation I is called a stable model of a ground program Π if I is the minimal Herbrand model of Π^I.

There are several *ASP* solvers. We will be using *clingo*. The following table shows the symbols used by *clingo* and their equivalent in the logic sense.

C.1.2 LP$^{\text{MLN}}$

LP^{MLN} [181] is a probabilistic extension of answer set programs with the concept of weighted rules which is derived from Markov Logic. In LP^{MLN}, a weight is assigned to each rule so that

the more rules a stable model satisfy, the more weights it gets.

A program Π is a finite set of weighted rules of the form:

$$w : A \leftarrow B \wedge N \tag{C.2}$$

where A is a disjunction of atoms, B is a conjunction of atoms, N is a negative formula constructed from atoms with a conjunction, disjunction, and negation, and w is a real number or the symbol α which represents the "infinite weight".

A rule $w : R$ is a soft rule if w is a real number, a hard rule if w is α denoting the infinite weight α. An LP^{MLN} program is ground if its rules contain no variables. Any LP^{MLN} program Π of signature σ with a ground LP^{MLN} program $gr_\sigma[\Pi]$ is obtained from the rules of Π by replacing every variable with every ground term of σ. The weight of a ground rule in $gr_\sigma[\Pi]$ is the same as the weight of the rule in Π from which the ground rule is obtained. By $\overline{\Pi}$ we denote the unweighted logic program obtained from Π, i.e.,

$$\overline{\Pi} = \{R | w : R \in \Pi\} \tag{C.3}$$

For an LP^{MLN} program Π, Π_I denotes the set of rules $w : R$ in Π such that $I \models R$ and $SM[\Pi]$ denotes the set $\{I \mid I \text{ is a stable model of } \Pi^I\}$. Now, the unnormalized weight of I under Π is defined as:

$$W_\Pi(I) = \begin{cases} \exp \sum_{w:R \in \Pi} w & if I \in SM[\Pi] \\ 0 & Otherwise \end{cases} \tag{C.4}$$

The normalized weight of an interpretation I under Π is defined as:

$$P_\Pi(I) = \lim_{\alpha \to \infty} \frac{W_\Pi(I)}{\sum_{J \in SM[\Pi]} W_\Pi(J)} \tag{C.5}$$

For any proposition A, $P_\Pi(A)$ is defined as:

$$P_\Pi(A) = \sum_{I, I \models A} P_\Pi(I) \tag{C.6}$$

where I is a stable model of Π.

And the conditional probability under Π for the propositions A and B is given as:

$$P_\Pi(A|B) = \frac{P_\Pi(A \wedge B)}{P_\Pi(B)} \tag{C.7}$$

LPMLN2ASP [181] is an implementation of LP^{MLN} using *ASP* solver *clingo*. Its input language is similar to that of *clingo* except that weights can be prepended to rules, in which case the rules become uncertain. The system is able to return most probable stable models as well as conditional and marginal probabilities of predicates. Probabilities of the evidence given by the model are translated into weights using the following formula:

$$w = f(p) = \ln \frac{p}{1-p} \tag{C.8}$$

C.2 Rule discovery

We performed experiments in discovering both positive and negative rules for two predicate *spouse* and $foundedBy$ and evaluating their quality in term of confidence score. For each predicate, we use random walk generation to generate multiple scenarios of 600 samples each with different homogeneity.

We report in Table. C.1 some statistics of our rule discovery on for *spouse* and $foundedBy$. Overall, many positive and negative rules are induced from both two predicates, but only small portion of them are meaningful (confidence higher than 0.5), except for spouse's negative rules where we can exploit up to 58 out of 83 possible rules. This may be due to the fact that spouse's negative rules appear more often than positive rules. Moreover, for $foundedBy$, among 74 induced positive rules, only 7 rules seems to be valid. Meanwhile, this gap looks much smaller in *spouse* (27 vs. 7). This may stress the importance of scenario size against our rule discovery: we need a $foundedBy$ size more than 600 samples to increase the selectivity of rule set and *spouse* can be mined with small size in input. Source code and test results, including data for two predicates and all exploited rules are available at `https://github.com/ppapotti/expclaim` (RUDIK part).

Predicate	#Execution	#Positive Rules	#Negative Rules
spouse	7	27/7/3[1]	83/58/6
$foundedBy$	5	74/7/2	35/8/0

Table C.1 – Statistics of Rudik Executions

Table. C.2 and Table. C.3 includes some representative positive and negative rules for *spouse* and $foundedBy$. Overall, we are able to find some meaningful rules for both predicates. Although it seems to be relatively difficult in case of negative $foundedBy$, we stills have 2, 3 rules that are useful, e.g $foundedBy(x,v0) \wedge child(v0,y) => \neg foundedBy(x,y)$ (y cannot create x if his parent already created it) or $foundationPlace(x,v0) \wedge locationCity(v1,v0) \wedge parentCompany(v1,y) => \neg foundedBy(x,y)$ (it's unlikely that a parent company establishes a branch at its location.). Moreover, type constraints (e.g (**Royalty, Royalty**), (**Fictional Character, Fictional Character**) for *spouse* or (**Oragnisation, Person**) for $foundedBy$ help to discover more expressive rules with higher confidence. Interestingly, considering a positive rule: $owner(x,y) => foundedBy(x,y)$ which only gives 0.57 confidence, if y is indeed a person who owns x, then this rule becomes very reliable (0.96).

Predicate	Rule	Confidence Score
Spouse	$parent(v0,x) \wedge parent(v0,y)$	0.84
	$parent(v0,x) \wedge parent(v0,y) \wedge \textbf{type}(\textbf{x},\textbf{Royalty}) \wedge \textbf{type}(\textbf{y},\textbf{Royalty})$	0.87
	$child(x,v0) \wedge child(y,v0)$	0.78
	$child(x,v0) \wedge parent(v0,y)$	0.87
	$successor(x,v0) \wedge spouse(v1,v0) \wedge predecessor(v1,y)$	0.25
foundedBy	$foundedBy(x,v0) \wedge foundedBy(v1,v0) \wedge foundedBy(v1,y)$	0.87
	$occupation(y,x) \wedge occupation(y,v0) \wedge foundedBy(v0,y)$	0.89
	$keyPerson(x,v0) \wedge foundedBy(x,v0) \wedge owner(x,y)$	0.57
	$keyPerson(x,v0) \wedge foundedBy(x,v0) \wedge owner(x,y) \wedge \textbf{type}(\textbf{x},\textbf{Organization}) \wedge \textbf{type}(\textbf{y},\textbf{Person})$	0.96
	$foundedBy(x,v0) \wedge developer(v1,v0) \wedge developer(v1,object)$	0.45
	$foundedBy(x,v0) \wedge developer(v1,v0) \wedge developer(v1,object) \wedge \textbf{type}(\textbf{x},\textbf{Organization}) \wedge \textbf{type}(\textbf{y},\textbf{Person})$	0.65

Table C.2 – Examples of positive rule exploited for $spouse$ and $foundedBy$

Predicate	Rule	Confidence Score
Spouse	$child(x,y)$	0.64
	$child(x,y) \wedge \textbf{type}(\textbf{x},\neg\textbf{FictionalCharacter}) \wedge \textbf{type}(\textbf{y},\neg\textbf{FictionalCharacter})$	0.75
	$child(x,y) \wedge birthPlace(x,v0) \wedge birthPlace(y,v0)$	0.83
	$parent(x,y)$	0.96
	$predecessor(x,v0) \wedge predecessor(v0,y)$	0.89
	$parent(v0,x) \wedge ! = (v0,v1) \wedge parent(v1,y)$	0.59
foundedBy	$foundedBy(x,v0) \wedge child(v0,y)$	0.71
	$foundationPlace(x,v0) \wedge locationCity(v1,v0) \wedge parentCompany(v1,y)$	0.66
	$affiliation(y,v0) \wedge country(v0,v1) \wedge location(x,v1)$	0.67
	$owner(x,v0) \wedge occupation(v0,y)$	0.49

Table C.3 – Examples of negative rule exploited for $spouse$ and $foundedBy$

Appendix D

RuleBERT: More on Reasoning with Soft Rules

Let σ be a signature as in first-order logic. An LP$^{\text{MLN}}$program Π is a finite set of weighted rules of the form:

$$w : A \leftarrow B \tag{D.1}$$

where A is a disjunction of atoms of σ, B is a conjunction of literals (atoms and negated atoms) of σ, and w is a real number or the symbol α.

When A is $\bot$ (the empty disjunction), the rule asserts that B should be false in the stable model. An LP$^{\text{MLN}}$ rule (D.1) is called *soft* if w is a real number or *hard* if w is α. An LP$^{\text{MLN}}$ program is *ground* if its rules contain no variables. An LP$^{\text{MLN}}$ program Π that contains variables is identified with a ground LP$^{\text{MLN}}$ program $gr_\sigma[\Pi]$, which is obtained from Π by replacing every variable with every ground term of σ. The weight of a ground rule in $gr_\sigma[\Pi]$ is the same as the weight of the corresponding rule in Π. By $\overline{\Pi}$ we denote the unweighted logic program obtained from Π, i.e., $\overline{\Pi} = \{R \mid w : R \in \Pi\}$.

For a ground LP$^{\text{MLN}}$ program Π, Π_I denotes the set of rules $w : R$ in Π such that I satisfies R (denoted $I \models R$) and SM$[\Pi]$ denotes the set $\{I \mid I$ is a (deterministic) stable model of $\overline{\Pi_I}\}$. The (unnormalized) weight of I under Π is defined as follows:

$$W_\Pi(I) = \begin{cases} exp(\sum\limits_{w:R\in\Pi_I} w) & \text{if } I \in \text{SM}[\Pi]; \\ 0 & \text{otherwise.} \end{cases}$$

The probability of I under Π is the normalized weight defined as follows:

$$P_\Pi(I) = \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_{J\in\text{SM}[\Pi]} W_\Pi(J)}.$$

In Answer Set programming (ASP), search problems are reduced to computing *stable models* (a.k.a. answer sets), a set of beliefs described by the program. In the case of a Horn program, the stable models coincide with the minimal models.

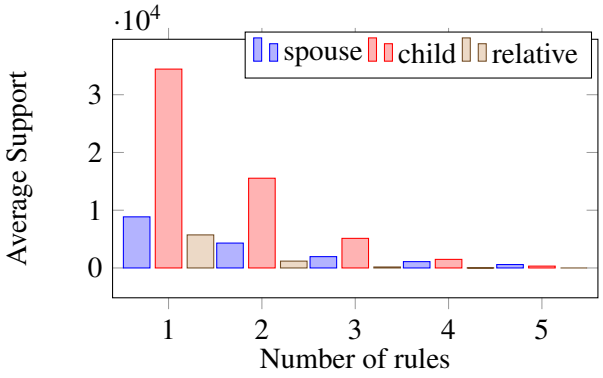


Figure D.1 – Support of the overlapping rules.

D.1 Rules Support

We designed an experiment to show the impact of increasing the number of overlapping rules on the same target predicate. The goal is to measure how often multiple rules are triggered for the same target triple. We measure this with the support of a rule, i.e., the number of triples in the knowledge base that satisfy all the atoms in the rule. To compute the support for more than one rule, we combine the premises of the rules. In this experiment, we picked three predicates (*spouse*, *child* and *relative*), and for each one we selected ten rules randomly. Next, we used DBpedia online endpoint[1] to compute the support for each combination of n (n=1,2,...,5) rules for each predicate. The results in Figure D.1 show that by increasing the number of rules, the support decreases for all predicates. For combinations with more than three rules, the support is very small.

D.2 More Experimental Details

For fine-tuning our models, we rely on Google Colaboratory which assigns random GPU clusters of various types. The number of parameters of our models is in the range of 355M parameters. Hyper-parameters for our models, shown in Table D.2, have been set manually to maximize accuracy. Execution times vary largely depending on the GPU at hand and the scenario, with fine tuning on a Tesla V100 taking from one hour for a single rule to a few hours for all the chaining experiments.

D.3 Ablation

Impact of the Data Size

Setting. We report the impact of the size of the fine-tuning data on the model performance. As shown in Table 5.6, the accuracy of the fine-tuned model is higher for rules with higher confidence. We therefore divide the rules in three categories: *High* contains rules with confidence greater than

[1] http://dbpedia.org/sparql

Dataset	Size
Mod0 Test(own)	2667
Mod1 Test(own)	4000
Mod2 Test(own)	5334
Mod3 Test(own)	6667
Mod4 Test(own)	8000
Mod5 Test(own)	9334
Test(D$\leq$5)	9334
Depth=0	16057
Depth=1	6608
Depth=2	5389
Depth=3	3993
Depth=4	2619
Depth=5	1336

Table D.1 – Number of examples in each of the test datasets for the chaining experiment.

Hyper-Parameter	**Value**
Learning Rate	1e-6
Weight Decay	0.1
Number of Epochs	3
Batch Size	16
Learning Rate Decay	Linear
Warmup Ratio	0.06

Table D.2 – Hyper-parameters for fine-tuning our model.

0.8, *Medium* has rules with confidence between 0.4 and 0.8, and *Low* is for the rest. There are six rules in the *Medium* category and the other two categories have five rules each. For each rule, we fine-tune seven models with 1k, 2k, 5k, 10k, 15k, 20k, to 30k examples.
Results. Figure D.3 shows that having more training data improves the accuracy in all scenarios. For all categories, there is a significant increase going from 10k to 15k examples and the impact is smaller with higher values. The highest increase is for rules with high confidence, and rules with medium confidence demonstrate larger increase compared to rules in the *Low* category.

D.3.1 Role of Example Formats

Setting. When we teach rules to PLMs, we rely on examples with real names from a fixed pool. However, our goal is to teach PLMs the semantics of the soft rule, not the facts in our examples. Thus, we further design an experiment to assess the impact of the format used in the example facts on the behavior of the model. We distinguish two formats for the generated facts: (*i*) real names such as *Alice* and *IBM*, and (*ii*) letters such as *A* and *B*. We first use each format in fine-tuning and we then test both formats. We end up with two test/train scenarios: one with the same format and one with different formats. For this study, we use just one rule: (*child(a,c)* $\wedge$ *parent(c,b)* $\rightarrow$

child(a,b) → negparent(a,b)
child(a,b) → nespouse(a,b)
child(a,b) → negchild(b,a)
child(a,b) → negrelation(b,a)
parent(a,b) → negparent(b,a)
parent(a,b) → nespouse(a,b)
spouse(a,b) → relative(b,a)
successor(a,b) → predecessor(b,a)
predecessor(a,b) → negsuccessor(a,b)
successor(a,b) → negspouse(a,b)
predecessor(a,b) → negspouse(a,b)
child(a,c) ∧ parent(c,b) → spouse(a,b)
child(b,a) ∧ child(c,a) → spouse(b,c)
parent(a,b) ∧ parent(b,c) → negparent(a,c)
parent(a,b) ∧ child(c,a) → spouse(b,c)
spouse(a,b) ∧ parent(c,a) → negspouse(b,c)
spouse(a,b) ∧ child(a,c) → negspouse(b,c)
successor(a,c) ∧ successor(b,c) → negspouse(a,b)
publisher(c,b) ∧ subsequentwork(c,a) → publisher(a,b)
publisher(c,b) ∧ previouswork(c,a) → publisher(a,b)

Figure D.2 – The set of rules used for fine tuning RULEBERT$_{20}$ in the experiment of Section 5.3.3 (unseen rules).

	Train Letter	**Train Name**
Test Letter	.981	.932
Test Name	.977	.985

Table D.3 – Impact of the example format on accuracy.

spouse(a,b)), with 30K examples for fine-tuning, and 2k for testing.

Results. The results in Table D.3 show that the model outcome does not depend heavily on using the same fact format for training and testing. With examples using letters in training, the results are slightly better in the case with two formats. We ultimately use names for test and train in our default configuration as it yields the best results.

D.4 Data Generation Example

We show an example of data generation for Algorithm 8. For simplicity, we show an example for a hard rule (rule confidence is implicitly equal to one) and show an example of a soft rule. We begin by setting the input parameters:

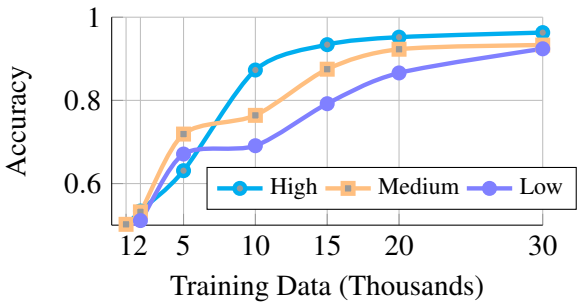


Figure D.3 – Impact of the training data size.

Algorithm 1 Input:

- $r =$ `child(A,C) ∧ parent(C,B) → spouse(A,B)`
- $n = 8$
- $m = 5$
- $pools =$ `{Alice,Bob,Carl,David,Eve}`

We set $n = 8$ to generate all the eight hypotheses. We start by generating a set of facts F (line 3), having predicates from the body of the rule with random polarity. We ensure that there are facts which trigger the rule. The number of facts should not exceed m. Example of generated facts F:

Generated Facts F:

- f_1: `negparent(Eve,Carl)`
- f_2: `child(Eve,David)`
- f_3: `parent(Carl,Bob)`
- f_4: `child(Alice,Carl)`

Four facts are generated in total. Facts f_3 and f_4 trigger rule r. We then feed the rule r and facts F into the LP$^{\text{MLN}}$reasoner (line 4). The output O is then:

LP$^{\text{MLN}}$Reasoner Output O:

- o_1: `child(Eve,David)`
- o_2: `child(Alice,Carl)`
- o_3: `parent(Carl,Bob)`
- o_4: `spouse(Alice,Bob)`
- o_5: `negchild(Eve,Carl)`

We start generating the hypotheses:

Generated Hypotheses H:

- h_1: `child(Eve,David)`
- h_2: `child(David,Eve)`
- h_3: `spouse(Alice,Bob)`
- h_4: `negspouse(Alice,Bob)`
- h_5: `child(David,Carl)`
- h_6: `negchild(David,Carl)`
- h_7: `spouse(Bob,Eve)`
- h_8: `negspouse(Bob,Eve)`

The hypothesis h_1 is obtained by sampling from F (line 5), making it a valid hypothesis. Then, the hypothesis h_2 is generated by altering h_1 with the function $Alter$(line 19-22). In this example, since *child* is not symmetric, h_2 is produced a switch of the subject and the object of h_1 to produce a false hypothesis (line 6).

The hypothesis h_3 is the outcome of rule r being triggered by facts f_3 and f_4 (line 7). In a similar fashion to h_2, we produce h_4 (line 8).

Hypothesis h_5 is sampled from the universe of all unsatisfied positive facts having a different predicate than that of the rule body (line 9), making it an invalid hypothesis, as it is not found in the O. Hypothesis h_6 is the negation of h_5, and, following CWA, it is a valid hypothesis (line 10).

Finally, hypothesis h_7 is sampled from the universe of unsatisfied rule-head atoms (line 11), and it is negated to produce hypothesis h_8.

We thus obtain eight different examples represented in symbolic knowledge, where each example contains the set of generated facts F, the rule r, and a single hypothesis h_i. The following is one example in symbolic knowledge:

Example #1 (Symbolic):

- *Rule* r = `child(A,C) ∧ parent(C,B) → spouse(A,B)`
- *Facts* F :
  - f_1: `negparent(Eve,Carl)`
  - f_2: `child(Eve,David)`
  - f_3: `parent(Carl,Bob)`
  - f_4: `child(Alice,Carl)`
- *Hypothesis* h_3 : `spouse(Alice,Bob)`

We then convert each example to synthetic English using pre-defined templates for the facts and rules. Here is Example #1 in synthetic English.

Example #1 (Synthetic English):

- *Rule* $r =$ If the child of the first person is the third person, and the parent of the third person is the second person, then the first person is the spouse of the second person.
- *Facts* F :
  - f_1**:** The parent of Eve is not Carl.
  - f_2**:** The child of Eve is David.
  - f_3**:** The parent of Carl is Bob.
  - f_4**:** The child of Alice is Carl.
- *Hypothesis* h_3 : The spouse of Alice is Bob.

The *Context* is defined as the combined set of facts and rule(s). Both *Context* and *Hypothesis* are fed as an input to the model.

Example #1 (Model Input):

- *Context* : The parent of Eve is not Carl. The child of Eve is David. If the child of the first person is the third person, and the parent of the third person is the second person, then the first person is the spouse of the second person. The parent of Carl is Bob. The child of Alice is Carl.
- *Hypothesis* : The spouse of Alice is Bob.

Résumé en français

Un *Knowledge Graph* est une représentation structurée d'information qui contient des entités du monde réel sous forme de nœuds, et les relations entre elles sous forme d'arêtes. Les entités et les relations dans un KG ont des descriptions sémantiques sous forme de types et de propriétés qui leur sont associés. Les KG représentent les données avec de grandes collections d'entités interconnectées. Généralement, un riche ensemble de types (classes) est disponible pour décrire les entités (par exemple, l'entité *Paris* est une *ville*, *France* est un *pays*), tandis que les prédicats décrivent leurs relations (une ville *estCapitale* d'un pays) et leurs propriétés (la France a une *population :62M*).

Les KG organisent les informations sous forme triplets avec *predicate* exprimant une relation binaire entre un *sujet* et un *objet*. Les KG stockent une grande quantité d'informations factuelles, et les triplets KG, ou les faits (*facts*), représentent des informations sur les entités du monde réel, leurs propriétés et leurs relations, telles que "Larry Page est le fondateur de Google".

La création d'un KG en RDF est une activité importante dans les systèmes d'information modernes et, au cours des 15 dernières années, beaucoup de grands KG académiques [3–6] et institutionnels [7, 14, 54]. ont été conçus. Par exemple, la version anglaise de DBpedia stocke 6M d'entités et 9B de triplets de relations. [2]. Les structures syntaxiques et sémantiques de la connaissance dans les KG sont utiles pour construire des applications telles que les Systèmes de Questions-Réponses et la Recherche Sémantique.

L'un des principaux défis que posent les KG est la qualité de leurs données. Cela est dû aux processus impliqués dans leur création et leur mise à jour. Les données structurées des KG sont généralement extraites de sources multiples, éventuellement du Web, sans validation humaine. Cela soulève deux problèmes principaux. Le premier problème est celui des erreurs factuelles. Des données incorrectes ou périmées peuvent être transmises des sources aux KG, ou bien le bruit peut provenir des extracteurs automatiques [9, 27]. Le deuxième problème est l'incomplétude. Comme un graphe est rarement complet dans la pratique, *Closed World Assumption* (CWA) n'est pas valable en pratique. (CWA) ne tient pas pas pour les KGs [9, 28], c'est-à-dire qu'il n'est pas possible de conclure qu'un fait manquant est faux. Nous suivons donc la *Open World Assumption* (OWA) et le considérons comme *inconnu*. De plus, dans de nombreux cas, le schéma du KG n'est pas fixe, c'est-à-dire que l'ensemble des prédicats change au fil du temps, et que de nouveaux faits peuvent être insérés sans contrôle d'intégrité.

En raison de ces problèmes, la quantité d'incomplétude et d'erreurs dans les KG peut être importante, avec jusqu'à 30% d'erreurs rapportées pour les données extraites du Web [29, 30]. Les KG peuvent contenir de nombreuses data, par exemple, WIKIDATA compte plus d'un milliard

[2] http ://wiki.dbpedia.org/dbpedia-version-2016-04

de faits et des millions d'entités, il n'est donc pas possible de vérifier manuellement les triplets pour trouver les erreurs et ajouter les faits manquants. Des outils sont nécessaires pour aider les humains dans la curation des KG. Une approche naturelle dans cette direction est d'extraire des *règles déclaratives*. Une fois validées, ces règles peuvent être exécutées à grande échelle sur le KG pour augmenter la qualité de ses data [10, 28, 31–33].

$$r_{pos} = \texttt{spouse}(a,b) \wedge \texttt{child}(a,v_0) \rightarrow \texttt{child}(b,v_0)$$
$$r_{neg} = \texttt{parent}(a,b) \wedge \texttt{birthDate}(b,v_0) \wedge \texttt{birthDate}(a,v_1) \wedge v_0 > v_1$$

Par exemple, la règle r_{pos} stipule que si deux personnes sont dans la relation de *conjoint*, chacune d'elles est le parent de l'enfant de l'autre, et la règle r_{neg} exprime qu'une personne ne peut avoir une date de naissance inférieure à celle de ses parents.

Ces règles doivent être élaborées manuellement pour être exécutées sur les KG. Ce processus peut s'avérer difficile, car les experts du domaine, qui connaissent la sémantique de l'ensemble de données en question, peuvent ne pas avoir les connaissances en informatique nécessaires pour exprimer ces règles formellement. De plus, l'écriture d'un ensemble de règles est chronophage, car il peut y en avoir des milliers qui sont nécessaires pour obtenir des résultats de bonne qualité. Dans ce contexte, un système d'extraction de règles est important pour aider les utilisateurs dans la conservation des données, ainsi que dans toute tâche impliquant un raisonnement sur le KG [35, 36].

L'extraction de règles (Rule mining) est le processus d'extraction automatique de règles logiques à partir du KG. Ces règles peuvent être exploitées dans la curation des KG en réduisant les incohérences ou en y ajoutant de nouveaux faits. Par exemple, r_{pos} peut nous aider à compléter le KG en ajoutant de nouvelles arêtes entre les entités tandis que r_{neg} nous aidera à supprimer les nœuds ou les arêtes qui ne la satisfont pas.

Trois problèmes principaux rendent l'extraction de règles à partir de KG difficile :

Qualité des données. La plupart des algorithmes de fouille de règles supposent que les données d'entrée comportent de très petites quantités d'erreurs [37–40], mais les KG sont incomplets et peuvent comporter des pourcentages élevés d'erreurs.

Open World Assumption. Certaines méthodes supposent que des exemples positifs et négatifs sont disponibles [41, 42]. Cependant, les KG ne contiennent que des déclarations positives, et il n'est pas possible de supposer le CWA, car il n'y a pas de solution évidente pour dériver des faits négatifs servant de contre-exemples.

Data Volume. Plusieurs algorithmes existants de fouille de règles exploitent le principe selon lequel le graphe d'entrée peut tenir entièrement dans la mémoire centrale principale [28, 32, 33, 43]. Comme les KG peuvent avoir une très grande taille, les méthodes existantes limitent la taille de l'espace de recherche en restreignant l'exploration à un langage de règles simples, ce qui fait que certains motifs importants dans les données peuvent être ratés.

De plus, comme on peut le voir dans r_{pos}, la plupart des règles logiques ne sont pas vraies dans tous les cas et pour cette raison, un score de confiance doit leur être attribué. Définir ces indices de confiance d'une manière qui représente la précision des règles est un autre défi qui doit être résolu dans la tâche de découverte automatique de règles.

Nous commençons cette thèse en proposant une méthode pour créer un Knowledge Graph pour le domaine de l'audit. Nous créons ce KG en utilisant des documents d'audit et des

taxonomies et les relations entre eux. Ensuite, nous étendons un système de fouille de règles pour améliorer ses performances en extrayant des règles conditionnelles et des règles sur des prédicats littéraux. Nous proposons également un score de confiance pour calculer la précision des règles positives et négatives. Nous avons également utilisé ces règles dans des tâches telles que la vérification des faits et le transfert de connaissances de sens commun à des modèles linguistiques pré-entraînés (Pre-trained Language Models). L'objectif principal de cette thèse est d'étudier de nouvelles approches pour la création et la curation continue de KGs. Pour cela, nous introduisons un algorithme pour trouver des entités et leurs relations et nous développons également un algorithme efficace de fouille de règles pour trouver des règles positives et négatives. Nous proposons également des mesures pour calculer la confiance des règles positives et négatives et les employons dans une application de vérification des faits.

Dans notre recherche, nous avons utilisé des KG généraux (*DBPedia*, *Wikidata*) au lieu du KG de KPMG. Cette décision a été prise pour deux raisons(i) leur KG n'est pas assez mature pour appliquer ces techniques, (ii) en raison de l'accord de confidentialité, nous ne pouvons pas exposer leur KG. Dans ce qui suit, nous résumerons nos contributions dans les différents domaines qui ont été abordés par cette thèse.

Création d'un KG d'audit (Chapitre 3)

Dans ce chapitre, nous présentons d'abord notre méthode d'identification automatique des nœuds pour la création du KG de KPMG, puis nous discutons de notre solution pour l'identification des relations entre les différents nœuds d'activité et de taxonomie de KPMG. Ces relations seront représentées comme des arêtes dans le KG. Nous n'avons pas pu trouver de cadre de conversion de texte en structure qui puisse fonctionner dans notre contexte et nous avons donc décidé de développer un cadre pour identifier les arêtes dans le KG de KPMG (conversion de texte en texte structuré). Nos expériences ont montré que le modèle proposé fonctionne bien pour d'autres tâches de correspondance.

La tâche d'identification des nœuds couvre la génération de deux types de nœud dans le KG de KPMG : les entités et les mots. Elle contient deux sous-tâches : i) trouver dans le corpus de documents les entités qui peuvent être représentatives, et ii) trouver les membres de la famille (mots) pour chaque entité représentative.

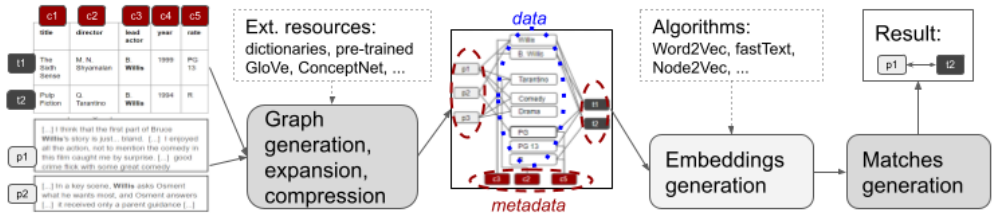


FIGURE 7.4 – Le cadre proposé : (1) les documents de texte et de données structurées sont modélisés conjointement dans un graphe, (ii) des embeddings sont produits pour les nœuds de données et de métadonnées (représentant des textes, des nœuds de taxonomie, des tuples), (iii) les nœuds de métadonnées sont mis en correspondance dans une approche non supervisée.

Pour trouver les relations entre différentes entités dans le KG de KPMG, nous proposons un cadre pour l'apprentissage de représentations de données et de textes qui (i) est adapté au domaine en question avec une modélisation conjointe de corpus hétérogènes et (ii) exploite les informations structurées disponibles pour améliorer la qualité des embeddings générés et du processus d'appariement. Cette méthode a été initialement conçue pour l'identification d'arêtes dans le KG de KPMG mais nos expériences montrent qu'elle est performante dans d'autres tâches.

La figure 7.4 montre les différents composants de notre cadre. Tout d'abord, il représente les documents textuels et les tableaux comme des nœuds et des arêtes dans un graphe non orienté. Ce graphe contient deux types de nœud principaux. Les nœuds *Data* représentent des tokens (mots) dans les corpus, soit dans des paragraphes de texte, soit dans des cellules de tableau. Les nœuds *Metadata* représentent les ID des tuples, des attributs et des paragraphes. Les arêtes du graphe représentent la relation entre les données et les métadonnées, par exemple, un tuple/attribut/paragraphe contient le token dans un nœud de données. Comme notre objectif est de faire correspondre les nœuds de métadonnées, nous cherchons à créer davantage de chemins entre les nœuds liés et à supprimer les connexions parasites. Le premier objectif est atteint dans une étape d'expansion qui exploite des ressources externes, telles que ConceptNet [21]. Le second objectif est obtenu en élaguant les arêtes et les nœuds à l'aide d'une technique de compression de graphes conçue pour notre tâche d'appariement.

Ensuite, nous générons un embedding pour chaque nœud du graphe. Nous nous appuyons sur des solutions existantes pour cette étape et l'algorithme actuel peut être remplacé au fur et à mesure que la communauté progresse dans cette tâche. Enfin, nous utilisons les embeddings pour les nœuds de métadonnées dans un algorithme non supervisé pour identifier les nœuds correspondants, comme le paragraphe et le tuple dans le premier exemple.

Le cadre permet aux utilisateurs d'améliorer la solution en fonction des exigences et des ressources dont ils disposent. S'il existe des ressources externes pertinentes, comme des dictionnaires de mots, elles peuvent être intégrées dans la construction du graphe pour fusionner les nœuds de données. Les graphes de connaissances et les ontologies peuvent être insérés dans l'étape d'expansion pour trouver davantage de relations entre les nœuds de métadonnées. Si un nouvel algorithme de génération d'embeddings est disponible, il peut être intégré dans la deuxième étape pour améliorer la qualité des embeddings.

Les résultats publiés dans ce chapitre sont basés sur le publication suivant :

> Naser Ahmadi, Hansjorg Sand, and Paolo Papotti, **Unsupervised Matching of Data and Text**, (Submitted for publication).

Découverte de règles conditionnelles (Chapitre 4)

Ce chapitre contient notre travail d'extension d'un système précédent de découverte de règles (RuDiK). Le but de notre travail est de permettre à RuDiK d'extraire des règles conditionnelles. De nombreuses règles ne sont valables que dans une zone géographique donnée, à un moment précis, ou pour des catégories d'entités spécifiques. Nous étendons les méthodes d'extraction de base pour capturer les règles conditionnelles, qui sont des règles qui ne s'appliquent qu'à un sous-ensemble des données et qui sont reconnues par une sélection avec une constante sur les

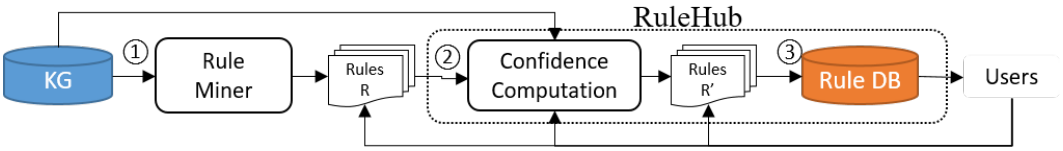


FIGURE 7.5 – Architecture du RuleHub.

valeurs d'une entité ou d'un type. Nous trouvons ces sous-ensembles de données, pour lesquels l'exploration conduit à de nouvelles règles qui ne sont pas identifiées par l'exploration générale, en utilisant le regroupement d'entités.

Nous présentons une méthode pour extraire les règles conditionnelles dans les KGs et nos expériences ont montré que cette technique est capable de générer plus de règles en comparaison avec la technique générique. Il est également démontré expérimentalement que les règles conditionnelles sont plus précises que les règles génériques et qu'elles couvrent un plus petit sous-ensemble de triplets.

Les résultats de ce chapitre sont basés sur l'article suivant :

> Naser Ahmadi, Phi Huynh, Vamsi Meduri, Stefano Ortona and Paolo Papotti, **Mining Expressive Rules in Knowledge Graphs**, ACM Journal of Data and Information Quality, 2019.

Un corpus de règles et comment les enseigner aux PLMs (Chapitre 5)

La première section de ce chapitre présente RuleHub, un système qui expose sur un site Web un corpus extensible de règles pour les KG publiques (`http://rudik.eurecom.fr`). RuleHub est conçu avec des outils de fouille de règles comme principale source de règles à l'esprit, comme le montre la figure 7.5. Les utilisateurs peuvent interroger ou parcourir le référentiel de règles, en fonction du KG et du prédicat qui les intéresse. De plus, ils peuvent spécifier et ajouter manuellement de nouvelles règles ou mettre à jour les règles existantes en fournissant plus de métadonnées, comme la confiance d'une règle. Nous pensons que la confiance est cruciale pour les règles, car très peu de règles sont complètement correctes ou fausses en général. Pour cette raison, nous fournissons également un module qui calcule la confiance pour une règle donnée sur un KG. Les règles pour n'importe quel KG et de n'importe quel type peuvent être traitées par le système. Dans ce chapitre, nous rapportons notre expérience de la construction de RuleHub et de son alimentation avec des règles découvertes par des systèmes de fouille de règles existants. Nous pensons que RuleHub peut être un catalyseur pour un travail collaboratif indispensable à la définition de métadonnées pour les KG publiques.

RuleHub présente les contributions suivantes :

- Nous introduisons une nouvelle méthode de calcul de la confiance pour les règles négatives, c'est-à-dire les règles qui identifient des contradictions dans les données.
- Nous présentons le premier corpus ouvert de règles (générées automatiquement) pour les KG publiques et détaillons son modèle de données et ses composants.

- Nous rapportons les leçons apprises lors de la création d'un tel corpus et les résultats expérimentaux montrant comment notre modèle de calcul de la confiance des règles présente une corrélation remarquable avec les valeurs de confiance annotées manuellement.

Dans la deuxième partie du chapitre 5, nous proposons deux méthodes d'extraction de règles logiques pour les KG de Wikidata et la troisième section de ce chapitre aborde le problème du transfert des connaissances de sens commun vers des modèles de langage pré-entraînés. Pour cette question, nous développons un modèle (**RuleBert**) pour transférer la connaissance dans les règles logiques extraites pour les KG à un modèle de langage. RuleBert présente les contributions suivantes :

- Nous introduisons le problème de l'enseignement de règles souples exprimées dans un langage synthétique à des PLM par le biais d'un réglage fin (modélisé comme une classification binaire).

- Nous créons et publions le premier jeu de données pour cette tâche. Ce jeu de données contient 3,2 millions d'exemples dérivés de 161 règles décrivant des modèles réels de sens commun avec la probabilité cible pour la tâche obtenue à partir d'un raisonneur formel.

- Nous présentons des techniques permettant de prédire la probabilité correcte du résultat du raisonnement pour les règles et les faits souples donnés. Notre solution s'appuie sur une fonction de perte révisée qui modélise efficacement l'incertitude des règles. Notre approche gère les règles multi-variables et s'étend bien aux exemples qui nécessitent un raisonnement sur des règles à entrées multiples.

- Nous montrons que notre approche permet d'obtenir des modèles à réglage fin qui donnent une probabilité de prédiction très proche de celle produite par un raisonneur formel. Un PLM réglé sur des règles souples, RULEBERT, peut raisonner efficacement sur des faits et des règles qui n'ont pas été vus lors de la formation, même lorsqu'il est réglé avec seulement 20 règles.

Ce chapitre est basé sur les publications suivantes :

Naser Ahmadi, Duyen Truong, Mai Dao, Stefano Ortona, and Paolo Papotti, **RuleHub : a Public Corpus of Rules for Knowledge Graphs**, ACM Journal of Data and Information Quality, 2020.

Naser Ahmadi, and Paolo Papotti, **Wikidata logical rules and where to find them**, Wiki Workshop, 2021.

Mohammed Saeed, Naser Ahmadi, Paolo Papotti, and Preslav Nakov, **Teaching Soft Rules to Pre-trained Language Models**, The 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2021.

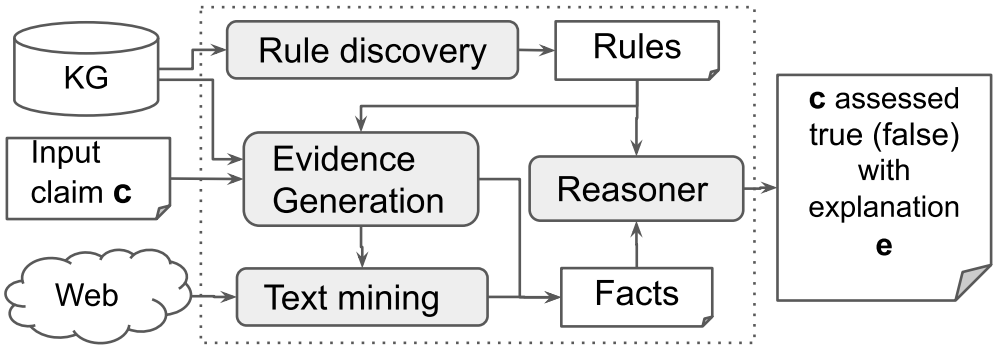


FIGURE 7.6 – Notre cadre de vérification des faits EXPCLAIM.

Vérification explicable des faits (Chapitre 6)

Nous présentons ExpClaim pour la vérification des faits basée sur des règles extraites de KGs. ExpClaim est un système de vérification des faits entièrement *automatisé et interprétable* qui exploite efficacement les preuves incertaines. La figure 7.6 montre notre cadre, EXPCLAIM. Le module *Rule discovery* prend en entrée le KG K pour générer les règles. Nous convertissons ensuite les règles découvertes Σ dans le langage d'entrée du raisonneur, où le poids d'une règle est son support. Pour une déclaration (claim) donnée $c : p(x,y)$, $p \in K$ et les règles Σ, le module *Evidence Generation* collecte les faits pertinents (triples satisfaisant le corps des règles) à partir du KG et du Web avec le module *Text mining*.

Nous transmettons ensuite les règles et les preuves au module *Reasoner*, où différents modes de calcul peuvent être utilisés pour déduire si $p(x,y)$ ou $negp(x,y)$ fait partie de l'ensemble de réponses. La sortie du raisonneur comprend une explication de la décision interprétable par l'humain. Les détails des principales étapes sont donnés ci-après.

Les résultats expérimentaux sur un KG réel montrent que notre méthode (i) obtient des résultats qualitatifs comparables ou meilleurs que les méthodes ML existantes de type boîte noire et (ii) produit des explications consommables par l'humain. ExpClaim est présenté dans l'article suivant :

> Naser Ahmadi, Joohyung Lee, Paolo Papotti and Mohammed Saeed, **Explainable Fact Checking with Probabilistic Answer Set Programming**, Conference for Truth and Trust Online (TTO), 2019.

Bibliography

[1] T. Safavi and D. Koutra, "Relational world knowledge representation in contextual language models: A review," *arXiv preprint arXiv:2104.05837*, 2021.

[2] A. Piscopo and E. Simperl, "What we talk about when we talk about wikidata quality: a literature survey," in *Proceedings of the 15th International Symposium on Open Collaboration*, 2019, pp. 1–11.

[3] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell, "Toward an architecture for never-ending language learning." in *AAAI*, 2010, pp. 1306–1313.

[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia-A crystallization point for the web of data," *Web Semantics: science, services and agents on the WWW*, vol. 7, no. 3, pp. 154–165, 2009.

[5] F. M. Suchanek, G. Kasneci, and G. Weikum, "YAGO: A core of semantic knowledge unifying wordnet and wikipedia," in *WWW*, 2007, pp. 697–706.

[6] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Comm. of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *SIGMOD*, 2008, pp. 1247–1250.

[8] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré, "Incremental knowledge base construction using DeepDive," *PVLDB*, vol. 8, no. 11, pp. 1310–1321, 2015.

[9] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang, "From data fusion to knowledge fusion," *PVLDB*, vol. 7, no. 10, pp. 881–892, 2014.

[10] O. Deshpande, D. S. Lamba, M. Tourn, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan, "Building, maintaining, and using knowledge bases: a report from the trenches," in *SIGMOD*, 2013, pp. 1209–1220.

[11] C. Unger, A. Freitas, and P. Cimiano, "An introduction to question answering over linked data," in *Reasoning Web International Summer School*. Springer, 2014, pp. 100–140.

[12] D. Diefenbach, V. Lopez, K. Singh, and P. Maret, "Core techniques of question answering systems over knowledge bases: a survey," *Knowledge and Information systems*, vol. 55, no. 3, pp. 529–569, 2018.

[13] H. Bast, B. Björn, and E. Haussmann, "Semantic search on text and knowledge bases," *Foundations and Trends in Information Retrieval*, vol. 10, no. 2-3, pp. 119–271, 2016.

[14] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: a web-scale approach to probabilistic knowledge fusion," in *KDD*, 2014, pp. 601–610.

[15] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic web*, vol. 8, no. 3, pp. 489–508, 2017.

[16] T. Steiner, R. Verborgh, R. Troncy, J. Gabarro, and R. Van de Walle, "Adding realtime coverage to the google knowledge graph," in *11th International Semantic Web Conference (ISWC 2012)*, vol. 914. Citeseer, 2012, pp. 65–68.

[17] X. Zou, "A survey on application of knowledge graph," in *Journal of Physics: Conference Series*, vol. 1487, no. 1. IOP Publishing, 2020, p. 012016.

[18] S. Auer and S. Mann, "Towards an open research knowledge graph," *The Serials Librarian*, vol. 76, no. 1-4, pp. 35–41, 2019.

[19] M. Kejriwal, *Domain-specific knowledge graph construction*. Springer, 2019.

[20] T. Adams, "Google and the future of search: Amit singhal and the knowledge graph," *The Guardian*, vol. 19, 2013.

[21] R. Speer, J. Chin, and C. Havasi, "Conceptnet 5.5: An open multilingual graph of general knowledge," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[22] M. Sap, R. L. Bras, E. Allaway, C. Bhagavatula, N. Lourie, H. Rashkin, B. Roof, N. A. Smith, and Y. Choi, "ATOMIC: an atlas of machine commonsense for if-then reasoning," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 3027–3035.

[23] M. Kejriwal, R. Shao, and P. Szekely, "Expert-guided entity extraction using expressive rules," in *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, 2019, pp. 1353–1356.

[24] B. Abu-Salih, "Domain-specific knowledge graphs: A survey," *Journal of Network and Computer Applications*, vol. 185, p. 103076, 2021.

[25] S. Wu, L. Hsiao, X. Cheng, B. Hancock, T. Rekatsinas, P. Levis, and C. Ré, "Fonduer: Knowledge base construction from richly formatted data," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM, 2018, pp. 1301–1316.

[26] N. Jain, "Domain-specific knowledge graph construction for semantic analysis," in *European Semantic Web Conference*. Springer, 2020, pp. 250–260.

[27] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti, "Extraction and integration of partially overlapping web sources," *PVLDB*, vol. 6, no. 10, pp. 805–816, 2013.

[28] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek, "Fast rule mining in ontological knowledge bases with AMIE+," *The VLDB Journal*, vol. 24, no. 6, pp. 707–730, 2015.

[29] F. M. Suchanek, M. Sozio, and G. Weikum, "SOFIE: A self-organizing framework for information extraction," in *WWW*, 2009, pp. 631–640.

[30] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker, "Temporal rules discovery for web data cleaning," *PVLDB*, vol. 9, no. 4, pp. 336–347, 2015.

[31] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang, "Detecting data errors: Where are we and what needs to be done?" *PVLDB*, vol. 9, no. 12, pp. 993–1004, 2016.

[32] Z. Abedjan and F. Naumann, "Amending RDF entities with new facts," in *ESWC*, 2014, pp. 131–143.

[33] Y. Chen, S. Goldberg, D. Z. Wang, and S. S. Johri, "Ontological pathfinding," in *SIGMOD*, 2016, pp. 835–846.

[34] P. Suganthan GC, C. Sun, H. Zhang, F. Yang, N. Rampalli, S. Prasad, E. Arcaute, G. Krishnan *et al.*, "Why big data industrial systems need rules and what we can do about it," in *SIGMOD*, 2015, pp. 265–276.

[35] M. H. Gad-Elrab, D. Stepanova, J. Urbani, and G. Weikum, "Exfakt: A framework for explaining facts over knowledge graphs and text," in *WSDM*, 2019, pp. 87–95.

[36] L. Bellomarini, E. Sallinger, and G. Gottlob, "The vadalog system: Datalog-based reasoning for knowledge graphs," *PVLDB*, vol. 11, no. 9, pp. 975–987, 2018.

[37] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[38] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," *PVLDB*, vol. 6, no. 13, pp. 1498–1509, 2013.

[39] Z. Abedjan, L. Golab, and F. Naumann, "Data profiling: A tutorial," in *SIGMOD*, 2017, pp. 1747–1751.

[40] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," *The computer journal*, vol. 42, no. 2, pp. 100–111, 1999.

[41] L. Dehaspe and H. Toivonen, "Discovery of frequent datalog patterns," *DMKD*, vol. 3, no. 1, pp. 7–36, 1999.

[42] S. Muggleton and L. De Raedt, "Inductive logic programming: Theory and methods," *The Journal of Logic Programming*, vol. 19, pp. 629–679, 1994.

[43] M. H. Farid, A. Roatis, I. F. Ilyas, H. Hoffmann, and X. Chu, "CLAMS: bringing quality to data lakes," in *SIGMOD*, 2016, pp. 2089–2092.

[44] N. Ahmadi, H. Sand, and P. Papotti, "Unsupervised matching of data and text," 2021.

[45] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *SIGMOD*, 2020.

[46] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos, "Valentine: Evaluating matching techniques for dataset discovery," *CoRR*, vol. abs/2010.07386, 2020. [Online]. Available: https://arxiv.org/abs/2010.07386

[47] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan, "Deep entity matching with pre-trained language models," *PVLDB*, 2021.

[48] N. Ahmadi, V.-P. Huynh, V. V. Meduri, S. Ortona, and P. Papotti, "Mining expressive rules in knowledge graphs," *Journal of Data and Information Quality (JDIQ)*, vol. 12, no. 2, pp. 1–27, 2020.

[49] N. Ahmadi, T.-T.-D. Truong, L.-H.-M. Dao, S. Ortona, and P. Papotti, "Rulehub: A public corpus of rules for knowledge graphs," *Journal of Data and Information Quality (JDIQ)*, vol. 12, no. 4, pp. 1–22, 2020.

[50] N. Ahmadi and P. Papotti, "Wikidata logical rules and where to find them," in *Companion Proceedings of the Web Conference 2021*, 2021, pp. 580–581.

[51] M. Saeed, N. Ahmadi, P. Nakov, and P. Papotti, "Rulebert: Teaching soft rules to pre-trained language models," *arXiv preprint arXiv:2109.13006*, 2021.

[52] N. Ahmadi, J. Lee, P. Papotti, and M. Saeed, "Explainable fact checking with probabilistic answer set programming," in *Conference for Truth and Trust Online (TTO)*, 2019.

[53] G. Weikum, "Knowledge graphs 2021: A data odyssey," *Proc. VLDB Endow.*, vol. 14, no. 12, pp. 3233–3238, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol14/p3233-weikum.pdf

[54] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. Welty, "Building Watson: An overview of the DeepQA project," *AI Mag.*, vol. 31, no. 3, pp. 59–79, 2010.

[55] A. Saeedi, E. Peukert, and E. Rahm, "Using link features for entity clustering in knowledge graphs," in *European Semantic Web Conference*. Springer, 2018, pp. 576–592.

[56] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion." in *AAAI*, 2015, pp. 2181–2187.

[57] D. J. Pearce and P. H. Kelly, "A dynamic topological sort algorithm for directed acyclic graphs," *Journal of Experimental Algorithmics (JEA)*, vol. 11, pp. 1–7, 2007.

[58] B. Golden, "Shortest-path algorithms: A comparison," *Operations Research*, vol. 24, no. 6, pp. 1164–1168, 1976.

[59] R. Mihalcea, P. Tarau, and E. Figa, "Pagerank on semantic networks, with application to word sense disambiguation," in *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, 2004, pp. 1126–1132.

[60] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[61] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1247–1261.

[62] C. Xiong, J. Callan, and T.-Y. Liu, "Word-entity duet representations for document ranking," in *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, 2017, pp. 763–772.

[63] G. Weikum, L. Dong, S. Razniewski, and F. Suchanek, "Machine knowledge: Creation and curation of comprehensive knowledge bases," *arXiv preprint arXiv:2009.11564*, 2020.

[64] Z. Zhao, S.-K. Han, and I.-M. So, "Architecture of knowledge graph construction techniques," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 19, pp. 1869–1883, 2018.

[65] H. Cunningham, "Gate: A framework and graphical development environment for robust nlp tools and applications," in *Proc. 40th annual meeting of the association for computational linguistics (ACL 2002)*, 2002, pp. 168–175.

[66] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Unsupervised named-entity extraction from the web: An experimental study," *Artificial intelligence*, vol. 165, no. 1, pp. 91–134, 2005.

[67] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[68] J. Li, A. Sun, J. Han, and C. Li, "A survey on deep learning for named entity recognition," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[69] S. Geetha, G. A. Mala, and N. Kanya, "A survey on information extraction using entity relation based methods," 2011.

[70] S. Sarawagi, *Information extraction*. Now Publishers Inc, 2008.

[71] X. Han, T. Gao, Y. Lin, H. Peng, Y. Yang, C. Xiao, Z. Liu, P. Li, M. Sun, and J. Zhou, "More data, more relations, more context and more openness: A review and outlook for relation extraction," *arXiv preprint arXiv:2004.03186*, 2020.

[72] H. Wang, M. Tan, M. Yu, S. Chang, D. Wang, K. Xu, X. Guo, and S. Potdar, "Extracting multiple-relations in one-pass with pre-trained transformers," *arXiv preprint arXiv:1902.01030*, 2019.

[73] S. Wu and Y. He, "Enriching pre-trained language model with entity information for relation classification," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 2361–2364.

[74] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Icml*, 2011.

[75] M. Nentwig, M. Hartung, A.-C. Ngonga Ngomo, and E. Rahm, "A survey of current link discovery frameworks," *Semantic Web*, vol. 8, no. 3, pp. 419–436, 2017.

[76] A. Bordes, X. Glorot, J. Weston, and Y. Bengio, "A semantic matching energy function for learning with multi-relational data," *Machine Learning*, vol. 94, no. 2, pp. 233–259, 2014.

[77] D. C. Faye, O. Cure, and G. Blin, "A survey of rdf storage approaches," *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées*, vol. 15, pp. 11–35, 2012.

[78] R. kumar Kaliyar, "Graph databases: A survey," in *International Conference on Computing, Communication & Automation*. IEEE, 2015, pp. 785–790.

[79] S. Ortona, V. V. Meduri, and P. Papotti, "Robust discovery of positive and negative rules in knowledge bases," in *ICDE*, 2018, pp. 1168–1179.

[80] M. Loster, F. Naumann, J. Ehmueller, and B. Feldmann, "Curex: A system for extracting, curating, and exploring domain-specific knowledge graphs from text," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 1883–1886.

[81] "Shapes constraint language (SHACL)," W3C, Tech. Rep., Jul. 2017. [Online]. Available: https://www.w3.org/TR/shacl/

[82] J. E. L. Gayo, E. Prud'hommeaux, H. R. Solbrig, and J. M. Á. Rodríguez, "Validating and describing linked data portals using rdf shape expressions." in *LDQ@ SEMANTICS*. Citeseer, 2014.

[83] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek, "Amie: association rule mining under incomplete evidence in ontological knowledge bases," in *WWW*. ACM, 2013, pp. 413–422.

[84] J. Lajus, L. Galárraga, and F. Suchanek, "Fast and exact rule mining with amie 3," in *European Semantic Web Conference*. Springer, 2020, pp. 36–52.

[85] Y. Dai, S. Wang, N. N. Xiong, and W. Guo, "A survey on knowledge graph embedding: Approaches, applications and benchmarks," *Electronics*, vol. 9, no. 5, p. 750, 2020.

[86] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013.

[87] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[88] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes." in *AAAI*, 2014, pp. 1112–1119.

[89] B. Shi and T. Weninger, "Proje: Embedding projection for knowledge graph completion," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[90] W. Qian, C. Fu, Y. Zhu, D. Cai, and X. He, "Translating embeddings for knowledge graph completion with relation attention mechanism." in *IJCAI*, 2018, pp. 4286–4292.

[91] J. Weston, A. Bordes, O. Yakhnenko, and N. Usunier, "Connecting language and knowledge bases with embedding models for relation extraction," *arXiv preprint arXiv:1307.7973*, 2013.

[92] J. L. Martinez-Rodriguez, I. Lopez-Arevalo, and A. B. Rios-Alvarado, "Openie-based approach for knowledge graph construction from text," *Expert Systems with Applications*, vol. 113, pp. 339–355, 2018.

[93] J. Marciniak, "Wordnet as a backbone of domain and application conceptualizations in systems with multimodal data," in *Proceedings of the LREC 2020 Workshop on Multimodal Wordnets (MMW2020)*. Marseille, France: The European Language Resources Association (ELRA), May 2020, pp. 25–32. [Online]. Available: https://www.aclweb.org/anthology/2020.mmw-1.5

[94] D. Ferrucci and A. Lally, "Uima: an architectural approach to unstructured information processing in the corporate research environment," *Natural Language Engineering*, vol. 10, no. 3-4, pp. 327–348, 2004.

[95] A. Ardalan, D. Paulsen, A. S. Saini, W. Cai, and A. Doan, "Toward data cleaning with a target accuracy: A case study for value normalization," *arXiv preprint arXiv:2101.05308*, 2021.

[96] G. Rizzo and R. Troncy, "Nerd: a framework for unifying named entity recognition and disambiguation extraction tools," in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 2012, pp. 73–76.

[97] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *Proceedings of the 2011 conference on empirical methods in natural language processing*, 2011, pp. 1535–1545.

[98] M. Hearst, "Automatic acquisition of hyponyms from large text corpora in proc," in *14th International Conference Computational Linguistics, Nantes France*, 1992.

[99] F. Mesquita, J. Schmidek, and D. Barbosa, "Effectiveness and efficiency of open relation extraction," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 447–457.

[100] M. A. Jaro, "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 414–420, 1989.

[101] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[102] L. Getoor and A. Machanavajjhala, "Entity resolution for big data," in *ACM SIGKDD*, 2013, pp. 1527–1527.

[103] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," in *ACL*, 2018, pp. 784–789.

[104] J. Guo, Y. Fan, X. Ji, and X. Cheng, "Matchzoo: a learning, practicing, and developing system for neural text matching," in *SIGIR*, 2019, pp. 1297–1300.

[105] W. Chen, H. Wang, J. Chen, Y. Zhang, H. Wang, S. Li, X. Zhou, and W. Y. Wang, "Tabfact: A large-scale dataset for table-based fact verification," in *ICLR*, 2020.

[106] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer, "Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2508–2521, 2020.

[107] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[108] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proc. VLDB Endow.*, vol. 11, no. 11, pp. 1454–1467, 2018.

[109] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *SIGMOD*. ACM, 2018, pp. 19–34.

[110] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *ACL*, 2019, pp. 4171–4186.

[111] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontañón, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big Bird: Transformers for longer sequences," *CoRR*, vol. abs/2007.14062, 2020. [Online]. Available: https://arxiv.org/abs/2007.14062

[112] S. Shaar, N. Babulkov, G. D. S. Martino, and P. Nakov, "That is a known lie: Detecting previously fact-checked claims," in *ACL*, 2020, pp. 3607–3618.

[113] A. Adhikari, A. Ram, R. Tang, and J. Lin, "Docbert: Bert for document classification," *arXiv preprint arXiv:1904.08398*, 2019.

[114] Y. Ibrahim, M. Riedewald, G. Weikum, and D. Zeinalipour-Yazti, "Bridging quantities in tables and text," in *ICDE*. IEEE, 2019, pp. 1010–1021.

[115] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos, "Tapas: Weakly supervised table parsing via pre-training," in *ACL*, 2020, pp. 4320–4333.

[116] P. Yin, G. Neubig, W. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," in *ACL*, 2020, pp. 8413–8426.

[117] H. Chen, S. Wadhwa, X. D. Li, and A. Zukov-Gregoric, "Yelm: End-to-end contextualized entity linking," *arXiv preprint arXiv:1911.03834*, 2019.

[118] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.

[119] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *NIPS*, 2015, pp. 3294–3302.

[120] C. De Boom, S. Van Canneyt, T. Demeester, and B. Dhoedt, "Representation learning for very short texts using weighted word embedding aggregation," *Pattern Recognition Letters*, vol. 80, pp. 150–156, 2016.

[121] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[122] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[123] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *SIGKDD*, 2014, pp. 701–710.

[124] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*, 2016, pp. 855–864.

[125] C. Fu, X. Han, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong, "End-to-end multi-perspective matching for entity resolution," in *IJCAI*, 2019.

[126] R. C. Fernandez and S. Madden, "Termite: a system for tunneling through heterogeneous data," in *aiDM*. ACM, 2019, pp. 7:1–7:8.

[127] M. Günther, P. Oehme, M. Thiele, and W. Lehner, "Learning from textual data in database systems," in *CIKM*, 2020, pp. 375–384.

[128] D. Freedman and P. Diaconis, "On the histogram as a density estimator: L2 theory," *Z. Wahrscheinlichkeitstheorie und verwandte Gebiete*, vol. 57, no. 4, pp. 453–476, 1981.

[129] G. A. Miller, *WordNet: An electronic lexical database*. MIT press, 1998.

[130] I. Yamada, H. Shindo, H. Takeda, and Y. Takefuji, "Joint learning of the embedding of words and entities for named entity disambiguation," in *SIGNLL*, 2016.

[131] E. J. Gerritse, F. Hasibi, and A. P. de Vries, "Graph-embedding empowered entity retrieval," in *European Conference on Information Retrieval*, 2020, pp. 97–110.

[132] P. Mcnamee and J. Mayfield, "Character n-gram tokenization for european language text retrieval," *Information retrieval*, vol. 7, no. 1-2, pp. 73–97, 2004.

[133] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in power-law networks," *Physical review E*, vol. 64, no. 4, p. 046135, 2001.

[134] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 62:1–62:34, 2018.

[135] M. P. Stumpf, C. Wiuf, and R. M. May, "Subnets of scale-free networks are not scale-free: sampling properties of networks," *Proceedings of the National Academy of Sciences*, vol. 102, no. 12, pp. 4221–4224, 2005.

[136] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 631–636.

[137] V. Krishnamurthy, M. Faloutsos, M. Chrobak, L. Lao, J.-H. Cui, and A. G. Percus, "Reducing large internet topologies for faster simulations," in *International Conference on Research in Networking*. Springer, 2005, pp. 328–341.

[138] N. K. Ahmed, J. Neville, and R. Kompella, "Network sampling: From static to streaming graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, no. 2, pp. 1–56, 2013.

[139] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 177–187.

[140] A. Rezvanian and M. R. Meybodi, "Sampling social networks using shortest paths," *Physica A: Statistical Mechanics and its Applications*, vol. 424, pp. 254–268, 2015.

[141] Y. Li, Z. Wu, S. Lin, H. Xie, M. Lv, Y. Xu, and J. C. Lui, "Walking with perception: Efficient random walk sampling via common neighbor awareness," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 962–973.

[142] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[143] K. Lee, H. Jo, J. Ko, S. Lim, and K. Shin, "Ssumm: Sparse summarization of massive graphs," in *SIGKDD*, 2020, pp. 144–154.

[144] S. Schoenmackers, O. Etzioni, D. S. Weld, and J. Davis, "Learning first-order horn clauses from web text," in *Empirical Methods in Natural Language Processing*, 2010, pp. 1088–1098.

[145] W. Fan and F. Geerts, *Foundations of Data Quality Management*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012. [Online]. Available: http://dx.doi.org/10.2200/S00439ED1V01Y201207DTM030

[146] S. Kruse and F. Naumann, "Efficient discovery of approximate dependencies," *PVLDB*, vol. 11, no. 7, pp. 759–772, 2018.

[147] W. Fan, Y. Wu, and J. Xu, "Functional dependencies for graphs," in *SIGMOD*, 2016, pp. 1843–1857.

[148] W. Fan, C. Hu, X. Liu, and P. Lu, "Discovering graph functional dependencies," in *SIGMOD*, 2018, pp. 427–439.

[149] G. Töpper, M. Knuth, and H. Sack, "Dbpedia ontology enrichment for inconsistency detection," in *I-SEMANTICS*, 2012, pp. 33–40.

[150] D. Wienand and H. Paulheim, "Detecting incorrect numerical data in dbpedia," in *ESWC*, 2014.

[151] B. Min, R. Grishman, L. Wan, C. Wang, and D. Gondek, "Distant supervision for relation extraction with an incomplete knowledge base." in *HLT-NAACL*, 2013, pp. 777–782.

[152] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[153] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[154] A. Bordes, J. Weston, R. Collobert, and Y. Bengio, "Learning structured embeddings of knowledge bases," in *AAAI*, 2011.

[155] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee, "Linked data on the web," in *WWW*, 2008, pp. 1265–1266.

[156] C. Mangold, "A survey and classification of semantic search approaches," *International Journal of Metadata, Semantics and Ontologies*, vol. 2, no. 1, pp. 23–34, 2007.

[157] M. Schmidt, M. Meier, and G. Lausen, "Foundations of sparql query optimization," in *ICDT*, 2010, pp. 4–33.

[158] T. P. Tanon, C. Bourgaux, and F. M. Suchanek, "Learning how to correct a knowledge base from the edit history," in *WWW*, 2019, pp. 1465–1475.

[159] Internet Engineering Task Force (IETF), "The javascript object notation (JSON) data interchange format," https://tools.ietf.org/html/std90.

[160] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," in *International semantic web conference*, 2006, pp. 30–43.

[161] M. G. Kendall, "The treatment of ties in ranking problems," *Biometrika*, vol. 33, no. 3, pp. 239–251, 1945.

[162] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2020. [Online]. Available: https://openreview.net/forum?id=SyxS0T4tvS

[163] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does bert look at? an analysis of bert's attention," in *BlackBoxNLP@ACL*, 2019.

[164] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in BERTology: What we know about how BERT works," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 842–866, 2020. [Online]. Available: https://www.aclweb.org/anthology/2020.tacl-1.54

[165] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of NLP models with CheckList," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 4902–4912. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.442

[166] N. Kassner and H. Schütze, "Negated and misprimed probes for pretrained language models: Birds can talk, but cannot fly," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 7811–7818. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.698

[167] J. Lee and Y. Wang, "Weighted rules under the stable model semantics." in *KR*, 2016, pp. 145–154.

[168] P. Clark, O. Tafjord, and K. Richardson, "Transformers as soft reasoners over language," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 7 2020, pp. 3882–3890. [Online]. Available: https://doi.org/10.24963/ijcai.2020/537

[169] Y.-A. Wang and Y.-N. Chen, "What do position embeddings learn? an empirical study of pre-trained language model positional encoding," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6840–6849. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-main.555

[170] M. Babakar and W. Moy, "The state of automated factchecking," *Full Fact*, 2016.

[171] N. Hassan, F. Arslan, C. Li, and M. Tremayne, "Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster," in *KDD*, 2017.

[172] I. Jaradat, P. Gencheva, A. Barrón-Cedeño, L. Màrquez, and P. Nakov, "ClaimRank: Detecting check-worthy claims in Arabic and English," in *NAACL-HTL*, 2018, pp. 26–30.

[173] B. Shi and T. Weninger, "Discriminative predicate path mining for fact checking in knowledge graphs," *Knowledge-Based Systems*, vol. 104, pp. 123–133, 2016.

[174] G. L. Ciampaglia, P. Shiralkar, L. M. Rocha, J. Bollen, F. Menczer, and A. Flammini, "Computational fact checking from knowledge networks," *PloS one*, vol. 10, no. 6, 2015.

[175] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *NIPS*, 2013, pp. 926–934.

[176] K. Popat, S. Mukherjee, A. Yates, and G. Weikum, "Declare: Debunking fake news and false claims using evidence-aware deep learning," in *EMNLP*, 2018, pp. 22–32.

[177] K. Popat, S. Mukherjee, J. Strötgen, and G. Weikum, "Credibility assessment of textual claims on the web," in *CIKM*, 2016.

[178] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," pp. 1070–1080, 1988.

[179] J. Lee, V. Lifschitz, and R. Palla, "A reductive semantics for counting and choice in answer set programming," in *AAAI*, 2008, pp. 472–479.

[180] P. Ferraris, J. Lee, and V. Lifschitz, "Stable models and circumscription," *Artificial Intelligence*, vol. 175, pp. 236–263, 2011.

[181] J. Lee, S. Talsania, and Y. Wang, "Computing LPMLN using ASP and MLN solvers," *Theory and Practice of Logic Programming*, 2017.

[182] P. Shiralkar, A. Flammini, F. Menczer, and G. L. Ciampaglia, "Finding streams in knowledge graphs to support fact checking," in *ICDM*, 2017, pp. 859–864.

[183] V. Huynh and P. Papotti, "Towards a benchmark for fact checking with knowledge bases," in *Companion of The Web Conference 2018 on The Web Conference (WWW)*, 2018.

[184] J. Lehmann, D. Gerber, M. Morsey, and A. N. Ngomo, "Defacto - deep fact validation," in *ISWC*, 2012, pp. 312–327.

[185] T. D. Cao, I. Manolescu, and X. Tannier, "Searching for truth in a database of statistics," in *WebDB*, 2018, pp. 4:1–4:6.

[186] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu, "Toward computational fact-checking," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 589–600, 2014.

[187] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[188] A. Srinivasan, "The Aleph manual," 2001.

[189] J. Leblay, "A declarative approach to data-driven fact checking," in *AAAI*, 2017, pp. 147–153.

[190] A. Pradhan, "Explainable fact checking by combining automated rule discovery with probabilistic answer set programming," Ph.D. dissertation, Arizona State University, 2018.

[191] J. Rammelaere and F. Geerts, "Explaining repaired data with CFDs," *PVLDB*, vol. 11, no. 11, pp. 1387–1399, 2018.

[192] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang, "Interactive and deterministic data cleaning," in *SIGMOD*, 2016, pp. 893–907.

[193] S. Sadiq, G. Governatori, and K. Namiri, "Modeling control objectives for business process compliance," in *International conference on business process management*. Springer, 2007, pp. 149–164.

[194] A. Silberschatz and A. Tuzhilin, "What makes patterns interesting in knowledge discovery systems," *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 6, pp. 970–974, 1996.

[195] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, "FEVER: a large-scale dataset for fact extraction and verification," in *NAACL-HLT*, 2018.

[196] A. Gatt and E. Krahmer, "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation," *Journal of Artificial Intelligence Research*, vol. 61, pp. 65–170, 2018.

[197] O. Tafjord, B. D. Mishra, and P. Clark, "Proofwriter: Generating implications, proofs, and abductive statements over natural language," 2020.

[198] V. Lifschitz, "What is answer set programming?" pp. 1594–1597, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1620270.1620340