INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAT025

Thèse de doctorat

TELECOM Paris

IP PARIS

Knowledge Base Curation using Constraints

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626
École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 7 septembre 2020, par

Thomas Pellissier Tanon

Composition du Jury :

Serge Abiteboul Directeur de recherche émérite, INRIA (DI ENS)	Examinateur
Antoine Amarilli Maître de conférence, Télécom Paris	Co-encadrant de thèse
Laure Berti-Equille Directeur de recherche, IRD	Présidente et Rapporteure
Markus Krötzsch Professeur, TU Dresden	Rapporteur
Nathalie Pernelle Professeur, Université Sorbonne Paris Nord	Examinatrice
Simon Razniewski Senior researcher, Max Planck Institute for Informatics	Examinateur
Fabian Suchanek Professeur, Télécom Paris	Directeur de thèse
Danai Symeonidou Chargé de recherche, INRA	Examinatrice

Knowledge Base Curation using Constraints

Thomas Pellissier Tanon

September 7th, 2020

Abstract

Knowledge bases are huge collections of primarily encyclopedic facts. They are widely used in entity recognition, structured search, question answering, and other tasks. These knowledge bases have to be curated, and this is a crucial but costly task. In this thesis, we are concerned with curating knowledge bases automatically using constraints.

Our first contribution aims at discovering constraints automatically. We improve standard rule mining approaches by using (in-)completeness meta-information. We show that this information can increase the quality of the learned rules significantly.

Our second contribution is the creation of a knowledge base, YAGO 4, where we statically enforce a set of constraints by removing the facts that do not comply with them.

Our last contribution is a method to correct constraint violations automatically. Our method uses the edit history of the knowledge base to see how users corrected violations in the past, in order to propose corrections for the present.

Remerciements

Tout d'abord, j'aimerais remercier mes directeurs de thèse Fabian Suchanek et Antoine Amarilli pour leur soutien durant ces trois années de thèse. Ce travail n'aurait pas été possible sans eux. Merci beaucoup pour m'avoir laissé choisir et conduire les directions dans lesquelles mener les travaux présentés dans cette thèse, tout en vous rendant très disponibles pour m'aider et me guider.

Merci tout spécialement à Fabian pour avoir relu et réécrit bon nombre de mes articles en supportant avec une patience admirable mes bien trop nombreuses fautes d'anglais.

Merci à mes rapporteurs, Laure Berti-Equille et Markus Krötzsch pour le temps passé à relire cette thèse. Merci à tous les membres de mon jury pour avoir accepté d'y participer.

Merci à toutes les personnes avec qui j'ai collaboré sur les travaux présentés dans cette thèse. Tout spécialement à Gerhard Weikum et Daria Stepanova qui m'ont accueilli pour un stage au Max Planck Institute for Informatics. Sans oublier Camille Bourgaux dont l'aide m'a été très précieuse. Mais aussi un grand merci à Simon Razniewski, Paramita Mirza Thomas Rebele et Lucie-Aimée Kaffee avec qui j'ai eu le plus grand plaisir à collaborer.

Merci aussi à Denny Vrandečić, Serge Abiteboul, Eddy Caron, Pierre Senellart et David Montoya pour m'avoir encadré avant cette thèse et permis de mûrir ce projet.

Merci aux membres de l'équipe DIG pour les bons moments passés ensemble. Entre autres, à Albert, Armand, Arnaud, Camille, Etienne, Favia, Jacob, Jean-Benoît, Jean-Louis, Jonathan, Julien, Julien, Louis, Marc, Marie, Maroua, Mauro, Mikaël, Miy-oung, Mostafa, Nathan, Ned, Oana, Pierre-Alexandre, Quentin, Thomas, Thomas et Ziad.

Merci enfin à ma famille pour tout le soutien qu'elle m'a donné durant ces trois années. Merci surtout à Mélissa pour son soutien sans faille malgré les trop nombreuses heures décalées passées à travailler.

Contents

1 Introduction **11**
1.1 Motivation 11
1.2 Presented Contributions 12
1.3 Other works 14

2 Preliminaries **17**
2.1 Knowledge bases 17
2.2 Queries 21
2.3 Rules and Rule Learning 22

I Mining Constraints **27**

3 Completeness-aware Rule Scoring **29**
3.1 Introduction 29
3.2 Related Work 31
3.3 Approach 33
3.4 Evaluation 38
3.5 Conclusion 43

4 Increasing the Number of Numerical Statements **45**
4.1 Introduction 45
4.2 Approach 46
4.3 Evaluation 47
4.4 Conclusion 49

II Enforcing Constraints Statically **51**

5 YAGO 4: A Reason-able Knowledge Base **53**
5.1 Introduction 53

5.2 Related Work . . . 55
5.3 Design . . . 57
5.3.1 Concise Taxonomy . . . 57
5.3.2 Legible Entities and Relations . . . 58
5.3.3 Well-Typed Values . . . 60
5.3.4 Semantic Constraints . . . 60
5.3.5 Annotations for Temporal Scope . . . 63
5.4 Knowledge Base . . . 64
5.4.1 Construction . . . 64
5.4.2 Data . . . 65
5.4.3 Access . . . 66
5.5 Conclusion . . . 68

III Enforcing Constraints Dynamically 71

6 Learning How to Correct a Knowledge Base from the Edit History 73
6.1 Introduction . . . 73
6.2 Related Work . . . 74
6.3 Constraints . . . 77
6.4 Corrections . . . 79
6.5 Extraction of the Relevant Past Corrections . . . 82
6.6 Correction Rule Mining . . . 85
6.7 Bass . . . 88
6.7.1 Bass-RL . . . 89
6.7.2 Bass . . . 93
6.8 Experiments on Wikidata . . . 95
6.8.1 Wikidata . . . 95
6.8.2 Dataset Construction . . . 96
6.8.3 Implementation of the Approaches . . . 99
6.8.4 Evaluation against the Test Set . . . 102
6.8.5 Bass Ablation Study . . . 105
6.8.6 User Evaluation . . . 107
6.9 Conclusion . . . 108

7 Querying the Edit History of Wikidata 111
7.1 Introduction . . . 111
7.2 Related Work . . . 112
7.3 System Overview . . . 113
7.4 Usage . . . 115

7.5 Conclusion 116

8 Conclusion **117**
8.1 Summary 117
8.2 Outlook 118

Bibliography **119**

A Résumé en français **131**
A.1 Introduction 131
A.1.1 Motivation 131
A.1.2 Contenu 132
A.1.3 Autres travaux 134
A.2 Préliminaires 136
A.3 Apprentissage de règles à l'aide de cardinalités 138
A.4 Apprentissage de corrections d'une base de connaissances à partir de son historique de modifications 141
A.5 Conclusion 145
A.5.1 Résumé 145
A.5.2 Perspective 146

Chapter 1

Introduction

1.1 Motivation

Knowledge bases (KBs) are sets of machine-readable facts. They contain *entities* and *relations* about them. For example an encyclopedic knowledge base could contain entities about people like Jean-François Champollion or places like Paris, and named relations like *livedIn*(`Jean-François_Champollion`, `Paris`) to state that Jean-François Champollion lived in Paris. Well-known knowledge bases in this area include Wikidata (Vrandecic and Krötzsch, 2014), YAGO (Suchanek, Kasneci, and Weikum, 2007), Freebase (Bollacker et al., 2008), DBpedia (Bizer et al., 2009) or the Google Knowledge Graph. Such knowledge bases are used to display fact sheets like in Wikipedia or in the Google and Bing search engines. They are also used to directly answers user questions like in Alexa or Siri. But knowledge bases are also used for other kinds of contents and use cases. For example, Amazon and eBay maintain knowledge bases of the products they sell and Uber maintains a food knowledge base to help its customers choose a restaurant.

Some knowledge bases are very large. For example, Wikidata contains 81M entities[1] and Freebase 39M. Some of these projects use software pipelines to build the knowledge base automatically from one or multiple existing sources. For example, DBpedia is built by a set of extractors from Wikipedia content. Others are using a crowd of contributors, paid or volunteer, to fill the knowledge base. This is the case of Wikidata, which has more than 40k different editors per month.

As a consequence, knowledge bases often exhibit quality problems, originating from edge cases in the conversion pipelines, good-faith mistakes, or vandalism in the crowd-sourced content. For example Zaveri, Kontokostas, et al., 2013 found that 12% of the DBpedia triples have some kind of issue. Even crowd-sourced knowledge bases often rely significantly on automated importation or curation

[1] As of April 10th, 2020.

pipelines. 43% of Wikidata edits in March 2020 have been done using automated tools.

To fight such problems, many knowledge bases contain a constraint system. One of the most basic kinds of constraint is stating that the values of a given property should have a given type. For example, a knowledge base could enforce that the values of the *birthPlace* property should be strings representing a date. Constraints can also be more complex stating e.g., that a person can have at most 2 parents (cardinality restrictions) or that an entity cannot be a person and a place at the same time (disjunctions). Some of these constraints are enforced by the knowledge base. For example, the OWL formalism itself requires that properties whose values are literals (string, dates...) have to be disjoint from the ones whose values are entities. However, these constraints are often violated in practice. For example, as of March 20th, 2020, Wikidata has 1M "domain" constraint violations and 4.4M "single value" constraint violations. Thus, there is a need for tools to filter out such problems from the knowledge base. There is also a need for tools that help the knowledge base curators repair these violations in an automated or semi-automated way.

This thesis provides some new approaches and techniques to improve the state on the art on this complex task.

1.2 Presented Contributions

The first chapter of this thesis, Chapter 2, presents general preliminaries on knowledge bases and rule mining. The main work is composed of 3 parts.

Part I, presents improvements to the rule mining problem. More precisely, Chapter 3 presents a novel approach that improves rules mining over incomplete knowledge bases by making use of cardinality information. This allows mining rules of higher quality. These can then be used in order to complete the knowledge base or as constraints to flag problems in the data. Our approach works by introducing a new rule quality estimation measure, the *completeness confidence*. The completeness confidence takes into account the number of expected objects for given subjects and predicates to better assess the quality of the rules. We show that this measure does not require cardinality information on all the knowledge base entities to be effective. We evaluated this completeness confidence both on real-world and synthetic datasets, showing that it outperforms existing measures both with respect to the quality of the mined rules and the predictions they produce.

Chapter 4 presents an approach to increase the number of available cardinality metadata, especially to improve the quality of the rules mined using the completeness confidence.

The content of these two chapters was published at ISWC 2017 where it was nominated for the best student paper award. An invited short version of the paper has been presented at IJCAI 2018:

> Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. "Completeness-Aware Rule Learning from Knowledge Graphs". Full paper at ISWC 2017. https://doi.org/10.1007/978-3-319-68288-4_30

> Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. "Completeness-aware Rule Learning from Knowledge Graphs". Invited paper at IJCAI 2018. https://doi.org/10.24963/ijcai.2018/749

Part II of this thesis presents an approach of static constraint enforcement. More precisely, it presents YAGO 4, a knowledge base is basically a simpler and cleaner version of Wikidata. We built this knowledge base using a declarative mapping and constraint enforcement pipeline. YAGO 4 can be seen as an example of a knowledge base that ensures quality by filtering out constraint violations. This work has been published as a resource paper at ESWC 2020:

> Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. "YAGO 4: A Reason-able Knowledge Base". Resource paper at ESWC 2020. https://doi.org/10.1007/978-3-030-49461-2_34

Part III presents contributions to the task of dynamically learning how to enforce constraints. Chapter 6 discusses a novel problem: learning how to fix constraint violations using the edit history of a knowledge base. We present a formalism of the problem and an algorithm to extract "past corrections" of constraint violations from the knowledge base history. To solve this problem, the chapter suggests two different approaches: one based on rule mining and the other one using neural networks, the latter providing better accuracy but no explanation of its predictions. We validated both approaches experimentally on Wikidata, showing substantial improvements over baselines. This work has been presented at TheWebConf 2019. The neural network approach is currently under review:

> Thomas Pellissier Tanon, Camille Bourgaux, and Fabian M. Suchanek. "Learning How to Correct a Knowledge Base from the Edit History". Full paper at WWW 2019. https://doi.org/10.1145/3308558.3313584

Thomas Pellissier Tanon and Fabian M. Suchanek. "Neural Knowledge Base Repairs". Under review at ISWC 2020.

Chapter 7 is an annex of the previous chapter. It presents a system we implemented to query the edit history of Wikidata efficiently. It was used to extract the data used to evaluate the approaches presented in the previous chapter. This work has been demoed at ESWC 2019:

Thomas Pellissier Tanon and Fabian M. Suchanek. "Querying the Edit History of Wikidata". Demo at ESWC 2019. `https://doi.org/10.1007/978-3-030-32327-1_32`

1.3 Other works

During my Ph.D., I contributed to some other works that are not presented in this thesis.

I presented my master thesis at the Linked Data on The Web workshop. This work was done with David Montoya under the supervision of Serge Abiteboul, Pierre Senellart, and Fabian M. Suchanek. The master thesis was about building a knowledge integration platform for personal information. This system is able to synchronize in both directions from datasources such as emails, calendars, and contact books. It also aligns and enriches the data. This work was also demoed at CIKM:

David Montoya, Thomas Pellissier Tanon, Serge Abiteboul, Pierre Senellart, and Fabian M. Suchanek. "A Knowledge Base for Personal Information Management". Paper at the LDOW workshop collocated with WWW 2018. `http://ceur-ws.org/Vol-2073/article-02.pdf`

David Montoya, Thomas Pellissier Tanon, Serge Abiteboul, and Fabian M.Suchanek. "Thymeflow, A Personal Knowledge Base with Spatio-temporal Data". Demo at CIKM 2016. `https://doi.org/10.1145/2983323.2983337`

I also published an earlier work on using grammatical dependencies for question answering over knowledge bases. This work was demoed at ESWC and the dataset to train such systems has been presented as a poster at ISWC:

Thomas Pellissier Tanon, Marcos Dias de Assunção, Eddy Caron, and Fabian M. Suchanek. "Demoing Platypus - A Multilingual Question Answering Platform for Wikidata". Demo at ESWC 2018. https://doi.org/10.1007/978-3-319-98192-5_21

Dennis Diefenbach, Thomas Pellissier Tanon, Kamal Deep Singh, and Pierre Maret. "Question Answering Benchmarks for Wikidata". Poster at ISWC 2017. http://ceur-ws.org/Vol-1963/paper555.pdf

Furthermore, together with Lucie-Aimée Kaffee, we conducted a study of the stability of the Wikidata schema. We analyzed the stability of the data based on the changes in the labels of properties in six languages. We found that the schema is overall stable, making it a reliable resource for external usage. This work was presented at the WikiWorshop:

Thomas Pellissier Tanon and Lucie-Aimée Kaffee. "Property Label Stability in Wikidata: Evolution and Convergence of Schemas in Collaborative Knowledge Bases". Short paper at the WikiWorkshop collocated with WWW 2018. https://doi.org/10.1145/3184558.3191643

I also helped a Ph.D. student in the team, Thomas Rebele, on his work on evaluating Datalog using Bash shell commands. I formalized the problem and provided a converter between Datalog and relational algebra. Our method allows preprocessing large tabular data in Datalog — without indexing the data. The Datalog query is translated to Unix Bash and can be executed in a shell. Our experiments have shown that, for the use case of data preprocessing, our approach is competitive with state-of-the-art systems in terms of scalability and speed, while at the same time requiring only a Bash shell on a Unix system. This work has been published at ISWC:

Thomas Rebele, Thomas Pellissier Tanon, and Fabian M. Suchanek. "Bash Datalog: Answering Datalog Queries with Unix Shell Commands". Full paper at ISWC 2018. https://doi.org/10.1007/978-3-030-00671-6_33

Chapter 2

Preliminaries

2.1 Knowledge bases

A knowledge base (KB) is an interlinked collection of factual information. A knowledge base can be represented as a graph. Nodes represent entities (like the human Elvis Presley, or the notion of a human) and directed labeled edges represent facts about these entities. For example, an edge labeled "type" from the node representing Elvis Presley to the node representing the notion of a human encodes that Elvis Presley is a human. See Figure 2.1 for a graphical example of a knowledge base. In this work, we use description logics (DL) (Baader et al., 2003) as a knowledge base language, and more specifically the ones that fall under the OWL 2 DL formalism (Motik, Patel-Schneider, and Grau, 2012).

Syntax

We assume a set $\mathsf{N_C}$ of *concept names* (unary predicates, also called classes), a set $\mathsf{N_R}$ of *role names* (binary predicates, also called properties), and a set $\mathsf{N_I}$ of *individuals* (also called constants).

Definition 2.1 (ABox)**.** An *ABox* (dataset) is a set of *concept assertions* and *role assertions* which are respectively of the form $A(a)$ or $R(a, b)$, where $A \in \mathsf{N_C}$, $R \in \mathsf{N_R}$, and $a, b \in \mathsf{N_I}$.

Definition 2.2 (TBox)**.** A *TBox* (ontology) is a set of axioms that expresses relationships between concepts and roles. (e.g., concept or role hierarchies, role domains and ranges...). Their form depends on the description logic $\mathcal{L}$ in question.

We only consider here description logics $\mathcal{L}$ that fall under the OWL 2 DL formalism (Motik, Patel-Schneider, and Grau, 2012).

Some commonly found building blocks in description logics formulas are the following:[1]

- $\top$ is the most general concept containing all individuals.
- $\bot$ is the empty concept.
- $\{a_1, \ldots, a_n\}$ is the concept containing only the individuals a_1, $\ldots$, a_n.
- $\neg A$ is the complement of the concept A (i.e., all individuals not in A).
- $A \sqcup B$ is the concept that contains all individuals in A or B.
- $A \sqcap B$ is the concept that contains all individuals that are in both A and B.
- $\exists R \cdot B$ is the concept containing all a such that there exists b with $R(a, b)$ and $B(b)$.
- R^- is the inverse role of R i.e., informally, $R(a, b)$ if, and only if, $R^-(b, a)$.

This notation can be used to create axioms like the following:

- $A \sqsubseteq B$ states that all individuals in the concept A are also in the concept B.
- $A \equiv B$ states that A and B are equivalent i.e., $A \sqsubseteq B$ and $B \sqsubseteq A$.
- $P \sqsubseteq R$ states that all facts of a role P are also in a role R.
- $P \equiv R$ states that P and R are equivalent i.e., $P \sqsubseteq R$ and $R \sqsubseteq P$.
- ($\texttt{func}\, R$) states that R if functional i.e., informally, for all a, b_1 and b_2 if $R(a, b_1)$ and $R(a, b_2)$ then $b_1 = b_2$.
- ($\texttt{trans}\, R$) states that R if transitive i.e., informally, for all a, b and c if $R(a, b)$ and $R(b, c)$ then $R(a, c)$.

Definition 2.3 (Knowledge base)**.** A *knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is a pair of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

The knowledge base can also be written as a set of *RDF triples* (Cyganiak et al., 2014) $\langle s, p, o \rangle$ where s is the *subject*, p is the *property*, and o the *object*. A concept assertion $A(a)$ is written as $\langle a, \texttt{rdf:type}, A \rangle$, and a role assertion $R(a, b)$ as $\langle a, R, b \rangle$. The TBox can also be written with RDF triples using the mapping defined in Motik and Patel-Schneider, 2012.

[1] For brevity we do not recall all OWL 2 DL formulas and axiom elements. They are all described in Motik, Patel-Schneider, and Parsia, 2012.

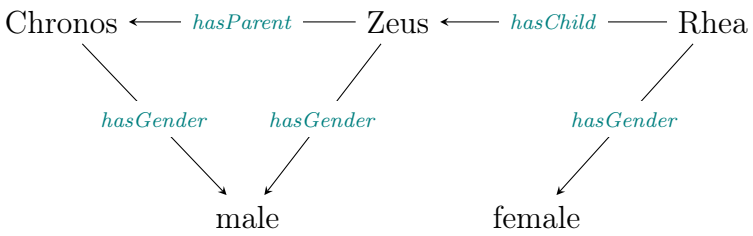

Figure 2.1 – Example knowledge base

Example 2.4. Here is an example of knowledge base about Greek gods:

$$\begin{aligned}\mathcal{T} = \{\ & \exists \mathit{hasParent} \sqsubseteq \mathit{Human}, \exists \mathit{hasChild} \sqsubseteq \mathit{Human}, \exists \mathit{hasGender} \sqsubseteq \mathit{Human}, \\ & \mathit{hasParent}^- \equiv \mathit{hasChild}, \exists \mathit{hasGender}^- \sqsubseteq \{\texttt{male}, \texttt{female}\}\ \} \\ \mathcal{A} = \{\ & \mathit{hasGender}(\texttt{Zeus}, \texttt{male}), \mathit{hasGender}(\texttt{Cronos}, \texttt{male}), \\ & \mathit{hasGender}(\texttt{Rhea}, \texttt{female}), \mathit{hasParent}(\texttt{Zeus}, \texttt{Chronos}), \\ & \mathit{hasChild}(\texttt{Rhea}, \texttt{Zeus}),\ \}\end{aligned}$$

The TBox states that if you have a parent, a child, or a gender then you are a human, that *hasChild* is the inverse property of *hasParent*, and that the two possible values for the *hasGender* relation are `male` and `female`. The ABox is represented graphically in Figure 2.1.

The ABox and the TBox entail the following facts: $\mathit{Human}(\texttt{Zeus})$, $\mathit{Human}(\texttt{Chronos})$, $\mathit{Human}(\texttt{Rhea})$, $\mathit{hasParent}(\texttt{Zeus}, \texttt{Rhea})$ and $\mathit{hasChild}(\texttt{Chronos}, \texttt{Zeus})$.

Semantics

We recall the standard semantics of description logic knowledge bases (Baader et al., 2003).

Definition 2.5 (Interpretation)**.** An *interpretation* has the form $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set and $\cdot^\mathcal{I}$ is a function that maps each $a \in \mathsf{N_I}$ to $a^\mathcal{I} \in \Delta^\mathcal{I}$, each $A \in \mathsf{N_C}$ to $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and each $R \in \mathsf{N_R}$ to $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$.

The function $\cdot^\mathcal{I}$ is straightforwardly extended to general concepts and roles. For example:[2]

- $\top^\mathcal{I} = \Delta^\mathcal{I}$
- $\bot^\mathcal{I} = \emptyset$

[2]See Motik, Patel-Schneider, and Grau, 2012 for the full semantics of all OWL 2 DL formulas.

- $\{a_1, \ldots, a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$
- $(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
- $(A \sqcup B)^{\mathcal{I}} = A^{\mathcal{I}} \cup B^{\mathcal{I}}$
- $(A \sqcap B)^{\mathcal{I}} = A^{\mathcal{I}} \cap B^{\mathcal{I}}$
- $(\exists R \cdot B)^{\mathcal{I}} = \{a \mid \exists b\, (a, b) \in R^{\mathcal{I}}, b \in B^{\mathcal{I}}\}$
- $(R^-)^{\mathcal{I}} = \{(b, a) \mid (a, b) \in R^{\mathcal{I}}\}$

An interpretation $\mathcal{I}$ *satisfies*:[3]

- $A(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$
- $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
- $A \sqsubseteq B$, if $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$
- $P \sqsubseteq R$, if $P^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
- $(\texttt{func}\, R)$ if $\forall a, b_1, b_2 \in \Delta^{\mathcal{I}}\ (a, b_1) \in R^{\mathcal{I}} \wedge (a, b_2) \in R^{\mathcal{I}} \implies b_1 = b_2$
- $(\texttt{trans}\, R)$ if $\forall a, b, c \in \Delta^{\mathcal{I}}\ (a, b) \in R^{\mathcal{I}} \wedge (b, c) \in R^{\mathcal{I}} \implies (a, c) \in R^{\mathcal{I}}$

We write $\mathcal{I} \models_{\mathcal{L}} \alpha$ if $\mathcal{I}$ satisfies the description logic axiom α according to the description logic used $\mathcal{L}$.

Definition 2.6 (Model)**.** An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ if $\mathcal{I}$ satisfies all axioms in $\mathcal{T}$ and all assertions in $\mathcal{A}$ according to $\mathcal{L}$.

Definition 2.7 (Consistency)**.** A knowledge base is *consistent* if it has a model.

Definition 2.8 (Entailment)**.** A knowledge base $\mathcal{K}$ *entails* a description logic axiom α in $\mathcal{L}$ if $\mathcal{I} \models_{\mathcal{L}} \alpha$ for every model $\mathcal{I}$ of $\mathcal{K}$. We write $\mathcal{K} \models_{\mathcal{L}} \alpha$ in this case.

Definition 2.9 (Unique name assumption)**.** The *unique name assumption* restricts the set of possible interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{K}$ to only the ones where $\cdot^{\mathcal{I}}$ is injective on individuals. It means that each individual $a \in \mathsf{N_I}$ has a different interpretation i.e., each real world entity is represented by at most one individual.

[3] See Motik, Patel-Schneider, and Grau, 2012 for the semantic of all OWL 2 DL axioms.

Ideal knowledge base

Following Darari, Nutt, et al., 2013, we define the the *ideal knowledge base* $\mathcal{K}^i$, which contains all correct facts over $\mathsf{N_C}$, $\mathsf{N_R}$ and $\mathsf{N_I}$ that hold in the real world.

Of course, $\mathcal{K}^i$ is an imaginary construct whose content is generally not known. We will present a method that can deduce instead to which extent the available knowledge base approximates/lacks information compared to the ideal knowledge base, as in "Einstein is missing 2 children and Feynman none".

2.2 Queries

The description logic formalism explained in the previous section allows us to do entailments on the knowledge base. However, answering queries like "who are the people who died where they were born" is not possible with the description logic formalism we described. This is why we now introduce some variations of the concept of *queries*.

Conjunctive and disjunctive queries

Definition 2.10 (Conjunctive query)**.** A *conjunctive query* takes the form $q(\vec{x}) = \exists \vec{y}\ \psi(\vec{x}, \vec{y})$, where ψ is a conjunction of atoms of the form $A(t)$ or $R(t, t')$ or of equalities $t = t'$, where t, t' are individual names from $\mathsf{N_I}$ or variables from $\vec{x} \cup \vec{y}$, $A \in \mathsf{N_C}$, and $R \in \mathsf{N_R}$.

Example 2.11. The query $q(x) = \mathit{Person}(x) \wedge \mathit{birthPlace}(x, \texttt{Paris})$ returns all the people born in Paris.

The query $q'(x) = \exists y\, \mathit{Person}(x) \wedge \mathit{birthPlace}(x, y) \wedge \mathit{deathPlace}(x, y)$ returns all the people who died where they were born.

If $\vec{x} = \emptyset$, q is a boolean conjunctive query. A boolean conjunctive query q is *satisfied* by an interpretation $\mathcal{I}$, written $\mathcal{I} \models q$, if there is a homomorphism π mapping the variables and individual names of q into $\Delta^{\mathcal{I}}$ such that: $\pi(a) = a^{\mathcal{I}}$ for every $a \in \mathsf{N_I}$, $\pi(t) \in A^{\mathcal{I}}$ for every concept atom $A(t)$ in ψ, $(\pi(t), \pi(t')) \in R^{\mathcal{I}}$ for every role atom $R(t, t')$ in ψ, and $\pi(t) = \pi(t')$ for every $t = t'$ in ψ. We also consider as boolean conjunctive queries the queries `true` and `false` which are respectively always and never satisfied by an interpretation. A boolean conjunctive query q is *entailed* by a knowledge base $\mathcal{K}$, written $\mathcal{K} \models q$, if, and only if, q is satisfied by every model of $\mathcal{K}$.

A tuple of constants $\vec{a}$ is a (certain) *answer* to a conjunctive query $q(\vec{x})$ on a knowledge base $\mathcal{K}$ if $\mathcal{K} \models q(\vec{a})$ where $q(\vec{a})$ is the boolean conjunctive query obtained by replacing the variables from $\vec{x}$ by the constants $\vec{a}$.

Definition 2.12 (Union of conjunctive queries)**.** A *union of conjunctive queries* is a disjunction of conjunctive queries and has as answers the union of the answers of the conjunctive queries it contains.

Propositional queries

Definition 2.13 (Propositional query)**.** A *propositional query* takes the form $q(\vec{x}) = \exists \vec{y}\ \psi(\vec{x}, \vec{y})$ where ψ is a propositional formula. Propositional formulas are defined inductively as follows:

- $A(t)$, $R(t, t')$, $t = t'$, `true` and `false` are propositional formulas where t, t' are individual names from $\mathsf{N_I}$ or variables from $\vec{x} \cup \vec{y}$, $A \in \mathsf{N_C}$ and $R \in \mathsf{N_R}$.
- $x \wedge y$, $x \vee y$ and $\neg x$ are propositional formulas if x and y are propositional formulas.

Example 2.14. The propositional query $q(x) = \mathit{Person}(x) \wedge \neg \mathit{Dead}(x)$ answers are all people who are not dead.

Let $\vec{a}$ be a vector of constants. We write $\mathcal{I} \models q(\vec{a})$ where $q(\vec{x}) = \exists \vec{y}\ \psi(\vec{x}, \vec{y})$ if there exists a homomorphism π mapping the variables and individual names of q into $\Delta^{\mathcal{I}}$ such that $\pi(a) = a^{\mathcal{I}}$ for every $a \in \mathsf{N_I}$ and if we do the following replacements then the resulting propositional formula can be evaluated to true: (1) Replace $A(t)$ by the valuation of $\pi(t) \in A^{\mathcal{I}}$ for every concept atom $A(t)$ in ψ. (2) Replace $R(t, t')$ by the valuation of $(\pi(t), \pi(t')) \in R^{\mathcal{I}}$ for every role atom $R(t, t')$ in ψ. (3) Replace $t = t'$ by the valuation of $\pi(t) = \pi(t')$ for every $t = t'$ in ψ.

2.3 Rules and Rule Learning

The previous sections have presented how to answer queries and derive new facts from the existing ones based on axioms written in description logics. However, these two methods require queries and axioms as input. In this section, we instead aim at automatically inferring new rules and facts from the ones already present in the knowledge base.

Association rule learning concerns the discovery of frequent patterns in a data set and the subsequent transformation of these patterns into rules. This task has been popularized by Agrawal, Imielinski, and Swami, 1993. Association rules in the relational format have been subject to intensive research in inductive logic programming (see, e.g., Dehaspe and De Raedt, 1997 as the seminal work in this direction) and more recently in the knowledge base community (see Galárraga, Teflioudi, et al., 2015 for a prominent work). In the following, we adapt basic notions in relational association rule mining to our case of interest.

Definition 2.15 (Rule)**.** A *rule* is of the form $r(\vec{x}) = b(\vec{x}) \to h(\vec{x})$, where b and h are both conjunctive queries.

We say that a rule $r(\vec{x}) = b(\vec{x}) \to h(\vec{x})$ is *satisfied* by an interpretation $\mathcal{I}$ if for any vector $\vec{a}$ of constants, we have $\mathcal{I} \models b(\vec{a}) \implies \mathcal{I} \models h(\vec{a})$.

We say that a knowledge base $\mathcal{K}$ *entails* a rule $r(\vec{x}) = b(\vec{x}) \to h(\vec{x})$ if for any vector $\vec{a}$ of constants, we have $\mathcal{K} \models b(\vec{a}) \implies \mathcal{K} \models h(\vec{a})$.

Classical scoring of association rules is based on *rule support*, *body support* and *confidence*, which are defined as follows:

Definition 2.16 (Query support)**.** As in Dehaspe and De Raedt, 1997, the *support* of a conjunctive query q in a knowledge base $\mathcal{K}$ is the number of distinct answers of q on $\mathcal{K}$:

$$supp(q) := |\{\vec{x} \mid \mathcal{K} \models q(\vec{x})\}|$$

The rule support metric has been introduced in order to see if a rule applies in a lot of cases, in order to mine general rules rather than rules applying in only a few cases:

Definition 2.17 (Rule support)**.** The *support* of a rule $r(\vec{x}) : b(\vec{x}) \to h(\vec{x})$ is the number of times a rule applies on $\mathcal{K}$:

$$supp(r) := supp(b \wedge h) = |\{\vec{x} \mid \mathcal{K} \models b(\vec{x}) \wedge h(\vec{x})\}|$$

The confidence is an often used metric to assess how much a rule applies on a given knowledge base:

Definition 2.18 (Standard confidence)**.** The *standard confidence* of a rule $r(\vec{x}) : b(\vec{x}) \to h(\vec{x})$ is the proportion of the number of times a rule actually applies with respect to the number of times it can apply:

$$conf(r) := \frac{supp(r)}{supp(b)}$$

Note that we always have $conf(r) \in [0, 1]$.

Example 2.19. Consider the knowledge base shown in Figure 2.2 with an empty TBox. Consider the rules r_1 and r_2:

$$r_1(x, y) : \exists z\ worksAt(x, z) \wedge educatedAt(y, z) \to hasChild(x, y)$$

$$r_2(x, z) : \exists y\ hasFather(x, y) \wedge hasChild(y, z) \to hasSibling(x, z)$$

The first rule states that workers of certain institutions often have children among the people educated there. The second rule states that if your father has children, then they are your siblings.

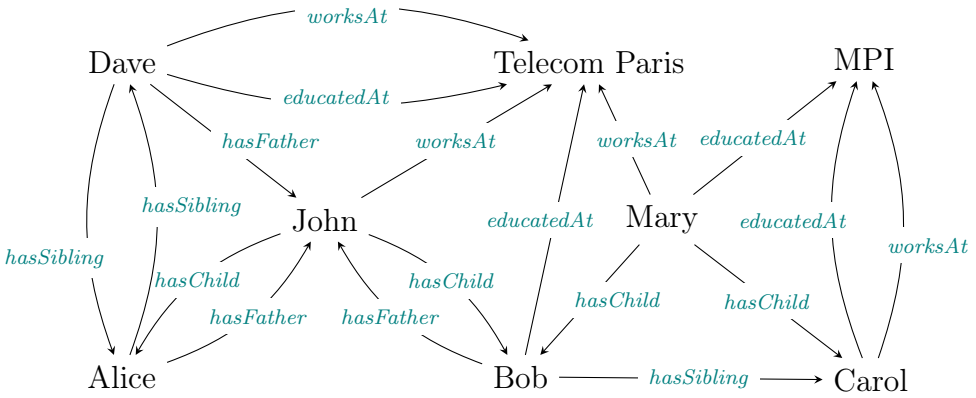

Figure 2.2 – Example knowledge base

The body of r_1 seen as a query, $b_1(x, y) = \exists z\, worksAt(x, z) \wedge educatedAt(y, z)$, has 8 possible answers in $\mathcal{K}$: (`Dave`, `Dave`), (`Dave`, `Bob`), (`John`, `Dave`)... Hence, the body support of the rule r_1 is $supp(b_1(x, y)) = 8$. The only two possible answers of the query $b_1(x, y) \wedge hasChild(x, y)$ are (`John`, `Bob`) and (`Mary`, `Bob`). Hence $supp(r_1) = 2$. So, have $conf(r_1) = \frac{2}{8}$.

Analogously, for r_2, $b_2(x, z) = \exists y\, hasFather(x, y) \wedge hasChild(y, z)$ has 6 answers and therefore $supp(b_2(x, z)) = 6$. However, only one of these answers is such that $hasSibling(x, z)$ holds, so $supp(r_2) = 1$. This leads to $conf(r_2) = \frac{1}{6}$. □

Support and confidence were originally developed for scoring rules over complete data. If data is missing, their interpretation is not straightforward and they can be misleading. For example, in Example 2.19, the rule r_1 has a confidence of $\frac{1}{4}$ and r_2 has a lower confidence of $\frac{1}{6}$, although r_1 is clearly wrong.

The reason is that the standard confidence aims first at avoiding wrong assertions in the predictions. It does so by considering as wrong all predicted assertions that are not already in the knowledge base, even if these assertions might be true in the real world. Other approaches aim at finding other compromises between predicting wrong facts and missing true facts. In Galárraga, Teflioudi, et al., 2015, the *confidence under the Partial Completeness Assumption* (PCA) has been proposed as a measure. This measure guesses negative facts by assuming that data is usually added to knowledge bases in batches. For example, if at least one child of John is known then most probably all of John's children are present in the knowledge base.

Definition 2.20 (PCA confidence)**.** The *PCA confidence* is defined for all rules of the form $r(x, y) : b(x, y) \to h(x, y)$ where x and y are variables and h a role assertion by:

$$conf_{pca}(r) := \frac{supp(r)}{supp_{pca}(r)}$$

where

$$supp_{pca}(r) := |\{(x, y) \mid \mathcal{K} \models b(x, y) \text{ and } \exists y' \in \mathsf{N_I}\, \mathcal{K} \models h(x, y')\}|$$

Example 2.21. We obtain $conf_{pca}(r_1) = \frac{2}{4}$. Indeed, since `Carol` and `Dave` are not known to have any children in the knowledge base, four existing body substitutions are not counted in the denominator. Meanwhile, we have $conf_{pca}(r_2) = \frac{1}{6}$, since all people that are predicted to have siblings by r_2 already have siblings in the available knowledge base. □

Knowledge base completion

After having mined rules, we want to use them to complete the knowledge base. That is, we want to add to the knowledge base all the assertions that are generated by the rules. Formally:

Definition 2.22 (Rule-based knowledge base completion)**.** Let $\mathcal{K}$ be a knowledge base and let r be a rule such that $r(\vec{x}) = b(\vec{x}) \rightarrow h_1(\vec{x}) \wedge \cdots \wedge h_n(\vec{x})$ where the h_i are concept or role assertions. Then the *completion* of $\mathcal{K} = (\mathcal{A}, \mathcal{T})$ by r is a knowledge base $\mathcal{K}_r = (\mathcal{A}_r, \mathcal{T})$ such that $\mathcal{A}_r = \mathcal{A} \cup \{h_1(\vec{a}), \ldots, h_n(\vec{a}) \mid \mathcal{K} \models b(\vec{a})\}$.

Example 2.23. If we follow Example 2.19, we have $\mathcal{A}^a_{r_1} = \mathcal{A} \cup \{$ *hasChild*(`John`, `Dave`), *hasChild*(`Carol`, `Mary`), *hasChild*(`Dave`, `Dave`), *hasChild*(`Carol`, `Carol`), *hasChild*(`Dave`, `Bob`), *hasChild*(`Mary`, `Dave`) $\}$. □

Note that the ideal knowledge base $\mathcal{K}^i$ introduced in Section 2.1 is the perfect completion of $\mathcal{K}^a$, i.e., it is supposed to contain all correct facts with entities and relations from $\Sigma_{\mathcal{K}^a}$ that hold in the current state of the world. The goal of rule-based knowledge base completion is to extract from $\mathcal{K}^a$ a set of rules $\mathcal{R}$ such that $\cup_{r \in \mathcal{R}} \mathcal{K}^a_r$ is as close to $\mathcal{K}^i$ as possible.

Rule evaluation

To evaluate the predictions of a rule $r(\vec{x}) : b(\vec{x}) \rightarrow h(\vec{x})$ against a set of expected predictions $\mathcal{P}$, we can use the following metrics:

Definition 2.24 (Precision)**.** The *precision* of a rule is the fraction of the predictions computed by the rule that are in the set of expected predictions $\mathcal{P}$. This measure aims at computing how many of the rule predictions are expected predictions. Formally:

$$precision(r) := \frac{|\{\vec{x} \mid \mathcal{K} \models b(\vec{x}) \wedge \mathcal{P} \models h(\vec{x})\}|}{|\{\vec{x} \mid \mathcal{K} \models b(\vec{x})\}|}$$

Definition 2.25 (Recall)**.** The *recall* is given by the fraction of the predictions computed by the rule r that are in the set of expected predictions $\mathcal{P}$. This measure aims at computing how many of the expected predictions are found by the rules. Formally:

$$recall(r) := \frac{|\{\vec{x} \mid \mathcal{K} \models b(\vec{x}) \wedge \mathcal{P} \models h(\vec{x})\}|}{|\{\vec{x} \mid \mathcal{P} \models h(\vec{x})\}|}$$

Definition 2.26 (F1 score)**.** The *F1 score* is the harmonic mean of precision and recall:

$$F_1(r) := 2\frac{precision(r) \cdot recall(r)}{precision(r) + recall(r)}$$

Example 2.27. Consider the same knowledge base and rules as in Example 2.19.

We evaluate the rule r_1 against the expected prediction set:

$$\begin{aligned}\mathcal{P} = \{\ & hasChild(\texttt{John}, \texttt{Dave}), hasChild(\texttt{John}, \texttt{Alice}), hasChild(\texttt{John}, \texttt{Bob}),\\ & hasChild(\texttt{Mary}, \texttt{Bob}), hasChild(\texttt{Mary}, \texttt{Carole}), hasChild(\texttt{Mary}, \texttt{Dave})\ \}\end{aligned}$$

Like presented in Example 2.19, r_1 predicts 8 facts. In these facts 4 are in $\mathcal{P}$: *hasChild*(`John`, `Dave`), *hasChild*(`John`, `Bob`), *hasChild*(`Mary`, `Bob`), and *hasChild*(`Mary`, `Dave`).

Hence we have $precision(r_1) = \frac{4}{8} = 0.5$ and $recall(r_1) = \frac{4}{6} = 0.66$. This gives us $F_1(r_1) = 2\frac{0.5 \cdot 0.66}{0.5 + 0.66} = 0.57$. □

Now that we have the preliminaries in place, we will move to the first contribution of this thesis. This contribution aims at refining the confidence measure when cardinality information is available in the knowledge base.

Part I

Mining Constraints

Chapter 3

Completeness-aware Rule Scoring

The work presented in this chapter has been done with Daria Stepanova, Simon Razniewski, Paramita Mirza and Gerhard Weikum and has been published at ISWC 2017. It was one of the papers nominated for the best student paper award. An invited short version of the paper has been published at IJCAI 2018.[1]

> Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. "Completeness-Aware Rule Learning from Knowledge Graphs". Full paper at ISWC 2017. `https://doi.org/10.1007/978-3-319-68288-4_30`

> Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. "Completeness-aware Rule Learning from Knowledge Graphs". Invited paper at IJCAI 2018. `https://doi.org/10.24963/ijcai.2018/749`

3.1 Introduction

Motivation

An important task over knowledge bases is *rule learning*. This task aims at learning new *rules* on the knowledge base. For example, the rule $r_1 : worksAt(x, z) \wedge educatedAt(y, z) \rightarrow hasChild(x, y)$ could be mined from the knowledge base displayed in Figure 3.1 page 36. This rule states that workers of certain institutions often have children among the people educated there, as this is frequently the

[1] Some rule quality measures presented in the ISWC paper are omitted here to focus on my personal contribution.

case for popular scientists. Rules are relevant for a variety of applications ranging from knowledge base curation, e.g., completion and error detection (Paulheim, 2017; Galárraga, Teflioudi, et al., 2015; Gad-Elrab et al., 2016), to data mining and semantic culturonomics (Suchanek and Preda, 2014). Section 2.3 Page 22 has formally defined the concept of rules and presented some existing methods to evaluate them.

However, since such rules are learned from incomplete data, they might be erroneous and might make incorrect predictions on missing facts. The already presented rule r_1 is clearly not universal and should be ranked lower than the rule $r_2 : \mathit{hasFather}(x, y) \wedge \mathit{hasChild}(y, z) \rightarrow \mathit{hasSibling}(x, z)$. However, standard rule measures like confidence (i.e., the conditional probability of the rule's head given its body) incorrectly favor r_1 over r_2 for the given knowledge base.

Recently, efforts have been put into detecting the concrete number of facts of certain types that hold in the real world (e.g., *"Einstein has 3 children"*) by exploiting Web extraction and crowd-sourcing methods (Mirza, Razniewski, and Nutt, 2016; Prasojo et al., 2016). Such cardinality information provides a lot of hints about the topology of knowledge bases and reveals parts that should be especially targeted by rule learning methods. It can also give hints to detect bad rules. For example, r_1 is probably going to create too many facts to comply with cardinality hints like "a person has at most two parents". However, surprisingly, despite its obvious importance, to date, no systematic way of making use of such information in rule learning exists.

In this chapter, we propose to exploit cardinality information about the expected number of edges in knowledge bases to better assess the quality of learned rules. Because knowledge bases are often incomplete in some areas and complete in others, our intuition is that some cardinality information might help to assess rules quality, at least by discarding rules which add facts in already complete areas like suggested earlier.

State of the art and its limitations

Galárraga, Teflioudi, et al., 2013 introduced a completeness-aware rule scoring based on the partial completeness assumption (PCA). The idea of the partial completeness assumption is that if at least one object for a given subject and a predicate is in a knowledge base (e.g., "Eduard is Einstein's child"), then all objects for that subject-predicate pair ("Einstein's children") are assumed to be known. This assumption was taken into account in rule scoring, and empirically it turned out to be indeed valid in real-world knowledge bases for some topics. However, it does not universally hold and does not correctly handle the case when edges in a graph are missing in a seeminlgy random fashion. For example, in encyclopedic knowledge bases like Wikidata, only well-known children are present in the knowl-

edge base, and not the less famous ones or the ones who died in their infancy. It is also the case for relations with no well-defined cardinalities, like "award received" or "position held". Often, only the most prominent values are provided for this relation. Similarly, Doppa et al., 2011 discussed the absence of contradiction as confirmation for "default" rules, i.e., rules that hold in most cases, leading the few contradicting facts to be highlighted and better covered in the knowledge base. Galárraga, Razniewski, et al., 2017 used crowd-sourcing to acquire completeness data. The acquired statements were then used in a post-processing step of rule learning to filter out predictions that violate these statements. However, this kind of filtering does not have any impact on the quality of the mined rules and the incorrect predictions for instances about which no completeness information exists.

Contributions

This work presents the first proper investigation of how cardinality information, and more specifically the number of edges that are expected to exist in the real world for a given subject-predicate pair in a knowledge base, can be used to improve rule learning. The contributions of our work are as follows:

- We present an approach that accounts for sparse data about the number of edges that should exist for given subject-predicate pairs in the ranking stage of rule learning. This is done by introducing a new ranking measure, the *completeness confidence*. This confidence generalizes both the standard confidence and the PCA confidence. Specifically, it performs identically to the two existing confidence measures if the corresponding cardinality metadata are provided. For example, if the knowledge base is stated to be globally complete, our completeness confidence will be the same as standard confidence, so it will also have the same performance.
- We implement our new ranking measure and evaluate it both on real-world and synthetic datasets, showing that they outperform existing measures both with respect to the quality of the mined rules and the predictions they produce.

3.2 Related Work

Rule learning

The problem of automatically learning patterns from knowledge bases has gained a lot of attention in recent years. Some relevant works are Galárraga, Teflioudi, et al., 2013; Galárraga, Teflioudi, et al., 2015; Zhichun Wang and Li, 2015, which focus on learning Horn rules and either ignore completeness information or make

use of completeness by filtering out predicted facts violating completeness in a post-processing step. On the contrary, we aim at injecting the statements into the learning process.

In the association rule mining community, some works focused on finding (interesting) *exception rules*, which are defined as rules with low support (rare) and high confidence (e.g., Taniar et al., 2008). Our work differs from this line of research because we do not necessarily look for rare interesting rules, but care about the quality of their predictions.

Another relevant stream of research is concerned with learning Description Logic TBoxes or schema (e.g., Lehmann et al., 2011). However, these techniques focus on learning concept definitions rather than nonmonotonic rules. The main difference between us and these approaches is that we aim at finding a hypothesis that consistently predicts unseen data, while DL learners focus more on learning models that perfectly describe the data.

In the context of inductive and abductive logic programming, learning rules from incomplete interpretations given as a set of positive facts along with a possibly incomplete set of negative ones was studied, e.g., in Law, Russo, and Broda, 2014. In contrast to our approach, this work does not exploit knowledge about the number of missing facts, and neither do the works on terminology induction, e.g., Sazonau, Uli Sattler, and Brown, 2015. Learning nonmonotonic rules in the presence of incompleteness was studied in hybrid settings (Józefowska, Lawrynowicz, and Lukaszewski, 2010; Lisi, 2010), where a background theory or a hypothesis can be represented as a combination of an ontology and Horn or nonmonotonic rules. The main point in these works is the assumption that there might be potentially missing facts in a given dataset. However, it is not explicitly mentioned which parts of the data are (in)complete like in our setting. Moreover, the emphasis of these works is on the complex reasoning interaction between the components, while we are more concerned with techniques for deriving rules with high predictive quality from large knowledge bases. The work by d'Amato et al., 2016 ranks rules using the ratio of correct versus incorrect predictions. These incorrect predictions are found using ontologies that allow determining incorrect facts. In contrast to our scenario of interest, in their work, the knowledge about the exact numbers of missing knowledge base facts has not been exploited. Since the publication of our original paper, Ho et al., 2018 presented an approach using numerical embeddings to predict possible missing facts. The authors used these predicted missing facts to refine the scoring of rules in incomplete areas. Muñoz, Minervini, and Nickles, 2019 used cardinality information to refine the learning of entity numerical embeddings to better predict facts missing in the knowledge base.

There are also many less relevant statistical approaches to completing knowledge bases based on, e.g., low-dimensional embeddings (Zhen Wang et al., 2014)

or tensor factorization (Nickel, Tresp, and Kriegel, 2012).

Completeness information

The idea of bridging the open and closed world assumption by using completeness information was first introduced in the database world in Levy, 1996; Etzioni, Golden, and Weld, 1997, and later adapted to the Semantic Web in Darari, Nutt, et al., 2013. For describing such settings, the common approach is to fix the complete parts (and assume that the rest is potentially incomplete).

Galárraga, Razniewski, et al., 2017 have extended the rule mining system AMIE to mine rules that predict where a knowledge base is complete or incomplete. The focus of the work is on learning association rules like *"If someone has a date of birth but no place of birth, then the place of birth is missing."* In contrast, we reason about the missing edges by trying to estimate the exact number of edges that should be present in a knowledge base. In Galárraga, Razniewski, et al., 2017 it has also been shown that completeness information can be used to improve the accuracy of fact prediction, by pruning out in a post-processing step those facts that are predicted in parts expected to be complete. In the present chapter, we take a more direct approach and inject completeness information already into the rule acquisition phase, to also prune away problematic rules, not only individual wrong predictions.

Our cardinality statements (e.g., "John has 3 children") encode knowledge about parts of a knowledge base that are (un)known, and thus should have points of contact with operators from epistemic logic; we leave the extended discussion on the matter for future work.

3.3 Approach

In the following we make the unique name assumption, i.e., we assume that each entity is represented by at most one individual (Definition 2.9 Page 20). We also use the distinction between ideal knowledge base and available knowledge base presented in Section 2.1 Page 21.

We use here all the definitions related to knowledge bases and rules presented in the preliminaries (Chapter 2). We also assume that the rule head contains a single role assertion $h(x, y)$. This does not introduce a loss of generality compared to rules with a conjunction of role assertions in the head because rules under the form $b(\vec{x}) \rightarrow h_1(\vec{x}) \wedge \cdots \wedge h_n(\vec{x})$ could be converted into an equivalent set of rules $b(\vec{x}) \rightarrow h_1(\vec{x}), \ldots, b(\vec{x}) \rightarrow h_n(\vec{x})$.

Scoring and ranking rules are core steps in association rule learning. A variety of measures for ranking rules have been proposed, with prominent ones being

confidence (Definition 2.20 Page 24), conviction (Brin et al., 1997), and lift[2].

The existing (in-)completeness-aware rule measure in the knowledge base context, the PCA confidence from Galárraga, Teflioudi, et al., 2015, already presented in Definition 2.20 Page 24, has two apparent shortcomings: First, it only counts as counterexamples those pairs (x, y) for which at least one $h(x, y')$ is in the current available knowledge base $\mathcal{K}^a$ for some y' and a rule's head predicate h. This means it may incorrectly give high scores to rules predicting facts for very incomplete relations, e.g., *place of baptism*. Second, it is not suited to relations that are not added in a "none or all" matter. For example, the most important values of an "awards" relation are often known, while the other values are often unknown or added to the knowledge base much later.

Thus, in this work, we focus on the improvements of rule scoring functions by making use of extra cardinality information. Before dwelling into the details of our approach we discuss the formal representation of such cardinality statements.

Cardinality Statements Overall, one can think of 4 different cardinality templates obtained by fixing the subject or object in a role assertion or considering a concept assertion and reporting the number of respective facts that hold in $\mathcal{K}^i$. E.g., for *hasChild*(`John`, `Mary`) we can count (1) the children of `John`; (2) the people of which `Mary` is a child of; or (3) the facts over *hasChild* relation. And for *Person*(`John`) we can count (4) the number of elements in the *Person* concept.

In practice, numerical statements for templates (1), (2), and (4) can be obtained using web extraction techniques (Mirza, Razniewski, and Nutt, 2016), from functional properties of relations, from closed concepts, or from crowd-sourcing. For (3) things get trickier. We leave this issue for future work and focus here only on templates (1), (2), and (4). We work in the following only on template (1). Indeed, we can rewrite (2) to (1) provided that inverse relations can be expressed in a knowledge base. For instance, $|\{s \mid \mathit{hasChild}(s, \texttt{john})\}| = |\{o \mid \mathit{hasParent}(\texttt{john}, o)\}|$ for the predicates *hasChild* and *hasParent*, which are inverses of one another. More, it is possible to rewrite (4) to (1) with some notation abuse by writing $\mathit{hasInstance}(C, o)$ the concept assertion $C(o)$.

We have now a simple way to report (the numerical restriction on) the absolute number of facts over a certain roles in the ideal knowledge base $\mathcal{K}^i$. To formalize it, we define the partial function *num* that takes as input a role $p \in \mathsf{N_R}$ and an individual $s \in \mathsf{N_I}$ and outputs a natural number corresponding to the number of facts in $\mathcal{K}^i$ over p with s as the first argument:

$$\mathit{num}(p, s) := |\{o \mid \mathcal{K}^i \models p(s, o)\}| \tag{3.1}$$

[2]The lift of a rule $r(\vec{x}) = b(\vec{x}) \to h(\vec{x})$ is defined by $\mathit{lift}(r) = \frac{\mathit{supp}(b \wedge h)}{\mathit{supp}(b) \cdot \mathit{supp}(h)}$.

num is a partial function because most of the time we do not know all cardinalities. For example, in a knowledge base about some people's family, the number of biological parents is always known whereas it might be the case that the number of children is known for some people only and the number of siblings is never known.

Naturally, the number of missing facts for a given p and s can be obtained as

$$miss(p, s) := num(p, s) - |\{o \mid \mathcal{K}^a \models p(s, o)\}| \tag{3.2}$$

Example 3.1. Consider the knowledge base in Figure 3.1 (the same one as the knowledge base presented in the preliminaries) and the following cardinality statements for it:

$num(hasChild, \texttt{John}) = 3$
$num(hasChild, \texttt{Mary}) = 3$
$num(hasChild, \texttt{Alice}) = 1$
$num(hasChild, \texttt{Carol}) = 0$
$num(hasChild, \texttt{Dave}) = 0$

$num(hasSibling, \texttt{Bob}) = 3$
$num(hasSibling, \texttt{Alice}) = 2$
$num(hasSibling, \texttt{Carol}) = 2$
$num(hasSibling, \texttt{Dave}) = 2$

We then have:

$miss(hasChild, \texttt{Mary}) = miss(hasChild, \texttt{John}) = miss(hasChild, \texttt{Alice}) = 1;$
$miss(hasChild, \texttt{Carol}) = miss(hasChild, \texttt{Dave}) = 0;$

$miss(hasSibling, \texttt{Bob}) = miss(hasSibling, \texttt{Carol}) = 2;$
$miss(hasSibling, \texttt{Alice}) = miss(hasSibling, \texttt{Dave}) = 1.$ □

We are now ready to define the *completeness-aware rule scoring problem*. Given a knowledge base and a set of cardinality statements, *completeness-aware rule scoring* aims to score rules not only by their predictive power on the known knowledge base like presented in Section 2.3 Page 22, but also with respect to the number of wrongly predicted facts in complete areas and the number of newly predicted facts in known incomplete areas.

In this work, we propose to explicitly rely on incompleteness information to determine whether to consider an assignation as a counterexample for a rule at hand or not.

To do that, we first define two indicators for a given rule r, reflecting the number of new predictions made by r in incomplete ($npi(r)$) and, respectively, complete ($npc(r)$) knowledge base parts:

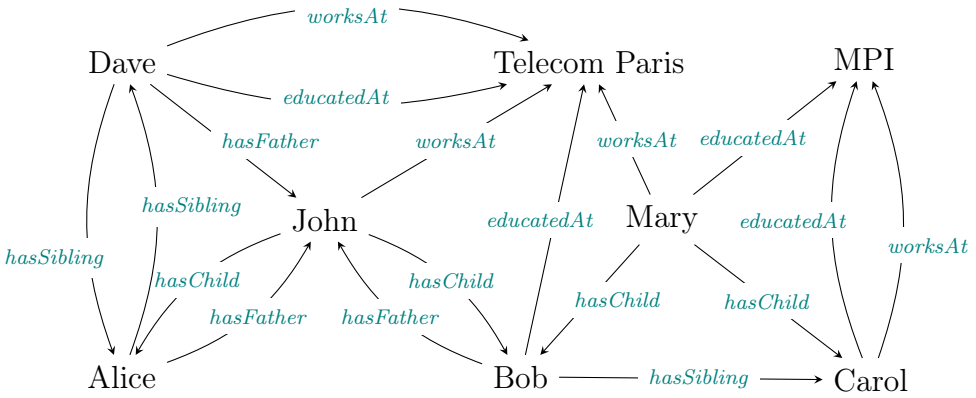

Figure 3.1 – Example knowledge base

Definition 3.2. The *Number of Predictions in Incomplete parts* of a rule $r(x, y) = b(x, y) \rightarrow h(x, y)$ is the number of new predictions of r that add objects in the places where it is known that values are missing. It is formally defined by:

$$npi(r) := \sum_{x \in \mathsf{N_I} \text{ and } miss(h,x) \text{ is defined}} min(pred(r, x), miss(h, x))$$

where $pred(r, x)$ is the number of new predictions of r for a given x:

$$pred(r, x) := |\{y \mid \mathcal{K}^a \models b(x, y) \wedge \mathcal{K}^a \not\models h(x, y)\}|$$

Definition 3.3. Similarly, the *Number of Predictions in Complete parts* of a rule $r(x, y) = b(x, y) \rightarrow h(x, y)$ is the number of new predictions of r that add objects in the places where it is known that $miss(h, x)$ objects are missing and the rules already predicted $miss(h, x)$ other objects. It is formally defined by:

$$npc(r) := \sum_{x \in \mathsf{N_I} \text{ and } miss(h,x) \text{ is defined}} max(pred(r, x) - miss(h, x), 0)$$

where $pred(r, x)$ is the number of new predictions of r for a given x:

$$pred(r, x) := |\{y \mid \mathcal{K}^a \models b(x, y) \wedge \mathcal{K}^a \not\models h(x, y)\}$$

Example 3.4. Consider the knowledge base whose ABox is displayed in Figure 3.1 with an empty TBox, and the cardinality statements described in Example 3.1.

We consider the rules:

$$r_1 : worksAt(x, z) \wedge educatedAt(y, z) \rightarrow hasChild(x, y)$$

$$r_2 : hasFather(x, y) \wedge hasChild(y, z) \rightarrow hasSibling(x, z)$$

r_1 predicts new *hasChild* values for `Carol`, `Dave`, `John`, and `Mary`. Among them, the expected cardinality is only known for `Carole` and `Dave`.

The two predicted assertions for `Carol` are *hasChild*(`Carol`, `Carol`) and *hasChild*(`Carol`, `Mary`). Among them, none is already in the knowledge base so $pred(r_1, \texttt{Carol}) = 2$. With a similar reasoning, we get $pred(r_1, \texttt{Dave}) = 2$.

The two predicted assertions for `John` are *hasChild*(`John`, `Dave`) and *hasChild*(`John`, `Bob`). However *hasChild*(`John`, `Bob`) is already in the knowledge base so $pred(r_1, \texttt{John}) = 1$. In the same way, we get $pred(r_1, \texttt{Mary}) = 1$.

Because $miss(hasChild, \texttt{Carol}) = 0$, $miss(hasChild, \texttt{Dave}) = 0$, $miss(hasChild, \texttt{John}) = 1$, and $miss(hasChild, \texttt{Mary}) = 1$, we have $npi(r_1) = min(2, 0)+min(2, 0)+min(1, 1) + min(1, 1) = 2$ and $npc(r_1) = max(2 - 0, 0) + max(2 - 0, 0) + max(1 - 1, 0) + max(1 - 1, 0) = 4$.

Similarly, we get $npi(r_2) = 4$ and $npc(r_2) = 1$. □

Exploiting these additional indicators for $r(x, y) : b(x, y) \rightarrow h(x, y)$ we obtain the following *completeness-aware confidence*:

Definition 3.5. The *completeness-aware confidence* confidence is defined by:

$$conf_{comp}(r) := \frac{supp(r)}{supp(b) - npi(r)}$$

Example 3.6. We consider the same knowledge base and rules as Example 3.4. Obviously, the rule r_2 should be preferred over r_1.

The body and rule supports of r_1 over the knowledge base are $supp(b_1) = 8$ and $supp(r_1) = 2$ respectively. Hence, we have $conf(r_1) = \frac{2}{8}$ and $conf(r_2) = \frac{1}{6}$.

As discussed before, we obtain $conf_{pca}(r_1) = \frac{2}{4}$. Indeed, since `Carol` and `Dave` are not known to have any children in the knowledge base, four existing body substitutions are not counted in the denominator. Meanwhile, we have $conf_{pca}(r_2) = \frac{1}{6}$, since all people that are predicted to have siblings by the rule already have siblings in the available knowledge base $\mathcal{K}^a$.

For our completeness confidence, $conf_{comp}(r_1) = \frac{2}{8-2} = \frac{2}{6}$ and $conf_{comp}(r_2) = \frac{1}{6-4} = \frac{1}{2}$, resulting in the desired rule ordering, which is not achieved by existing measures. □

Our completeness confidence generalizes both the standard and the PCA confidence. Indeed, under the Closed World Assumption, the knowledge base is supposed to be fully complete, i.e., for all $p \in \mathsf{N_R}, s \in \mathsf{N_I}$ we have $miss(p, s) = 0$. Similarly, if we assume the Partial Completeness Assumption, then for all pairs p, s such that at least one $o \in \mathsf{N_I}$ exists with $\mathcal{K}^a \models p(s, o)$, $miss(p, s) = 0$ and $miss(p, s) = +\infty$ for the others.

Proposition 3.7. *For every knowledge base $\mathcal{K}$ and rule r it holds that*

(i) under the Closed World Assumption (CWA) $conf_{comp}(r) = conf(r)$;

(ii) under the Partial Completeness Assumption (PCA) $conf_{comp}(r) = conf_{pca}(r)$.

Proof. **(i)** Under the Closed World Assumption, it holds that for all $p \in \mathsf{N_R}$, $s \in \mathsf{N_I}$, $miss(p, s) = 0$. Thus, for all rules r, we have that $npi(r) = 0$, and hence, $conf_{comp}(r) = conf(r)$.
(ii) Under the Partial Completeness Assumption, it is assumed that for all $p \in \mathsf{N_R}$ and $s \in \mathsf{N_I}$ it holds that $miss(p, s) = 0$ if $\exists o\, \mathcal{K}^a \models p(s, o)$ and $miss(p, s) = +\infty$ if not. Hence, for all r we have $npi(r) = \sum_x predict(r, x)$, where

$$predict(r, x) = \begin{cases} 0, \text{ if } \exists y\, \mathcal{K}^a \models h(x, y) \\ pred(r, x), \text{ if } \forall y'\, \mathcal{K}^a \not\models h(x, y') \end{cases}$$

From this we get

$$conf_{comp}(r) = \frac{supp(r)}{supp(b) - \sum_{x \text{ such that } \forall y'\, \mathcal{K}^a \not\models h(x,y')} pred(r, x)}$$

The denominator of the latter formula counts all matches of the rule body and subtracts from them those for which the head $h(x, y)$ is predicted and $\forall y'\, \mathcal{K}^a \not\models h(x, y')$. Hence, we end up counting only body substitutions with $\mathcal{K}^a \models h(x, y')$ for at least one y', i.e., $|\{(x, y) \mid \exists y\, \mathcal{K}^a \models b(x, y) \wedge \exists y'\, \mathcal{K}^a \models h(x, y')\}| = supp_{pca}(r)$. Hence, $conf_{comp}(r) = \frac{supp(r)}{supp_{pca}(r)} = conf_{pca}(r)$. □

3.4 Evaluation

We have implemented our **c**ompleteness-**a**ware **r**ule **l**earning approach into a C++ system prototype CARL[3], following a standard relational learning algorithm implementation such as Goethals and Bussche, 2002. While our general methodology can be applied to mining rules of arbitrary form, in the experimental evaluation to restrict the rule search space we focus only on rules with two atoms in their body of the form

$$p(x, y) \wedge q(y, z) \rightarrow r(x, z) \tag{3.3}$$

Note that here p, q, and r are not allowed to denote inverse property expressions. We aim to compare the predictive quality of the top k rules mined by our completeness-aware approach with the ones learned by standard rule learning

[3]The source code and all the data are available at `https://github.com/Tpt/CARL`.

methods: (1) the PCA confidence (Definition 2.20 Page 24) from AMIE (Galárraga, Teflioudi, et al., 2015) and (2) the standard confidence (Definition 2.18 Page 23) used e.g., by WarmeR (Goethals and Bussche, 2002).

Because the goal of our new confidence is to provide a better rule quality evaluation measure, we only compare with other rule quality measures that have already proved their quality and usefulness, and we do not compare with fact prediction approaches or with baselines.

Dataset

We used two datasets for the evaluation: (i) *WikidataPeople*, which is a dataset we have created from the Wikidata knowledge base, containing 2.4M facts over 9 predicates[4] about biographical information and family relationships of people; and (ii) *LUBM*, which is a synthetic dataset describing the structure of a university (Guo, Pan, and Heflin, 2005).

For the WikidataPeople dataset, the approximation of the ideal knowledge base ($\mathcal{K}^i$) is obtained by exploiting available information about inverse relations (e.g., *hasParent* is the inverse of *hasChild*), functional relations (e.g., *hasFather*, *hasMother*) as well as manually hand-crafted solid rules from the family domain like

$$hasParent(x, z) \wedge hasParent(y, z) \wedge x \neq y \rightarrow hasSibling(x, y).^5$$

We generated cardinality statements from WikidataPeople $\mathcal{K}^i$ by exploiting properties of functional relations, e.g., *hasBirthPlace*, *hasFather*, *hasMother* must be uniquely defined. We considered also that everybody with a *hasDeathDate* has a *hasDeathPlace*. For the other relations, the PCA (Galárraga, Teflioudi, et al., 2015) is used. This resulted in 10M cardinality statements.

LUBM $\mathcal{K}^i$, with 1.2M facts, was constructed by running the LUBM data generator for 10 universities, removing all `rdf:type` triples, and introducing inverse predicates. 464K cardinality statements were obtained by counting the number of existing objects for each subject-predicate pair after completing the graph using the provided OWL ontology, i.e., assuming the partial completeness assumption on the whole dataset.

Experimental Setup

To assess the effect of our proposed measures, we first construct versions of the available knowledge base ($\mathcal{K}^a$) by removing parts of the data from $\mathcal{K}^i$ and introducing a synthetic bias in the data (i.e., leaving many facts in $\mathcal{K}^a$ for some relations

[4] *hasFather*, *hasMother*, *hasStepParent*, *hasSibling*, *hasSpouse*, *hasChild*, *hasBirthPlace*, *hasDeathPlace*, and *hasNationality*

[5] see `https://github.com/Tpt/CARL/tree/master/eval/wikidata` for details

and few for others). The synthetic bias is needed to simulate our scenario of interest, where some parts of $\mathcal{K}^a$ are very incomplete while others are fairly complete, which is indeed the case in real-world knowledge bases. In Wikidata, for instance, sibling information is only reported for only 3% of non-living people, while children data is known for 4%.

For that, we proceed in two steps: First, we define a *global ratio*, which determines a uniform percentage of data retained in the available graph. This ratio is useful to check how the different measures perform with respect to the (in-)completeness of the knowledge base. To further refine this, we then define a *predicate ratio* individually for each predicate to simulate the real world discrepancies between fairly complete and very incomplete areas. For the WikidataPeople knowledge base, this ratio is chosen as *(i)* 0.8 for *hasFather* and *hasMother*; *(ii)* 0.5 for *hasSpouse*, *hasStepParent*, *hasBirthPlace*, *hasDeathPlace* and *hasNationality*; *(iii)* 0.2 for *hasChild*; and *(iv)* 0.1 for *hasSibling*. For the LUBM dataset, the predicate ratio is uniformly defined as 1 for regular predicates and 0.5 for inverse predicates. Note that this static predicate ratio is here to simulate the completeness gap in the knowledge base between predicates. Hence, the predicate ratio is fixed as part of the design of the experimental setup, and we do not make it vary or otherwise study it in our experimental results

For a given predicate, the final ratio of facts in $\mathcal{K}^a$ retained from those in $\mathcal{K}^i$ is then computed as $min(1, 2 \cdot k \cdot n)$, where k is the predicate ratio and n is the global ratio.

Note that, after randomly removing facts, the partial completeness assumption no longer holds on the data. This gives our method a competitive edge over competing approaches such as AMIE which assume the partial completeness assumption.

The assessment of the rules learned from different versions of the available knowledge base is performed by comparing rule predictions with the approximation of $\mathcal{K}^i$. More specifically, every learned rule is assigned a *quality score*, defined as the ratio of the number of predictions made by the rule in $\mathcal{K}^i \setminus \mathcal{K}^a$ over the number of all predictions outside $\mathcal{K}^a$.

$$qualityScore(r) = \frac{|\mathcal{K}_r^a \cap \mathcal{K}^i \setminus \mathcal{K}^a|}{|\mathcal{K}_r^a \setminus \mathcal{K}^a|} \tag{3.4}$$

This scoring naturally allows us to control the percentage of rule predictions that hit our approximation of $\mathcal{K}^i$, similar to standard recall estimation in machine learning (c.f. Definition 2.25 Page 26).

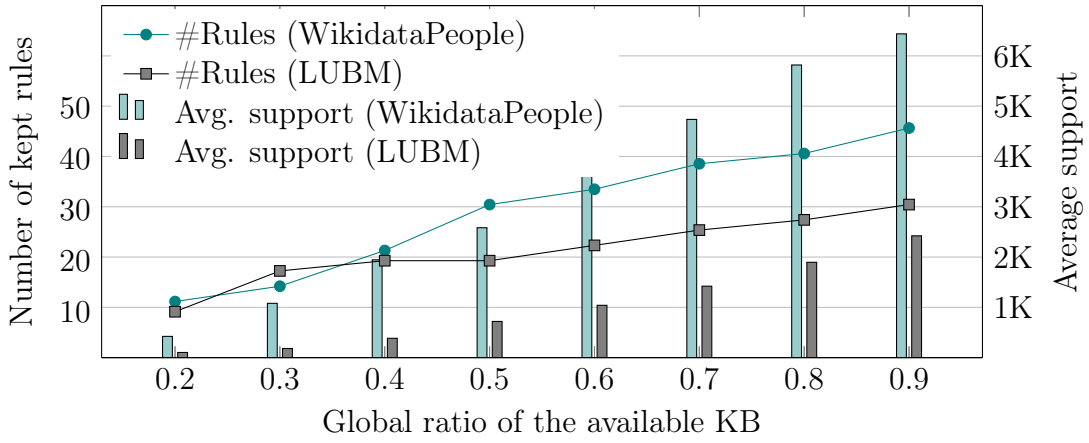

Figure 3.2 – Number of kept rules and their average support for WikidataPeople and LUBM datasets

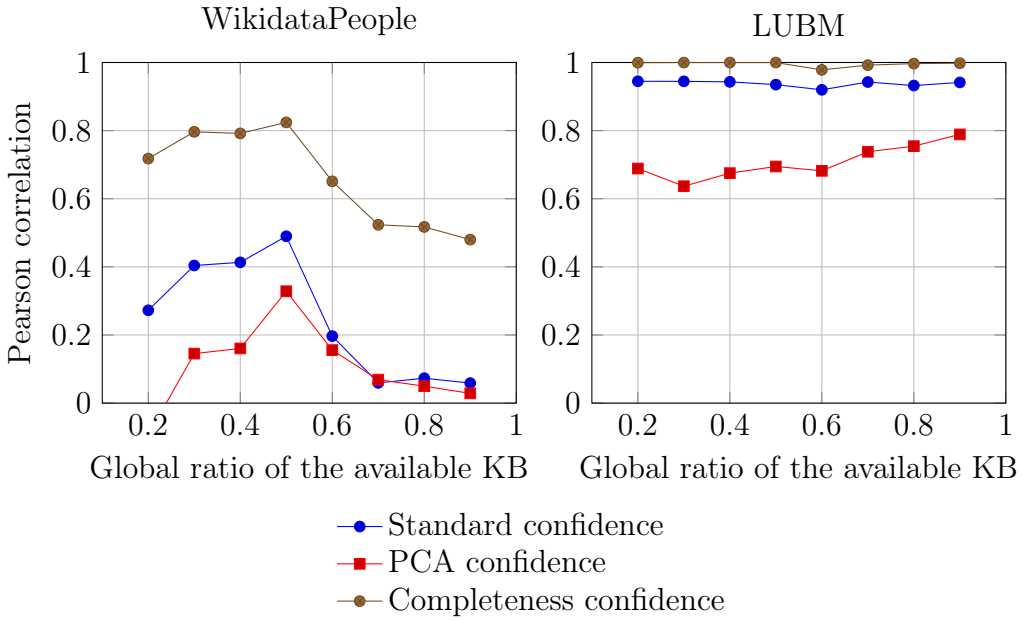

Figure 3.3 – Evaluation results for WikidataPeople and LUBM datasets

Table 3.1 – Example of a good and a bad rule mined from WikidataPeople with global ratio of 0.5

Rule r	$conf(r)$	$conf_{pca}(r)$	$conf_{comp}(r)$
$hasSibling(x,y) \wedge hasSibling(y,z)$ $\rightarrow hasSibling(x,z)$	0.10	0.10	0.89
$hasMother(x,y) \wedge hasSpouse(y,z)$ $\rightarrow hasStepParent(x,z)$	0.0015	0.48	0.0015

Results

From every version of the available knowledge base, we have mined all possible rules with two atoms in their body of the form of Equation 3.3 (page 38). For that, we search all the possible tuples (p,q,r) such that the conjunctive query $(x,y) \wedge q(y,z) \wedge r(x,z)$ has at least one answer. We kept only rules r with $conf(r) \geq 0.001$ and $supp(r) \geq 10$, whose *head coverage*[6] is greater than 0.001. Figure 3.2 shows the number of kept rules and their average support (Definition 2.17 Page 23) for each global ratio used for generating $\mathcal{K}^a$.

The evaluation results for WikidataPeople and LUBM datasets are in Figure 3.3. The horizontal axis displays the global ratio used for generating $\mathcal{K}^a$. We compared different rule ranking methods as previously discussed. The Pearson correlation coefficient (vertical axis) between each ranking measure and the quality score of rules (Equation 3.4) is used to evaluate the measures' effectiveness. The Pearson correlation coefficient between two random variables X and Y with numerical values is defined by $\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$ with cov being the covariance and σ the standard deviation. We measured the Pearson correlation coefficient since apart from the ranking order (as captured by, e.g., Spearman's rank correlation coefficient), the absolute values of the measures are also insightful for our setting.

Since facts are randomly missing in the considered versions of $\mathcal{K}^a$, the PCA confidence performs worse than the standard confidence for given datasets, while our completeness confidence significantly outperforms both (see Table 3.1 for examples).

For both knowledge bases, the completeness confidence outperforms the rest of the measures. It is noteworthy that the standard confidence performs considerably better on the LUBM knowledge base with a correlation coefficient higher than 0.9 than on the WikidataPeople knowledge base. Still, completeness confidence shows better results, reaching a nearly perfect correlation of 0.99. We hypothesize that

[6] *Head coverage* is the ratio of the number of predicted facts that are in $\mathcal{K}^a$ over the number of facts matching the rule head.

this is due to the bias between the different predicates of the LUBM knowledge base being less strong than in the WikidataPeople knowledge base, where some predicates are missing a lot of facts, while others just a few.

3.5 Conclusion

We have defined the problem of learning rules from incomplete knowledge bases enriched with the exact numbers of missing edges of certain types. We proposed a novel rule ranking measure that effectively makes use of sparse cardinality information, the *completeness confidence*. Our new measure has been injected in the rule learning prototype CARL and evaluated on real-world and synthetic knowledge bases, demonstrating significant improvements both with respect to the quality of mined rules and with respect to the predictions they produce.

For future work, it would interesting to expend the experiments. We could benchmark our completeness confidence against Galárraga, Razniewski, et al., 2017 to check which method produces the best predictions and which has the best runtime. We could also adapt the AMIE rule learning algorithm to make it work with our completeness confidence instead of the PCA confidence.

Chapter 4

Increasing the Number of Numerical Statements

The work presented in this chapter has been done with Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum and has been published at ISWC 2017 alongside the previous chapter content.

> Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum (2017). "Completeness-Aware Rule Learning from Knowledge Graphs". In: ISWC, pp. 507–525. https://doi.org/10.1007/978-3-319-68288-4_30

4.1 Introduction

We have shown in the previous chapter that the exploitation of numerical completeness statements is beneficial for rule quality assessment. A natural question is where to acquire such statements in real-world settings. Like already mentioned in the previous chapter, various works have shown that numerical assertions can be frequently found on the Web (Darari, Nutt, et al., 2013), obtained via crowd-sourcing (Darari, Razniewski, et al., 2016), text mining (Mirza, Razniewski, Darari, et al., 2017), or even using Hoeffding's inequality to find upper bounds (Giacometti, Markhoff, and Soulet, 2019). Other work like Galárraga, Razniewski, et al., 2017 aims at mining if the knowledge base is complete for a given subject and predicate, a much more restricting set that mining the actual (in-)completeness statement. We believe that mining numerical correlations concerning knowledge base edges and then assembling them into rules is a valuable and a modular approach to increase the amount of completeness information, which we present in

what follows. This approach works by mining rules like "If a person has more than 2 siblings, then their parents are likely to have more than 3 children".

4.2 Approach

In the following we make the unique name assumption, i.e., we assume that each entity is represented by at most one individual (Definition 2.9 Page 20).

We start with an available knowledge base $\mathcal{K}^a = (\mathcal{T}, \mathcal{A}^a)$ and some statements of the form described in the Equation 3.1 from Section 3.3 of the previous chapter, namely:

$$num(p, s) := |\{o \mid \mathcal{K}^i \models p(s, o)\}|$$

where $\mathcal{K}^i = (\mathcal{T}, \mathcal{A}^i)$ is the ideal knowledge base.

Step 1 Let $p_{\leq k}$ and $p_{\geq k}$ where $k \in \mathbb{N}$ be fresh concepts not present in $\mathsf{N_C}$, which describe bounds on the number of outgoing p-edges for a given constant. For every cardinality $num(p, s) = k$, we create the facts $p_{\leq k}(s)$ and $p_{\geq k}(s)$. For the pairs $p \in \mathsf{N_R}, s \in \mathsf{N_I}$ with no available cardinality statements we construct the facts $p_{\geq n}(s)$ where $n = |\{o \mid \mathcal{K}^a \models p(s, o)\}|$, encoding that there may be outgoing p-edges that are missing from s, as the graph is believed to be incomplete by default. We store all constructed facts over p_{card} in $\mathcal{S}$.

We then complete the domain of each p_{card} predicate as follows. For every $p_{\leq k}(s) \in \mathcal{S}$, if $p_{\leq k'}(s') \in \mathcal{S}$ for some $s' \in \mathsf{N_I}$ and $k' > k$, we construct the rule $p_{\leq k}(x) \rightarrow p_{\leq k'}(x)$ Similarly, for every $p_{\geq k}(s) \in \mathcal{S}$, if $p_{\geq k'}(s') \in \mathcal{S}$ where $k' < k$, we create $p_{\geq k}(x) \rightarrow p_{\geq k'}(x)$.

The constructed rules are then applied to the facts in $\mathcal{S}$ to obtain an extended set $\mathcal{A}^{card}$ of facts over p_{card}. The latter step is crucial when using a rule mining system that is not doing arithmetic inferences (such as "$x > 4$ implies $x > 3$").

Step 2 We then use a standard rule learning system, AMIE (Galárraga, Teflioudi, et al., 2015), on $\mathcal{A}^a \cup \mathcal{A}^{card}$ to mine rules like:

(1) $p'_{card}(x) \rightarrow p_{card}(x)$

(2) $p'_{card}(x) \wedge p''_{card}(x) \rightarrow p_{card}(x)$

(3) $p'_{card}(x) \wedge r(x, y) \rightarrow p_{card}(x)$

(4) $p'_{card}(x) \wedge r(x, y) \wedge p''_{card}(y) \rightarrow p_{card}(x)$

(5) $r(x, y) \wedge p''_{card}(y) \rightarrow p_{card}(x)$

We rank the obtained rules based on confidence and select the ones that pass operator-chosen minimal confidence and minimal support thresholds into the set $\mathcal{R}$.

Step 3 Finally, in the last step we use the obtained ruleset $\mathcal{R}$ to derive further numerical statements together with weights assigned to them. For that we compute $\mathcal{A}' = \bigcup_{r \in \mathcal{R}} \{\mathcal{A}^{card} \cup \mathcal{A}^a\}_r$. The weights of the statements are inherited from the rules that derived them. We then employ two simple heuristics: (i) Given multiple rules predicting the same fact, the highest weight for it is kept. We then post-process predictions made by different rules for the same subject-predicate pair as follows: (ii) If $p_{\leq k}(s), p_{\geq k'}(s) \in \mathcal{A}'$ for $k' > k$, we remove from $\mathcal{A}'$ predictions with the lowest weight thus resolving the conflict on the numerical bounds.

From the obtained graph we reconstruct cardinality statements as follows.

- Given $p_{\leq k}(s), p_{\geq k}(s) \in \mathcal{A}'$ with weights w and w' we create a cardinality statement $num(p, s) = k$ with the weight $min(w, w')$.
- If $p_{\leq k}(s), p_{\geq k'}(s) \in \mathcal{A}'$ for $k' < k$, then we set $k' \leq num(p, s) \leq k$.
- Among two facts $p_{\leq k}(s), p_{\leq k'}(s)$ (respectively $p_{\geq k}(s)$, $p_{\geq k'}(s)$) with $k < k'$ (respectively $k > k'$) the first ones are kept and represented using the two previous reconstructions.

Regular facts in $\mathcal{A}'$ are similarly translated into their numerical representations.

Example 4.1. Consider the knowledge base in Figure 3.1 Page 36 and the following cardinality statements for it: $num(hasChild, \texttt{john}) = num(hasSibling, \texttt{bob}) = 3$.

Among others, $\mathcal{A}^{card}$ contains the facts: $hasChild_{\geq 3}(\texttt{john})$, $hasSibling_{\geq 3}(\texttt{bob})$, $hasChild_{\geq 2}(\texttt{mary})$, $hasChild_{\geq 2}(\texttt{john})$, $hasSibling_{\geq 2}(\texttt{bob})$, $hasSibling_{\geq 1}(\texttt{dave})$, and $hasSibling_{\geq 1}(\texttt{alice})$.

On the graph $\mathcal{A}^a \cup \mathcal{A}^{card}$, the confidence of $hasFather(x, y) \wedge hasChild_{\geq 3}(y) \rightarrow hasSibling_{\geq 2}(x)$ is $\frac{1}{3}$ and 1 for $hasFather(x, y) \wedge hasChild_{\geq 3}(y) \rightarrow hasSibling_{\geq 1}(x)$. □

4.3 Evaluation

To evaluate our method for the automated acquisition of cardinality statements from a knowledge base, we reused the WikidataPeople dataset described in Section 3.4 Page 39. Different from the previous chapter, we do not complete the dataset with basic rules (like materializing the inverse relations).

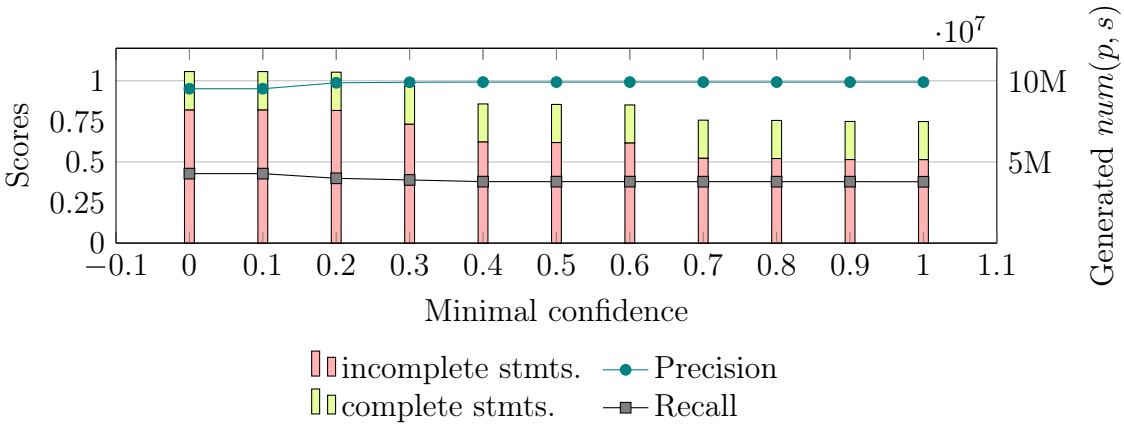

Figure 4.1 – Number of (in-)complete statements for generated cardinalities $num(p, s)$, and quality of predicted cardinalities.

Dataset

We have collected around 282K cardinality statements from various sources:

- Wikidata schema, i.e., *hasFather*, *hasMother*, *hasBirthPlace*, and *hasDeathPlace* are functional properties and, thus, should have at most one value.
- The 7.5K values of the Wikidata predicate *numberOfChildren*;
- 663 *novalue* statements from Wikidata;
- 86K cardinality statements from Mirza, Razniewski, and Nutt, 2016 for the *hasChild* predicate of Wikidata;
- 182K cardinality statements are extracted from human-curated and complete Freebase facts (1.6M). The mapping to Wikidata has been done using tools from Pellissier Tanon, Vrandečić, et al., 2016.

Experimental Setup

For the sake of simplicity and easy reproducibility, we set aside 20% of the cardinality statements, chosen uniformly at random, to build a validation set, while the rest were incorporated into the WikidataPeople knowledge base following the previously described approach. We then ran our rule learning algorithm to mine cardinality rules. We have set the minimal support threshold to 200 and the minimal confidence threshold to 0.01. Examples of mined rules along with their standard confidences include

- $hasSibling(x, y) \wedge hasSibling_{\geq 4}(y) \rightarrow hasSibling_{\geq 3}(x)$: 0.97
- $hasFather(y, x) \wedge hasSibling_{\geq 4}(y) \rightarrow hasChild_{\geq 3}(x)$: 0.90.

The learned rules were then applied to the enriched WikidataPeople knowledge base to retrieve new exact cardinalities $num(p, s)$ by only keeping (p, s) pairs where the higher and lower bounds matched. The minimum of the standard confidence of the best rules used to get the upper and lower bounds was assigned as the final confidence of each $num(p, s)$.

Results

We aim to evaluate whether we can accurately recover the cardinality statements in the validation set (which is our gold standard) by utilizing the learned cardinality rules. For different minimal confidence thresholds, the quality of the predicted cardinalities is measured with standard precision and recall (c.f. Definitions 2.24 and 2.25, Page 25), which is presented in Figure 4.1.

To our knowledge, this is the first work to mine cardinalities from existing ones in the knowledge base, so we do not provide here comparisons with other works. Indeed, Giacometti, Markhoff, and Soulet, 2019 has been published after the evaluation was conducted and published.

We get a nearly perfect precision and a fair recall (around 40%) for the generated cardinalities, which amount to 7.5M-10M depending on the threshold. Around one third of $num(p, s)$ statements indicate completeness of the knowledge base for some given (p, s) pairs.

The top rules mined often present very relaxed cardinality bounds like $hasFather(x, y) \wedge hasChild_{\geq 4}(y) \rightarrow hasSibling_{\geq 1}(x)$. Indeed these rules have better support because of the facts already present in the knowledge base which provide lower bounds on the cardinalities. A more in-depth study of the mined rules is left for future work.

4.4 Conclusion

In this chapter we have presented a new approach to increase the number of cardinality metadata in a knowledge base. This approach works by mining rules on the upper and lower bounds of the number of expected objects for a given subject and predicate. An evaluation allows us to conclude that our proposed approach is effective. This approach can be used to improve the performance of the completeness confidence presented in the previous chapter.

For future work, provided that sufficiently many similar numerical correlations about edge numbers are extracted, one could attempt to induce more general constraints involving arithmetic functions like the "number of siblings of a person is

bounded by the number of his parents' children minus 1" or "the sum of the number of a person's brothers and the number of their sisters equals the number of their siblings". Similarly, the simple heuristic used to build back cardinality assertions from the mined upper and lower bounds provides room for more advanced voting/weighting schemes and inconsistency resolution in the case of conflicting cardinality assertions.

This chapter concludes Part I of this thesis, where we have focused on acquiring new rule based constraints to complete knowledge bases. In Parts II and III, we will focus on how to enforce them, especially to repair constraints violations.

Part II

Enforcing Constraints Statically

Chapter 5

YAGO 4: A Reason-able Knowledge Base

The work presented in this section has been done with Gerhard Weikum and Fabian Suchanek and has been published as a resource paper at ESWC 2020. Most of the design and the implementation have been done by the author of this thesis.

> Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. "YAGO 4: A Reason-able Knowledge Base". Resource paper at ESWC 2020. https://doi.org/10.1007/978-3-030-49461-2_34

5.1 Introduction

In Part I, we have presented an approach to mine new constraints. In this part, we aim at enforcing them. This chapter presents an example of knowledge base, YAGO 4, that statically enforces constraints on the data during its generation process.

YAGO (Suchanek, Kasneci, and Weikum, 2007; Hoffart et al., 2011; Mahdisoltani, Biega, and Suchanek, 2015; Rebele, Suchanek, et al., 2016) was one of the first academic projects to build a knowledge base automatically. The main idea of YAGO was to harvest information about entities from the infoboxes and categories of Wikipedia and to combine this data with an ontological backbone derived from classes in WordNet (Fellbaum, 1998). Since Wikipedia is an excellent repository of entities, and WordNet is a widely used lexical resource, the combination proved useful. YAGO sent each fact through a pipeline of filtering, constraint checking, and de-duplication steps. This procedure scrutinized noisy input and boosted the quality of the final knowledge base, to a manually verified accuracy of 95%. This

precision was possible thanks to the tight control that the YAGO creators had over the extraction process, the filtering process, the ontological type system, the choice of the relations, and the semantic constraints. However, despite new versions YAGO 2 and YAGO 3 with substantial jumps in scope and size, the focus on Wikipedia infoboxes meant that YAGO has not arrived at the same scale as Freebase or Wikidata.

Meanwhile, Wikidata (Vrandecic and Krötzsch, 2014) has evolved into the world's foremost publicly available knowledge base. It is a community effort where anybody can contribute facts – either by manually adding or curating statements in the online interface or by bulk-loading data. Wikidata has motivated more than 40,000 people who contribute at least once a month. The result is a public knowledge base with 70M named entities, very good long-tail coverage, and impressive detail[1].

At the same time, Wikidata has been designed by its founders to be a collection of information, not as a collection of universally agreed-upon knowledge (Vrandecic and Krötzsch, 2014). It may intentionally contain contradictory statements, each with different sources or validity areas. Therefore, Wikidata does not enforce semantic constraints, such as "each person has exactly one father". Furthermore, the large user community has led to a proliferation of relations and classes: Wikidata contains 6.7k relations, of which only 2.6k have more than 1000 facts, and it contains around 2.4M classes[2], of which 80% have less than 10 instances. Many instances (e.g., all cities) are placed in the taxonomy under more than 60 classes. This complexity is the trade-off that Wikidata has found to accommodate its large user community. For downstream applications, the convoluted and often confusing type system of Wikidata makes browsing and question answering tedious. Moreover, there is little hope to run strict classical reasoners (e.g., for OWL 2) in a meaningful way, as the knowledge base contains many small inconsistencies. Hence, every possible statement is deducible regardless of whether it is intuitively correct or false. Some of these issues have been pointed out in the comprehensive study of knowledge base quality by Zaveri, Rula, et al., 2016.

Example

To illustrate the shortcomings of the verbose and sometimes confusing type hierarchy of Wikidata, consider the entities *Notre Dame de Paris* (`http://www.wikidata.org/entity/Q2981`) and *Potala Palace* (`http://www.wikidata.org/entity/Q71229`) both of which are landmarks of two world religions.

[1] All the numbers given in the chapter about Wikidata are valid as of Feb. 24, 2020.

[2] Wikidata does not have a strong concept of a "class"; we use this term to denote entities that have superclasses (i.e., appear as subject of "subclass of" triples).

As of November 2019, Notre Dame was an instance of types *catholic cathedral* and *minor basilicas*, with a rich set of superclasses.

The Potala Palace in Lhasa was an instance of *palace* and *tourist attraction*. Interestingly, the latter did not have Notre Dame de Paris as an instance, neither directly nor indirectly. So a query for tourist attractions would have found the Potala Palace but not Notre Dame. This problem has been now fixed in Wikidata. However, one could expect that similar problems exist.

Moreover, the class *tourist attraction* was a subclass of *geographic object* which was an instance of the class *geometric concept* which, in turn, had superclass *mathematical concept*. As a consequence, a query for mathematical concepts returned entities like *tensor*, *polynomial*, *differential equation*... and the *Potala Palace* as answers.

Contribution

In this chapter, we describe the new YAGO version, YAGO 4, which aims to combine the best of the two worlds: It collects the facts about instances from Wikidata, but it forces them into a rigorous type hierarchy with semantic constraints. The complex taxonomy of Wikidata is replaced by the simpler and clean taxonomy of schema.org (Guha, Brickley, and Macbeth, 2016). The classes are equipped with SHACL constraints (Knublauch and Kontokostas, 2017) that specify disjointness, applicable relations, and cardinalities. This way, YAGO 4 transfers the rationale of the original YAGO from the combination of Wikipedia and WordNet to the combination of Wikidata and schema.org. The result is a new knowledge base, which is not just large, but also logically consistent, so that OWL-based reasoning is feasible. Hence we call YAGO 4 a "reason-able" knowledge base. The new resource is available at `https://yago-knowledge.org` under a permissive license (Creative Commons Attribution-ShareAlike). YAGO 4 also comes with a browser and a SPARQL endpoint. Figure 5.1 shows an excerpt from the new YAGO in the online browser.

5.2 Related Work

The Linked Open Data cloud contains several dozen general-purpose knowledge bases[3]. YAGO 4 is not intended to replace these knowledge bases, but rather to serve as an addition to this ecosystem with unique characteristics that complement the other players. For example, DBpedia also has a new version that ingests facts from Wikidata (Ismayilov et al., 2018), with a well-designed pipeline that allows harvesting of different knowledge sources (Frey et al., 2019). This new DBpedia

[3] `https://www.lod-cloud.net/`

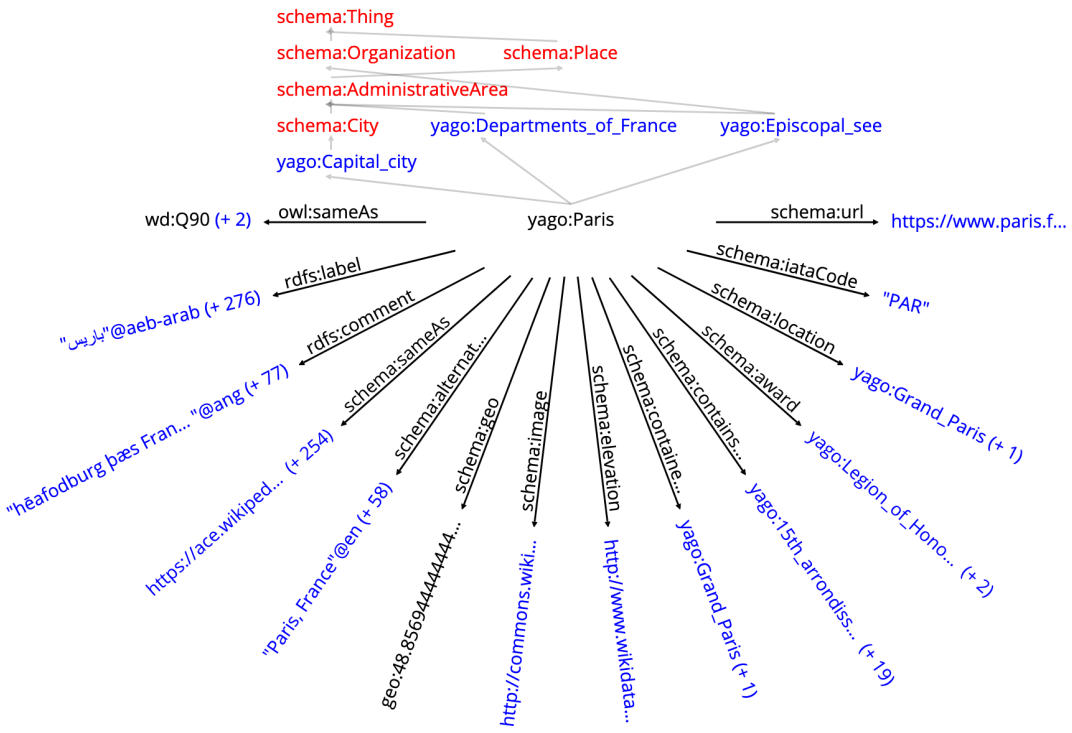

Figure 5.1 – The YAGO 4 Browser. On the website, hovering reveals the full name of abbreviated items, and all red and blue items are clickable.

and YAGO 4 have made different design choices, resulting in different strengths and limitations. Whereas (Frey et al., 2019) key priority has been data mapping and integration, our key priority has been to strengthen the logical rigor of the knowledge base, to support OWL and other reasoners. This is why YAGO 4 builds on schema.org and adds its own constraint system, which is much more elaborate than what DBpedia enforces.

5.3 Design

The construction of the YAGO 4 knowledge base is driven by several design decisions, which we explain and motivate next. The overarching point is to center YAGO 4 around a well-founded notion of classes. For example, a *Person* is defined as a subclass of *Thing*, and has an explicit set of possible relations such as *birthDate*, *affiliation*, etc. Conversely, other relations such as *capitalOf*, *headquarter* or *population* are not applicable to instances of the class *Person*. This overarching principle of semantic consistency unfolds into several design choices.

5.3.1 Concise Taxonomy

Wikidata contains a very detailed taxonomy to which the community contributes by adding *instanceOf* and *subclassOf* statements. However, the resulting class hierarchy is so deep and convoluted that it is not easy to grasp and that browsing it is rather tedious. For example, Paris is an instance of 60 classes, 20 of which are called using terms like "unit", "entity", "subject", or "object". Moreover, the class hierarchy is not stable: any contributor can add or remove *subclassOf* links between any two classes. Potentially, this could lead to millions of entities being classified differently, just by a single edit. On the other hand, schema.org, the second major input to YAGO 4, has established itself as a reference taxonomy on the Web, beyond its initial aim at helping search engines to index web pages. It is stable, well maintained, and changes can only be made by agreement in the W3C Schema.org Community Group[4]. At the same time, schema.org does not provide fine-grained classes such as "electric cars" or "villages" – which only Wikidata has. Schema.org also does not have any biochemical classes (such as proteins etc.).

We address the latter problem by using *Bioschemas* (Gray, Goble, and Jimenez, 2017)[5]. This project extends schema.org in the field of the life sciences – a field that is not covered in schema.org, and that is very prominent in Wikidata. We manually merged 6 Bioschemas classes into schema.org, referring to the merged taxonomy as the "schema.org taxonomy" for simplicity.

[4] `https://www.w3.org/community/schemaorg/`
[5] `https://bioschemas.org`

To achieve the stability of schema.org while preserving the fine-grained classes of Wikidata, we found the following solution: The top-level taxonomy of YAGO 4 is taken from the schema.org taxonomy, and the leaf-level classes are taken from Wikidata. For this purpose, we manually mapped 241 classes of schema.org to Wikidata classes. Classes of schema.org that could not be mapped, mostly shopping-related or social-media classes such as `schema:LikeAction`, were removed. We encode this information in a declarative manner, by using the new relation `yago:fromClass`, as shown here:

```
schema:Person yago:fromClass wd:Q215627
```

We keep the meaning of the schema.org classes. For example, `schema:Person` is defined to cover also fictional people, so we did the same in YAGO 4.

With these inputs, the YAGO 4 taxonomy is then constructed as follows: For each instance in Wikidata, we consider each possible path in the Wikidata taxonomy to classes mapped to schema.org. If the first class on the path has an English Wikipedia article, we include it in YAGO 4. The rationale is that only classes with an English Wikipedia article are of sufficient interest for a wider audience and use cases. We then continue the path to the root in the Wikidata taxonomy, discarding all classes on the way, until we hit a class that has been mapped to schema.org. We continue our path to the root (`schema:Thing`) in the schema.org taxonomy, adding all classes on the way to YAGO 4.

We discard all Wikidata classes that have less than 10 direct instances. This threshold serves to ignore classes that have little value in use cases or are rather exotic. We further remove subclasses of a small list of meta-level Wikidata classes. Finally, we drop subclasses of pairs of classes for which we enforce disjointness constraints. These design choices allow us to model villages and cars, while significantly reducing the size of the taxonomy. From the 2.4M original Wikidata classes, we kept only 10k classes, shrinking the taxonomy by 99.6%. We also discard 11M instances (14%) – two-thirds of which (7.5M) are Wikipedia-specific meta-entities (disambiguation page, category, wikitext template, etc.). Our strategy capitalizes on the stable backbone of schema.org while being able to augment YAGO 4 with new data coming from Wikidata.

5.3.2 Legible Entities and Relations

In the following we use the usual RDF notation for IRI prefixes. For example `yago:` stands for `http://yago-knowledge.org/resource/` and `schema:` for `http://schema.org/`.

YAGO 4 is stored in the RDF format. Unlike Wikidata, we chose to give human-readable URIs to all entities, in order to make the knowledge base more accessible for interactive use. If an entity has a Wikipedia page (which we know

because Wikidata links it to Wikipedia), we take the Wikipedia title as the entity name. Otherwise, we concatenate the English label of the entity with its Wikidata identifier (e.g., `Bischmisheim_Q866094`). Our previous study, Pellissier Tanon and Kaffee, 2018, suggests that the Wikidata labels are fairly stable, leading to fairly stable YAGO URIs. If the entity has no English label, we stay with the Wikidata identifier. Using languages other than English for fallback would have made the URIs less stable: this is because the addition of labels in other languages (especially English) is frequent in Wikidata, and we expect it to be more frequent than changes in the English labels. We make the necessary changes to arrive at a valid local IRI name, and add the IRI prefix of YAGO, `http://yago-knowledge.org/resource/`. This gives the vast majority of entities human-readable names, without introducing duplicates or ambiguity. Note that our choices mean that our URIs may not be fully stable, and may change if Wikidata labels change. This choice is because we preferred to focus on readability rather than stability: Wikidata IRIs are better suited to be used as a permanent link target, and we provide a mapping between Wikidata and YAGO, that can help alleviate any issues with the instability of our URIs.

Wikidata has a very rich set of relations, but many of these have only very few facts. Indeed 61% of them have less than 1000 facts and 85% of them less than 10k. For YAGO 4, we chose to follow the successful model of previous YAGO versions, which only kept a limited number of possible relations for each class. We chose the relations from schema.org, which are each attached to a class. While these relations are conservative in coverage, they have emerged as a useful reference. We mapped 116 of these relations manually to the relations of Wikidata. We store this information similarly to the class mapping by using the new relation, `yago:fromProperty`, as shown here:

```
schema:birthPlace yago:fromProperty wdt:P569
```

The pipeline for knowledge base construction implements these mappings (c.f. Section 5.4.1). This process discards around 7k relations from Wikidata, leaving only the 116 mapped relations. As a by-product, it gives human-readable names to all relations. Example relations are `schema:birthPlace` `schema:founder`, and `schema:containedInPlace`. We use RDF and RDFS relations whenever possible, including `rdfs:label` and `rdfs:comment` instead of `schema:name` and `schema:description`. For example, the fact `wd:Q42` `wdt:P31` `wd:Q5` from Wikidata becomes

```
yago:Douglas_Adams rdf:type schema:Person
```

5.3.3 Well-Typed Values

YAGO 4 has not just well-typed entities, but also well-typed literals. For this purpose, we translate the data values of Wikidata to RDF terms. External URIs are converted into `xsd:anyURI` literals after normalizing them.[6] We chose to keep external URIs as literals and not as entities, because we do not make any statements about URIs. Time values are converted to `xsd:dateTime`, `xsd:date`, `xsd:gYearMonth` or `xsd:gYear`, depending on the time precision. We discard the other time values whose precision could not be mapped to an XML schema type. Globe coordinates are mapped to `schema:GeoCoordinates` resources. Quantities are mapped to `schema:QuantitativeValue` resources (keeping the unit and precision). If there is no unit and an empty precision range, we map to `xsd:integer` where possible. If the unit is a duration unit (minutes, seconds...) and the precision range is empty, we map to `xsd:duration`. In this way, the vast majority of values are migrated to standard RDF typed literals.

5.3.4 Semantic Constraints

YAGO 4 has hand-crafted semantic constraints that not just keep the data clean, but also allow logical reasoning on the data. We model constraints in the W3C standards SHACL (Knublauch and Kontokostas, 2017) and OWL. SHACL is a language to validate RDF graphs. It allows expressing constraints on the graph using shapes. There are two kinds of shapes, *node shapes* and *property shapes*. *Node shapes* are applied to a set of nodes. For example, the following RDF triples declare a node shape `yago:PersonShape` that should be enforced on all instances of `schema:Person`:

```
yago:PersonShape a sh:NodeShape
yago:PersonShape sh:targetClass schema:Person
```

After declaring node shapes, it is possible to apply shapes on facts using *property shapes*. These property shapes are applied to the outgoing edges of the nodes on which a node shape is enforced. For example, the following RDF triples declare a property shape `yago:birthDateShape` which should be enforced on all facts with a `schema:birthDate` property and a subject for which `yago:PersonShape` is enforced:

```
yago:PersonShape sh:property yago:birthDateShape
yago:birthDateShape sh:path schema:birthDate
```

[6] We follow the normalization suggested by RFC 2986 Section 6.2.

It is possible to state various constraints on the property shape (range, cardinality...). They are going to be described in the following paragraphs.

YAGO 4 currently has the following constraints:

Disjointness

We specify 6 major top-level classes: `schema:Event`, `schema:Organization`, `schema:Person`, `schema:Place`, `schema:CreativeWork`, and `bioschemas:BioChemicalEntity`. With the exception of `schema:Organization`/`schema:Place`, these are pairwise disjoint; so that these classes cannot have any instances in common. We use OWL to express, for example:

```
schema:Person owl:disjointWith schema:CreativeWork
```

Note that organizations are not disjoint from places, because many organizations are also located somewhere.

Domain and Range

Each relation comes with a domain and range constraint, meaning that a relation such as *birthPlace* can apply only to a person and a place. RDFS can specify the domain and range of relations by help of the predicates `rdfs:domain` and `rdfs:range`, but our constraints have a different semantics: If a knowledge base contained the fact ⟨`London`, `schema:birthPlace`, `Paris`⟩, then the statement ⟨`schema:birthPlace`, `rdfs:domain`, `schema:Person`⟩ would simply deduce that London must be a person. In contrast, our constraints would flag the knowledge base as inconsistent. We use SHACL to express these constraints. For example, these triples declare that the values of the `schema:birthPlace` property should comply with the `yago:PlaceShape` shape that covers instances of `schema:Place`:

```
yago:PersonShape sh:property yago:birthPlaceProperty
yago:birthPlaceShape sh:path schema:birthPlace
yago:birthPlaceShape sh:node yago:PlaceShape
yago:PlaceShape a sh:NodeShape
yago:PlaceShape sh:targetClass schema:Place
```

The same property can be used to describe entities of different classes. For example `schema:telephone` can be used to describe both persons and organizations. In this case, the same property is going to be in the shapes of several classes. The domain of the property is then the union of all these classes.

In the same spirit, we also support disjunction in property ranges. For example, the range of `schema:author` is `schema:Person` union `schema:Organization`.

Following the same argument, the range of the `schema:birthDate` property is the union of datatypes `xsd:dateTime`, `xsd:date`, `xsd:gYearMonth` and `xsd:gYear` to allow different calendar value precisions. Our range constraints also include the validation of `xsd:string` literals via regular expressions, as in this example:

```
yago:PersonShape sh:property yago:telephoneShape
yago:telephoneShape sh:path schema:telephone
yago:telephoneShape sh:datatype xsd:string
yago:telephoneShape sh:pattern "+\d{1,3} ..."
```

Functional Constraints

A functional constraint says that a relation can have at most one object for a subject. Several of our relations are functional, e.g., *birthPlace* or *gender*. Again, we use SHACL:

```
yago:PersonShape sh:property yago:birthPlaceShape
yago:birthPlaceShape sh:maxCount "1"^^xsd:integer
```

Cardinality Constraints

Going beyond functional constraints, we can also specify the maximal number of objects in general. For example, people can have only two parents in YAGO 4. We use again the SHACL `sh:maxCount` property.

Enforcement of Constraints

YAGO 4 assumes that, for every class, the only allowed properties are the ones explicitly allowed by the domain constraints. This amounts to interpreting the SHACL constraints under a "closed world assumption". The constraints are automatically enforced during the construction of the knowledge base (see Section 5.4.1), and so the data of YAGO 4 satisfies all constraints. Overall, the enforcement of constraints leads to the removal of 132M facts from Wikidata (i.e., 28% of all the facts). Since the constraints are enforced when constructing the knowledge base, we can then add the deductive `rdfs:domain` and `rdfs:range` facts to YAGO 4 and be sure that they will not be used to deduce new facts that violate the constraints.

The ontology automatically generated from the constraints declaration uses the OWL 2 axioms *DisjointClasses*, *ObjectPropertyDomain*, *DataPropertyDomain*, *ObjectPropertyRange*, *DataPropertyRange*, *ObjectUnionOf*, *FunctionalDataProperty*, *FunctionalObjectProperty*, and falls into the OWL DL flavor. Statistics about the mapping and constraints are shown in Table 5.1.

Table 5.1 – Schema and mapping statistics

Item	Number
Schema.org classes	235
Bioschemas.org classes	6
Object properties	100
Datatype properties	41
Node shapes	49
Property shapes	217
Domain constraints	217
Object range constraints	132
Datatype range constraints	57
Regex constraints	21
Disjoint constraints	18

5.3.5 Annotations for Temporal Scope

Following previous YAGO versions, YAGO 4 also attaches temporal information to its facts. We harvest these from the Wikidata qualifier system, which annotates facts with their validity time, provenance, and other meta information. We express the temporal scopes of facts by the relations `schema:startDate` and `schema:endDate`. Instead of relying on a custom format for these annotations, we made use of the RDF* model proposal (Hartig, 2017), which has received good traction in recent years. The RDF* model allows RDF triples to be subjects and objects of other triples. These "inner triples" are often written using the $\ll s\ p\ o \gg$ notation. For example, we state that Douglas Adams lived in Santa Barbara until 2001 as follows:

`≪ Douglas_Adams schema:homeLocation Santa_Barbara ≫ schema:endDate 2001`

This asserts that the fact encoded by the triple `Douglas_Adams schema:homeLocation Santa_Barbara` was true until 2001.

In terms of semantics, a subtle point is that we must distinguish between two existing semantics of RDF*: the Property Graph (PG) semantics and the Separate Assertions (SA) mode[7]. The property graph semantics asserts the inner triples whereas the separate assertions mode does not. For instance, in the previous example, the PG semantics would imply that Douglas Adams still lives in Santa Barbara, which is not what we mean. By contrast, the SA mode would imply that

[7]These semantics names are taken from ongoing discussions about the formalization of the RDF* specification: `https://lists.w3.org/Archives/Public/public-rdf-star/2019Aug/0001.html`

Douglas Adams only lived in Santa Barbara until 2001, without indicating where he currently lives. For this reason, in the design of YAGO4, we use the SA mode of RDF* globally.

5.4 Knowledge Base

5.4.1 Construction

We have designed a system that builds YAGO 4 automatically from (1) a Wikidata dump and (2) the SHACL shape definitions of Section 5.3. We keep only the so-called "truthy" Wikidata statements[8], i.e., for each subject and predicate we keep only the statements with the "best" rank (a.k.a. "preferred" if a statement with such rank exists, "normal" if not).

The knowledge base building system constructs the class hierarchy, the entities, and the facts as outlined in Section 5.3. Its main purpose is then to enforce the constraints (Section 5.3.4). If a resource is an instance of disjoint classes, we drop the two `rdf:type` relations leading to this conflict. We drop all instances that are not instances of any class. We enforce domain, range, and regular-expression constraints by pruning all candidate facts that would violate a constraint. Finally, we check the cardinality constraints, removing all objects if there are too many for a given subject. For example, if a functional property has multiple values with the same Wikidata rank, they are all removed from YAGO.

Our system is implemented in the Rust programming language[9], using the `Iterator` infrastructure to ingest and output data streams. We use the already existing stream operators, which resemble those of relational algebra (map/project, filter, flat map, collect/materialize into a hash structure). We also implemented new operators particularly for YAGO 4 (stream-hash join, stream-hash anti join, group-by, and transitive closure). For example, the `owl:sameAs` links between YAGO 4 and Freebase can be extracted from the values of the Wikidata property `wdt:P646` ("Freebase Id") by the algebraic operator plan presented in Figure 5.2. In this figure, π is the projection operator, σ the selection, $\bowtie$ the inner join, *Wikidata* the table of all Wikidata triples (s, p, o), and *WikidataToYagoMapping* the mapping between Wikidata and YAGO instances (*wd*, *yago*). To avoid reading the full Wikidata N-Triples dump each time, we first load the Wikidata dump into the RocksDB key-value store to index its content[10]. This index allows for efficiently selecting triples based on a predicate or a (predicate, subject) tuple,

[8] `https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format#Truthy_statements`

[9] `https://www.rust-lang.org/`

[10] `https://rocksdb.org/`

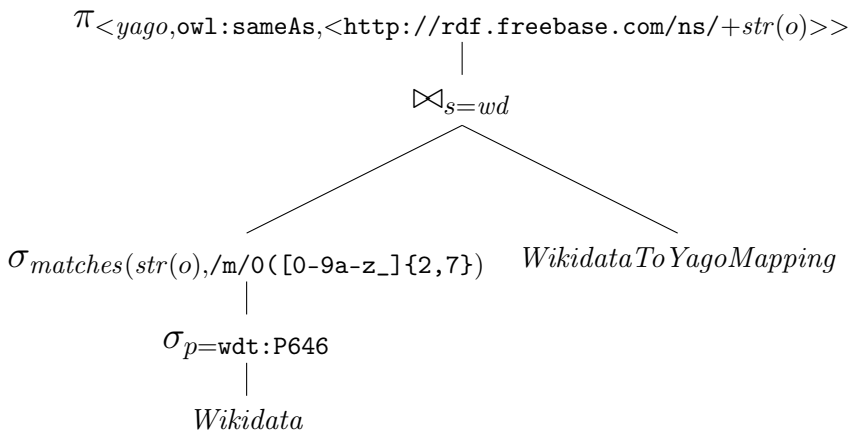

Figure 5.2 – Algebric operator plan generating the `owl:sameAs` relations between YAGO and Freebase using the values of the Wikidata property `wdt:P646` ("Freebase Id").

and getting back a stream of triples from the database.

The advantage of having operator plans in Rust is that our system can specify the query plan (i.e., the best execution order). We also get benefit from a programming language with a compiler able to carry performance optimizations, generating highly efficient native code. After having loaded the data into RocksDB, a process that takes around a day, our execution plan generates the Wikipedia-flavored YAGO 4 (see below) in two hours on a commodity server.

We ran our system on a dump of 78M Wikidata items. 8M of these are entities related to Wikimedia websites, such as categories. From the 474M Wikidata facts whose properties have been mapped to schema.org, we filtered out 89M of them because of the domain constraints and 42M more because of the range and regex constraints. The cardinality constraints lead to the removal of an extra 0.6M facts.

5.4.2 Data

YAGO 4 is made available in three "flavors":

- **Full:** This flavor uses all data from Wikidata, resulting in a very large knowledge base.

- **Wikipedia:** This smaller flavor of YAGO 4 contains only the instances that have a Wikipedia article (in any language).

- **English Wikipedia:** This is an additional restriction of the Wikipedia flavor, containing only instances that have an English Wikipedia article.

All three flavors of YAGO 4 are built in the same way, and have the same schema, with 116 properties and the same taxonomy of 140 top-level classes from schema.org and bioschemas.org, and the same subset of Wikidata classes. Table 5.2 shows statistics for the three YAGO 4 variants, generated from the Wikidata N-Triples dump of November 25, 2019.
Each flavor of YAGO 4 is split into the following files:

- **Taxonomy:** The full taxonomy of classes.
- **Full-types:** All `rdf:type` relations.
- **Simple-types:** The `rdf:type` relations to schema.org classes (i.e., without the classes generated from Wikidata).
- **Labels:** All entity labels (`rdfs:label`, `schema:alternateName`, and `rdfs:comment`).
- **Facts:** The facts that are not labels.
- **Annotations:** The fact annotations encoded in RDF* (Hartig, 2017).
- **SameAs:** The `owl:sameAs` links to Wikidata, DBpedia, and Freebase and the `schema:sameAs` links to all the Wikipedias.
- **Schema:** The schema.org classes and properties, in OWL 2 DL.
- **Shapes:** The SHACL constraints used to generate YAGO 4.

Each file is a compressed N-Triples file, so that standard tools can directly ingest the data.

5.4.3 Access

Web Page. The YAGO 4 knowledge base is available on the website `https://yago-knowledge.org`. The Web page offers an introduction to YAGO, documentation ("Getting started"), and a list of publications and contributors. The Web page also has a schema diagram that lists all top-level classes with their associated relations and constraints.

Table 5.2 – Size statistics for YAGO 4 in the flavors Full (F), Wikipedia (W), and English Wikipedia (E), Wikidata, DBpedia (obtained from the DBPedia SPARQL server on 2020-03-04) and Yago 3.0.2.

	YAGO F	YAGO W	YAGO E	Wikidata	DBpedia	YAGO 3
Classes	10124	10124	10124	2.4M	484k	475k
Classes from Wikidata	9883	9883	9883	2.4M	222	0
Relations	143	143	143	7k	3k	102
Individuals	67M	15M	5M	78M	5M	5M
Labels	303M	137M	66M	371M	22M	32M
Descriptions	1399M	139M	50M	2146M	12M	69k
Aliases	68M	21M	14M	71M	0	9M
`rdf:type`	70M	16M	5M	77M	114M	17M
Facts	343M	48M	20M	974M	131M	10M
Avg. # facts per entity	5.1	3.2	4	12.5	26	2
sameAs to Wikidata	67M	15M	5M	N.A.	816k	0
sameAs to DBpedia	5M	5M	5M	0	N.A.	4M
sameAs to Freebase	1M	1M	1M	1M	157k	0
sameAs to Wikipedia	43M	43M	26M	66M	13M	5M
Fact annotations	2.5M	2.2M	1.7M	220M	0	3M
Dump size	60GB	7GB	3GB	127GB	99GB	2GB

License. The entire YAGO 4 knowledge base, as well as all previous versions and the logo, can be downloaded from the Web page. YAGO 4 is available under a Creative Commons Attribution-ShareAlike License. The reason for this choice is that, while Wikidata is in the public domain, schema.org is under a Creative Commons Attribution-ShareAlike License.[11]

Source Code. We have released the source code for constructing YAGO 4 on GitHub at `https://github.com/yago-naga/yago4` under the GNU GPL v3+ license.

SPARQL Endpoint. YAGO 4 comes with a responsive SPARQL endpoint, which can be used as an API or interactively. The URL of the endpoint is `https://yago-knowledge.org/sparql/query`. The YAGO URIs are also all dereferenceable, thus complying with the Semantic Web best practices.

Browser. YAGO 4 comes with a graphical knowledge base browser, with an example shown in Figure 5.1. For each entity, the browser visualizes the outgoing relationships in a star-shape around the entity. Above the entity, the browser shows the hierarchy of all classes of which the entity is a (transitive) instance, including those with multiple inheritances. If an entity has more than one object for a given relation, a relation-specific screen shows all objects of that relation for the entity. For size reasons, the browser shows only the Wikipedia flavor of YAGO.

Applications. YAGO has already been used in quite many projects (Rebele, Suchanek, et al., 2016), including question answering, entity recognition, and semantic text analysis. We believe that the new version of YAGO opens up the door to an entire array of new applications because it is possible to perform logical reasoning on YAGO 4. Not only is the knowledge base equipped with semantic constraints, but it is also provably consistent. We have checked the "English Wikipedia" flavor of YAGO 4 with the OWL 2 DL reasoner HermiT (Glimm et al., 2014), proving its logical consistency.[12] This makes it possible to perform advanced kinds of logical inference on YAGO 4.

5.5 Conclusion

This chapter has presented YAGO 4, the newest version of the YAGO knowledge base. The unique characteristics of YAGO 4 are to combine the wealth of facts

[11] `https://schema.org/docs/terms.html`

[12] HermiT was unable to load the "Full" flavor due to a memory overflow, but it contains the same taxonomy and the same constraints as the "English Wikipedia" flavor.

from Wikidata with the clean and human-readable taxonomy from schema.org, together with semantic constraints that enforce logical consistency. This way, the resulting knowledge base can be processed with OWL reasoners and is also more user-friendly for browsing and question answering.

In the future, it would be nice to provide a tool alongside YAGO 4 that would be able to give the original fact(s) in Wikidata which led to a given YAGO fact. This tool would allow YAGO 4 users to know which fact to edit in Wikidata, or whether the conversion pipeline is wrong, in case they found a problem in YAGO. Adding reasoning capabilities to the public SPARQL endpoint would also be interesting to make use of the consistency of YAGO 4.

However, the YAGO 4 process of removing all offending facts when fixing semantic constraint violations lead to losing 27% of facts with respect to Wikidata. Therefore, the next chapter will present a better way to fix these violations.

Part III

Enforcing Constraints Dynamically

Chapter 6

Learning How to Correct a Knowledge Base from the Edit History

Most of the work presented in this section has been done with Camille Bourgaux and Fabian Suchanek and has been published at WWW.[1]

> Thomas Pellissier Tanon, Camille Bourgaux, and Fabian M. Suchanek. "Learning How to Correct a Knowledge Base from the Edit History". Full paper at WWW 2019. `https://doi.org/10.1145/3308558.3313584`

The neural network approach *Bass* (Section 6.7) and the evaluation against the test dataset (Section 6.8) are new and are currently under review at ISWC.

> Thomas Pellissier Tanon and Fabian M. Suchanek. "Neural Knowledge Base Repairs". Under review at ISWC 2020.

6.1 Introduction

In the previous chapter, we have presented a knowledge base that statically enforces constraints. This is done by removing all facts violating these constraints. However, this method leads to an important amount of fact loss (27%). Therefore, we now focus on a better way to fix these violations.

[1]The formalization of constraints in description logics presented in the paper is omitted here to only focus on my personal contribution. For this reason, some parts of this thesis (in particular Section 6.3) differ significantly from the conference version of this work.

In this chapter, we aim at learning how to repair constraint violations. Our goal is to help a knowledge base editor by suggesting how to clean the data locally (providing a solution to a particular constraint violation) or globally (providing rules that can be automatically applied to all constraint violations of a given form once validated by the editor). To do that, we take advantage of the *edit history* of the knowledge base. We propose two approaches to learn correction suggestions. The first one mines *correction rules* that express how different kinds of constraint violations are usually solved. The second one trains a neural network that predicts constraint violation corrections. To the best of our knowledge, this is the first work that builds on past user corrections to infer possible new ones. We validate our framework experimentally on Wikidata, for which the whole edit history of more than 700 million edits is available. Our experiments show substantial improvements over baselines. More concretely, our contributions are as follows:

- a formal definition of the problem of correction rule mining,
- a correction rule mining algorithm, together with an implementation for Wikidata, CorHist,[2]
- a deep neural network algorithm, together with an implementation for Wikidata, Bass,[3]
- a suggestion tool for users to correct data based on our mined correction rules,[4]
- an experimental evaluation based both on the prediction of the corrections in the history and on user validation of the suggested local corrections.

6.2 Related Work

We start with a brief discussion of works relevant to our problem along three axes: constraints for knowledge bases, knowledge base cleaning, and rule learning.

Constraints

Constraints have long been used in databases and knowledge bases to express rules that the data should follow. Databases typically operate under the closed world

[2]Available at `https://github.com/Tpt/corhist`.

[3]This implementation is not publicly available yet because of the double-blind requirement of the peer-review process of ISWC 2020.

[4]Available at `https://tools.wmflabs.org/wikidata-game/distributed/#game=43`.

assumption, where missing facts are considered to be false. This allows for "completeness" constraints such as tuple generating dependencies. knowledge bases, in contrast, operate under the open-world assumption, where missing facts are not necessarily false. They thus classically have only "correctness" constraints, such as disjointness or functionality axioms (corresponding to special cases of denial constraints and equality generating dependencies in databases).

To express also completeness constraints, several works propose to use description logics, with varying semantics (Motik, Horrocks, and Ulrike Sattler, 2009; Tao et al., 2010). Another possibility is to use queries that should or should not hold as constraints (see e.g., Kontokostas et al., 2014 for methods for writing constraint queries in SPARQL). Other approaches define constraint languages to specify conditions for RDF graph (Cyganiak et al., 2014) validation, such as SHACL (Knublauch and Kontokostas, 2017) or ShEx (Boneva, Gayo, and Prud'hommeaux, 2017). It has been argued in Patel-Schneider, 2015 that description logics under the closed world assumption are also suitable for constraint checking in RDF, which can then be implemented with SPARQL queries. In our work, we depart from this path by using association rules as constraints.

Contrary to the above works, we do not aim at *expressing* constraints, but at *repairing* their violations. The correction rules we learn for this purpose are similar in spirit to active integrity constraints (Flesca, Greco, and Zumpano, 2004), which specify for each constraint a set of possible repair actions. This type of constraint has recently been applied to description logic knowledge bases as well (Rantsoudis, Feuillade, and Herzig, 2017). Conditioned active integrity constraints add conditions for choosing among the possible actions, and we propose, in a similar spirit, to take into account the context of the constraint violation to correct it. Different from these existing works, our goal is to mine correction rules automatically from the edit history of the knowledge base.

Knowledge base cleaning

Several recent approaches have dealt with the interactive cleaning of knowledge bases. The proposed methods detect when a constraint is violated, compute the responsible facts, and then interact with the user to find out how to update the knowledge base. The goal is then to minimize the number of questions the user has to answer. This is done in various ways, which include taking into account the dependencies among the facts to check, or the interaction between several constraint violations to define heuristics to choose the best question to ask the user (Bergman et al., 2015; Bienvenu, Bourgaux, and Goasdoué, 2016; Assadi, Milo, and Novgorodov, 2018; Arioua and Bonifati, 2018).

Other approaches to improve the quality of a knowledge base rely on statistics, clustering, or structural aspects of the knowledge bases. Chen et al., 2020 uses

knowledge base embeddings, lexical distance, and constraint-based refinements to predict corrections. Our work goes beyond the state of the current knowledge base and learns from the edit history instead. Paulheim and Bizer, 2014 uses statistics to add missing types to the knowledge base and to detect wrong statements. Liang et al., 2017 exploits the observation that cycles in the knowledge base often contain wrong "isA" relations. G. d. Melo, 2013 cleans "sameAs" relations based on the shape of the existing identity facts and on differences in the graph. Lertvittayakumjorn, Kertkeidkachorn, and Ichise, 2017 uses graph and text distances to look for values for a given subject and predicate. A. Melo and Paulheim, 2017 fixes the subject or object of existing relations based on type relations and string matching. Again other approaches, like Acosta et al., 2018, use crowdsourcing to detect Linked Data quality issues. We refer the reader to Section 7.2 of Acosta et al., 2018 for a recent overview of approaches for data quality assessment.

Our method also exploits knowledge base constraints. However, it differs from the above in that it *learns the corrections automatically* from the edit history. It thus taps a source of knowledge that has so far not been exploited.

Some works in completely different domains also use the idea of learning repairs from past corrections. For example, Bader et al., 2019 learns how to fix errors in source code based on previous error corrections.

Rule learning

Mining logical rules by finding correlations in a dataset is a well-established research topic. In particular, learning patterns in the data can be used for completing knowledge bases as presented in Chapter 3. An algorithm for learning conjunctive patterns from a knowledge base enriched with a set of rules is described in Józefowska, Lawrynowicz, and Lukaszewski, 2010. Methods similar to association rule mining have also been used for the induction of new ontological rules from a knowledge base (Sazonau, Uli Sattler, and Brown, 2015). A more recent trend is to use embedding-based models for knowledge base completion. A comparison between these models and usual rule learning approaches is reported in Meilicke et al., 2018 and significant recent works in this area include Ho et al., 2018; B. Yang et al., 2014; F. Yang, Z. Yang, and Cohen, 2017.

Graph neural networks

Neural networks are often applied to knowledge base related tasks (Wu et al., 2019). However, there does not seem to be any work in the literature that uses neural networks to predict repairs in knowledge bases. Some works use neural networks to reason on top of knowledge bases (Hohenecker and Lukasiewicz, 2018). Several other works tackle the knowledge graph completion task by mining rules (Ho et al.,

2018) or by link prediction. Our approach takes inspiration from these methods, but ultimately tackles a different problem: We do not want to predict links, but the correction of the violation of a constraint. We refer to Wu et al., 2019 for a detailed survey about graph neural networks.

In this chapter, we use a vanilla rule mining algorithm inspired by Galárraga, Teflioudi, et al., 2013 and a combination of usual neural network components. Our contribution is not the rule mining and the neural network per se, but the application of rule mining and deep learning to the edit history of a knowledge base to mine correction rules. This avenue has, to the best of our knowledge, never been investigated.

6.3 Constraints

This section defines the constraints that can be imposed on a knowledge base. In this chapter we make the unique name assumption, i.e., we assume that each entity is represented by at most one individual (Definition 2.9 Page 20).

Definition 6.1 (Constraint)**.** A constraint $\Gamma(\vec{x})$ is a rule under the form $b(\vec{x}) \rightarrow h(\vec{x})$ where $\vec{x}$ is a sequence of variables, b a propositional query (c.f. Definition 2.13 Page 22) and h a union of conjunctive queries (c.f. Definition 2.12 Page 22).

Example 6.2. As a running example, we consider the following knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and set of constraints $\mathcal{C}$ inspired by Wikidata. We assume that the description logic subset used is the one that covers only concept inclusions.

Our TBox $\mathcal{T}$ expresses that human beings and deities are persons. Our ABox $\mathcal{A}$ provides information on several individuals. Our constraints $\mathcal{C}$ state that there are two possible genders, that those who have a mother or are a mother must be persons or animals, that a mother must have gender female, and that if a has mother b, then b must have child a.

$$\mathcal{T} = \{\mathit{Human} \sqsubseteq \mathit{Person}, \mathit{Deity} \sqsubseteq \mathit{Person}\}$$

$$\begin{aligned}\mathcal{A} = \{\ & \mathit{Deity}(\texttt{Zeus}),\ \mathit{Deity}(\texttt{Rhea}),\ \mathit{hasGender}(\texttt{Zeus}, \texttt{masculine}),\\ & \mathit{hasGender}(\texttt{Rhea}, \texttt{female}),\ \mathit{hasMother}(\texttt{Zeus}, \texttt{Rhea}),\\ & \mathit{hasChild}(\texttt{Rhea}, \texttt{Zeus}),\ \mathit{Human}(\texttt{Spinoza}),\\ & \mathit{hasMother}(\texttt{Spinoza}, \texttt{Marques})\ \}\end{aligned}$$

$$\begin{aligned}\mathcal{C} = \{\, &\Gamma_0(x) : \exists y\ \mathit{hasGender}(y, x) \rightarrow x = \texttt{male} \vee x = \texttt{female},\\ &\Gamma_1(x) : \exists y\ \mathit{hasMother}(x, y) \rightarrow \mathit{Person}(x) \vee \mathit{Animal}(x),\\ &\Gamma_2(x) : \exists y\ \mathit{hasMother}(y, x) \rightarrow \mathit{Person}(x) \vee \mathit{Animal}(x),\\ &\Gamma_3(x) : \exists y\ \mathit{hasMother}(y, x) \rightarrow \mathit{hasGender}(x, \texttt{female}),\\ &\Gamma_4(x, y) : \mathit{hasMother}(x, y) \rightarrow \mathit{hasChild}(y, x)\ \}\end{aligned}$$

We say that a knowledge base $\mathcal{K}$ *satisfies a constraint* $\Gamma \in \mathcal{C}$ where $\Gamma(\vec{x}) : b(\vec{x}) \rightarrow h(\vec{x})$ if for all x, $\mathcal{K} \models b(\vec{x}) \implies \mathcal{K} \models h(\vec{x})$. Otherwise, $\mathcal{K}$ *violates* Γ.

Example 6.3. In our running example, the knowledge base $\mathcal{K}$ satisfies Γ_1 since $\{x \mid \mathcal{K} \models \exists y\ \mathit{hasMother}(x, y)\} = \{\texttt{Zeus}, \texttt{Spinoza}\}$ and $\mathcal{K} \models \mathit{Person}(\texttt{Zeus})$ and $\mathcal{K} \models \mathit{Person}(\texttt{Spinoza})$. However, it violates Γ_2 because $\mathcal{K} \not\models \mathit{Person}(\texttt{Marques}) \vee \mathit{Animal}(\texttt{Marques})$ while $\mathcal{K} \models \exists y\ \mathit{hasMother}(y, \texttt{Marques})$.

It also violates Γ_3 and Γ_4 for similar reasons.

Finally, $\{\mathit{hasGender}(\texttt{Zeus}, \texttt{masculine})\}$ violates Γ_0 because of the unique name assumption (which enforces that the interpretation of `masculine` differs from those of `male` and `female`). By monotonicity, $\mathcal{K}$ violates Γ_0.

Proposition 6.4. *Every constraint $\Gamma \in \mathcal{C}$ can be rewritten as a set of rules $\Gamma_i(\vec{x}) : b_i(\vec{x}) \rightarrow h_i(\vec{x})$, $i \in \{1, \ldots n, \}$, where $b_i(\vec{x})$ is a conjunctive query and $h_i(\vec{x})$ is a union of conjunctive queries and for every knowledge base $\mathcal{K}$, $\mathcal{K}$ satisfies Γ if, and only if, $\mathcal{K}$ satisfies all Γ_i, $i \in \{1, \ldots n, \}$.*

Proof. Let's assume we have a constraint $\Gamma(\vec{x}) : b(\vec{x}) \rightarrow h(\vec{x})$ where b is a propositional query and h a union of conjunctive queries.

It is possible to rewrite b in disjunctive normal form such that $b(\vec{x}) = \exists \vec{y} \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} b_{i,j}(\vec{x}, \vec{y})$ where $b_{i,j}$ is an optionally negated role, concept or equality assertion. We end up with $\Gamma(\vec{x}) : \bigvee_{i=1}^{n} \exists \vec{y}\ \bigwedge_{j=1}^{m} b_{i,j}(\vec{x}, \vec{y}) \rightarrow h(\vec{x})$. This could be rewritten with an equivalent set of rules $\Gamma_i(\vec{x}) : \exists \vec{y_i}\ \bigwedge_{j=1}^{m} b_{i,j}(\vec{x}, \vec{y_i}) \rightarrow h(\vec{x})$ with $i \in 1, \ldots, n$. Indeed, Γ is satisfied if, and only if, all Γ_i are satisfied.

We could then simplify the rules Γ_i. Γ_i could be written $\Gamma_i(\vec{x}) : \exists \vec{y}\ b(\vec{x}, \vec{y}) \wedge \neg b^n(\vec{x}, \vec{y}) \rightarrow h(\vec{x})$ where $b(\vec{x}, \vec{y})$ is a conjunction of roles, concepts or equality assertions and $b^n(\vec{x}, \vec{y})$ a disjunction of them (we have factorized the negation).

We could then define $\vec{x'}$ to be the concatenation of $\vec{x}$ and $\vec{y}$ and end up with $\Gamma_i(\vec{x'}) = b(\vec{x'}) \wedge \neg b^n(\vec{x'}) \rightarrow h(\vec{x'})$ that is equivalent to $\Gamma_i(\vec{x'}) : b(\vec{x'}) \rightarrow h(\vec{x'}) \vee b^n(\vec{x'})$.

$h(\vec{x'}) \vee b^n(\vec{x'})$ is a union of conjunctive queries. Hence we found a set of queries Γ_i equivalent to Γ and having the expected properties. □

Constraint violations. A *constraint instance* $\Gamma(\vec{a})$ of a constraint $\Gamma(\vec{x})$ is obtained by replacing the variables $\vec{x}$ by the individual names $\vec{a}$ in $\Gamma(\vec{x})$. This notion allows us to define constraint violations:

Definition 6.5. A *violation of a constraint* $\Gamma(\vec{x})$ in $\mathcal{K}$ is a minimal subset $\mathcal{V} \subseteq \mathcal{K}$ such that there exists $\vec{a}$ such that $\mathcal{V}$ violates $\Gamma(\vec{a})$ and $\mathcal{K}$ violates $\Gamma(\vec{a})$.

In this definition, the requirement that $\mathcal{K}$ violates $\Gamma(\vec{a})$ may seem superfluous. Yet, if Γ is $Human(x) \rightarrow \exists y\ hasGender(x, y)$, it may be the case that some $\mathcal{V} \subseteq \mathcal{K}$ violates $\Gamma(\vec{a})$ (e.g., $\{Human(\texttt{Zeus})\}$), while $\mathcal{K}$ satisfies it (e.g., $\{Human(\texttt{Zeus}), hasGender(\texttt{Zeus}, \texttt{male})\}$).

Example 6.6. In our running example, it is easy to see that the subset $\mathcal{V}_0 = \{hasGender(\texttt{Zeus}, \texttt{masculine})\}$ is a violation of Γ_0. Consider now $\mathcal{V} = \{hasMother(\texttt{Spinoza}, \texttt{Marques})\}$. $\mathcal{V}$ is a violation of Γ_2, Γ_3 and Γ_4. Indeed, it violates $\Gamma_2(\texttt{Marques})$, $\Gamma_3(\texttt{Marques})$, and $\Gamma_4(\texttt{Spinoza}, \texttt{Marques})$ and $\mathcal{K}$ does not satisfy any of these constraint instances. However, even if $\mathcal{V}$ violates $\Gamma_1(\texttt{Spinoza})$, $\mathcal{V}$ is not a violation of Γ_1 because $\{Human(\texttt{Spinoza}), Human \sqsubseteq Person\} \subseteq \mathcal{K}$ satisfies the head of $\Gamma_1(\texttt{Spinoza})$.

If a constraint instance $\Gamma(\vec{a})$ of $\Gamma(\vec{x}) : b(\vec{x}) \rightarrow h(\vec{x})$ is violated, $\mathcal{K} \models b(\vec{a})$ and $\mathcal{K} \not\models h(\vec{a})$, so its violations are the minimal subsets of $\mathcal{K}$ responsible for $\mathcal{K} \models b(\vec{a})$.

Connection to justifications. We conclude the section by pointing out that our notion of *violations* is closely related to the notion of *justifications* introduced in previous works in the context of explaining why a given axiom is entailed by an ontology. A *justification* (also known as an explanation, axiom pinpointing, or MinAs) for the entailment of a boolean conjunctive query is a minimal subset of the knowledge base that entails the boolean conjunctive query (Schlobach and Cornet, 2003; Kalyanpur et al., 2007).

Proposition 6.7. *If $\mathcal{K}$ violates $\Gamma(\vec{a}) : b(\vec{a}) \rightarrow h(\vec{a})$, a subset $\mathcal{V} \subseteq \mathcal{K}$ is a violation of $\Gamma(\vec{a})$ if, and only if, $\mathcal{V}$ is a justification of $\mathcal{K} \models b(\vec{a})$.*

Proof. Like we stated earlier, $\mathcal{V} \subseteq \mathcal{K}$ is a violation of $\Gamma(\vec{a})$ if, and only if, $\mathcal{V}$ is a smallest subset $\mathcal{K}$ responsible for $\mathcal{K} \models b(\vec{a})$. This exactly corresponds to the definition of a justification of $\mathcal{K} \models b(\vec{a})$. □

6.4 Corrections

We now turn to the correction of constraint violations.

Solutions. We will make use of atomic modifications of the knowledge base to define solutions to constraint violations.

Definition 6.8 (Atomic modification)**.** An *atomic modification* of a knowledge base $\mathcal{K}$ consists of two sets of triples $(\mathcal{M}^+, \mathcal{M}^-)$ such that $\mathcal{M}^+ \cap \mathcal{K} = \emptyset$ and $\mathcal{M}^- \subseteq \mathcal{K}$. Furthermore, one of the following must hold:

1. $\mathcal{M}^+ = \emptyset$ and $\mathcal{M}^-$ consists of an assertion (we say that $(\mathcal{M}^+, \mathcal{M}^-)$ is a *deletion*), or

2. $\mathcal{M}^- = \emptyset$ and $\mathcal{M}^+$ consists of an assertion (we say that $(\mathcal{M}^+, \mathcal{M}^-)$ is an *addition*), or

3. $\mathcal{M}^-$ consists of an assertion $\langle s, p, o \rangle$ and $\mathcal{M}^+$ consists of an assertion $\langle s', p', o' \rangle$ which differs from $\langle s, p, o \rangle$ by only one element (we say that $(\mathcal{M}^+, \mathcal{M}^-)$ is a *replacement*).

Since the sets $\mathcal{M}^+$ and $\mathcal{M}^-$ contain at most one triple, we slightly abuse the notation and identify the singletons with their elements (e.g., we will denote the addition of $\langle s, p, o \rangle$ simply by $(\langle s, p, o \rangle, \emptyset)$).

During the history of a knowledge base, users can change not just the assertions of the knowledge base, but also the TBox. However, the TBox is typically much smaller and more stable than the ABox. Therefore, the edit history of the TBox is not a rich ground for correction rule mining. It is why we do not consider changes in the TBox as part of the atomic modification definition.

Note that a replacement is equivalent to a sequence of a deletion and an addition. We chose to keep it as an atomic modification because it corresponds to common knowledge base curation tasks, such as correcting an erroneous object for a given subject and predicate, or fixing a predicate misuse. Atomic modifications can be used to solve a constraint violation, as follows:

Definition 6.9 (Solution)**.** A *solution* to a violation $\mathcal{V}$ of a constraint instance $\Gamma(\vec{a})$ of $\Gamma(\vec{x}) : b(\vec{x}) \rightarrow h(\vec{x})$ in $\mathcal{K}$ is an atomic modification $(\mathcal{M}^+, \mathcal{M}^-)$ such that there exists $\mathcal{K}' \subseteq \mathcal{K}$ such that $(\mathcal{V} \cup \mathcal{K}' \cup \mathcal{M}^+) \setminus \mathcal{M}^-$ satisfies $\Gamma(\vec{a})$. We call $(\mathcal{M}^+, \mathcal{M}^-)$ a solution to $\mathcal{V}$ for $\Gamma(\vec{a})$ in $\mathcal{K}$.

Note that $\Gamma(\vec{a})$ can still be violated in $(\mathcal{K} \cup \mathcal{M}^+) \setminus \mathcal{M}^-$ if $\mathcal{K}$ contains other violations of $\Gamma(\vec{a})$ for which $(\mathcal{M}^+, \mathcal{M}^-)$ is not a solution. For example, if $\Gamma(a) : \exists x\ R(a, x) \wedge A(a) \rightarrow \texttt{false}$, and $\mathcal{K} = \{R(a, b), R(a, c), A(a)\}$, the deletion of $R(a, b)$ is a solution to the violation $\{R(a, b), A(a)\}$, but $\{R(a, c), A(a)\}$ still violates $\Gamma(a)$. Note also that every constraint violation has at least one solution, which consists of the deletion of any of its elements. Solutions may also be additions or replacements, as in the following example:

Example 6.10. In our running example, the deletion ($\emptyset$, *hasGender*(`Zeus`, `masculine`)) and the replacement (*hasGender*(`Zeus`, `male`), *hasGender*(`Zeus`, `masculine`)) are two possible solutions to $\mathcal{V}_0$ for Γ_0(`masculine`).

The deletion ($\emptyset$, *hasMother*(`Spinoza`, `Marques`)) is a solution to $\mathcal{V}$ for the three constraint instances Γ_2(`Marques`), Γ_3(`Marques`) and Γ_4(`Spinoza`, `Marques`). The additions (*Human*(`Marques`), $\emptyset$), (*hasGender*(`Marques`, `female`), $\emptyset$) and (*hasChild*(`Marques`, `Spinza`), $\emptyset$) are solutions to $\mathcal{V}$ for respectively Γ_2(`Marques`), Γ_3(`Marques`) and Γ_4(`Spinoza`, `Marques`).

Good solutions. Our goal is to find "good" solutions to constraint violations, i.e., solutions that make the knowledge base as close to the real world as possible. The basic requirement for a "good" solution is that it deletes only erroneous facts and that it adds only true facts. We also prefer replacements over deletions as long as they fulfill this condition. For instance, in our running example, the replacement (*hasGender*(`Zeus`, `male`), *hasGender*(`Zeus`, `masculine`)) is better than the deletion ($\emptyset$, *hasGender*(`Zeus`, `masculine`)), because it corrects erroneous information instead of simply erasing it.

The main difficulty in finding good solutions to constraint violations is that we do not have access to an oracle that knows the validity of all facts. This is the problem that all knowledge base cleaning approaches face (c.f. Section 6.2). Our idea is to exploit the history of the knowledge base modifications to learn how to correct constraint violations.

Definition 6.11 (Edit history)**.** The *edit history* of a knowledge base is a sequence of knowledge bases $(\mathcal{K}_i)_{0\leq i\leq p} = (\mathcal{T}_i \cup \mathcal{A}_i)_{0\leq i\leq p}$ such that $\mathcal{K}_{i+1} = (\mathcal{K}_i \cup \mathcal{M}_i^+) \setminus \mathcal{M}_i^-$, where $(\mathcal{M}_i^+, \mathcal{M}_i^-)$ is an atomic modification.

The edit history allows us to pinpoint how constraint violations have been corrected in the past. In order to avoid learning from vandalism or mistakes, we consider only those corrections that have not been reversed:

Definition 6.12 (Past correction)**.** Let $\mathcal{K}_p$ be the current state of the knowledge base. A *past correction* is a solution $(\mathcal{M}^+, \mathcal{M}^-)$ to a violation $\mathcal{V}$ of a constraint instance $\Gamma(\vec{a})$ in $\mathcal{K}_i$ such that $\mathcal{M}^+ \subseteq \mathcal{B}$, $\mathcal{M}^- \subseteq \mathcal{D}$ where $\mathcal{K}_p = (\mathcal{K}_i \cup \mathcal{B}) \setminus \mathcal{D}$ with $\mathcal{B} \cap \mathcal{D} = \emptyset$.

Intuitively, $(\mathcal{B}, \mathcal{D})$ corresponds to the sequence of additions and deletions that leads from $\mathcal{K}_i$ to the current state of the knowledge base $\mathcal{K}_p$, that contains the solution, and that does not "undo" it.

Relevant past corrections. We are interested in learning solutions that correct constraint violations in the *current* knowledge base $\mathcal{K}_p$. We thus consider only

those past corrections that would have been corrections also under the current TBox. For example, assume that the TBox contained $C \sqsubseteq B$. Assume that $C(a)$ was added to correct a violation of the constraint $A(x) \rightarrow B(x)$. If, in the meantime, the inclusion $C \sqsubseteq B$ has been removed, we do not want to learn from this past correction. The following definition formalizes these requirements.

Definition 6.13 (Relevant Past Correction)**.** A *relevant past correction* ($\mathcal{M}^+$, $\mathcal{M}^-$) to a violation $\mathcal{V}$ of a constraint instance $\Gamma(\vec{a})$ in $\mathcal{K}_i$ is a past correction such that $(\mathcal{V} \cap \mathcal{A}_i) \cup \mathcal{T}_p$ contains a violation $\mathcal{V}'$ of $\Gamma(\vec{a})$ such that ($\mathcal{M}^+$, $\mathcal{M}^-$) is also a solution to $\mathcal{V}'$ in $\mathcal{A}_i \cup \mathcal{T}_p$.

We will now see how we can use the relevant past corrections to mine correction rules.

6.5 Extraction of the Relevant Past Corrections

In this section, we propose an approach to extract past corrections from the knowledge base history that can be used as ground truth to train constraint correction predictors.

We assume here that the description logic used allows rewriting a conjunctive query $q(\vec{x})$ with respect to $\mathcal{T}$ into a union of conjunctive queries $q'(\vec{x})$ such that for every ABox $\mathcal{A}$, answering $q(\vec{x})$ over $(\mathcal{T}, \mathcal{A})$ amounts to answering $q'(\vec{x})$ over $(\emptyset, \mathcal{A})$. For example, this is the case of the description logics *DL-Lite*$_r$ (Calvanese et al., 2007) and *flat QL* (Kontchakov and Zakharyaschev, 2014).
Algorithm 1 constructs the set of relevant past corrections from the knowledge base history. It consists of three main steps. First, it constructs patterns to spot knowledge base modifications that could be part of a relevant past correction. Then it uses these patterns to extract atomic modifications that solved some violations in the past. Finally, the relevant past corrections are obtained by pruning those that have been reversed.

Let us explain our algorithm with our running example. Consider the constraint $\Gamma_0(x) : \exists y\ hasGender(y, x) \rightarrow x = \texttt{male} \vee x = \texttt{female}$. Assume that the triple $\langle \texttt{Zeus}, hasGender, \texttt{masculine} \rangle$ was added between $\mathcal{K}_1$ and $\mathcal{K}_2$, but then replaced by the triple $\langle \texttt{Zeus}, hasGender, \texttt{male} \rangle$ between $\mathcal{K}_{100}$ and $\mathcal{K}_{101}$.

The first goal of the algorithm is to find out that the removal of the triple $\langle \texttt{Zeus}, hasGender, \texttt{masculine} \rangle$ between $\mathcal{K}_{100}$ and $\mathcal{K}_{101}$ (as part of the replacement) may be part of a relevant past correction. We call this deletion a *correction seed*. Looking for correction seeds instead of computing the constraint violations for all constraints on all knowledge base versions has the advantage of significantly reducing the search space.

Algorithm 1 Construction of `PCDataset`

Input: set of constraints $\mathcal{C}$, current TBox $\mathcal{T}_p$, history $(\mathcal{K}_i)_{0 \leq i \leq p}$
Output: set of relevant past corrections `PCDataset`

// Construct correction seed patterns
for all $\Gamma \in \mathcal{C}$ such that $\Gamma(\vec{x}) : b(\vec{x}) \rightarrow h(\vec{x})$ **do**
  $\texttt{Patterns}(\Gamma) := \{(?, D(\vec{x})) \mid D(\vec{x}) \in b'(\vec{x}), b'(\vec{x}) \in \texttt{rewrite}(b(\vec{x}), \mathcal{T}_p)\}$
  if $h \neq \texttt{false}$ **then**
    $\texttt{Patterns}(\Gamma) = \texttt{Patterns}(\Gamma) \cup (\{(A(\vec{x}), ?) \mid A(\vec{x}) \in h'(\vec{x}), h'(\vec{x}) \in \texttt{rewrite}(h(\vec{x}), \mathcal{T}_p)\})$
// Extract past corrections
for $0 \leq i \leq p-1$ **do**
  if $(\mathcal{M}_i^+, \mathcal{M}_i^-)$ such that $\mathcal{K}_{i+1} = (\mathcal{K}_i \cup \mathcal{M}_i^+) \setminus \mathcal{M}_i^-$ matches some pattern in $\texttt{Patterns}(\Gamma)$ **then**
    $\texttt{PCDataset} = \texttt{PCDataset} \cup \{\langle (\mathcal{M}_i^+, \mathcal{M}_i^-), \Gamma(\vec{a}), \mathcal{V}, i \rangle \mid \mathcal{V} \in \texttt{Violations}(\mathcal{K}_i, \Gamma(\vec{a})) \setminus \texttt{Violations}(\mathcal{K}_{i+1}, \Gamma(\vec{a}))\}$
// Remove reversed past corrections
for $\langle (\mathcal{M}_i^+, \mathcal{M}_i^-), \Gamma(\vec{a}), \mathcal{V}, i \rangle \in \texttt{PCDataset}$ **do**
  if $\mathcal{M}_i^+ \not\subseteq \mathcal{K}_p$ or $\mathcal{M}_i^- \cap \mathcal{K}_p \neq \emptyset$ **then**
    $\texttt{PCDataset} = \texttt{PCDataset} \setminus \{\langle (\mathcal{M}_i^+, \mathcal{M}_i^-), \Gamma(\vec{a}), \mathcal{V}, i \rangle\}$

To find such correction seeds efficiently, the first step of the algorithm precomputes for each constraint a set of atomic modification patterns that the possible correction seeds would match. In the example there would be only one pattern: the deletion pattern $(?, \langle ?, \mathit{hasGender}, ?\rangle)$, where ? can be anything so that it matches both the deletion of $\langle ?, \mathit{hasGender}, ?\rangle$ and its replacements. Since we only consider past corrections that involve assertions and want them to be relevant for the current TBox, computing the correction seed patterns can be done via query rewriting of the conjunctive queries in the body $b(\vec{x})$ and the head $h(\vec{x})$ of the constraint with respect to $\mathcal{T}_p$. Indeed, we have chosen to restrict ourselves to description logics that allow this rewriting. Each atom that occurs in the rewriting of the body of a constraint corresponds to a deletion pattern, and each atom that occurs in the rewriting of the head of a constraint corresponds to an addition pattern. We collect the patterns for the constraint Γ in the set Patterns(Γ).

The second step of the algorithm verifies, for each correction seed, whether it solved some constraint violation in the past – i.e., whether $\mathcal{K}_i$ contains some violations of some constraint instances that are not in $\mathcal{K}_{i+1}$. If so, the modification between $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$ is a solution that solved these violations in $\mathcal{K}_i$. In the example we would have found the violation $\{\langle \texttt{Zeus}, \mathit{hasGender}, \texttt{masculine}\rangle\}$ of $\Gamma_0(\texttt{masculine})$ in $\mathcal{K}_{100}$, which is not in $\mathcal{K}_{101}$. So we would have extracted that $(\langle \texttt{Zeus}, \mathit{hasGender}, \texttt{male}\rangle, \langle \texttt{Zeus}, \mathit{hasGender}, \texttt{masculine}\rangle)$ is a solution that solved the violation $\{\langle \texttt{Zeus}, \mathit{hasGender}, \texttt{masculine}\rangle\}$ of $\Gamma_0(\texttt{masculine})$ in $\mathcal{K}_{100}$. We store this information as a tuple in the relevant past corrections dataset (the `PCDataset`), as shown in Example 6.14. Finding the constraint instances violated in $\mathcal{K}_i$ or $\mathcal{K}_{i+1}$ is done via conjunctive query answering (Proposition 6.4), and computing their violations amounts to computing conjunctive query justifications (Proposition 6.7).

The final step of the algorithm removes corrections that have been reversed. The result is thus the set of relevant past corrections.

Example 6.14. Here is an example of what a row in the dataset `PCDataset` of relevant past corrections extracted for our running example could look like:

- Constraint instance: $\Gamma_0(\texttt{masculine})$
- Violation: $\{\langle \texttt{Zeus}, \mathit{hasGender}, \texttt{masculine}\rangle\}$
- Relevant past correction:
  $(\{\langle \texttt{Zeus}, \mathit{hasGender}, \texttt{male}\rangle\}, \{\langle \texttt{Zeus}, \mathit{hasGender}, \texttt{masculine}\rangle\})$
- Knowledge base version index: 100

6.6 Correction Rule Mining

In this section, we propose an approach based on rule mining to learn *correction rules* for building solutions to constraint violations. We name this approach *CorHist* like "*Cor*rections from *Hist*ory".

Correction rules. The previous algorithm has given us a list of relevant past corrections (the PCDataset, exemplified in Example 6.14). We now present our approach to mine *correction rules* from this dataset and the knowledge base history.

Definition 6.15 (Correction rule)**.** A *correction rule* is of the form

$$r : [\Gamma(\vec{x})] : \mathcal{E}(\vec{x}, \vec{y}, \vec{z}) \to (\mathcal{M}^+(\vec{x}, \vec{y}), \mathcal{M}^-(\vec{x}, \vec{y})), \text{ where}$$

- $\Gamma(\vec{x})$ is a constraint that can be partially instantiated, i.e., some of its variables have been replaced by constants,
- $(\mathcal{M}^+(\vec{x}, \vec{y}), \mathcal{M}^-(\vec{x}, \vec{y}))$ is an atomic modification where the same partial instantiation of $\vec{x}$ is done,
- $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ is a set of atoms called the *context* of the violation such that $\mathcal{M}^-(\vec{x}, \vec{y}) \subseteq \mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ and such that the same partial instantiation of $\vec{x}$ is done.

A correction rule can be *applied* to a knowledge base $\mathcal{K}$ when there exist tuples of constants $\vec{a}, \vec{b}$ such that $\mathcal{K}$ violates $\Gamma(\vec{a})$ (recall that this can be decided via conjunctive query answering by Proposition 6.4), $\vec{a}$ is compatible with the partial instentiation of $\vec{x}$ in the rule, and $\mathcal{K} \models \exists \vec{z}\, \mathcal{E}(\vec{a}, \vec{b}, \vec{z})$. The *result of the rule application* is then $(\mathcal{M}^+(\vec{a}, \vec{b}), \mathcal{M}^-(\vec{a}, \vec{b}))$.

Note that while the variables from $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ that do not appear in $\Gamma(\vec{x})$ or in the head of r can be existentially quantified, those that occur in the head of r have to be free: they have to be mapped to individuals occurring in the knowledge base to construct the result.

Example 6.16. In our running example, we would like to learn the following correction rules:

$$\begin{aligned} r_1 := [\Gamma_0(\texttt{masculine})] : \{&\mathit{hasGender}(y, \texttt{masculine})\} \\ &\to (\mathit{hasGender}(y, \texttt{male}), \mathit{hasGender}(y, \texttt{masculine})) \\ r_2 := [\Gamma_2(x)] : \{&\mathit{hasMother}(y, x), \texttt{Human}(y)\} \\ &\to (\texttt{Human}(x), \emptyset) \end{aligned}$$

The context of the second rule says that if x is the mother of a human, then x must also be a human. The rule obtained by replacing `Human` by `Animal` would express how to solve a violation of Γ_2 in the context where y is an animal.

Mining correction rules. We mine correction rules with Algorithm 2. This algorithm is an adaptation of the algorithm in Galárraga, Teflioudi, et al., 2013; Galárraga, Teflioudi, et al., 2015 to our context, where we learn rules, not from a knowledge base, but from the `PCDataset` and the knowledge base history. We first adapt the definitions of the confidence and support already presented in Chapter 2 to our case. The *support of the body* of a correction rule r for a constraint Γ is the number of violations of Γ stored in the `PCDataset` that could have been corrected by applying r. Such violations are associated with an instance $\Gamma(\vec{a})$ of the partially instantiated $\Gamma(\vec{x})$ that appears in r and with an index i such that $\mathcal{K}_i \models \exists \vec{z}\, \mathcal{E}(\vec{a}, \vec{b}, \vec{z})$ for some $\vec{b}$. These two conditions imply that r could be applied to the knowledge base $\mathcal{K}_i$. Moreover, we need to check that the result of applying r to $\mathcal{K}_i$ actually gives a solution to $\mathcal{V}$. For example, consider the constraint $\Gamma(x) : \exists y\ R(x, y) \wedge A(x) \rightarrow \texttt{false}$ and the `PCDataset` that contains the two past corrections $(\emptyset, R(a, b))$ for $\Gamma(a)$ at revision i and $(\emptyset, R(a, c))$ for $\Gamma(a)$ at revision j where $a, b, c \in \mathsf{N_I}$. Both violations count for the support of the body of $[\Gamma(x)] : \exists y\ R(x, y) \rightarrow (\emptyset, R(x, y))$ but only the second one counts for the support of the body of $[\Gamma(x)] : R(x, c) \rightarrow (\emptyset, R(x, c))$, even in the case where $\mathcal{K}_i \models R(a, c)$.

Formally,

$$sup_{body}(r) = |\{\mathcal{V} \mid \langle ?, \Gamma(\vec{a}), \mathcal{V}, i\rangle \in \texttt{PCDataset}, \exists \vec{b}\ \mathcal{K}_i \models \exists \vec{z} \mathcal{E}(\vec{a}, \vec{b}, \vec{z})$$
and the result of the application of r to $\mathcal{K}$ at revision i is a solution to $\mathcal{V}\}|$.

The *support of the rule* r measures when the past correction is exactly the result of the application of the rule in the cases where it could be applied. Formally,

$$sup_{rule}(r) = |\{\mathcal{V} \mid \langle(\mathcal{M}^+(\vec{a}, \vec{b}), \mathcal{M}^-(\vec{a}, \vec{b})), \Gamma(\vec{a}), \mathcal{V}, i\rangle \in \texttt{PCDataset},$$
$$\text{and } \mathcal{K}_i \models \exists \vec{z}\ \mathcal{E}(\vec{a}, \vec{b}, \vec{z})\}|.$$

Finally, as usual, the *confidence of a correction rule* r is $conf(r) = \frac{sup_{rule}(r)}{sup_{body}(r)}$.

Algorithm 2 shows our mining algorithm. It takes as input the PCDataset computed by Algorithm 1, the knowledge base history, a minimum support threshold *minsup*, a minimum confidence threshold *minconf*, and a regularization threshold θ. These thresholds are chosen empirically (see Section 6.8.3). The algorithm produces correction rules (Definition 6.15). For this purpose, it first generates a trivial rule r_0 for each entry of the `PCDataset` (line 4). This rule has as context simply the deletion part of the constraint past correction.

This trivial rule is then transformed into several more general rules, which we call *basic rules*, each of which is obtained from r_0 by replacing some of the constants by variables (line 5). Formally, the algorithm uses all partial substitutions σ from constants to distinct fresh variables. It retains only those basic rules that meet the minimum support and confidence thresholds (line 6).

Algorithm 2 Correction rule mining

Input: `PCDataset`, $(\mathcal{K}_i)_{0\leq i\leq p}$, `minsup`, `minconf`, θ
Output: correction rules

1: // Generate basic rules
2: `BasicR` := $\emptyset$
3: **for all** $\langle(\mathcal{M}^+(\vec{a},\vec{b}),\mathcal{M}^-(\vec{a},\vec{b})),\Gamma(\vec{a}),\mathcal{V},i\rangle \in$ `PCDataset` **do**
4: $r_0 := [\Gamma(\vec{a})] : \mathcal{M}^-(\vec{a},\vec{b}) \rightarrow (\mathcal{M}^+(\vec{a},\vec{b}),\mathcal{M}^-(\vec{a},\vec{b}))$
5: `BasicR` = `BasicR` $\cup \{\sigma(r_0) \mid C \subseteq \vec{a}\cup\vec{b}, \sigma : C \rightarrowtail Var,$
6: $sup_{rule}(\sigma(r_0)) \geq$ `minsup`, $conf(\sigma(r_0)) \geq$ `minconf`$\}$
7: // Refine the context part of the rules
8: $q := []$, q.`enqueueAll`(`BasicR`)
9: **while** q is not empty **do**
10: $r := q$.`dequeue`()
11: Output r
12: **for all** operators op **do**
13: **for all** $r' \in op(r)$ **do**
14: **if** $sup_{rule}(r') \geq$ `minsup` and $conf(r') \geq conf(r) + \theta$ **then**
15: q.`enqueue`(r')

In the second step, the algorithm incrementally refines each rule by building up its context part $\mathcal{E}(\vec{x},\vec{y},\vec{z})$ (lines 8-15). This works similarly to the mining algorithm of Galárraga, Teflioudi, et al., 2013: Each refinement step adds one atom built from the knowledge base concept and role names and the variables and constants that appear in the rule, plus at most one fresh variable (lines 12-13). The possible operators are:

- Add a new concept for an existing constraint variable of $\vec{x}$. For example, if x is bound to `Zeus` and $Person$(`Zeus`) is in the knowledge base, then $Person(x)$ is going to be returned as possible atom.
- Add a new role for an existing variable of $\vec{x}$, the other role parameter being assigned to a fresh variable. For example, if x is bound to `Zeus` and $hasGender$(`Zeus`, `male`) and $hasChild$(`Chronos`, `Zeus`) are in the knowledge base, then $hasGender(x,y)$ and $hasChild(y,x)$ are going to be returned as a possible atoms where y is a fresh variable.
- Add a new role for an existing variable of $\vec{x}$, the other role parameter being assigned to an existing variable. For example, if x is bound to `Zeus` and y to `Chronos` and $hasChild$(`Chronos`, `Zeus`) is in the knowledge base, then $hasChild(y,x)$ are going to be returned as a possible atom.

- Add a new role for an existing variable of $\vec{x}$, the second role parameter being assigned to a constant. For example, if x is bound to `Zeus` and $\mathit{hasGender}(\texttt{Zeus}, \texttt{male})$ and $\mathit{hasChild}(\texttt{Chronos}, \texttt{Zeus})$ are in the knowledge base, then $\mathit{hasGender}(x, \texttt{male})$ is going to be returned as a possible atom, but not $\mathit{hasChild}(\texttt{Chronos}, x)$.

If the resulting rule meets the minimum support threshold and improves the confidence by at least θ (line 14), the rule is retained (line 15).

Applying correction rules. When all rules have been mined, they are sorted by decreasing confidence, breaking ties with the help of the support (as it is done in Liu, Hsu, and Ma, 1998 to build classifiers from rules). This set of rules then forms a program that can be used to fix constraint violations as follows. Given a violation $\mathcal{V}$ of a constraint Γ in $\mathcal{K}$, choose the first rule r in the program that is relevant for Γ (i.e., that contains $[\Gamma(\vec{x})]$ where $\Gamma(\vec{x})$ is a partially instantiated version of Γ). Then check whether r can be applied to $\mathcal{V}$. The correction is the result of the rule application.

Example 6.17. Assume we mined the rules r_1 and r_2 of the preceding example with confidence 0.9 and 0.8 respectively, and another rule $r_3 := [\Gamma_0(x)] : \{\mathit{hasGender}(x, y)\} \to (\emptyset, \mathit{hasGender}(x, y))$ with confidence 0.5. The correction program is (r_1, r_2, r_3).

To correct a violation of Γ_0, i.e., a wrong value for the *hasGender* property, the program first checks whether r_1 is applicable. If so, it replaces `male` by `masculine`. Otherwise, it falls back to r_3 and removes the wrong value. To correct a violation of Γ_2, it ignores r_1 that is not related to Γ_2 and either applies r_2 if the context matches or does nothing.

6.7 Bass

CorHist suffers from a systematic weakness: it can take into account only triples that explicitly appear in a correction rule – and not shallow signals from the state of the knowledge base. To overcome this limitation, we design a new neural network architecture to implement a correction predictor.

Our predictor takes as input the violated constraint, the violation, and the facts about the entities that are mentioned in the violation. It predicts as output a triple to add and/or a triple to delete. At training time, the input and the outputs come from the edit history of the knowledge base. At prediction time, the input comes from the current state of the knowledge base. In the following, we first present

a "conversion" of CorHist to a neural network called *Bass-RL*, before adding new components to improve its performance, leading to our final *Bass* architecture[5].

6.7.1 Bass-RL

The goal of Bass-RL is to build a neural network that mirrors exactly the functioning of the CorHist rule mining approach. Figure 6.1 shows the basic architecture of our network. It is composed of 3 input components, which all feed into the "edit predictor". The dimensions of the internal layers are parametrized by a constant d, which is a hyperparameter of the network. The inputs are:

- **The constraint.** We give to each constraint a unique integer id and then we use a trained embedding matrix to create a d-dimensional vector from the one-hot encoding of the constraint id. This vector is then fed into the edit predictor. This vector gives the network the information about which type of constraint we aim to fix.
- **The violation.** For each of the k triples in the constraint violation, we encode the predicate and the object with the help of the "term embedding" component. These encodings are concatenated and fed into the edit predictor. The subject of the violation triple is not used.
- **The facts about the entities** that appear in the constraint violation. We encode only the predicates and the objects of these facts because the subject is already known to the network. For example, if we consider the violation triple $\langle \texttt{JohnDoe}, birthPlace, \texttt{Paris} \rangle$, the two entities are `JohnDoe` and `Paris`. We consider all facts having `JohnDoe` or `Paris` as subject, and we embed the pairs of predicates and objects of each of these facts e.g., $\{(hasType, \texttt{Person}), (hasGender, \texttt{Male}), \ldots\}$ for `JohnDoe` and $\{(hasType, \texttt{schema:Place}), (inCountry, \texttt{France}), \ldots\}$ for `Paris`. There are $2k$ mentioned entities, 2 for each of the k triples in the constraint violation. Each fact is embedded using the "entity fact embedding" component. Then the output vectors are concatenated and fed into the edit predictor.

Let us now describe each component in detail.

Term embedding

We embed RDF terms as follows: predicates are one-hot encoded and then embedded into a space of dimension d using a trained embedding matrix $\mathcal{P}$. We embed

[5]The "Bass" name derives from "CorHist" because a bass is a singer with a "deep" voice, and "chorist" is an old English word for "singer".

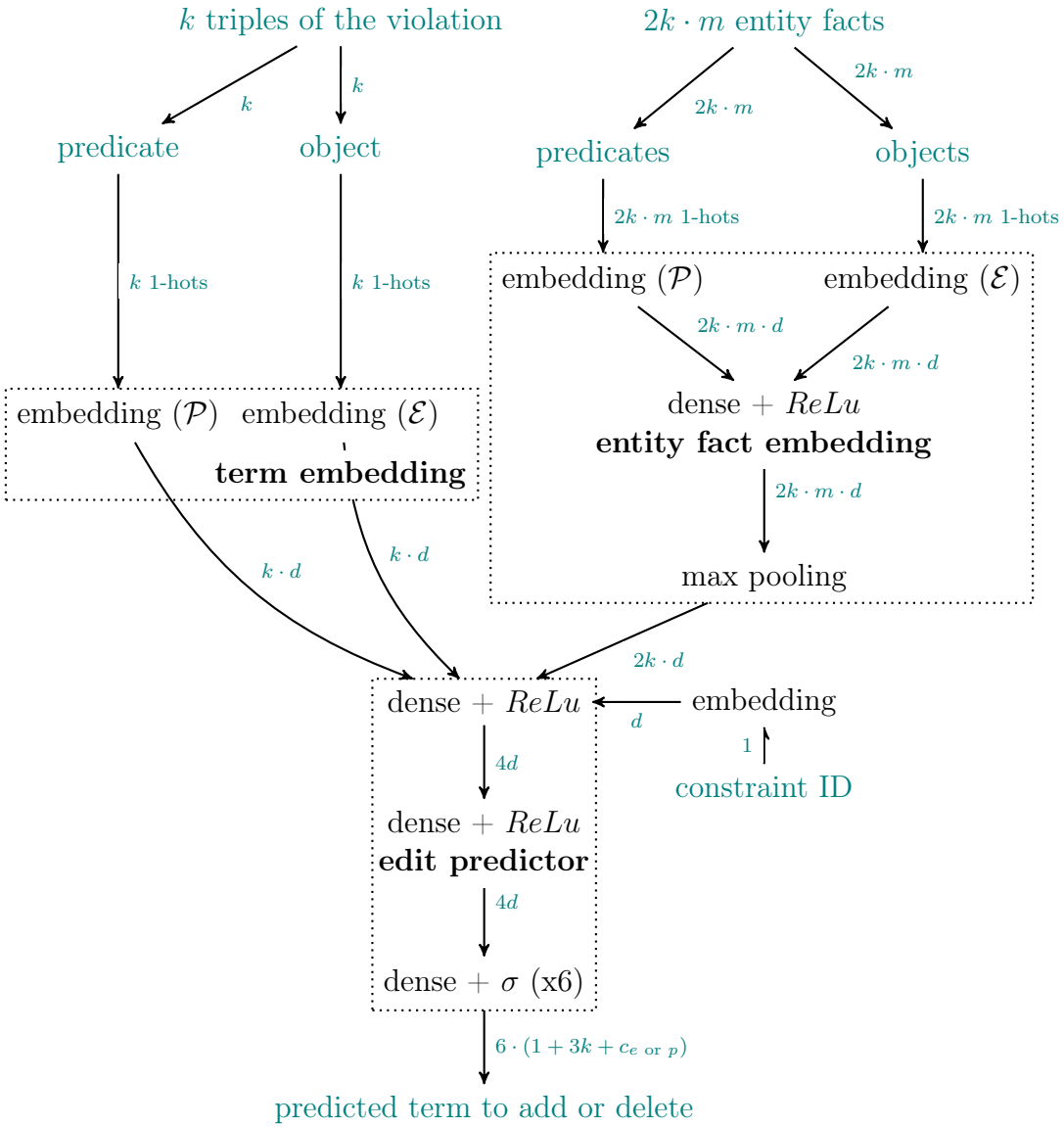

Figure 6.1 – Network architecture of Bass-RL. d is the vector dimension hyperparameter, k is the max number of triples in the constraint violation, and m is the max number of facts per entity.

similarly the objects using a trained matrix $\mathcal{E}$. These matrices $\mathcal{E}$ and $\mathcal{P}$ are shared by all object and predicate embedding operations. This embedding allows the edit predictor to act on specific predicates and entities in the triples of the constraint violation.

Entity fact embedding

Similarly to CorHist, we give the neural network the facts about the $2k$ entities involved in the constraint violations. A classical approach for this purpose would be to use entity embeddings, i.e., to embed the entity itself. However, entity embeddings have two drawbacks: First, they require expensive pre-training. Second, and more crucially, they do not work with new entities. Therefore, we embed not the entity, but the facts that the entity is involved in. While the objects of these facts will still be encoded using learned embeddings (and thus cannot be entities that are unknown at training time), the subjects can be entities that have never been seen at training time. This allows the network to check constraints on newly-added entities.

We encode the (predicate, object) facts of each entity mentioned in the constraint violation as follows: we embed the predicates by reusing the same predicate embedding matrix $\mathcal{P}$ described previously. We embed similarly the objects by reusing the same entity embedding matrix $\mathcal{E}$. Then we combine the predicate and the object using a dense layer[6] with a rectified linear unit (ReLU) non linearity[7]. Then we merge the obtained embeddings for each (predicate, object) using a max-pooling layer to get a single vector for the entity. Formally, if we have the input embedding vectors $\begin{pmatrix} x_1^1 \\ \vdots \\ x_d^1 \end{pmatrix}, \ldots, \begin{pmatrix} x_1^k \\ \vdots \\ x_d^k \end{pmatrix}$ we would return the vector $\begin{pmatrix} max(x_1^1, \ldots, x_1^k) \\ \vdots \\ max(x_d^1, \ldots, x_d^k) \end{pmatrix}$.

Edit predictor

The edit predictor takes as input the previous components, i.e., the constraint id embedding, the embeddings of the k violation triple predicates and objects, and the embedding of the facts about the $2k$ entities mentioned in the constraint violation. We aim here first at combining this input to generate a vector that represents

[6] A dense layer with non-linearity f is a function $X \to f(M \cdot X)$ where X is a d dimensional vector and M a matrix of dimensions $d \times d$

[7] This function is defined as $ReLU(x) = max(0, x)$ applied to each vector term.

the violation by meaningfully combining the already listed inputs. We feed the inputs into a multilayer perceptron i.e., a sequence of dense layers, each followed by a non-linear function. We use two dense layers of dimension $4d$ with the ReLU non-linearity. We use multiple layers to allow the network to do a sequence of combinations of the weights given by the inputs.

After computing this vector, we now have to "decode" it into an edit prediction. Our network outputs two triples – one to add and one to delete. Each triple is given by its three components (subject, predicate, and object). Each output component could of course just be the one-hot encoding of a predicate or entity. However, then the network would have to learn each instantiation of a constraint individually (as in "If the subject of the violation triple is Elvis, then the subject of the addition triple should be Elvis", "If the subject is Madonna, then..."). Therefore, we allow the network to output a code, as in "The subject of the addition triple is the object of the first constraint violation triple". To permit nevertheless the output of constants as well, we combine both approaches. With this, each of the 6 outputs (2×3 triple components) works in the same way, classifying the output term into one of the following options:

- 1 class to state that the output is not existent (the triple should not be returned), or unknown. We call it the class 0.

- $3k$ classes to state that the output term is the same as the subject/predicate or object of one of the k triples in the constraint violation. For example, the class 1 corresponds to the subject of the first violation triple, the class 2 to its predicate, the class 3 to its object, the class 4 to the subject of the second violation triple, etc.

- c_p or c_e classes to state that the output is one of the c_p predicates (for the predicate to add or delete outputs) or one of the c_e entities (for the other outputs) from the list of predicates/entities found at least t times in the expected outputs from the training data (where t is a hyperparameter).

This leads to $1 + 3k + c_e$ possible classes (and so, neurons) for each of the four subject/object outputs and $1 + 3k + c_p$ for the two predicate outputs. Each of these outputs are implemented like regular classifiers using a dense layer with a softmax non-linearity[8]. The outputs returns a $1 + 3k + c_{e \text{ or } p}$ dimensional vector. The kth vector output is the predicted probability of kth class.

To retrieve the final edit, we consider the output for both the "delete" triple and the "add" triple. If the three outputs for the subject, predicate, and object

[8]The *softmax* function is defined by $\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{D} e^{x_j}}$ for all $i \in \{1, \ldots, D\}$ if there are D possible classes.

of the triple give known values (a known entity or predicate or a known violation triple term given in the input) we build the triple to add or to delete from these outputs. If the three outputs give the "not defined" class, we return no triple. This means that the edit does not contain a triple to add, or a triple to delete, respectively. If there are only one or two components returning "not defined", we assume that the network has not been able to predict a correction.

6.7.2 Bass

We now present our improved repair predictor, *Bass*. Figure 6.2 shows the architecture of our network. We designed two improvements over Bass-RL:

RDF literal embedding

We want our predictor to be able to act on RDF literals. We recall that RDF literals are string of unicode characters, optionally annotated with a language tag (e.g., `"example"@en` where `example` is the literal value and `en` is the language tag) or a datatype identifier (e.g., `"2020-09-07"^^xsd:date` where `xsd:date` stands for `<http://www.w3.org/2001/XMLSchema#date>` to state that `2020-09-07` should be interpreted as a date). For example, if every book can have at most one ISBN identifier, and if a given book has two ISBNs, `2-7654-1005-4`, and `abc`, it is easy to decide that `abc` should be abandoned, if we allow the network to access the literals.

For this purpose, we first remove the datatype IRI. This leads to strings like `"Elvis Presley"@en` for a language tagged string or `"42"` for an integer. To split these strings into components that could be embedded separately, we tokenize the string with the BERT tokenizer (Devlin et al., 2019) and apply an embedding matrix on each token. We then apply a max pooling on the embedding sequence to get another d-dimensional vector. We use the BERT tokenizer to provide better support for out-of-vocabulary words. It also allows us to better handle complex values like numbers and dates that would not be properly split by a tokenizer splitting on whitespaces. It also allows keeping more significant elements than a simple char based encoding. For example the date literal `"2020-09-07"` is going to be tokenized into `[CLS]`, `"`, `2020`, `-`, `09`, `-`, `07`, `-`, `"`, `[SEP]`. And the label `"Elvis Presley"@en` into `[CLS]`, `"`, `elvis`, `pres`, `##ley`, `"`, `@`, `en`, `[SEP]`. This input is not used when the objects are IRIs or blank nodes.

Constraint embedding

Bass-RL, like CorHist, encoded each constraint by an ID. This does not allow the network to generalize over each constraint, to treat similar constraints similarly, or

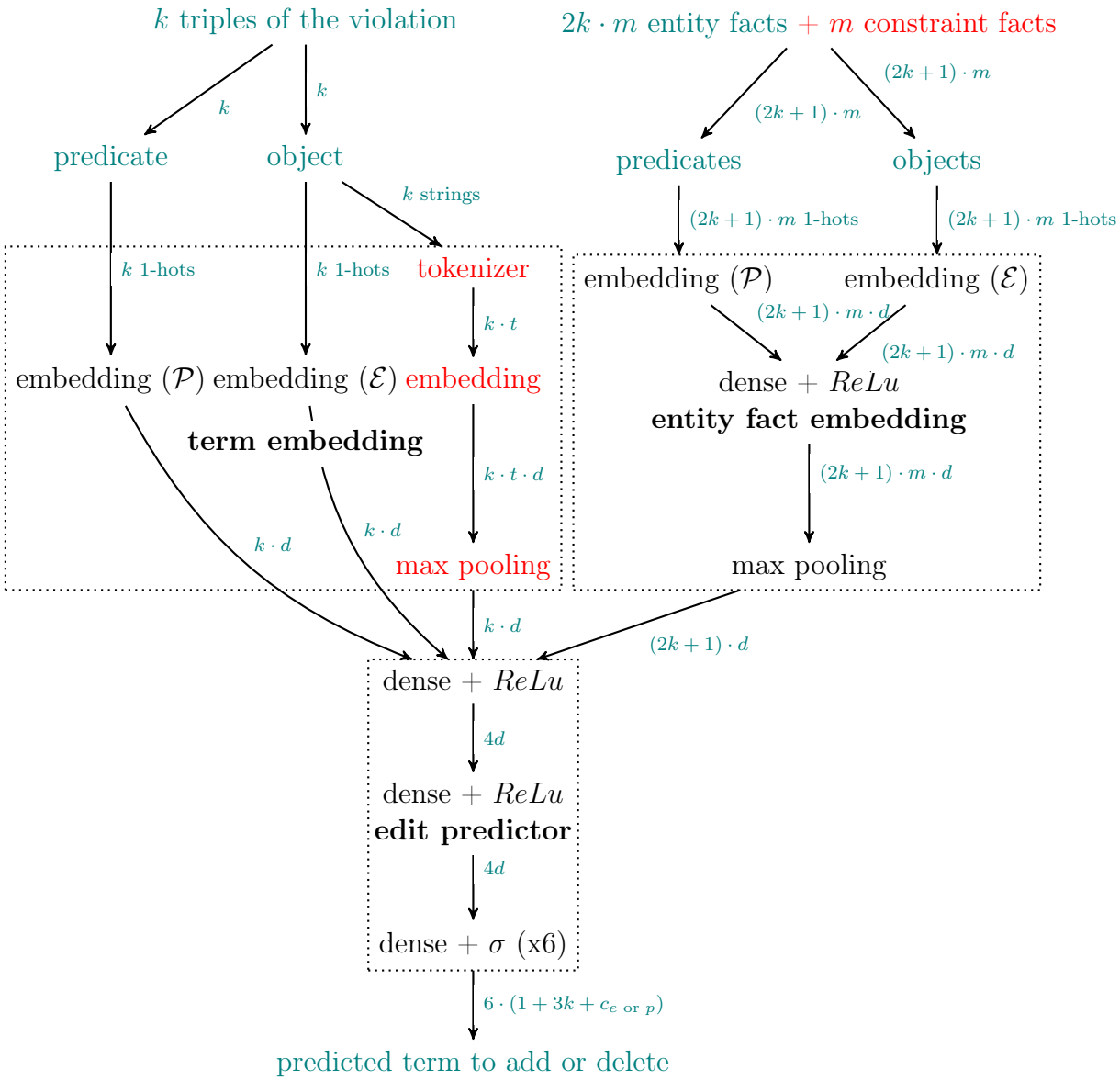

Figure 6.2 – Bass network architecture when there is just one violation triple. d is the vector dimension hyperparameter, k the max number of triples in the constraint violation, m the max number of facts per entity, and t the max number of string tokens. Additions compared to Bass-RL (Figure 6.1) are in red.

to learn new constraints. To remedy this shortcoming, we encode each constraint by a set of (predicate, object) pairs and we encode this set using the entity fact embedding component introduced earlier.

To encode each constraint by a set of (predicate, object) pairs, we rely on *constraint shapes*. The constraint shape of a constraint Γ is a constraint Γ^s where all constants in Γ have been replaced by fresh variables. For example, the shape of the constraint $\Gamma_1(x) : \mathit{Person}(x) \rightarrow \exists y\ \mathit{hasBirthPlace}(x, y)$ is $\Gamma_1^s(x) : c_1(x) \rightarrow \exists y\, p_1(x, y)$. Two constraint shapes are *equivalent* if they have the same components up to a renaming of variables. With this definition, we encode a constraint $\Gamma(\vec{x}) : b(\vec{x}) \rightarrow \exists \vec{y}\ \bigvee_i h_i(\vec{x}, \vec{y})$ by the following set of property value pairs:

- (`bass:constraintShape`, c) where c is an identifier assigned to the equivalence class of the constraint shape Γ^s.
- (p_i, o_i) for each $\langle s_i, p_i, o_i \rangle$ in b and h_i where o_i is a constant.
- (p_i^-, s_i) for each $\langle s_i, p_i, o_i \rangle$ in b and h_i where s_i is a constant and p_i^- is the inverse property of p_i.[9]

These components give us our new Bass network.

6.8 Experiments on Wikidata

This section presents an experimental evaluation of CorHist and Bass against Wikidata content.

6.8.1 Wikidata

Wikidata is a generalist collaborative knowledge base. Wikidata has already been described in the previous chapters. The data about each Wikidata entity is stored in a versioned JSON blob, and there are more than 700M revisions. Wikidata encodes facts not in plain RDF triples but in a reified representation, in which each main $\langle s, p, o \rangle$ triple can be annotated with qualifiers and provenance information (Vrandecic and Krötzsch, 2014).

Wikidata knows the property `instanceOf` which is similar to `rdf:type`. It does not have a formally defined TBox, but knows properties such as `subClassOf`, `subPropertyOf`, and `inverseOf`. However, only the property `subClassOf` is used to flag constraint violations. Therefore, we use only this property in our TBox, which thus contains only simple concept inclusions.

[9] Using a new IRI if there is no inverse property of p_i already in the knowledge base.

We consider the set $\mathcal{C}$ of constraints built from ten types of Wikidata property constraints (see Table 6.1). They are the top Wikidata property constraints that can be expressed in rules, covering the majority of the most used constraints, as well as 71% of Wikidata property constraints. The remaining constraints are mainly about string format validation with regular expressions (52% of the remaining constraints) and qualifiers (31% of them).

6.8.2 Dataset Construction

We stored the RDF version (Erxleben et al., 2014; Hernández, Hogan, and Krötzsch, 2015) of the Wikidata edit history in an RDF store built for this task. We used named graphs for the global state of Wikidata after each revision, and the triple additions and deletions. Our dataset stores 390M annotated triples about 49M items extracted from the July 1st, 2018 full database dump. The storage system is described in detail in Chapter 7.

We extended Wikidata with facts about the external web pages mentioned in the knowledge base. These facts are considered by CorHist and Bass implementations just like regular Wikidata facts. However, because they describe external elements, these additional facts are considered to be always in the knowledge base and so, do not change between Wikidata revisions and could not appear in the mined corrections. They can only be used as context in the rules mined by CorHist and Bass. This works as follows: For every URL s that appears in the violation triples, we add two new facts: The first one is the triple $\langle s, \texttt{history:pageStatusCode}, \texttt{XXX}\rangle$, where `XXX` is the HTTP response code of that URL. The other is $\langle s, \texttt{history:pageContainsLabel}, o\rangle$, where o is the object of a triple connecting o to s, and one of o's labels (found with the `rdfs:label` relation) appears in the HTML page of s. For example, if we consider the triple

$$\langle \texttt{DouglasAdams}, isSameAs, \texttt{<https://viaf.org/viaf/113230702>}\rangle$$

from Wikidata, we fetch the external URL from `viaf.org`, and we add the two triples

$$\langle \texttt{<https://viaf.org/viaf/113230702>}, \texttt{history:pageStatusCode}, 200\rangle$$

$$\langle \texttt{<.../113230702>}, \texttt{history:pageContainsLabel}, \texttt{Douglas\_Adams}\rangle$$

because the VIAF website returned a HTTP code 200 for the page and because the label `"Douglas Adams"@en` of the entity `Douglas_Adams` appears on the page.

We extracted the relevant past corrections as explained in Section 6.5. Wikidata revisions do not correspond exactly to atomic modifications in our sense. For example, Wikidata bots can change multiple unrelated facts about the same entity at the same time. Wikidata users also sometimes prefer to delete a statement

Table 6.1 – Wikidata property constraints. R is the property for which the constraint is given. A constraint has several lines when it uses a property whose set of values may be specified or not. #constr. is the total number of constraints of the given type in Wikidata. #triples is the sum for all these constraints of the numbers of triples with the property R on which they apply. #viol. is the number of violations for this constraint in Wikidata on July 1st, 2018. #past cor. is the number of past corrections we extracted from Wikidata history. t.o. indicates that we were not able to extract all past corrections because of timeout so that we sample them (we then indicate the number of corrections we extracted). Note that here R and R' are not allowed to denote inverse property expressions.

Name	Rule form	#constr.	#triples	#viol.	#past cor.
Type[a]	$\exists y R(x,y) \rightarrow A_1(x) \vee \ldots \vee A_n(x)$	2575	249M	3465k	16M (t.o.)
Value type	$\exists y R(y,x) \rightarrow A_1(x) \vee \ldots \vee A_n(x)$	696	67M	3062k	19M (t.o.)
One-of	$\exists y R(y,x) \rightarrow x = a_1 \vee \ldots \vee x = a_n$	116	4.2M	4k	19k
Item requires statement	$\exists y R(x,y) \rightarrow R'(x,a_1) \vee \ldots \vee R'(x,a_n)$ $\exists y R(x,y) \rightarrow \exists z R'(x,z)$	3102	255M	3710k	15M (t.o.)
Value requires statement	$\exists y R(y,x) \rightarrow R'(x,a_1) \vee \ldots \vee R'(x,a_n)$ $\exists y R(y,x) \rightarrow \exists z R'(x,z)$	243	85M	1345k	6M (t.o.)
Conflict with	$\exists y R(x,y) \wedge (R'(x,a_1) \vee \ldots) \rightarrow$ `false` $\exists yz R(x,y) \wedge R'(x,z) \rightarrow$ `false`	781	4543M	68k	449k
Inverse[b]	$R(x,y) \rightarrow R'(y,x)$	152	6M	409k	3M
Single value	$R(x,y) \wedge R(x,z) \rightarrow y = z$	2909	90M	389k	491k
Distinct values	$R(y,x) \wedge R(z,x) \rightarrow y = z$	2843	78M	322k	16M

[a] The Wikidata constraint *Type* can be qualified so that instead of expressing that the subject of the property belongs to a given class using the property *instanceOf*, it states that the subject has to be related to the class by the property *subClassOf*, or either by *instanceOf* or by *subClassOf*. We ignore these cases, which are marginal: only 5% of *Type* constraints are qualified with property *subClassOf*, and 0.4% with "*instanceOf* or *subClassOf*". The same goes analogously for *Value type*, which concerns the object of the statement.

[b] *Inverse* and *Symetric* are two distinct kinds of constraints in Wikidata but we treat them together since *Symetric* is actually a special case of *Inverse*.

then add another one with the same property instead of directly modifying the value, in order to clear the existing qualifiers and references. Therefore, we artificially created a replacement modification for every deletion with a neighboring addition by the same user, which shares at least two components of the triple (analogously for additions). For example, if the correction seed is the deletion of $\langle \texttt{Zeus}, hasGender, \texttt{masculine} \rangle$, and if this revision or a neighboring one adds $\langle \texttt{Zeus}, hasGender, \texttt{male} \rangle$, then we consider this a replacement. However, if the same revision added the triple $\langle \texttt{Zeus}, hasMother, \texttt{Rhea} \rangle$, then we would not consider this a replacement, because it does not share two components with the first one.

Since the TBox consists of simple concept inclusions and the constraint bodies contain only roles, the deletion patterns for correction seeds correspond directly to the atoms of the constraint body. In the same vein, only atoms in the head of the *Type* or *Value type* constraints need to be rewritten. For example, if we consider the *Type* constraint $\exists y\ hasGender(x, y) \rightarrow Person(x)$ and if in Wikidata the subclasses of *Person* are *Human*, *FictionalHuman* and *Deity* then we would rewrite the constraint head to $Person(x)$ to $Person(x) \wedge Human(x) \wedge FictionalHuman(x) \wedge Deity(x)$. To find the constraint violations solved by a correction seed, we make use of the fact that the correction seed allows us to know the constraint instance $\Gamma(\vec{a})$, and we look for matches of the constraint instance body.

To speed up the execution for the four constraint types which have the highest numbers of past corrections, *Type*, *Value type*, *Item requires statement* and *Value requires statement*, we did not extract all the past corrections but sample them as follows. We compute only the relevant past corrections that were applied between $\mathcal{K}_i$ and $\mathcal{K}_{i+1}$ where i is a multiple of $s = max(1, N/10^6)$ with N the number of triples with the property R of the constraint at hand. Note that this sampling might be biased because edits in Wikidata are not evenly distributed on its history. For instance, imagine that we only process the revisions $\mathcal{K}_i$ where i is odd, and that two automated tools are editing Wikidata at the same time and at the same speed, with both tools making their edits alternatively. If this happens, then the proposed sampling method would only detect the changes of one of the tools, and not of the other. Even with these limitations, this sampling allows us to get a ground truth with sufficient coverage for rule mining for each constraint. Indeed, in practice, this extraction limitation only affects 0.9% of *Type*, 2% of *Value type*, 0.5% of *Item requires statement*, and 3% of *Value requires statement* constraints. All corrections are extracted for the other constraints.

In the final dataset, we limited the number of past corrections to 200k per constraint type by doing a random sampling in all the extracted past corrections. This limitation is made to facilitate the use of the dataset and to allow fetching all mentioned Web pages in a week. We split the dataset into an 80% training set, a

10% cross-validation set, and a 10% test set. To simplify the re-use of our dataset, we extended it with the facts about the entities mentioned in the violation triples at the time of the violation correction. This allows CorHist and Bass to be trained without having to access the edit history.

6.8.3 Implementation of the Approaches

CorHist

The output of CorHist is a set of correction rules that form a program (Section 6.6). To evaluate such a program, we apply it to each of the constraint violations stored in the test part of the `PCDataset`, using the knowledge base state just before the revision when the correction happened to evaluate the body and the head of the rule. Then we check whether the correction we compute is the same as the one associated with the constraint violation in the test part of the `PCDataset`.

We then report the comparison of the predicted correction with the expected correction from the `PCDataset` using the precision, recall and F1 metrics (Definitions 2.24, 2.25, and 2.26 Page 25). CorHist mines rules as explained in Section 6.6. We trained it on the training part of the `PCDataset`. In order to decrease the computation time, we only allow three atoms in $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$. We use a minimal support threshold $minsup = 10$, a minimal confidence threshold $minconf = 0.1$, and a regularization threshold $\theta = 0.05$. These thresholds have been set in order to limit the computation time of rule learning. After mining the rules with the minimal confidence threshold of 0.1, the cross-validation part of the `PCDataset` is used to determine a refined minimal confidence threshold that maximizes the F1 score of the obtained program. This allows us to pick the best possible minimal threshold according to the cross-validation dataset. The test set is used to evaluate the final program. The CorHist implementation is available on GitHub[10].

Example 6.18 (Example of mined rules by CorHist)**.** Here are some examples of rules mined by CorHist for each of the constraint types:

Type. We consider the constraint $\Gamma(s) : \exists o\ isAListOf(s, o) \rightarrow List(s)$. The following rule states that if a Wikidata item violates Γ and if the item is an instance of *WikiDisambiguationPage*, then the *isAListOf* facts should be removed:

$$[\Gamma(s)] : isAListOf(s, o) \wedge WikiDisambiguationPage(s) \rightarrow (\emptyset, isAListOf(s, o))$$

Value type. We consider the constraint $\Gamma(o) : \exists s\ foundInTaxon(s, o) \rightarrow Taxon(o)$. The following rule states that if the taxon is stated to be found in `human`

[10] `https://github.com/Tpt/corhist`

and has parts then the value of *foundInTaxon* should be replaced by `homoSapiens`:

$$[\Gamma(\texttt{human})] : \mathit{foundInTaxon}(s, \texttt{human}) \wedge \mathit{hasPart}(s, v) \rightarrow (\mathit{foundInTaxon}(s, \texttt{homoSapiens}), \mathit{foundInTaxon}(s, \texttt{human}))$$

One-of. We consider the constraint $\Gamma(o) : \exists s\ \mathit{mannerOfDeath}(s, o) \rightarrow o = \ldots$. The following rule fixes misuses by stating that if the value of *mannerOfDeath* is `trafficAccident`, then the predicate should be replace by *causeOfDeath*:

$$[\Gamma(\texttt{trafficAccident})] : \mathit{mannerOfDeath}(s, \texttt{trafficAccident}) \rightarrow (\mathit{causeOfDeath}(s, \texttt{trafficAccident}), \mathit{mannerOfDeath}(s, \texttt{trafficAccident}))$$

Item requires statement. We consider the constraint $\Gamma(s) : \exists o\ \mathit{heritageStatus}(s, o) \rightarrow \exists o\ \mathit{country}(s, o)$. The following rule states that if the monument has for heritage status `monumentInFornminnesregistret`, then it is located in Sweden, completing the knowledge base:

$$[\Gamma(s)] : \mathit{heritageStatus}(s, \texttt{monumentInFornminnesregistret}) \rightarrow (\mathit{country}(s, \texttt{Sweden}), \emptyset)$$

Value requires statement. We consider the constraint $\Gamma(o) : \exists s\ \mathit{residence}(s, o) \rightarrow \exists o_2\ \mathit{country}(o, o_2)$. The following rule states that if an entity violates Γ and has diplomatic relations, then it is a country and so has for country itself:

$$[\Gamma(o)] : \mathit{diplomaticRelation}(o, v) \rightarrow (\mathit{country}(o, o), \emptyset)$$

Conflict. We consider the constraint $\Gamma(s) : \exists o_1, o_2\ \mathit{filmplID}(s, o_1) \wedge \mathit{filmplFilmID}(s, o_2) \rightarrow \texttt{false}$. The following rule states that if the entity has a value for *filmplFilmID*, then *filmplID* should be removed:

$$[\Gamma(s)] : \mathit{filmplID}(s, o) \rightarrow (\emptyset, \mathit{filmplID}(s, o))$$

Inverse/Symmetric. We consider the constraint $\Gamma(s, o) : \mathit{geneticAssociation}(s, o) \rightarrow \mathit{geneticAssociation}(o, s)$. The following rule states that the symmetric facts should be materialized for *geneticAssociation*:

$$[\Gamma(s, o)] : \rightarrow (\mathit{geneticAssociation}(o, s), \emptyset)$$

Single value. We consider the constraint $\Gamma(o_1, o_2) : \exists s\ \mathit{sexOrGender}(s, o_1) \wedge \mathit{sexOrGender}\ (s, o_2) \rightarrow o_1 = o_2$. The following rule states that if someone is in a sport team and has two values for *sexOrGender* and one of these values is

`maleOrganism`, then the value `maleOrganism` for their *sexOrGender* should be removed:

$$[\Gamma(s)] : sexOrGender(s, \texttt{maleOrganism}) \wedge sportsTeam(s, v) \rightarrow (\emptyset, sexOrGender(s, \texttt{maleOrganism}))$$

Distinct values. We consider the constraint $\Gamma(s_1, s_2) : \exists s_1, s_2\ ncbiLocusTag(s_1, o) \wedge\ ncbiLocusTag(s_2, o) \rightarrow s_1 = s_2$. The following rule states that there are two entities with the same *ncbiLocusTag* and one has a value for the property *molecularFunction*, then *ncbiLocusTag* should be removed for this entity:

$$[\Gamma(s_1, s_2)] : ncbiLocusTag(s_1, o) \wedge molecularFunction(s_1, v) \rightarrow (\emptyset, ncbiLocusTag(s_1, o))$$

This concludes our presentation of examples of types of rules that we mine (Example 6.18).

Bass

We implemented Bass with the Keras API (Chollet et al., 2015) of Tensorflow 2 (Abadi et al., 2016). For the BERT tokenization, we use the HuggingFace "tokenizers" library (Wolf et al., 2019). With $\mathcal{E}$ and $\mathcal{P}$, we embed only the entities and predicates with at least 100 occurrences. We do the same for the output by setting $t = 100$. We choose to set all the embedding sizes to $d = 128$. We use the constraint type identifier ("single value", "value type"...) to identify the constraint shape, and we use the Wikidata statement that encodes the constraint to generate the (predicate, object) triples that describe the constraint. The Bass implementation is not available publicly yet to comply with the double-blind review constraints of ISWC 2020.

We trained Bass on the same training set as CorHist. We did so, by training for all constraint types at the same time, using the sum of categorical cross-entropy loss[11] for the 6 classification outputs and the gradient-descent based Adam optimizer (Kingma and Ba, 2015). We used the validation set to keep the best epoch according to the loss against the cross-validation dataset. We trained the model with a mini-batch size of 256. We have chosen a large mini-batch size to compute the loss function on enough samples in an attempt to have enough samples of different constraints to make the loss meaningful at each learning step. The best

[11] The categorical cross-entropy is a distance function between the expected predictions and the actual predictions defined as $CE\left(\begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix}, \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix}\right) = \sum_{i=0}^{n} t_i \log(p_i)$ where $t_i \in [0, 1]$ is the expected prediction for the class i, and $p_i \in [0, 1]$ is the predicted output for the same class i.

model has been found after the 3rd epoch on the full training dataset. After loading the dataset into memory, training took 18min using an Nvidia Quadro P3000 mobile GPU, an Intel Core i7-7700HQ CPU, and 32GB of RAM. We evaluate Bass predictions using the same metrics as CorHist: precision, recall, and F1 score.

Note that the Bass optimization target is not the same as the F1 score used for CorHist. Unlike the F1 score, the categorical cross-entropy is a distance function on the neural network outputs, allowing us to easily apply the gradient descent algorithm using the loss function associated with this distance. We make the assumption that this advantage in terms of simplifying the training outweighs the disadvantage for Bass of being optimized for this metric, instead of the F1 score that it will be evaluated with. The experimental evaluation will show that, even with this disadvantage, Bass achieves higher F1 scores than CorHist.

6.8.4 Evaluation against the Test Set

Table 6.2 presents the results of the evaluation of the mined corrections against the test set. We computed both the micro and macro average of the precision, recall, and F1 score per constraint for each constraint type. The macro average is an unweighted average over all constraints, i.e., all constraints are weighted the same. The micro average is an unweighted average over all violations, amounting to an average over all constraints weighted by their number of violations. Both numbers are important: The micro average gives more weight to constraints that have many violations to fix. It thus measures the overall impact of the applied correction on the dataset. However, if a few constraints covered most of the violations, then it would be easier to formulate some violation rules. Our method, in contrast, can also find ways by itself to solve constraints with fewer violations, but together contribute a large mass of corrections. To illustrate this, we also report the macro average: It measures the average performance across different constraints.

We compare our approach with two baselines: The first one, called "delete", is the most basic and uses the fact that all Wikidata constraint bodies contain an atom of the form $R(x, y)$ and the TBox contains only concept inclusions so that all constraint violations contain an assertion that matches $R(x, y)$. The "delete" baseline simply deletes this assertion. We define an additional baseline, "add", which tries to add a new triple to solve the constraint violation. For *Inverse* and *Symmetric* constraints this baseline adds the missing reverse edge and performs very well. For *Item requires statement*, *Value requires statement*, *Type* and *Value type*, it adds the missing triple only if it is possible to know the set of the possible expected value from the constraint rule (if there are multiple possible values, it picks one at uniformly at random for each prediction). For example, for *Type* constraints of the form $\exists y R(x, y) \rightarrow A(x)$ it applies the correction $(A(x), \emptyset)$. For *Item requires statement* constraints of the form $\exists y R(x, y) \rightarrow R'(x, a)$, the "add"

baseline applies $(R'(x, a), \emptyset)$ (similarly for *Value requires statement* constraints). If there are multiple a_i, it chooses one randomly for each constraint. The other constraint types could only be solved by a deletion. Hence, the "add" baseline is not defined for them. We also included in the aggregated results a variant of CorHist, called *CorHist-min* where we do not add context to rules ($\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ is always empty) without changing the other hyperparameters. To keep the table legible, we do not report the performance of CorHist-min and Bass-RL for each individual constraint type.

As shown in Table 6.2, the precision of our approaches strongly outperforms the two baselines – often by a very high margin. Regarding the recall, we manage to keep a reasonable, and sometimes even good, recall (see best F1 scores in Table 6.2). CorHist-min presents a higher precision than CorHist (+5% in micro average and +4% in macro average) at the cost of worst recall (-5% in micro average and -4% in macro average). This suggests that overall adding context to the rules seems more to help to find new rules that cover constraints violations that were not already solved by CorHist-min than it allows to refine rules already mined and kept by CorHist-min.

As shown in the evaluations, Bass slightly outperforms CorHist. The strongest improvements are seen for the constraint types "Conflict with" and "Single value", which both concern the removal of one value between two choices. The simple move from a rule learning algorithm (CorHist) to a neural network (Bass-RL) provides a small performance improvement of 1% in micro average F-score and none in macro average F-score. However, the recall of Bass in macro average is higher by 4% than the CorHist one, at the cost of a precision lower by 21%. This shows that Bass-RL attempts to predict an output more often than CorHist but fails more often at predicting the correct output. This lower recall and higher accuracy of CorHist may be caused by the support and precision thresholds of the rule mining approach which forbid predictions in case the rule does not seem to be significantly useful, a "safety" that Bass does not have. However, this "safety", which is useful also for performance reasons, might make CorHist's recall too low, and hence harm its performance. If the low CorHist recall is not caused by the support and precision thresholds, the small win of Bass-RL related to CorHist in micro average F1 score (1%) might be thanks to the fact that Bass-RL can take into account all facts about the violation entities, and so can draw holistic conclusions from the context facts. This comparison suggests that neural networks are more able to draw predictions on unseen cases, at the cost of precision. Indeed, the rule mining approach is able to provide a better precision thanks to its support and precision thresholds.

The addition of textual data and of a structured representation of the constraint, which is hard to take into account with a rule mining approach, allows

Table 6.2 – Evaluation of the correction rules mined by Bass and CorHist and comparison with the baselines. Best F scores in bold.

Constraint type		Micro average			Macro average		
		Prec.	Rec.	F	Prec.	Rec.	F
Type	add	0.53	0.17	0.26	0.28	0.10	0.14
	delete	0.04	0.04	0.04	0.08	0.08	0.08
	CorHist	0.86	0.75	0.80	0.89	0.34	0.49
	Bass	0.92	0.79	**0.85**	0.83	0.36	**0.50**
Value type	add	0.20	0.07	0.10	0.35	0.11	0.16
	delete	0.01	0.01	0.01	0.04	0.04	0.04
	CorHist	0.70	0.63	0.66	0.81	0.43	**0.56**
	Bass	0.78	0.69	**0.73**	0.70	0.28	0.40
One-of	delete	0.27	0.27	0.27	0.43	0.43	0.43
	CorHist	0.84	0.72	**0.78**	0.84	0.34	**0.48**
	Bass	0.86	0.71	**0.78**	0.77	0.26	0.39
Item requires statement	add	0.99	0.11	0.20	0.92	0.13	0.22
	delete	0.02	0.02	0.02	0.07	0.07	0.07
	CorHist	0.85	0.36	**0.51**	0.94	0.19	**0.32**
	Bass	0.89	0.35	0.50	0.76	0.17	0.28
Value requires statement[a]	delete	0.02	0.02	0.02	0.09	0.09	0.09
	CorHist	0.90	0.69	0.78	0.89	0.39	**0.55**
	Bass	0.98	0.75	**0.85**	0.72	0.32	0.44
Conflict with	delete	0.39	0.39	0.39	0.44	0.44	0.44
	CorHist	0.87	0.84	0.86	0.83	0.46	0.59
	Bass	0.91	0.86	**0.88**	0.77	0.71	**0.74**
Inverse/Sym.	add	0.91	0.91	0.91	0.82	0.82	0.82
	delete	0.07	0.07	0.07	0.11	0.11	0.11
	CorHist	0.94	0.92	0.93	0.90	0.73	**0.81**
	Bass	0.97	0.94	**0.95**	0.87	0.58	0.69
Single value	delete	0.45	0.45	0.45	0.42	0.42	0.42
	CorHist	0.55	0.50	0.53	0.74	0.23	0.36
	Bass	0.74	0.64	**0.69**	0.60	0.52	**0.56**
Distinct values	delete	0.55	0.55	0.55	0.45	0.45	0.45
	CorHist	0.58	0.57	**0.57**	0.80	0.26	0.39
	Bass	0.59	0.56	**0.57**	0.48	0.43	**0.46**
Total	add	0.46	0.14	0.22	0.33	0.11	0.16
	delete	0.24	0.24	0.24	0.22	0.22	0.22
	CorHist-min	0.80	0.59	0.67	0.89	0.26	0.40
	CorHist	0.75	0.64	0.69	0.85	0.30	0.44
	Bass-RL	0.77	0.65	0.70	0.64	0.34	0.44
	Bass	0.80	0.68	**0.73**	0.69	0.39	**0.49**

[a] The "add" baseline was not able to predict anything and is omitted here.

Bass to outperform Bass-RL by 5% in macro average F-score and by 3% in micro average, suggesting that textual data and structured representation of the constraints help to solve violations of under-represented constraints. However, CorHist, being based on rules, can explain why a given correction is made of a constraint violation, whereas Bass is unable to provide such explanations.

6.8.5 Bass Ablation Study

To understand the contribution of each component of Bass, we remove the components one by one and measure the performance. This leads to the following variants of Bass:

- *Bass without object literals*: The RDF literals embedding component described in Section 6.7.2 is removed.
- *Bass without entity facts*: The embeddings of the facts about the entities that appear in the violation triples described in Section 6.7.1 is removed.
- *Bass without entity facts*: The constraint is not given at all to the neural network. The network only predicts the correction from the violation triples and the facts about the entities that appear in the violation triples.
- *Bass with one hidden layer edit predictor*: The perceptron described in Section 6.7.1 which combines the various embeddings built by the neural network is composed of one dense layer instead of two.

We also add another ablation, *Bass minimal*. This variant removes from Bass-RL the constraint embedding and the entity fact embedding. This leads to a network where the edit prediction component receives only the violation triple predicates and objects, embedded with $\mathcal{P}$ and $\mathcal{E}$.

Additionally to the ablation, we investigate some possible variants of Bass:

- *Bass with constraint ids*: We replace the constraint description input by the Bass-RL constraint ID encoding. This gives the exact constraint to the model, without the possibility to generalize from the constraint description.
- *Bass with BiLSTM literals*: We replace the max-pooling layer in the literal embeddings by a bidirectional long short-term memory network (BiLSTM) (Hochreiter and Schmidhuber, 1997). BiLSTMs are commonly used layers for sequence embeddings and have been used successfully on tasks like natural language translation, suggesting possible good performance for literal embeddings.

Table 6.3 – Bass ablation study results.

	Micro average			Macro average		
Approach	Prec.	Rec.	F	Prec.	Rec.	F
Bass	0.80	0.68	**0.73**	0.69	0.39	**0.49**
Bass-RL	0.77	0.65	0.70	0.64	0.34	0.44
Bass minimal	0.65	0.53	0.58	0.50	0.25	0.33
Bass without object literals	0.77	0.65	0.71	0.66	0.36	0.47
Bass without entity facts	0.70	0.59	0.64	0.54	0.31	0.39
Bass without constraint	0.77	0.65	0.71	0.60	0.35	0.44
Bass with one hidden layer edit predictor	0.79	0.67	0.72	0.69	0.38	**0.49**
Bass with constraint ids	0.80	0.68	**0.73**	0.66	0.37	0.47
Bass with BiLSTM literals	0.78	0.66	0.71	0.65	0.37	0.47
Bass with entity facts attention	0.76	0.65	0.70	0.63	0.36	0.46
CorHist	0.75	0.64	0.69	0.85	0.30	0.44
Deletion baseline	0.24	0.24	0.24	0.22	0.22	0.22
Addition baseline	0.46	0.14	0.22	0.33	0.11	0.16

- *Bass with entity fact attention*: We replace the max-pooling layer that aggregates the embeddings of the (predicate, object) tuples of the involved entities by an attention layer. This attention layer uses for context the constraint embedding. This allows the network to change the relative importance of some entity facts based on the considered constraint.We define the attention following Luong, Pham, and Manning, 2015 as $\sigma(q \cdot V^\top) \cdot V$, where σ is the softmax function, q the query vector (here: the context embedding), and V the value sequence matrix.

Discussion

Using the description of the constraints instead of the constraint ids does not change the micro average score, but increases the macro average scores. This means that the change helps for constraints with only a few past corrections, which have a weaker weight in the micro average. We thus observe a better generalization with respect to the constraints. Embedding the object literal values brings 2% of improvement on both F-scores, showing it brings some value on the 13% of violations where one of the objects is a literal that is not a date or a geographical coordinate. The addition of a bidirectional LSTM and attention turns out to be detrimental to the performance of Bass.

Table 6.4 – Human evaluation of the suggested corrections.

Constraint type	Suggested	"Apply"	"Wrong"	Approval
Type	9908	252	312	0.45
Value type	2374	195	208	0.48
One-of	239	14	47	0.23
Item requires stm.	41	8	32	0.2
Value requires stm.	1024	790	178	0.82
Conflict with	3254	1717	203	0.89
Inverse/Symmetric	28138	20247	1720	0.92
Single value	3264	41	71	0.37
Distinct values	921	8	23	0.26
All	49163	23272	2794	0.89

6.8.6 User Evaluation

To see whether our corrections are accepted by the community, we designed a user study. We created a tool that suggests our corrections to Wikidata users for validation (available at `https://tools.wmflabs.org/wikidata-game/distributed/#game=43`). The user can choose a constraint type, and the tool then suggests corrections for random violations of constraints of this type (Figure 6.3). The violations for which corrections are suggested are provided by `query.wikidata.org`, which limits their number for performance reasons. For each proposed correction, the user has to choose between three options: apply the proposed correction to Wikidata, tag it as wrong, or get another correction to review. We use a simpler version of CorHist to predict the constraint violation than the one presented in this thesis. The version used for this human evaluation is the same as the one presented in Pellissier Tanon, Bourgaux, and Suchanek, 2019. We ran the experiment for 3 months and 47 Wikidata users participated.

The results are presented in Table 6.4. The number of corrections reviewed is highly unbalanced between the kinds of constraints, mainly because a few users evaluate a lot of suggestions, and have a predilection for some kinds of constraints. It is thus difficult to conclude those kinds of constraints for which very few corrections have been evaluated. However, we can still make some interesting observations. In particular, looking at the proposed corrections marked as wrong gives us insights about possible weaknesses of our approach.

For the constraints which got a significant number of evaluations, our approach seems to perform well for *Inverse* and *Symmetric*, *Conflict with* and *Value requires*

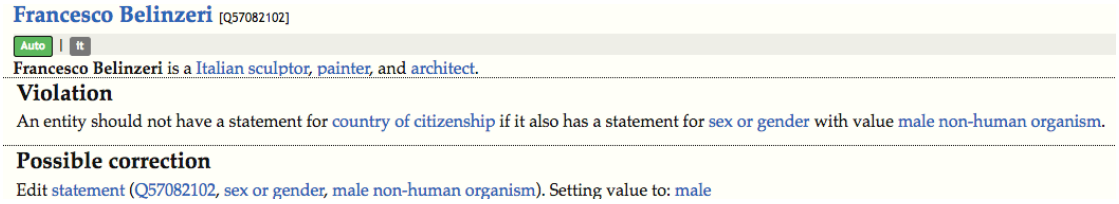

Figure 6.3 – Example of a replacement correction suggested by CorHist for a violation of the constraint $\Gamma(x) = \exists c\ countryOfCitizenship(x, c) \wedge sexOrGender(x,$ `maleOrg`$) \rightarrow$ `false`.

statement constraints, with approval rates above 80%.

The other approval rates are lower. This is partly due to biases in the data. For example, when gender is missing, our approach proposes the value "male" by default, because of the over-representation of men in Wikidata (Klein et al., 2016). This is in line with a common problem with machine learning approaches that reflects and amplifies the biases contained in the data. For example, see Zhao et al., 2019, a recent work on trying to remove the bias of word embeddings. Another issue is the quality of the constraints, which in Wikidata are sometimes questionable or difficult to understand (e.g., an incomplete set of possible types or values for completeness or *One-of* constraints).

Most of the approval rates are lower than the precision reported by the evaluation based on past corrections. We do not know why the results are different, one hypothesis is that the "easy" constraints violations are solved quickly, sometimes with the help of bots. Thus, in the current state of Wikidata, it could be the case that most of the remaining constraint violations are "hard", in the sense that they cannot be easily fixed. Investigating this in more detail is left for future work.

However, even lower approval scores do not mean that our approach would be useless: Psychological research (Funk and Dickson, 2011) shows that people find it much easier to choose from given options than to come up with an answer by themselves. The actual time needed to come up with an answer may vary, but if it takes just 3 times longer to come up with an answer than to accept or reject our proposed correction, then achieving a precision of 40% is already useful.

6.9 Conclusion

In this chapter, we have introduced the problem of learning how to fix constraint violations from a knowledge base history. We have presented two methods to address this problem, one based on rule mining and another on neural networks, the latter providing better performances but without explainability. Our experimental evaluation on Wikidata shows significant improvement over static baselines. Our

tool is live on Wikidata and has already allowed users to correct more than 23k constraint violations.

While our evaluation focused on Wikidata for which the whole edit history was available, our method can be applied in other settings. For example, it might be interesting to investigate to what extent the approaches presented here could be used to clean automatically extracted knowledge bases, using as ground truth the changes done in the knowledge base data sources. It will be also useful to do again the user evaluation with the latest version of the CorHist algorithm and with the Bass algorithm to see which one of the two works best with the violations currently present in Wikidata. Another possible improvement for the user evaluation tool is to allow the users to validate not only the corrections but the rules themselves, to apply them directly on all of Wikidata.

To evaluate the methods presented in this chapter we used Wikidata. However, to apply the past correction extraction algorithm, we needed a system that can easily and efficiently query Wikidata at any time in its edit history. This led us to build a database system tailored for this usage. It is described in the next chapter.

Chapter 7

Querying the Edit History of Wikidata

The work presented in this section has been done with Fabian Suchanek and has been demoed at ESWC 2019.

> Thomas Pellissier Tanon and Fabian M. Suchanek. "Querying the Edit History of Wikidata". Demo at ESWC 2019. https://doi.org/10.1007/978-3-030-32327-1_32

7.1 Introduction

To be able to implement our mining approach from Section 6.5, we needed to be able to evaluate queries at any time of the Wikidata history. Easy access to the Wikidata edit history is useful also to Wikidata contributors who wish to trace the progress of Wikidata over time, measure the number of contributions by bots, or identify areas of vandalism.

However, the only way to access the full historical revisions of Wikidata is to download a 250GB set of compressed XML dumps, which contain a JSON blob for each revision. This dump is hard to manipulate and even harder to index. It would be prohibitively expensive to create one RDF graph for each of the 700 million revisions, each with billions of triples. If one indexes only the differences between the revisions, one loses the global state at each revision.

In this chapter, we propose a system that smartly indexes the revisions, so that the full history of Wikidata edits can be made available as a SPARQL endpoint. Our endpoint can:

1. Retrieve the diff, i.e., the set of added and/or removed triples, for any Wikidata revision.

2. Evaluate any triple pattern against the global state of Wikidata after a given revision.

3. Retrieve the revisions that have added/removed triples matching a given triple pattern.

With this, we can answer questions such as "How many cities existed in Wikidata across time?", "How many entities were modified exclusively by bots?", or "How were different values for the gender property used across time in Wikidata?". All these queries can be asked in standard SPARQL, without any additions to the language. This model also addresses the requirements of Chapter 6.

7.2 Related Work

SPARQL endpoints for versioned RDF datasets have been proposed by several authors. Fernández, Polleres, and Umbrich, 2015 first discussed the problem of querying linked data archives. Fernández, Umbrich, et al., 2016 provides an extensive discussion of known solutions. Our work is concerned with an actual implementation of such a versioned RDF store for Wikidata. This causes practical problems of size that have not been considered in previous works.

Several systems allow storing versioned RDF: Quad stores (e.g., Bishop et al., 2011) and archives (e.g., Fernández, Martínez-Prieto, et al., 2018) could be used to store each triple annotated with their revision. However, this would mean that each triple would have to be stored once for every revision in which it appears. In the case of Wikidata, this would amount to quadrillions of triples to be stored. Pugliese, Udrea, and Subrahmanian, 2008 presented `tgrin` system that allows annotating RDF triples with a timestamp. However, the system does not provide time range support or, indeed, an available implementation. R&Wbase (Sande et al., 2013) is a wrapper on top of a SPARQL 1.1 endpoint that provides a git-like system for RDF graphs. However, it stores only the diff between different revisions. Therefore, it does not allow efficient querying of the full graph state at a given point of time. Neumann and Weikum, 2010 presents x-RDF-3X, a SPARQL database that annotates each triple with the timestamp at which it was added or removed. It annotates triples with validity ranges, thus permitting queries for any state of the database. It also provides advanced consistency features. However, by design, it does not allow loading data with already known timestamps or version IDs. Thus, it is not usable in our case. The system `v-RDFCSA` from Cerdeira-Pena et al., 2016 allows efficient storage and retrieval of versioned triples. However,

it does not allow subsequent additions of revisions and does not support revision ranges. Therefore, the system is hard to use with the huge number of revisions that Wikidata brings. OSTRICH (Taelman, Sande, and Verborgh, 2018) allows storing version-annotated triples, as well as querying them based on the version ID. It uses an HDT file for storing the base version and then stores a changeset with respect to this initial version. This system is not tailored to the case of Wikidata, where the base version is empty. Thus, the system would have to store the full global state for each revision.

7.3 System Overview

Our goal is to provide a SPARQL endpoint that allows querying not just for the differences between Wikidata revisions, but also for the global state after each revision. At the same time, we want to use only existing SPARQL features. For this purpose, we designed the following data model based on RDF 1.1 named graphs (Cyganiak et al., 2014):

Global State Graph: We will have one named graph per revision, which contains the global state of Wikidata after the revision has been saved.

Addition Graph: For each revision, we will have one graph that contains all the triples that have been added by this revision.

Deletion Graph: In the same spirit, we will have one graph for each revision that stores the triples that were removed.

Default Graph: The default graph contains triples that encode metadata about each revision: the revision author, the revision timestamp, the id of the modified entity, the previous revision of the same entity, the previous revision in Wikidata, and the IRIs of the additions, deletions, and global state graphs.

This data model allows us to query both the global state of Wikidata after each revision and the modifications brought by reach revision – without any change of the SPARQL 1.1 semantics or syntax (Harris, Seaborne, and Prud'hommeaux, 2013). We use the same schema as the official Wikidata dumps (Erxleben et al., 2014).

To store our data model, we cannot just load it as-is into a triple store. The revision graphs alone would occupy exabytes of storage. Indeed, the global state after each Wikidata revision contains billion of triples, and a naive storage would mean to duplicate all of the triples for each revision. Therefore, we implemented our own system. Our system is composed of a storage component able to store triples annotated with a graph name and query them using triple patterns. Then

we have a SPARQL evaluator interacting with the storage component using triple patterns.

The storage component is based on RocksDB[1], which we already used in Chapter 5 for YAGO 4. RocksDB is a scalable key-value store, optimized for a mixed workload between query and edits. This choice anticipates the possibility of live updates from Wikidata in the future. We create the following indexes into RocksDB:

Dictionary Indexes: Following a well-known practice in RDF storage systems, we represent every string by an integer id. We use one dictionary index to map the strings to their ids, and another one to map the ids to the strings.

Content Indexes: Each triple (s, p, o) appears in three content indexes. The indexes have as keys the permutations *spo*, *pos*, and *osp*, respectively, and as value a set of revision ranges. Each range is of the form $[start, end[$, where *start* is the id of the revision that introduced the triple, and *end* is the id of the revision that removed it (or $+\infty$ if the triple has not been removed).

Revision indexes: We use two revision indexes, which give the set of triples that have been added or removed by a given revision. Since the Wikidata edits affect only a single entity and usually only a single statement, the number of triples added and removed per edit is small. Therefore, we can easily store all of them in the value part of the key-value store.

Meta indexes: We use several metadata indexes to store, for each revision, its author, the previous revision, etc.

Thanks to the content indexes, it is easy to retrieve for a given triple pattern when the triples have been added and, if relevant, when they have been removed. The indexes can also be used to evaluate a triple pattern against the global state graph, by filtering the results against the revision ranges. This is efficient because out of the 4931M triples that have existed in the Wikidata history as of July 1st, 2018, only 475M has been removed. Therefore, the largest revision contains 90% of the total number of Wikidata triples that ever existed.

These indexes allow us to evaluate all possible triple patterns, optionally annotated with a graph name. If the triple pattern targets revision metadata, we use the metadata indexes. If it targets the content triples, we proceed with the following operations on top of the RocksDB indexes:

1. If the graph name is set to an addition graph or a deletion graph, we use the revision indexes to retrieve all the triples that were added or deleted, and we evaluate the triple pattern on them.

[1] `https://rocksdb.org/`

2. Otherwise, we do a prefix search in one of the content indexes, building the prefix from the bound parts of the triple pattern and choosing the index that allows us to have the longest prefix, and so the highest selectivity.
   (a) If the graph name is set and is a global state graph, we filter the triples that are returned from the prefix search by using the revision range as a filter.
   (b) Otherwise, we iterate through the matching triples and annotate them with graph names by using, for each triple, the revision range to find the ids of the addition graphs and the deletion graphs in which the triple appears. We do not return the ids of the global state graphs, because a query for the graph identifiers of a single triple pattern could return hundreds of millions of ids.

This storage system allows us to answer queries for a single triple pattern annotated with a graph name. To answer SPARQL queries, we plug our triple pattern evaluator into Eclipse RDF4J[2]. This system evaluates an arbitrary SPARQL 1.1 query by repeated calls to our triple pattern evaluator. It supports a wide range of query optimizations, including join reordering with handwritten static cardinality estimations that are based on the structure of the RDF dataset. For example, every revision has exactly a single author, and the number of triples per Wikidata entity is usually small.

Due to storage space constraints on the server provided by the Wikimedia Foundation, the facts annotations (qualifiers and references) are not loaded in our demo endpoint[3]. We cover the range from the creation of Wikidata to July 1st, 2018. Our demonstration instance stores more than 700M revisions and 390M content triples about 49M items. The RocksDB indexes are using 64Go on disk with the RocksDB gzip compression system, after having compacted the database.

7.4 Usage

Our system allows users to query the entire Wikidata edit history. For example, Figure 7.1 asks for the 10 most frequent replacements for the value of the "gender" property. The query retrieves first the set of triples with the gender property (`wdt:P21`) (Line 2), annotated with the name of addition graphs. Then, it retrieves the names of the deletion graphs for each revision that have seen such additions (Line 3). Finally, it retrieves any deleted triple with the same subject and predicate (Line 4). After that, it uses the usual SPARQL 1.1 features to compute the final result (Lines 1 and 5).

[2] `http://rdf4j.org/`
[3] `https://wdhqs.wmflabs.org`

```
1 SELECT ?genderAdd ?genderDel (COUNT(?revision) AS ?c) WHERE {
2   GRAPH ?addGraph { ?s wdt:P21 ?genderAdd }
3   ?revision hist:additions ?addGraph ; hist:deletions ?delGraph .
4   GRAPH ?delGraph { ?s wdt:P21 ?genderDel }
5 } GROUP BY ?genderAdd ?genderDel ORDER BY DESC(?c) LIMIT 10
```

Figure 7.1 – Retrieving the most common replacements of a "gender" value.

Such an analysis yields some interesting insights: First, we can observe that users correct erroneous gender values (such as "man") by their intended variants ("male"). Second, a slightly modified variant of the query allows us to quantify the gender gap in Wikidata over time: while the absolute difference between the number of men and women in Wikidata keeps increasing over time, the relative difference decreases. Finally, a similar query allows us to trace the increasing presence of other sex/gender values, such as "trans-male" or "non-binary" in the dataset.

7.5 Conclusion

In this chapter, we proposed a system that efficiently indexes the entire Wikidata edit history, and that allows users to answer arbitrary SPARQL 1.1 queries on it. Our system allows queries on both the revision diffs and the global state of Wikidata after each revision. Our system is available online at `https://wdhqs.wmflabs.org`. We also provide an extensive help page[4], together with a set of example queries. Between March 2019 and May 2020, it has served more than 2M queries.

However, our endpoint has still a major limitation: updating its contents requires new full Wikidata revisions dumps. These dumps are only released every few months. This prevents us to have speedy updates of the endpoint content and forbids interesting use-cases like vandalism detection and or live suggestions based on the latest user edits. Hence, our main future work is making our endpoint replicate Wikidata edits in real-time using the live edit stream provided by Wikidata.

We have successfully used this system for the approaches presented in the previous chapter to learn how to fix constraint violations. This concludes our efforts at learning to dynamically enforce constraints on a knowledge base.

[4] `https://www.wikidata.org/wiki/Wikidata:History_Query_Service`

Chapter 8

Conclusion

8.1 Summary

In this thesis, we have studied the correction and completion of knowledge bases. We made three core contributions.

In Part I, we have defined the problem of learning rules from incomplete knowledge bases. We introduced the *completeness confidence* ranking measure that effectively makes use of the cardinality information contained in the knowledge base. Our new measure has been evaluated on real-world and synthetic knowledge bases, demonstrating significant improvements both with respect to the quality of mined rules and with respect to the predictions that they produce. We have also proposed a method to automatically increase the coverage of cardinality information. This method can be used to improve the efficiency of our new measure.

In Part II, we have presented is YAGO 4, a knowledge base that, during its construction, enforces constraints over the knowledge base by applying simple static repair strategies. This method allowed us to build a large scale knowledge base that can be used with regular OWL reasoners.

In Part III, we aimed at avoiding the loss of data that our static repair strategies induced in YAGO 4. To achieve that, we introduced the problem of learning how to fix constraint violations from a knowledge base history. We have presented two methods to this end. One, CorHist, is based on rule mining. Its advantage is that it can explain its corrections. The other, Bass, is based on neural networks and offers better accuracy. However, it is not able to explain its predictions. An experimental evaluation on Wikidata showed that these two approaches are significantly outperforming static baselines to find the correct correction. We also provided the system as a tool to the Wikidata community, allowing it to fix more than 20k constraints violations.

8.2 Outlook

In this thesis, we have shown how constraints can be used in practice to clean knowledge bases. Many research challenges are still open.

The approach used to increase the number of cardinality statements could be expanded to mine rules on numerical values. For example, it could be used to learn that if a country is member of the OCDE then its GDP is higher than a certain threshold, or that all Roman consuls are born before the 10th century.

The work on learning how to correct violations of constraints from past corrections could be extended to textual data. For example, one could try to mine how to fix common spelling mistakes or improve the formatting of written works based on the changes previously done. Another application might be to mine from a source code edit history the common formatting changes and simple bug fixes to automatically suggest them when a contributor opens a code review request.

We hope that, by opening the door to these challenges, our work can contribute to making knowledge bases and other contents cleaner, and thus ultimately ever more useful.

Bibliography

Abadi, Martı'n, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2016). "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *CoRR*. URL: `http://arxiv.org/abs/1603.04467` (cit. on p. 101).

Acosta, Maribel, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Fabian Flöck, and Jens Lehmann (2018). "Detecting Linked Data quality issues via crowdsourcing: A DBpedia study". In: *Semantic Web* 9.3, pp. 303–335. DOI: `10.3233/SW-160239` (cit. on pp. 76, 142).

Agrawal, Rakesh, Tomasz Imielinski, and Arun N. Swami (1993). "Mining Association Rules between Sets of Items in Large Databases". In: *SIGMOD*, pp. 207–216. DOI: `10.1145/170035.170072` (cit. on p. 22).

Arioua, Abdallah and Angela Bonifati (2018). "User-guided Repairing of Inconsistent Knowledge Bases". In: *EDBT*, pp. 133–144. DOI: `10.5441/002/edbt.2018.13` (cit. on pp. 75, 142).

Assadi, Ahmad, Tova Milo, and Slava Novgorodov (2018). "Cleaning Data with Constraints and Experts". In: *International Workshop on the Web and Databases*, 1:1–1:6. DOI: `10.1145/3201463.3201464` (cit. on pp. 75, 142).

Baader, Franz, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press. ISBN: 0-521-78176-0 (cit. on pp. 17, 19).

Bader, Johannes, Andrew Scott, Michael Pradel, and Satish Chandra (2019). "Getafix: Learning to Fix Bugs Automatically". In: *Proc. ACM Program. Lang.* 159:1–159:27. DOI: `10.1145/3360585` (cit. on p. 76).

Bergman, Moria, Tova Milo, Slava Novgorodov, and Wang-Chiew Tan (2015). "QOCO: A Query Oriented Data Cleaning System with Oracles". In: *VLDB* 8.12, pp. 1900–1903. DOI: 10.14778/2824032.2824096 (cit. on pp. 75, 142).

Bienvenu, Meghyn, Camille Bourgaux, and François Goasdoué (2016). "Query-Driven Repairing of Inconsistent DL-Lite Knowledge Bases". In: *IJCAI*, pp. 957–964. URL: https://www.ijcai.org/Proceedings/16/Papers/140.pdf (cit. on pp. 75, 142).

Bishop, Barry, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov (2011). "OWLIM: A Family of Scalable Semantic Repositories". In: *Semantic Web* 2.1, pp. 33–42. DOI: 10.3233/SW-2011-0026 (cit. on p. 112).

Bizer, Christian, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann (2009). "DBpedia - A Crystallization Point for the Web of Data". In: *Journal of Web Semantics* 7.3, pp. 154–165. DOI: 10.1016/j.websem.2009.07.002 (cit. on pp. 11, 131).

Bollacker, Kurt D., Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor (2008). "Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge". In: *SIGMOD*, pp. 1247–1250. DOI: 10.1145/1376616.1376746 (cit. on pp. 11, 131).

Boneva, Iovka, José Emilio Labra Gayo, and Eric G. Prud'hommeaux (2017). "Semantics and Validation of Shapes Schemas for RDF". In: *ISWC*, pp. 104–120. DOI: 10.1007/978-3-319-68288-4_7 (cit. on p. 75).

Brin, Sergey, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur (1997). "Dynamic Itemset Counting and Implication Rules for Market Basket Data". In: *SIGMOD*, pp. 255–264. DOI: 10.1145/253260.253325 (cit. on p. 34).

Calvanese, Diego, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati (2007). "Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family". In: *J. Autom. Reasoning* 39.3, pp. 385–429. DOI: 10.1007/s10817-007-9078-x (cit. on p. 82).

Cerdeira-Pena, Ana, Antonio Fariña, Javier D. Fernández, and Miguel A. Martı'nez-Prieto (2016). "Self-Indexing RDF Archives". In: *DCC*, pp. 526–535. DOI: 10.1109/DCC.2016.40 (cit. on p. 112).

Chen, Jiaoyan, Xi Chen, Ian Horrocks, Ernesto Jiménez-Ruiz, and Erik B. Myklebust (2020). "Correcting Knowledge Base Assertions". In: *CoRR*. URL: https://arxiv.org/abs/2001.06917 (cit. on p. 75).

Chollet, François et al. (2015). *Keras*. URL: https://keras.io (cit. on p. 101).

Cyganiak, Richard, David Wood, Markus Lanthaler, Graham Klyne, Jeremy J Carroll, and Brian McBride (2014). *RDF 1.1 concepts and abstract syntax*. W3C Recommendation. World Wide Web Consortium. URL: https://www.w3.org/TR/rdf11-concepts/ (cit. on pp. 18, 75, 113, 136).

d'Amato, Claudia, Steffen Staab, Andrea GB Tettamanzi, Tran Duc Minh, and Fabien Gandon (2016). "Ontology Enrichment by Discovering Multi-Relational Association Rules from Ontological Knowledge Bases". In: *SAC*, pp. 333–338. DOI: 10.1145/2851613.2851842 (cit. on p. 32).

Darari, Fariz, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski (2013). "Completeness Statements about RDF Data Sources and Their Use for Query Answering". In: *ISWC*, pp. 170–187. DOI: 10.1007/978-3-642-41335-3_5 (cit. on pp. 21, 33, 45, 139).

Darari, Fariz, Simon Razniewski, Radityo Eko Prasojo, and Werner Nutt (2016). "Enabling Fine-Grained RDF Data Completeness Assessment". In: *ICWE*, pp. 170–187. DOI: 10.1007/978-3-319-38791-8_10 (cit. on pp. 45, 139).

Dehaspe, Luc and Luc De Raedt (1997). "Mining Association Rules in Multiple Relations". In: *ILP*, pp. 125–132. DOI: 10.1007/3540635149_40 (cit. on pp. 22, 23).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*, pp. 4171–4186. DOI: 10.18653/v1/n19-1423 (cit. on p. 93).

Diefenbach, Dennis, Thomas Pellissier Tanon, Kamal Deep Singh, and Pierre Maret (2017). "Question Answering Benchmarks for Wikidata". In: *ISWC*. URL: http://ceur-ws.org/Vol-1963/paper555.pdf.

Doppa, Janardhan Rao, Shahed Sorower, Mohammad Nasr Esfahani, John Walker Orr, Thomas G. Dietterich, Xiaoli Z. Fern, Prasad Tadepalli, and Jed Irvine (2011). "Learning Rules from Incomplete Examples via Implicit Mention Models". In: *ACML*, pp. 197–212. URL: http://proceedings.mlr.press/v20/doppa11/doppa11.pdf (cit. on pp. 31, 139).

Erxleben, Fredo, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić (2014). "Introducing Wikidata to the Linked Data Web". In: *ISWC*, pp. 50–65. DOI: 10.1007/978-3-319-11964-9_4 (cit. on pp. 96, 113).

Etzioni, Oren, Keith Golden, and Daniel S. Weld (1997). "Sound and Efficient Closed-World Reasoning for Planning". In: *Artif. Intell.* 89.1-2, pp. 113–148. DOI: 10.1016/S0004-3702(96)00026-4 (cit. on p. 33).

Fellbaum, Christiane, ed. (1998). *WordNet: An Electronic Lexical Database*. MIT Press. ISBN: 9780262061971. URL: https://mitpress.mit.edu/books/wordnet (cit. on p. 53).

Fernández, Javier D., Miguel A. Martínez-Prieto, Axel Polleres, and Julian Reindorf (2018). "HDTQ: Managing RDF Datasets in Compressed Space". In: *ESWC*, pp. 191–208. DOI: 10.1007/978-3-319-93417-4_13 (cit. on p. 112).

Fernández, Javier D., Axel Polleres, and Jürgen Umbrich (2015). "Towards Efficient Archiving of Dynamic Linked Open Data". In: *DIACRON @ ESWC*, pp. 34–49. URL: http://ceur-ws.org/Vol-1377/paper6.pdf (cit. on p. 112).

Fernández, Javier D., Jürgen Umbrich, Axel Polleres, and Magnus Knuth (2016). "Evaluating Query and Storage Strategies for RDF Archives". In: *SEMANTICS*, pp. 41–48. DOI: 10.1145/2993318.2993333 (cit. on p. 112).

Flesca, Sergio, Sergio Greco, and Ester Zumpano (2004). "Active Integrity Constraints". In: *PPDP*, pp. 98–107. DOI: 10.1145/1013963.1013977 (cit. on p. 75).

Frey, Johannes, Marvin Hofer, Daniel Obraczka, Jens Lehmann, and Sebastian Hellmann (2019). "DBpedia FlexiFusion the Best of Wikipedia > Wikidata > Your Data". In: *ISWC*, pp. 96–112. DOI: 10.1007/978-3-030-30796-7_7 (cit. on pp. 55, 57).

Funk, Steven C. and K. Laurie Dickson (2011). "Multiple-Choice and Short-Answer Exam Performance in a College Classroom". In: *Teaching of Psychology* 38.4, pp. 273–277. DOI: 10.1177/0098628311421329 (cit. on p. 108).

Gad-Elrab, Mohamed H., Daria Stepanova, Jacopo Urbani, and Gerhard Weikum (2016). "Exception-enriched rule learning from knowledge graphs". In: *ISWC*, pp. 234–251. DOI: 10.1007/978-3-319-46523-4_15 (cit. on p. 30).

Galárraga, Luis, Simon Razniewski, Antoine Amarilli, and Fabian M. Suchanek (2017). "Predicting Completeness in Knowledge Bases". In: *WSDM*, pp. 375–383. DOI: 10.1145/3018661.3018739 (cit. on pp. 31, 33, 43, 45, 139).

Galárraga, Luis, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek (2013). "AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases". In: *WWW*, pp. 413–422. DOI: 10.1145/2488388.2488425 (cit. on pp. 30, 31, 77, 86, 87).

— (2015). "Fast Rule Mining in Ontological Knowledge Bases with AMIE+". In: *VLDB* 24.6, pp. 707–730. DOI: 10.1007/s00778-015-0394-1 (cit. on pp. 22, 24, 30, 31, 34, 39, 46, 86, 138, 144).

Giacometti, Arnaud, Béatrice Markhoff, and Arnaud Soulet (2019). "Mining Significant Maximum Cardinalities in Knowledge Bases". In: *ISWC*. Vol. 11778, pp. 182–199. DOI: 10.1007/978-3-030-30793-6_11 (cit. on pp. 45, 49).

Glimm, Birte, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang (2014). "HermiT: An OWL 2 Reasoner". In: *J. Autom. Reasoning* 53.3, pp. 245–269. DOI: 10.1007/s10817-014-9305-1 (cit. on p. 68).

Goethals, Bart and Jan Van den Bussche (2002). "Relational Association Rules: Getting WARMeR". In: *Pattern Det. and Disc. Workshop*, pp. 125–139. DOI: 10.1007/3-540-45728-3_10 (cit. on pp. 38, 39).

Gray, Alasdair J. G., Carole A. Goble, and Rafael Jimenez (2017). "Bioschemas: From Potato Salad to Protein Annotation". In: *ISWC*. URL: http://ceur-ws.org/Vol-1963/paper579.pdf (cit. on p. 57).

Guha, Ramanathan V., Dan Brickley, and Steve Macbeth (2016). "Schema.org: Evolution of Structured Data on the Web". In: *Commun. ACM* 59.2, pp. 44–51. DOI: 10.1145/2844544 (cit. on p. 55).

Guo, Yuanbo, Zhengxiang Pan, and Jeff Heflin (2005). "LUBM: A benchmark for OWL knowledge base systems". In: *J. Web Semant.* 3.2-3, pp. 158–182. DOI: 10.1016/j.websem.2005.06.005 (cit. on pp. 39, 141).

Harris, Steve, Andy Seaborne, and Eric Prud'hommeaux (2013). *SPARQL 1.1 Query Language*. W3C Recommendation. World Wide Web Consortium. URL: https://www.w3.org/TR/sparql11-query/ (cit. on p. 113).

Hartig, Olaf (2017). "Foundations of RDF⋆ and SPARQL⋆ (An Alternative Approach to Statement-Level Metadata in RDF)". In: *Alberto Mendelzon Workshop on Foundations of Data Management and the Web*. URL: http://ceur-ws.org/Vol-1912/paper12.pdf (cit. on pp. 63, 66).

Hernández, Daniel, Aidan Hogan, and Markus Krötzsch (2015). "Reifying RDF: What Works Well With Wikidata?" In: *SSWS@ISWC*, pp. 32–47. URL: http://ceur-ws.org/Vol-1457/SSWS2015_paper3.pdf (cit. on p. 96).

Ho, Vinh Thinh, Daria Stepanova, Mohamed H. Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum (2018). "Rule Learning from Knowledge Graphs Guided by Embedding Models". In: *ISWC*, pp. 72–90. DOI: 10.1007/978-3-030-00671-6_5 (cit. on pp. 32, 76).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. on p. 105).

Hoffart, Johannes, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum (2011). "YAGO2: Exploring and Querying World Knowledge in Time, Space, Context, and Many Languages". In: *WWW*, pp. 229–232. DOI: 10.1145/1963192.1963296 (cit. on p. 53).

Hohenecker, Patrick and Thomas Lukasiewicz (2018). "Ontology Reasoning with Deep Neural Networks". In: *CoRR*. URL: http://arxiv.org/abs/1808.07980 (cit. on p. 76).

Ismayilov, Ali, Dimitris Kontokostas, Sören Auer, Jens Lehmann, and Sebastian Hellmann (2018). "Wikidata through the Eyes of DBpedia". In: *Semantic Web* 9.4, pp. 493–503. DOI: 10.3233/SW-170277 (cit. on p. 55).

Józefowska, Joanna, Agnieszka Lawrynowicz, and Tomasz Lukaszewski (2010). "The Role of Semantics in Mining Frequent Patterns from Knowledge Bases in Description Logics with Rules". In: *TPLP* 10.3, pp. 251–289. DOI: 10.1017/S1471068410000098 (cit. on pp. 32, 76).

Kalyanpur, Aditya, Bijan Parsia, Matthew Horridge, and Evren Sirin (2007). "Finding All Justifications of OWL DL Entailments". In: *ISWC*, pp. 267–280. DOI: 10.1007/978-3-540-76298-0_20 (cit. on p. 79).

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *ICLR*. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 101).

Klein, Maximilian, Harsh Gupta, Vivek Rai, Piotr Konieczny, and Haiyi Zhu (2016). "Monitoring the Gender Gap with Wikidata Human Gender Indicators". In: *OpenSym*, 16:1–16:9. DOI: 10.1145/2957792.2957798 (cit. on p. 108).

Knublauch, Holger and Dimitris Kontokostas (2017). *Shapes Constraint Language (SHACL)*. W3C Recommendation. World Wide Web Consortium. URL: https://www.w3.org/TR/shacl/ (cit. on pp. 55, 60, 75).

Kontchakov, Roman and Michael Zakharyaschev (2014). "An Introduction to Description Logics and Query Rewriting". In: *Reasoning Web. Reasoning on the Web in the Big Data Era*, pp. 195–244. DOI: 10.1007/978-3-319-10587-1_5 (cit. on p. 82).

Kontokostas, Dimitris, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri (2014). "Test-Driven Evaluation of Linked Data Quality". In: *WWW*, pp. 747–758. DOI: 10.1145/2566486.2568002 (cit. on p. 75).

Law, Mark, Alessandra Russo, and Krysia Broda (2014). "Inductive Learning of Answer Set Programs". In: *JELIA*. Vol. 8761, pp. 311–325. DOI: 10.1007/978-3-319-11558-0_22 (cit. on p. 32).

Lehmann, Jens, Sören Auer, Lorenz Bühmann, and Sebastian Tramp (2011). "Class expression learning for ontology engineering". In: *J. Web Sem.* 9.1, pp. 71–81. DOI: 10.1016/j.websem.2011.01.001 (cit. on p. 32).

Lertvittayakumjorn, Piyawat, Natthawut Kertkeidkachorn, and Ryutaro Ichise (2017). "Correcting Range Violation Errors in DBpedia". In: *ISWC*. URL: http://ceur-ws.org/Vol-1963/paper508.pdf (cit. on p. 76).

Levy, Alon Y. (1996). "Obtaining Complete Answers from Incomplete Databases". In: *VLDB*, pp. 402–412. URL: http://www.vldb.org/conf/1996/P402.PDF (cit. on p. 33).

Liang, Jiaqing, Yanghua Xiao, Yi Zhang, Seung-won Hwang, and Haixun Wang (2017). "Graph-Based Wrong IsA Relation Detection in a Large-Scale Lexical Taxonomy". In: *AAAI*, pp. 1178–1184. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14268 (cit. on pp. 76, 142).

Lisi, Francesca A. (2010). "Inductive Logic Programming in Databases: From Datalog to DL+log". In: *Theory Pract. Log. Program.* 10.3, pp. 331–359. DOI: 10.1017/S1471068410000116 (cit. on p. 32).

Liu, Bing, Wynne Hsu, and Yiming Ma (1998). "Integrating Classification and Association Rule Mining". In: *KDD*, pp. 80–86. URL: http://www.aaai.org/Library/KDD/1998/kdd98-012.php (cit. on pp. 88, 144).

Luong, Thang, Hieu Pham, and Christopher D. Manning (2015). "Effective Approaches to Attention-based Neural Machine Translation". In: *EMNLP*, pp. 1412–1421. DOI: 10.18653/v1/d15-1166 (cit. on p. 106).

Mahdisoltani, Farzaneh, Joanna Biega, and Fabian M. Suchanek (2015). "YAGO3: A Knowledge Base from Multilingual Wikipedias". In: *CIDR*. URL: http://cidrdb.org/cidr2015/Papers/CIDR15_Paper1.pdf (cit. on p. 53).

Meilicke, Christian, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt (2018). "Fine-Grained Evaluation of Rule- and Embedding-Based Systems for Knowledge Graph Completion". In: *ISWC*, pp. 3–20. DOI: 10.1007/978-3-030-00671-6_1 (cit. on p. 76).

Melo, André and Heiko Paulheim (2017). "An Approach to Correction of Erroneous Links in Knowledge Graphs". In: *K-CAP*. URL: http://ceur-ws.org/Vol-2065/paper12.pdf (cit. on p. 76).

Melo, Gerard de (2013). "Not Quite the Same: Identity Constraints for the Web of Linked Data". In: *AAAI*. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6491 (cit. on p. 76).

Mirza, Paramita, Simon Razniewski, Fariz Darari, and Gerhard Weikum (2017). "Cardinal Virtues: Extracting Relation Cardinalities from Text". In: *ACL*, pp. 347–351. DOI: 10.18653/v1/P17-2055 (cit. on pp. 45, 139, 140).

Mirza, Paramita, Simon Razniewski, and Werner Nutt (2016). "Expanding Wikidata's Parenthood Information by 178%, or How To Mine Relation Cardinality Information". In: *ISWC*. URL: http://ceur-ws.org/Vol-1690/paper4.pdf (cit. on pp. 30, 34, 48).

Montoya, David, Thomas Pellissier Tanon, Serge Abiteboul, Pierre Senellart, and Fabian M. Suchanek (2018). "A Knowledge Base for Personal Information Management". In: *LDOW@WWW*. URL: http://ceur-ws.org/Vol-2073/article-02.pdf.

Montoya, David, Thomas Pellissier Tanon, Serge Abiteboul, and Fabian M. Suchanek (2016). "Thymeflow, A Personal Knowledge Base with Spatio-temporal Data". In: *CIKM*, pp. 2477–2480. DOI: 10.1145/2983323.2983337.

Motik, Boris, Ian Horrocks, and Ulrike Sattler (2009). "Bridging the gap between OWL and relational databases". In: *J. Web Sem.* 7.2, pp. 74–89. DOI: 10.1016/j.websem.2009.02.001 (cit. on p. 75).

Motik, Boris and Peter F. Patel-Schneider (2012). *OWL 2 Web Ontology Language Mapping to RDF Graphs*. W3C Recommendation. World Wide Web Consortium. URL: http://www.w3.org/TR/owl2-mapping-to-rdf/ (cit. on p. 18).

Motik, Boris, Peter F. Patel-Schneider, and Bernardo Cuenca Grau (2012). *OWL 2 Web Ontology Language Direct Semantics*. W3C Recommendation. World Wide Web Consortium. URL: https://www.w3.org/TR/owl2-direct-semantics/ (cit. on pp. 17, 19, 20).

Motik, Boris, Peter F. Patel-Schneider, and Bijan Parsia (2012). *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. W3C Recommendation. World Wide Web Consortium. URL: http://www.w3.org/TR/owl2-syntax/ (cit. on p. 18).

Muñoz, Emir, Pasquale Minervini, and Matthias Nickles (2019). "Embedding cardinality constraints in neural link predictors". In: *SIGAPP*. DOI: 10.1145/3297280.3297502 (cit. on p. 32).

Neumann, Thomas and Gerhard Weikum (2010). "x-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases". In: *VLDB* 3.1, pp. 256–263. URL: http://www.vldb.org/pvldb/vldb2010/pvldb_vol3/R22.pdf (cit. on p. 112).

Nickel, Maximilian, Volker Tresp, and Hans-Peter Kriegel (2012). "Factorizing YAGO: Scalable Machine Learning for Linked Data". In: *WWW*, pp. 271–280. DOI: 10.1145/2187836.2187874 (cit. on p. 33).

Patel-Schneider, Peter F. (2015). "Using Description Logics for RDF Constraint Checking and Closed-World Recognition". In: *AAAI*, pp. 247–253. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9531 (cit. on p. 75).

Paulheim, Heiko (2017). "Knowledge graph refinement: A survey of approaches and evaluation methods". In: *SW* 8.3, pp. 489–508. DOI: 10.3233/SW-160218 (cit. on p. 30).

Paulheim, Heiko and Christian Bizer (2014). "Improving the Quality of Linked Data Using Statistical Distributions". In: *Int. J. Semantic Web Inf. Syst.* 10.2, pp. 63–86. DOI: 10.4018/ijswis.2014040104 (cit. on pp. 76, 142).

Pellissier Tanon, Thomas, Marcos Dias de Assunção, Eddy Caron, and Fabian M. Suchanek (2018). "Demoing Platypus - A Multilingual Question Answering Platform for Wikidata". In: *ESWC*, pp. 111–116. DOI: 10.1007/978-3-319-98192-5_21.

Pellissier Tanon, Thomas, Camille Bourgaux, and Fabian M. Suchanek (2019). "Learning How to Correct a Knowledge Base from the Edit History". In: *WWW*, pp. 1465–1475. DOI: 10.1145/3308558.331358 (cit. on p. 107).

Pellissier Tanon, Thomas and Lucie-Aimée Kaffee (2018). "Property Label Stability in Wikidata: Evolution and Convergence of Schemas in Collaborative Knowledge Bases". In: *WikiWorkshop@WWW*, pp. 1801–1803. DOI: 10.1145/3184558.3191643 (cit. on p. 59).

Pellissier Tanon, Thomas, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum (2017). "Completeness-Aware Rule Learning from Knowledge Graphs". In: *ISWC*, pp. 507–525. DOI: 10.1007/978-3-319-68288-4_30.
— (2018). "Completeness-aware Rule Learning from Knowledge Graphs". In: *IJCAI*, pp. 5339–5343. DOI: 10.24963/ijcai.2018/749.
Pellissier Tanon, Thomas and Fabian M. Suchanek (2019). "Querying the Edit History of Wikidata". In: *ESWC*, pp. 161–166. DOI: 10.1007/978-3-030-32327-1_32.
Pellissier Tanon, Thomas, Denny Vrandečić, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher (2016). "From Freebase to Wikidata: The great migration". In: *WWW*, pp. 1419–1428. DOI: 10.1145/2872427.2874809 (cit. on p. 48).
Pellissier Tanon, Thomas, Gerhard Weikum, and Fabian M. Suchanek (2020). "YAGO 4: A Reason-able Knowledge Base". In: *ESWC*, pp. 583–596. DOI: 10.1007/978-3-030-49461-2_34.
Prasojo, Radityo Eko, Fariz Darari, Simon Razniewski, and Werner Nutt (2016). "Managing and Consuming Completeness Information for Wikidata Using COOL-WD". In: *COLD@ISWC*. URL: http://ceur-ws.org/Vol-1666/paper-02.pdf (cit. on pp. 30, 139).
Pugliese, Andrea, Octavian Udrea, and V. S. Subrahmanian (2008). "Scaling RDF with Time". In: *WWW*, pp. 605–614. DOI: 10.1145/1367497.1367579 (cit. on p. 112).
Rantsoudis, Christos, Guillaume Feuillade, and Andreas Herzig (2017). "Repairing ABoxes through Active Integrity Constraints". In: *DL*. URL: http://ceur-ws.org/Vol-1879/paper41.pdf (cit. on p. 75).
Razniewski, Simon, Fabian M. Suchanek, and Werner Nutt (2016). "But what do we actually know". In: *AKBC*, pp. 40–44 (cit. on p. 139).
Rebele, Thomas, Thomas Pellissier Tanon, and Fabian M. Suchanek (2018). "Bash Datalog: Answering Datalog Queries with Unix Shell Commands". In: *ISWC*, pp. 566–582. DOI: 10.1007/978-3-030-00671-6_33.
Rebele, Thomas, Fabian M. Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum (2016). "YAGO: A Multilingual Knowledge Base from Wikipedia, Wordnet, and Geonames". In: *ISWC*, pp. 177–185. DOI: 10.1007/978-3-319-46547-0_19 (cit. on pp. 53, 68).
Sande, Miel Vander, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens, and Rik Van de Walle (2013). "R&Wbase: Git for Triples". In: *LDOW @ WWW*. URL: http://ceur-ws.org/Vol-996/papers/ldow2013-paper-01.pdf (cit. on p. 112).
Sazonau, Viachaslau, Uli Sattler, and Gavin Brown (2015). "General Terminology Induction in OWL". In: *ISWC*, pp. 533–550. DOI: 10.1007/978-3-319-25007-6_31 (cit. on pp. 32, 76).

Schlobach, Stefan and Ronald Cornet (2003). "Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies". In: *IJCAI*, pp. 355–362. URL: http://ijcai.org/Proceedings/03/Papers/053.pdf (cit. on p. 79).

Suchanek, Fabian M., Gjergji Kasneci, and Gerhard Weikum (2007). "Yago: a core of semantic knowledge". In: *WWW*, pp. 697–706. DOI: 10.1145/1242572.1242667 (cit. on pp. 11, 53, 131).

Suchanek, Fabian M. and Nicoleta Preda (2014). "Semantic Culturomics". In: *VLDB* 7.12, pp. 1215–1218. URL: http://www.vldb.org/pvldb/vol7/p1215-suchanek.pdf (cit. on p. 30).

Taelman, Ruben, Miel Vander Sande, and Ruben Verborgh (2018). "OSTRICH: Versioned Random-Access Triple Store". In: *WWW*, pp. 127–130. DOI: 10.1145/3184558.3186960 (cit. on p. 113).

Taniar, David, Wenny Rahayu, Vincent Lee, and Olena Daly (2008). "Exception rules in association rule mining". In: *Applied Mathematics and Computation* 205.2, pp. 735–750. DOI: 10.1016/j.amc.2008.05.020 (cit. on p. 32).

Tao, Jiao, Evren Sirin, Jie Bao, and Deborah L. McGuinness (2010). "Integrity Constraints in OWL". In: *AAAI*. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1931 (cit. on p. 75).

Vrandecic, Denny and Markus Krötzsch (2014). "Wikidata: a free collaborative knowledgebase". In: *Commun. ACM* 57.10, pp. 78–85. DOI: 10.1145/2629489 (cit. on pp. 11, 54, 95, 131, 141, 145).

Wang, Zhen, Jianwen Zhang, Jianlin Feng, and Zheng Chen (2014). "Knowledge Graph Embedding by Translating on Hyperplanes". In: *AAAI*, pp. 1112–1119. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531 (cit. on p. 32).

Wang, Zhichun and Juan-Zi Li (2015). "RDF2Rules: Learning Rules from RDF Knowledge Bases by Mining Frequent Predicate Cycles". In: *CoRR*. URL: http://arxiv.org/abs/1512.07734 (cit. on p. 31).

Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew (2019). "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR*. URL: http://arxiv.org/abs/1910.03771 (cit. on p. 101).

Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu (2019). "A Comprehensive Survey on Graph Neural Networks". In: *CoRR*. URL: http://arxiv.org/abs/1901.00596 (cit. on pp. 76, 77).

Yang, Bishan, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng (2014). "Embedding Entities and Relations for Learning and Inference in Knowledge Bases". In: *CoRR*. URL: http://arxiv.org/abs/1412.6575 (cit. on p. 76).

Yang, Fan, Zhilin Yang, and William W. Cohen (2017). "Differentiable Learning of Logical Rules for Knowledge Base Reasoning". In: *NIPS*, pp. 2316–2325. URL: http://papers.nips.cc/paper/6826-differentiable-learning-of-logical-rules-for-knowledge-base-reasoning (cit. on p. 76).

Zaveri, Amrapali, Dimitris Kontokostas, Mohamed Ahmed Sherif, Lorenz Bühmann, Mohamed Morsey, Sören Auer, and Jens Lehmann (2013). "User-driven Quality Evaluation of DBpedia". In: *I-SEMANTICS*, pp. 97–104. DOI: 10.1145/2506182.2506195 (cit. on pp. 11, 132).

Zaveri, Amrapali, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer (2016). "Quality Assessment for Linked Data: A Survey". In: *Semantic Web* 7.1, pp. 63–93. DOI: 10.3233/SW-150175 (cit. on p. 54).

Zhao, Jieyu, Tianlu Wang, Mark Yatskar, Ryan Cotterell, Vicente Ordonez, and Kai-Wei Chang (2019). "Gender Bias in Contextualized Word Embeddings". In: *NAACL-HLT*, pp. 629–634. DOI: 10.18653/v1/n19-1064 (cit. on p. 108).

Annexe A

Résumé en français

A.1 Introduction

A.1.1 Motivation

Les *bases de connaissances* sont des ensembles de faits lisibles par une machine. Elles contiennent des *entités* et des *relations* décrivant ces entités. Par exemple, une base de connaissances encyclopédique pourrait contenir des entités sur des personnes comme Jean-François Champollion ou des lieux comme Paris, et des relations nommées comme ⟨`Jean-François_Champollion`, `schema:homeLocation`, `Paris`⟩ pour indiquer que Jean-François Champollion a vécu à Paris. Des bases de connaissances bien connues dans ce domaine sont, entre autres, Wikidata (Vrandecic et Krötzsch, 2014), YAGO (Suchanek, Kasneci et Weikum, 2007), Freebase (Bollacker et al., 2008), DBpedia (Bizer et al., 2009) ou encore le Google Knowledge Graph. Ces bases de connaissances sont utilisées pour afficher des fiches d'information comme sur Wikipédia ou dans les moteurs de recherche Google et Bing. Elles permettent également de répondre directement aux questions des utilisateurs comme avec Alexa ou Siri. Mais les bases de connaissances sont également mises à profit pour d'autres types de contenus et de cas d'utilisation. Par exemple, Amazon et eBay maintiennent des bases de connaissances sur les produits qu'ils vendent et Uber tient une base de connaissances sur les aliments pour aider ses clients à choisir un restaurant.

Certaines bases de connaissances sont très vastes. Par exemple, Wikidata contient 81 millions d'entités[1] et Freebase 39 millions. Certains de ces projets utilisent des logiciels pour construire automatiquement la base de connaissances à partir d'une ou plusieurs sources existantes. Par exemple, DBpedia est construite

[1] Les chiffres sont à jour en date du 10 avril 2020.

par un ensemble d'extracteurs à partir du contenu Wikipédia. D'autres utilisent des contributeurs, rémunérés ou bénévoles, pour remplir la base de connaissances. C'est le cas de Wikidata, qui compte plus de 40 000 éditeurs différents par mois.

En conséquence, les bases de connaissances présentent souvent des problèmes de qualité, qui proviennent de bugs dans les pipelines de conversion, d'erreurs de bonne foi ou de vandalisme dans les contenus fournis par des contributeurs. Par exemple, Zaveri, Kontokostas et al., 2013 ont constaté que 12% des triplets de DBpedia présentent un problème. Même les bases de connaissances provenant de contributeurs s'appuient souvent de manière significative sur des filières d'importation ou de conservation automatisées. 43% des modifications de Wikidata en mars 2020 ont été effectuées à l'aide d'outils automatisés.

Pour lutter contre ces problèmes, de nombreuses bases de connaissances contiennent un système de contraintes. L'un des types de contraintes le plus courant est d'indiquer que les valeurs d'une propriété donnée doivent avoir un type donné. Par exemple, une base de connaissances pourrait imposer que les valeurs de la propriété `schema:birthPlace` soient des chaînes de caractères représentant une date. Les contraintes peuvent également être plus complexes comme, par exemple, une personne peut avoir au maximum deux parents (restrictions de cardinalité), une entité ne peut pas être une personne et un lieu en même temps (hypothèse de disjonctions). Certaines de ces contraintes sont maintenues par la base de connaissances. Par exemple, le formalisme des logiques de descriptions exige que les propriétés dont les valeurs sont des littéraux (chaîne de caractères, dates...) doivent être disjointes de celles dont les valeurs sont des entités. Cependant, ces contraintes sont souvent violées dans la pratique. Par exemple, en date du 20 mars 2020, Wikidata contient 1 million de violations de contraintes de type "domaine" et 4,4 millions de violations de contraintes de type "valeur unique". Il est donc nécessaire de disposer d'outils permettant de filtrer ces problèmes dans les bases de connaissances. Il y a également un besoin d'outils qui aident les personnes maintenant la base de connaissances à réparer ces violations de manière automatisée ou semi-automatisée.

Cette thèse fournit de nouvelles approches et techniques pour améliorer l'état de l'art sur cette tâche.

A.1.2 Contenu

Le premier chapitre de cette thèse est consacré aux préliminaires généraux sur les bases de connaissances et l'extraction de règles. Il est résumé dans la section A.2. Le cœur de la thèse est composé de trois parties.

La première partie présente les améliorations apportées au problème de l'extraction de règles. Elle est résumée dans la section A.3. Plus précisément, cette partie présente une nouvelle approche qui améliore l'extraction de règles sur des bases de

connaissances incomplètes en utilisant des informations de cardinalité. L'objectif de cette tâche est de permettre l'extraction de règles de meilleure qualité. Celles-ci peuvent ensuite être utilisées pour compléter la base de connaissances ou comme contraintes pour signaler des problèmes dans les données. Notre approche fonctionne en introduisant une nouvelle mesure d'estimation de la qualité des règles, la *confiance de complétude*. La confiance de complétude prend en compte le nombre d'objets attendus pour des sujets donnés et permet de mieux évaluer la qualité des règles. Nous montrons que cette mesure ne nécessite pas d'informations de cardinalité sur toutes les entités de la base de connaissances pour être efficace. Nous avons évalué cette confiance de complétude à la fois sur des ensembles de données réelles et synthétiques, montrant qu'elle surpasse les mesures existantes à la fois pour la qualité des règles extraites et pour les prédictions formulées par ces règles.

Cette partie présente aussi une approche visant à augmenter le nombre d'informations de cardinalité disponibles, notamment pour améliorer la qualité des règles exploitées en utilisant la *confiance de complétude*.

Ce travail a été publié à la conférence ISWC 2017 où il a été nominé pour le prix du meilleur article écrit par un étudiant. Une version abrégée de l'article a été présentée à IJCAI 2018 :

> Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. "Completeness-Aware Rule Learning from Knowledge Graphs". Article de recherche à ISWC 2017. `https://doi.org/10.1007/978-3-319-68288-4_30`

> Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. "Completeness-aware Rule Learning from Knowledge Graphs". Article invité à IJCAI 2018. `https://doi.org/10.24963/ijcai.2018/749`

La seconde partie de cette thèse présente une approche basée sur l'application statique de contraintes. Plus précisément, elle présente YAGO 4, une base de connaissances qui est essentiellement une version plus simple et plus propre de Wikidata. Nous avons construit cette base de connaissances en utilisant une approche déclarative et un pipeline d'application de contraintes. YAGO 4 peut être considéré comme un exemple de base de connaissances qui assure sa qualité en filtrant les violations de contraintes. Cette seconde partie est omise dans ce résumé. Ce travail a été publié comme ressource à la conférence ESWC 2020 :

> Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. "YAGO 4 : A Reason-able Knowledge Base". Article "ressource" à ESWC

2020. https://doi.org/10.1007/978-3-030-49461-2_34

La troisième et dernière partie présente une approche afin d'apprendre de manière dynamique à faire respecter des contraintes. Elle est résumée dans la section A.4. Elle aborde un problème nouveau : apprendre à corriger les violations de contraintes en utilisant l'historique d'éditions d'une base de connaissances. Nous présentons une formalisation du problème et un algorithme pour extraire les *corrections passées* des violations de contraintes de l'historique de la base de connaissances. Pour résoudre ce problème, le chapitre propose deux approches différentes : l'une basée sur l'extraction de règles et l'autre utilisant des réseaux de neurones, cette dernière fournissant une meilleure précision mais aucune garantie d'explicabilité pour ses prédictions. Nous avons validé les deux approches expérimentalement sur Wikidata, montrant des améliorations substantielles par rapport aux bases de référence. Ces travaux ont été présentés à la conférence TheWebConf 2019. L'approche via un réseau de neurones est actuellement en cours de relecture :

> Thomas Pellissier Tanon, Camille Bourgaux, and Fabian M. Suchanek. "Learning How to Correct a Knowledge Base from the Edit History". Article de recherche à WWW 2019. https://doi.org/10.1145/3308558.3313584

> Thomas Pellissier Tanon and Fabian M. Suchanek. "Neural Knowledge Base Repairs". En cours de relecture à ISWC 2020.

Cette partie propose aussi une annexe omise dans ce résumé. Cette annexe présente un système que nous avons mis en place pour interroger efficacement l'historique des modifications de Wikidata. Il a servi à extraire les données utilisées pour évaluer les approches présentées dans le chapitre précédent. Ce travail a fait l'objet d'une démonstration à la conférence ESWC 2019 :

> Thomas Pellissier Tanon and Fabian M. Suchanek. "Querying the Edit History of Wikidata". Démonstration à ESWC 2019. https://doi.org/10.1007/978-3-030-32327-1_32

A.1.3 Autres travaux

Au cours de mon doctorat, j'ai contribué à d'autres travaux qui ne sont pas présentés dans cette thèse.

J'ai présenté ma thèse de master lors du *workshop Linked Data on The Web*. Ce travail a été réalisé avec David Montoya sous la supervision de Serge Abiteboul, Pierre Senellart, et Fabian Suchanek. La thèse de master portait sur la construction d'une plateforme d'intégration des connaissances pour les informations personnelles. Ce système est capable de se synchroniser dans les deux sens à partir de sources de données telles que les courriels, le calendrier et les carnets d'adresses. Il permet également d'aligner et d'enrichir les données. Ce travail a également fait l'objet d'une démonstration à la conférence CIKM :

> David Montoya, Thomas Pellissier Tanon, Serge Abiteboul, Pierre Senellart, and Fabian M. Suchanek. "A Knowledge Base for Personal Information Management". Article au *workshop* LDOW qui a eu lieu à l'occasion de la conférence WWW 2018. `http://ceur-ws.org/Vol-2073/article-02.pdf`
>
> David Montoya, Thomas Pellissier Tanon, Serge Abiteboul, and Fabian M.Suchanek. "Thymeflow, A Personal Knowledge Base with Spatio-temporal Data". Démonstration à CIKM 2016. `https://doi.org/10.1145/2983323.2983337`

J'ai également publié un travail antérieur sur l'utilisation des relations grammaticales afin de répondre à des questions en langues naturelles sur des bases de connaissances. Ce travail a fait l'objet d'une démonstration à la conférence ESWC et des ensembles des données nécessaires à l'entraînement et l'évaluation de ces systèmes ont été présentés sous forme de poster à la conférence ISWC :

> Thomas Pellissier Tanon, Marcos Dias de Assunção, Eddy Caron et Fabian M. Suchanek. "Demoing Platypus - A Multilingual Question Answering Platform for Wikidata". Démonstration à ESWC 2018. `https://doi.org/10.1007/978-3-319-98192-5_21`
>
> Dennis Diefenbach, Thomas Pellissier Tanon, Kamal Deep Singh, and Pierre Maret. "Question Answering Benchmarks for Wikidata". Poster à ISWC 2017. `http://ceur-ws.org/Vol-1963/paper555.pdf`

De plus, avec Lucie-Aimée Kaffee, nous avons mené une étude sur la stabilité du schéma de Wikidata. Nous avons effectué cette analyse en nous basant sur les changements de labels des propriétés dans six langues. Nous avons constaté que le schéma est globalement stable, ce qui en fait une ressource fiable pour une utilisation externe. Ce travail a été présenté au WikiWorshop :

Thomas Pellissier Tanon et Lucie-Aimée Kaffee. “Property Label Stability in Wikidata : Evolution and Convergence of Schemas in Collaborative Knowledge Bases”. Article court lors du WikiWorkshop qui a eu lieu à l’occasion de la conférence WWW 2018. `https://doi.org/10.1145/3184558.3191643`

J’ai également aidé un doctorant de l’équipe, Thomas Rebele, dans son travail portant sur l’exécution Datalog avec l’interpréteur de commandes Bash. J’ai formalisé le problème et fourni un convertisseur entre Datalog et l’algèbre relationnelle. Notre méthode permet d’analyser de grandes données tabulaires avec Datalog – sans qu’il soit nécessaire d’indexer les données. La requête Datalog est traduite en Bash Unix et peut être exécutée dans un shell. Nos expérimentations ont montré que, pour le cas d’utilisation du prétraitement des données, notre approche est compétitive par rapport aux systèmes de pointe en termes d’évolutivité et de rapidité, tout en ne nécessitant qu’un terminal Bash sur un système Unix. Ce travail a été publié durant la conférence ISWC :

Thomas Rebele, Thomas Pellissier Tanon, and Fabian M. Suchanek. “Bash Datalog : Answering Datalog Queries with Unix Shell Commands”. Article de recherche à ISWC 2018. `https://doi.org/10.1007/978-3-030-00671-6_33`

A.2 Préliminaires

Bases de connaissances. Les bases de connaissances représentent l’information sous forme de graphes. Les sommets encodent des entités comme Jean-François Champollion ou la ville de Paris, et les arêtes nommées servent à encoder les relations entre les entités. Ces arêtes sont souvent appelées *triplets*. Un triplet est composé du sommet de départ, du nom de l’arête et du sommet d’arrivée. Par exemple, $\langle$`Jean-François_Champollion`, `schema:birthPlace`, `Paris`$\rangle$ encode que Jean-François Champollion est né à Paris. La figure A.1 propose un exemple graphique de base de connaissances. On utilisera dans la suite le modèle RDF (Cyganiak et al., 2014) qui formalise ces notions. Le contenu d’une base de connaissances $\mathcal{K}$ peut être vu comme un ensemble de triplets. La plupart des grosses bases de connaissances, comme par exemple celles à portée encyclopédique, utilisent l’hypothèse du monde ouvert. Elles supposent que la base de connaissances ne contient concrètement qu’une partie de ce qu’elle devrait idéalement contenir. Dans ce résumé nous utilisons la notation RDF usuelle pour les préfixes d’URIs. Par exemple, `schema:` est un raccourci pour `http://schema.org/`.

Apprentissage de règles. L'apprentissage des règles d'association concerne la découverte de schémas fréquents dans un ensemble de données et la transformation ultérieure de ces schémas en règles.

Definition A.1 (Requête conjonctive)**.** Une *requête conjonctive* q sur $\mathcal{K}$ est de la forme $\langle x_1, p_1, y_1\rangle \wedge \cdots \wedge \langle x_m, p_m, y_m\rangle$, où les x_i et y_i sont des variables ou des constantes et les p_i sont des identifiants de relation.

La *réponse* de q sur $\mathcal{K}$ est l'ensemble

$$q(\mathcal{K}) := \{(\nu(x_1), \ldots, \nu(x_m), \nu(y_1), \ldots, \nu(y_m)) \mid \forall i : \langle \nu(x_i), p_i, \nu(y_i)\rangle \in \mathcal{K}\}$$

où ν est une fonction qui met en correspondance les variables et les constantes avec les éléments de $\mathcal{K}$.

Definition A.2 (Règle)**.** Une *règle* est de la forme $r(\vec{x}) : b(\vec{x}) \to h(\vec{x})$, où b et h sont des requêtes conjonctives.

Elle exprime que pour tout $\vec{a}$, si $\vec{a} \in b(\mathcal{K})$ alors $\vec{a} \in h(\mathcal{K})$.

La manière classique d'évaluer la qualité des règles est basée sur le *support d'une requête*, *support d'une règle* et la *confiance*. Ces mesures sont définies par :

Definition A.3 (Support d'une requête)**.** Le *support* d'une requête conjonctive q dans une base de connaissances $\mathcal{K}$ est le nombre de réponses distinctes de q sur $\mathcal{K}$:

$$supp(q) := |q(\mathcal{K})|$$

Definition A.4 (Support d'une règle)**.** Le *support* d'une règle $r(\vec{x}) : b(\vec{x}) \to h(\vec{x})$ est le nombre de fois où une règle s'applique sur $\mathcal{K}$:

$$supp(r) := supp(b \wedge h) = |\{\vec{x} \mid \vec{x} \in b(\mathcal{K}) \wedge \vec{x} \in h(\mathcal{K})\}|$$

Definition A.5 (Confiance standard)**.** La *confiance standard* d'une règle $r(\vec{x}) : b(\vec{x}) \to h(\vec{x})$ est la proportion du nombre de fois où une règle s'applique par rapport au nombre de fois où elle pourrait s'appliquer :

$$conf(r) := \frac{supp(r)}{supp(b)}$$

On remarque qu'on a toujours $conf(r) \in [0, 1]$.

Exemple A.6. Considérons la base de connaissances $\mathcal{K}$ dans la figure A.1 et, en plus, ces règles qui en sont extraites :

$r_1(x, y) = \langle x, \texttt{schema:worksFor}, z\rangle \wedge \langle y, \texttt{educated}, z\rangle \to \langle x, \texttt{schema:children}, y\rangle$

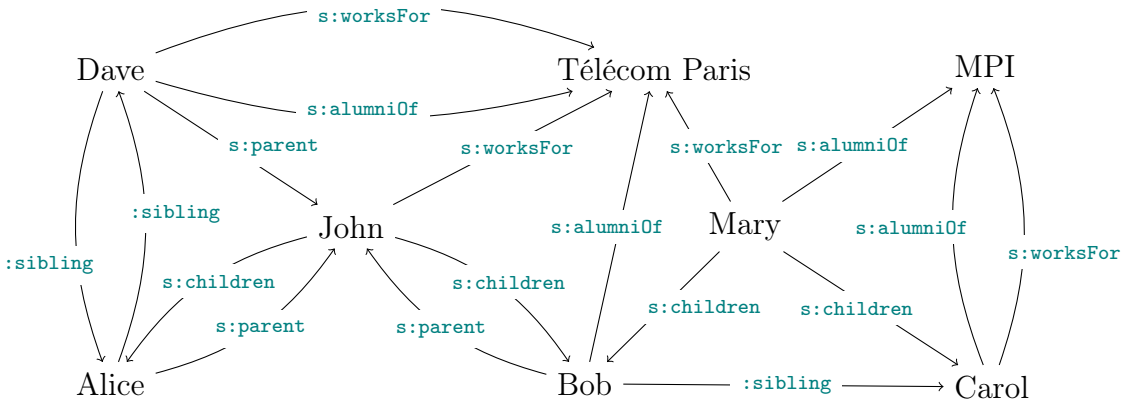

FIGURE A.1 – Exemple de base de connaissances

$$r_2(x, z) = \langle x, \texttt{schema:parent}, y\rangle \wedge \langle y, \texttt{schema:children}, z\rangle \rightarrow \langle x, \texttt{:sibling}, z\rangle$$

Les supports du corps de la règle r_1 et celui de la règle r_1 elle-même sur la base de connaissances sont respectivement $supp(\langle x, \texttt{schema:worksFor}, z\rangle \wedge \langle y, \texttt{educated}, z\rangle) = 8$ et $supp(r_1) = 2$ respectivement. Nous avons donc $conf(r_1) = \frac{2}{8}$. De la même manière, $conf(r_2) = \frac{1}{6}$. □

A.3 Apprentissage de règles à l'aide de cardinalités

L'apprentissage des règles est souvent utilisé pour compléter les bases de connaissances. Cependant, comme ces règles sont apprises à partir de données incomplètes, elles peuvent être erronées et peuvent faire des prédictions incorrectes sur des faits manquants. Par exemple, comme nous l'avons vu dans l'exemple A.6, la règle r_1 n'est clairement pas universelle et doit être classée plus bas que la règle r_2 même si la confiance favorise r_1 par rapport à r_2 pour la base de connaissances donnée.

Dans GALÁRRAGA, TEFLIOUDI et al., 2015, la *confiance sous l'hypothèse de complétude partielle*, ou *confiance PCA*, a été proposée comme une amélioration de la mesure de confiance. Elle devine les faits probablement faux en supposant que les données sont généralement ajoutées aux bases de connaissances par lots, c'est-à-dire que si au moins un enfant de Jean est connu, alors très probablement tous les enfants de Jean sont présents dans la base de connaissances. Formellement, la *confiance PCA* pour une règle $r(x, y) : b(x, y) \rightarrow \langle x, h, y\rangle$ est définie comme

$$conf_{pca}(r) := \frac{supp(r)}{supp_{pca}(r)} \tag{A.1}$$

où

$$supp_{pca}(r) := |\{(x, y) \mid (x, y) \in b(\mathcal{K}) \text{ et } \exists y' \in \mathsf{N_I}\ \langle x, h, y' \rangle \in \mathcal{K}\}|$$

Exemple A.7. On obtient $conf_{pca}(r_1) = \frac{2}{4}$. En effet, puisque `Carol` et `Dave` n'ont pas d'enfants connus dans la base de connaissances, quatre possibles assignations du corps de la règle ne sont pas comptées dans le dénominateur. Nous avons aussi $conf_{pca}(r_2) = \frac{1}{6}$, puisque toutes les personnes dont r_2 prévoit qu'elles auront des frères et sœurs ont déjà des frères et sœurs connus dans la base de connaissances. □

Cette hypothèse a été prise en compte pour classer les règles apprises, elle s'est avérée empiriquement valable pour certaines bases de connaissances. Cependant, la confiance PCA traite de manière inappropriée les cas où les arêtes d'une base de connaissances sont manquantes de manière aléatoire. Elle pose aussi des problèmes comme celui présenté dans l'exemple A.7. La question de savoir s'il faut compter l'absence de contradiction comme confirmation des règles par défaut a été examinée dans Doppa et al., 2011. Dans Galárraga, Razniewski et al., 2017, où de nouvelles données sur l'exhaustivité ont été tirées d'une base de connaissances en prenant comme source les données d'exhaustivité obtenues par *crowd-sourcing*. Les déclarations acquises ont été utilisées dans une étape de post-traitement de l'apprentissage des règles pour filtrer les prédictions non conformes. Cependant, ce type de filtrage n'a pas d'impact sur la qualité des règles exploitées et n'empêche pas les prévisions incorrectes pour les cas sur lesquels il n'existe aucune information d'exhaustivité.

Des efforts ont été déployés pour ajouter des informations sur l'exhaustivité aux bases de connaissances (Razniewski, Suchanek et Nutt, 2016 ; Darari, Nutt et al., 2013). Cela pourrait se faire en détectant le nombre concret de faits de certains types qui existent dans le monde réel (par exemple, "Einstein a 3 enfants"), ceci en faisant de l'extraction automatique sur le Web, ou par le *crowd-sourcing* (Prasojo et al., 2016 ; Darari, Razniewski et al., 2016 ; Mirza, Razniewski, Darari et al., 2017). Ces données fournissent de nombreux indices sur la topologie de la base de connaissances, et révèlent des parties qui devraient être particulièrement ciblées par les méthodes d'apprentissage des règles. Cependant, à ce jour, il n'existe aucun moyen systématique d'utiliser ces informations dans l'apprentissage des règles. Nous nous concentrons donc sur l'amélioration des fonctions de notation des règles en utilisant les informations supplémentaires d'(in-)complétude. Avant d'entrer dans les détails de notre approche, nous discutons de la représentation formelle de ces informations.

Déclarations de cardinalités. Nous représentons les informations de complétude en utilisant des *déclarations de cardinalité* qui indiquent le nombre absolu

d'objets pour une certaines relation et un certain sujet dans la base de connaissances idéale représentant le monde réel $\mathcal{K}^i$. Plus précisément, nous définissons la fonction partielle *num* qui prend comme entrée un prédicat p et une constante s et qui produit un nombre correspondant au nombre de faits dans $\mathcal{K}^i$ sur p avec s comme premier argument :

$$num(p, s) := |\{o \mid \langle s, p, o\rangle \in \mathcal{K}^i\}| \tag{A.2}$$

Ces déclarations de cardinalité peuvent être obtenues à l'aide de techniques d'extraction sur le web (Mirza, Razniewski, Darari et al., 2017). Avec ces déclarations, il est également possible d'encoder des cardinalités sur le nombre de sujets pour un prédicat et un objet donnés, à condition que des relations inverses puissent être exprimées dans la base de connaissances.

Naturellement, le nombre de faits manquants pour un p et s donné peut être obtenu par :

$$miss(p, s) := num(p, s) - |\{o \mid \langle s, p, o\rangle \in \mathcal{K}^a\}| \tag{A.3}$$

Exemple A.8. Considérons la base de connaissances de la figure A.1 et les déclarations de cardinalité suivantes pour celle-ci :

$num($`schema:children`,`John`$) = num($`schema:children`, `Mary`$) = 3$
$num($`schema:children`, `Alice`$) = 1$
$num($`schema:children`, `Carol`$) = num($`schema:children`, `Dave`$) = 0$
$num($`:sibling`, `Bob`$) = 3$
$num($`:sibling`, `Alice`$) = num($`:sibling`, `Carol`$) = 2$
$num($`:sibling`, `Dave`$) = 2$

Nous avons alors, par exemple :

$$miss(\texttt{schema:children}, \texttt{Mary}) = miss(\texttt{schema:children}, \texttt{John}) = 1$$

Il est possible d'utiliser l'extraction de règles pour compléter l'ensemble des déclarations de cardinalité disponibles. Cela peut être fait en créant de nouveaux faits $p_{\leq k}(s)$ et $p_{\geq k}(s)$ qui indiquent respectivement que $num(p, s) \leq k$ et $num(p, s) \geq k$. Cette méthode est présentée de manière plus détaillée dans le chapitre 4.

Confiance de complétude. Nous proposons de nous appuyer explicitement sur des informations de complétude pour déterminer s'il convient ou non de considérer une instance comme un contre-exemple pour une règle donnée.

Pour ce faire, nous définissons d'abord un indicateur pour une règle $r(x, y) : b(x, y) \rightarrow \langle x, h, y\rangle$, reflétant le nombre de nouvelles prédictions faites par r dans les parties incomplètes de la base de connaissances :

$$npi(r) := \sum_x min(|\{y \mid \langle x, h, y\rangle \in \mathcal{K}_r \backslash \mathcal{K}\}|, miss(h, x)) \tag{A.4}$$

Notez que $\mathcal{K}_r$ est la complétion de $\mathcal{K}$ en appliquant la règle r et que la sommation est faite exactement sur les entités pour lesquelles *miss* est défini. L'exploitation de cet indicateur supplémentaire pour $r(x, y) : b(x, y) \rightarrow \langle x, h, y \rangle$ permet de définir la *confiance de complétude* :

$$conf_{comp}(r) := \frac{supp(r)}{supp(b) - npi(r)} \tag{A.5}$$

Exemple A.9. La règle r_2 correspond davantage au monde réel que r_1 et doit donc être préférée à celle-ci. Pour notre nouvelle confiance de complétude, nous obtenons $conf_{comp}(r_1) = \frac{2}{6}$ et $conf_{comp}(r_2) = \frac{1}{2}$, ce qui donne l'ordre souhaité entre les règles, non atteint via les mesures existantes (c.f. les exemple A.6 et A.7). □

Dans l'hypothèse du monde fermé $conf_{comp}(r) = conf(r)$ car on a $\forall x$: $miss(h, x) = 0$. De même, dans l'hypothèse PCA $conf_{comp}(r) = conf_{pca}(r)$. Ainsi, notre confiance dans l'exhaustivité est une mesure plus générale que la confiance standard et la confiance PCA.

Évaluation. Nous avons expérimenté notre nouvelle confiance de complétude à la fois sur un ensemble de données du monde réel, un sous-ensemble de Wikidata (Vrandecic et Krötzsch, 2014), et un ensemble de données synthétiques, LUBM (Guo, Pan et Heflin, 2005). Nous les avons complétés, en utilisant des règles et l'ontologie LUBM OWL, pour obtenir une approximation de la base de connaissances idéale. Nous avons calculé le facteur de corrélation de Pearson entre le classement des règles possibles obtenu via la confiance, la confiance PCA et la confiance de complétude sur la base de connaissances disponible, et le classement des mêmes règles sur la base de connaissances idéale en utilisant la confiance. Notre confiance de complétude est nettement supérieure aux deux autres confiances sur ces deux ensembles de données.

A.4 Apprentissage de corrections d'une base de connaissances à partir de son historique de modifications

Un moyen d'éviter ou au moins de détecter certains des problèmes dans les données des bases de connaissances est d'imposer des *contraintes* sur la base de connaissances. Ces contraintes peuvent imposer que certaines informations soient présentes (par exemple, imposer que chaque être humain ait une date de naissance), ou que certaines déclarations ne soient pas absentes (par exemple, s'assurer qu'une personne n'est pas aussi une ville). Dans ce qui suit, nous exprimons les

contraintes sous forme de règles. Par exemple $\Gamma_1(x) : \exists y \ \langle x, \texttt{schema:parent}, y\rangle \rightarrow \langle x, \texttt{rdf:type}, \texttt{schema:Person}\rangle$ exprime qu'une entité x qui a un parent dans $\mathcal{K}$ doit aussi avoir pour type `schema:Person`. Une contrainte Γ est violée si le corps de Γ correspond à $\mathcal{K}$ mais pas la tête.

Ici, nous cherchons à apprendre comment réparer les violations de contraintes. Notre but est d'aider les personnes modifiant les bases de connaissances en suggérant comment nettoyer les données localement (en fournissant une solution à une violation de contrainte particulière) ou globalement (en fournissant des règles qui peuvent être automatiquement appliquées à toutes les violations de contraintes d'une forme donnée une fois validées par l'éditeur). Pour ce faire, nous profitons de l'*historique d'édition* de la base de connaissances. Nous l'utilisons pour exploiter les *règles de correction* qui expriment comment les différents types de violations de contraintes sont généralement résolus.

À notre connaissance, il s'agit du premier travail qui s'appuie sur les corrections passées des utilisateurs afin d'en deviner de nouvelles. En effet, plusieurs approches récentes ont proposé des méthodes pour détecter le moment où une contrainte est violée, calculer les faits responsables, puis interagir avec l'utilisateur pour savoir comment mettre à jour la base de connaissances. L'objectif est alors de minimiser le nombre de questions auxquelles l'utilisateur doit répondre. Cela se fait de différentes manières, qui incluent la prise en compte des dépendances entre les faits à vérifier ou encore l'interaction entre plusieurs violations de contraintes pour définir des heuristiques afin de choisir la meilleure question à poser à l'utilisateur (Bergman et al., 2015 ; Bienvenu, Bourgaux et Goasdoué, 2016 ; Assadi, Milo et Novgorodov, 2018 ; Arioua et Bonifati, 2018). D'autres approches visant à améliorer la qualité d'une base de connaissances reposent sur les statistiques, le regroupement ou les aspects structurels des bases de connaissances. Paulheim et Bizer, 2014 utilisent les statistiques pour ajouter les types manquants à la base de connaissances, et pour détecter les déclarations erronées. Liang et al., 2017 exploitent l'observation que les cycles dans la base de connaissances contiennent souvent de mauvaises relations `rdf:type`. Là encore, d'autres approches (Acosta et al., 2018) utilisent le crowdsourcing pour détecter les problèmes de qualité des données liées. Mais, contrairement à la nôtre, toutes ces approches ne donnent pas de valeur aux corrections apportées par les utilisateurs dans le passé.

Une règle de correction est de la forme :

$$r : \Gamma(\vec{x}), \mathcal{E}(\vec{x}, \vec{y}, \vec{z}) \rightarrow (\mathcal{M}^+(\vec{x}, \vec{y}), \mathcal{M}^-(\vec{x}, \vec{y})) \tag{A.6}$$

où

- $\Gamma(\vec{x})$ est une contrainte qui peut être partiellement instanciée, c'est-à-dire que certaines de ses variables ont été remplacées par des constantes,

- $(\mathcal{M}^+(\vec{x},\vec{y}), \mathcal{M}^-(\vec{x},\vec{y}))$ est la correction elle-même, une paire d'ensembles d'au plus un triplet où $\mathcal{M}^+(\vec{x},\vec{y})$ est l'ensemble des triplets qui doivent être ajoutés à la base de connaissances et $\mathcal{M}^-(\vec{x},\vec{y})$ l'ensemble des triplets qui doivent être supprimés,

- $\mathcal{E}(\vec{x},\vec{y},\vec{z})$ est un ensemble de triplets avec des variables et des constantes appelé le *contexte* de la violation.

Une règle de correction peut être *appliquée* à une base de connaisance $\mathcal{K}$ lorsqu'il existe des n-uplets de constantes $\vec{a},\vec{b}$ tels que $\mathcal{K}$ viole la contrainte $\Gamma(\vec{a})$ et $\exists\vec{z}\ \mathcal{E}(\vec{a},\vec{b},\vec{z}) \subseteq \mathcal{K}$. Le *résultat de l'application de la règle* est alors la correction $(\mathcal{M}^+(\vec{a},\vec{b}), \mathcal{M}^-(\vec{a},\vec{b}))$.

Extractions des anciennes corrections. Nous allons maintenant expliquer notre algorithme d'extraction des anciennes corrections depuis l'historique de modifications à l'aide d'un exemple : Considérons la contrainte $\Gamma_0(x) = \exists y \langle y, \texttt{schema:genre}, x\rangle \rightarrow x = \texttt{schema:Male} \vee x = \texttt{schema:Female}$. Supposons que $\langle \texttt{Zeus}, \texttt{schema:gender}, \texttt{masculin}\rangle$ a été ajouté entre $\mathcal{K}_1$ et $\mathcal{K}_2$, mais a ensuite été remplacé par $\langle \texttt{Zeus}, \texttt{schema:gender}, \texttt{schema:Male}\rangle$ entre la version $\mathcal{K}_{100}$ et la version $\mathcal{K}_{101}$.

Le premier objectif de l'algorithme est de découvrir que la suppression de $\langle \texttt{Zeus}, \texttt{schema:gender}, \texttt{masculin}\rangle$ entre $\mathcal{K}_{100}$ et $\mathcal{K}_{101}$ (dans le cadre du remplacement) peut faire partie d'une correction passée pertinente. Nous appelons cette suppression un *début de correction*. La recherche de débuts de corrections a l'avantage de réduire considérablement l'espace de recherche par rapport au fait de calculer les violations de contraintes pour toutes les contraintes sur toutes les versions de la base de connaissances.

Pour trouver efficacement ces débuts de correction, la première étape de l'algorithme précalcule pour chaque contrainte un ensemble de motifs de modification atomique auxquels les débuts de correction possibles correspondraient. Dans l'exemple, il n'y aurait qu'un seul motif : le motif de suppression $(?, \langle ?, \texttt{schema:gender}, ?\rangle)$, où ? peut être n'importe quoi de sorte qu'il corresponde à la fois à la suppression de $\langle ?, \texttt{schema:gender}, ?\rangle$ et à ses remplacements.

La deuxième étape de l'algorithme vérifie, pour chaque début de correction, si elle a résolu une violation de contrainte dans le passé – c'est-à-dire si $\mathcal{K}_i$ contient certaines violations de certaines instances de contraintes qui ne sont pas dans $\mathcal{K}_{i+1}$. Si c'est le cas, la modification entre $\mathcal{K}_i$ et $\mathcal{K}_{i+1}$ est une solution qui a résolu ces violations dans $\mathcal{K}_i$. Dans l'exemple, nous aurions trouvé la violation $\langle \texttt{Zeus}, \texttt{schema:gender}, \texttt{masculin}\rangle$ de $\Gamma_0(\texttt{masculin})$ dans $\mathcal{K}_{100}$, qui n'est pas dans $\mathcal{K}_{101}$. Nous aurions donc extrait que $(\langle \texttt{Zeus}, \texttt{schema:gender}, \texttt{schema:Male}\rangle, \langle \texttt{Zeus}, \texttt{schema:gender}, \texttt{masculin}\rangle)$

est une solution qui a résolu la violation $\{\langle \texttt{Zeus}, \texttt{schema:gender}, \texttt{masculin}\rangle\}$ de $\Gamma_0(\texttt{masculin})$ en $\mathcal{K}_{100}$. Nous stockons ces informations sous la forme d'un tuple dans l'ensemble de données des corrections passées pertinentes (l'ensemble `PCDataset`).

L'étape finale de l'algorithme supprime les corrections qui ont été inversées. Le résultat est donc l'ensemble des corrections passées pertinentes.

Recherche de règles de corrections. Nous cherchons des règles de correction en adaptant l'algorithme de Galárraga, Teflioudi et al., 2015 à notre contexte, où nous apprenons les règles non pas à partir d'une base de connaissances mais à partir du `PCDataset` et de l'historique de la base de connaissances. Nous pouvons étendre la définition de support et de support du corps de la règle à notre cas en comptant le nombre de tuples dans le `PCDataset` correspondant à $\Gamma(\vec{a})$, et, le cas échéant, $(\mathcal{M}^+, \mathcal{M}^-)$ est tel que $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$ a une réponse sur $\mathcal{K}_i$ avec i l'identifiant de révision qui correspond à la correction passée considérée. Une définition de la confiance en découle.

Notre algorithme prend en entrée le `PCDataset`, l'historique de la base de connaissances, un seuil de support minimum, un seuil de confiance minimum et un seuil de régularisation θ. Ces seuils sont choisis de manière empirique.

L'algorithme produit des règles de correction. À cette fin, il génère d'abord une règle triviale r_0 pour chaque entrée du `PCDataset`. Le contexte de cette règle est simplement la partie suppression de la correction passée de la contrainte.

Cette règle triviale est ensuite transformée en plusieurs règles plus générales, que nous appelons *règles de base*, chacune d'entre elles étant obtenue à partir de r_0 en remplaçant certaines des constantes par des variables. Formellement, l'algorithme utilise toutes les substitutions partielles σ des constantes par des variables fraîches distinctes. Il ne retient que les règles de base qui respectent les seuils de soutien et de confiance minimums.

Dans la deuxième étape, l'algorithme affine progressivement chaque règle en construisant sa partie contexte $\mathcal{E}(\vec{x}, \vec{y}, \vec{z})$. Cela fonctionne de manière similaire à l'algorithme d'extraction de Galárraga, Teflioudi et al., 2015 : Chaque étape d'affinement ajoute un atome construit à partir des triplets de la base de connaissances, en remplaçant le sujet ou l'objet du triplet par une variable qui apparaît dans la règle de base et en conservant l'autre élément constant ou en le remplaçant par une nouvelle variable. Si la règle résultante atteint le seuil minimum de prise en charge et améliore la confiance d'au moins θ, la règle est retenue.

Application des règles de correction. Lorsque toutes les règles ont été extraites, elles sont triées par ordre de confiance décroissant, et, pour les cas d'égalité, à l'aide du soutien (comme cela se fait dans Liu, Hsu et Ma, 1998 pour construire

des classificateurs à partir des règles). Cet ensemble de règles forme ensuite un programme qui peut être utilisé pour corriger les violations de contraintes comme suit. Étant donné une violation $\mathcal{V}$ d'une contrainte Γ dans $\mathcal{K}$, choisir la première règle r dans le programme qui est pertinente pour Γ (i.e., qui contient $[\Gamma(\vec{x})]$ où $\Gamma(\vec{x})$ est une version partiellement instanciée de Γ), puis vérifier si r peut être appliquée à $\mathcal{V}$. La correction est le résultat de l'application de la règle.

Évaluation. Nous avons mis en place une version légèrement simplifiée de notre approche sur Wikidata (Vrandecic et Krötzsch, 2014). Nous avons pris en compte un ensemble de 10 types de contraintes de propriété de Wikidata, couvrant 71% d'entre elles. Notre jeu de données se compose d'un ensemble de 700M de révisions de Wikidata, sur lequel nous avons miné plus de 75M de corrections passées. Pour cela, nous avons construit un système de requête pour stocker l'historique des modifications de Wikidata, permettant d'implémenter facilement les requêtes nécessaires pour extraire les corrections passées et pour exécuter notre algorithme d'exploration des règles. Ensuite, nous avons exécuté l'algorithme d'extraction de règles sur 80% des corrections passées et utilisé les 20% restants pour évaluer la précision et le rappel de notre approche. Nous l'avons comparé avec deux approches de référence. La première supprime un triplet du corps des règles de contraintes. La seconde ajoute un triplet pour résoudre la violation si les règles de contrainte ont dans leur tête un seul triplet et que toutes les variables de ce triplet sont connues (s'il y a plusieurs objet possibles, on en choisit un au hasard lors de chaque prédiction). Les résultats de l'évaluation montrent que la précision de notre approche est nettement supérieure aux deux approches de référence.

A.5 Conclusion

A.5.1 Résumé

Dans cette thèse, nous avons étudié la correction et l'enrichissement des bases de connaissances. Nous avons présenté trois contributions.

Dans la première partie, nous avons défini le problème de l'apprentissage des règles à partir de bases de connaissances incomplètes. Nous avons introduit la *confiance de complétude* qui utilise les informations de cardinalité contenues dans la base de connaissances afin d'estimer la qualité des règles. Notre nouvelle mesure a été évaluée sur des bases de connaissances réelles et synthétiques, démontrant des améliorations significatives tant en ce qui concerne la qualité des règles obtenues que les prévisions qu'elles produisent. Nous avons également proposé une méthode pour augmenter automatiquement le nombre de ces informations de cardinalité.

Cette méthode peut être utilisée pour améliorer l'efficacité de notre nouvelle mesure.

Dans la seconde partie omise dans ce résumé, nous avons présenté YAGO 4, une base de connaissances qui, pendant sa construction, impose des contraintes à la base de connaissances en appliquant des stratégies simples de réparation. Cette méthode nous a permis de construire une base de connaissances à grande échelle qui peut être utilisée avec les raisonneurs OWL.

Dans la troisième et dernière partie, nous avons cherché à éviter la perte de données que l'approche statique de réparation causait dans YAGO 4. Pour y parvenir, nous avons présenté le problème où l'on souhaite apprendre à réparer les violations de contraintes à partir de l'historique d'une base de connaissances. Nous avons proposé deux méthodes à cette fin. La première, CorHist, est basée sur l'exploitation des règles. Son avantage est qu'elle permet d'expliquer les corrections qu'elle propose. L'autre, Bass, est basée sur un réseau de neurones et offre une meilleure précision. Cependant, elle n'est pas capable d'expliquer ses prédictions. Une évaluation expérimentale sur Wikidata a montré que ces deux approches sont nettement plus performantes que les approches statiques pour trouver la bonne correction. Nous avons également fourni le système comme outil à la communauté Wikidata, lui permettant de corriger plus de 20 000 violations de contraintes.

A.5.2 Perspective

Dans cette thèse, nous avons montré comment les contraintes peuvent être utilisées en pratique pour nettoyer les bases de connaissances. De nombreux défis de recherche sont encore ouverts.

L'approche utilisée pour augmenter le nombre de déclarations de cardinalité pourrait être étendue afin d'apprendre des règles sur les valeurs numériques. Par exemple, elle pourrait être utilisée pour apprendre que si un pays est membre de l'OCDE, son PIB est alors supérieur à un certain seuil, ou que tous les consuls romains sont nés avant le 10^{e} siècle.

Les travaux visant à apprendre comment corriger les violations de contraintes à partir des corrections passées pourraient être étendus aux données textuelles. Par exemple, on pourrait essayer d'étudier comment corriger les fautes d'orthographe courantes ou améliorer la typographie des textes en fonction des modifications apportées précédemment. Une autre application pourrait consister à extraire d'un historique de modification du code source des corrections de bogues simples pour les suggérer automatiquement lorsqu'un contributeur ouvre une demande de révision de code.

Nous espérons qu'en ouvrant la porte à ces défis, ce travail pourra contribuer à rendre les bases de connaissances et autres contenus plus propres, et donc en fin de compte toujours plus utiles.

DOCTORAL SCHOOL

Titre: Maintenance des bases de connaissances à l'aide de contraintes

Mots clés: Bases de connaissances, contraintes, règles

Résumé: Les bases de connaissances sont des ensembles de faits, souvent sur des sujets encyclopédiques. Elles sont souvent utilisées pour la reconnaissance d'entités nommées, la recherche structurée, la réponse automatique à des questions, etc. Ces bases de connaissances doivent être maintenues, ce qui est une tâche cruciale mais coûteuse. Le sujet de cette thèse est la maintenance automatique de bases de connaissances à l'aide de contraintes.
La première contribution de cette thèse est à propos de la découverte automatique de contraintes. Elle améliore les approches classiques d'apprentissage de règles en utilisant des méta-informations de complétude des données. Elle montre que que ces informations permettent d'améliorer de manière significative la qualité des règles trouvées.
La seconde contribution est la création d'une base de connaissances, YAGO 4, qui assure le respect d'une série de contraintes en supprimant les faits qui n'y correspondent pas.
La troisième contribution est une méthode pour corriger automatiquement les violations de contraintes. Cette méthode utilise l'historique des modifications de la base de connaissances afin de proposer des corrections, ceci à partir de la manière avec laquelle les utilisateurs de la base de connaissances ont déjà corrigé des violations similaires.

Title: Knowledge Base Curation using Constraints

Keywords: Knowledge Base, Constraints, Rules

Abstract: Knowledge bases are huge collections of primarily encyclopedic facts. They are widely used in entity recognition, structured search, question answering, and other tasks. These knowledge bases have to be curated, and this is a crucial but costly task. In this thesis, we are concerned with curating knowledge bases automatically using constraints.
Our first contribution aims at discovering constraints automatically. We improve standard rule mining approaches by using (in-)completeness meta-information. We show that this information can increase the quality of the learned rules significantly.
Our second contribution is the creation of a knowledge base, YAGO 4, where we statically enforce a set of constraints by removing the facts that do not comply with them.
Our last contribution is a method to correct constraint violations automatically. Our method uses the edit history of the knowledge base to see how users corrected violations in the past, in order to propose corrections for the present.

Institut Polytechnique de Paris
91120 Palaiseau, France