International Research School in Artificial Intelligence in Bergen

AIB 2022, June 7–11, 2022, University of Bergen, Norway

Edited by

Camille Bourgaux
Ana Ozaki
Rafael Peñaloza

OASICS

Editors

Camille Bourgaux
DI ENS, ENS, CNRS, PSL University & Inria, France
Camille.Bourgaux@ens.fr

Ana Ozaki
University of Bergen, Norway
Ana.Ozaki@uib.no

Rafael Peñaloza
University of Milano-Bicocca, Italy
rafael.penalozanyssen@unimib.it

ACM Classification 2012
Computing methodologies → Knowledge representation and reasoning; Computing methodologies → Machine learning

ISBN 978-3-95977-228-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-228-0.

Publication date
June, 2022

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/OASIcs.AIB.2022.0

ISBN 978-3-95977-228-0 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Invited Papers

International Research School in Artificial Intelligence in Bergen (AIB 2022).
Editors: Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

These proceedings present the tutorial papers accompanying the lectures of the *First International Research School Artificial Intelligence in Bergen*, AIB 2022, held during June 7–11 2022 in Bergen, Norway. The Artificial Intelligence in Bergen research school aims at disseminating recent advances on AI. It is mainly intended for masters and Ph.D. students, postdocs, and researchers wishing to learn more about the theme of the research school, and is co-located with a workshop where participants are invited to present their work. The broad theme of the school this year was

"Knowledge Graphs and Machine Learning".

Knowledge graphs have gained a lot of popularity as a flexible way of modeling data at large scale. In addition to classical, symbolic, reasoning methods based on ontologies, recent years have seen an increasing trend of using machine learning techniques to complete, correct, or reason with knowledge graphs. The six lectures by renowned researchers given at AIB 2022 provide an overview of the various research topics related to knowledge graphs, with a particular focus on reasoning and the combination of machine learning and symbolic methods.

We would like to thank those who contributed to this first edition of AIB. First and foremost the lecturers and their co-authors. Second, the program committee members for their reviews of these lecture notes. Finally, we are grateful to the organizing committee for the local organization and the website. We also thank the Research Council of Norway, project numbers 316022 and 332921, the Meltzer Research Fund, and the University of Bergen.

March 2022
Camille Bourgaux, Ana Ozaki, Rafael Peñaloza
AIB 2022 co-chairs

International Research School in Artificial Intelligence in Bergen (AIB 2022).
Editors: Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program chairs

- Camille Bourgaux (CNRS, DI ENS)
- Ana Ozaki (University of Bergen)
- Rafael Peñaloza (University of Milano-Bicocca)

Program committee

- Roberto Confalonieri (Free University of Bozen-Bolzano)
- Julien Corman (Free University of Bozen-Bolzano)
- Jesse Davis (Katholieke Universiteit Leuven)
- Jeff Horty (University of Maryland)
- Yazmin A. Ibanez-Garcia (School of Informatics and Computer Science, Cardiff University)
- Egor Kostylev (University of Oslo)
- Özgür Lütfü Özcep (Institute of Information Systems, University of Lübeck)
- Matteo Palmonari (University of Milan-Bicocca)
- Jeff Z. Pan (University of Edinburgh)
- Nico Potyka (Universitaet Stuttgart)
- Felix Weitkämper (Ludwigs-Maximilians-Universität München)
- Lu Zhou (Kansas State University)

Organizing committee

- Ana Ozaki
- Ricardo Guimarães
- Cosimo Persia
- Philip Turk
- Victor Botelho

Knowledge Graphs: A Guided Tour

Aidan Hogan
DCC, University of Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data (IMFD), Santiago, Chile

Abstract

Much has been written about knowledge graphs in the past years by authors coming from diverse communities. The goal of these lecture notes is to provide a guided tour to the secondary and tertiary literature concerning knowledge graphs where the reader can learn more about particular topics. In particular, we collate together brief summaries of relevant books, book collections, book chapters, journal articles and other publications that provide introductions, primers, surveys and perspectives regarding: knowledge graphs in general; graph data models and query languages; semantics in the form of graph schemata, ontologies and rules; graph theory, algorithms and analytics; graph learning, in the form of knowledge graph embeddings and graph neural networks; and the knowledge graph life-cycle, which incorporates works on constructing, refining and publishing knowledge graphs. Where available, we highlight and provide direct links to open access literature.

2012 ACM Subject Classification Information systems → Graph-based database models; Information systems → Information integration; Computing methodologies → Artificial intelligence

Keywords and phrases knowledge graphs

Digital Object Identifier 10.4230/OASIcs.AIB.2022.1

Category Invited Paper

Funding *Aidan Hogan*: Supported by Fondecyt Grant No. 1181896 and by ANID – Millennium Science Initiative Program – Code ICN17_002.

Acknowledgements I wish to thank Camille Bourgaux, Ana Ozaki and Rafael Peñaloza for providing the idea for these lecture notes.

1 Introduction

Knowledge graphs have gained significant attention in recent years, both in industry and academia, as a way to integrate and leverage data and knowledge from diverse sources at large scale. Though the term "knowledge graph" had been used as far back as the 70's [60], it was the announcement of the Google Knowledge Graph [65] in 2012 around which a community began to crystallise [11]. The Google Knowledge Graph (in uppercase) was intended to be a proper name: the name, specifically, of a large, internal, graph-structured knowledge base housed within Google, which aimed to enhance Google's semantic search capabilities. Per the original blog post, Google envisaged using the knowledge graph as a way to shift from search based on "strings" to search based on "things" (or *entities*). Rather than performing string matching on searches like "`taj mahal`", the goal was to understand that this search likely refers to a thing – an *entity*: a mausoleum in Agra, India, on the bank of the river Yamuna (represented, in turn, by a node in the Google Knowledge Graph).

Arguably the greatest impact of the Google Knowledge Graph, however, was unanticipated. Other companies began to adopt the phrase "knowledge graph" in order to describe similar initiatives. Announcements of knowledge graphs by other well-known companies, such as Airbnb [16], Amazon [43], eBay [54], Facebook [51], IBM [18], LinkedIn [28], Microsoft [63], Uber [25], to name but a few, began to emerge. Meanwhile, "knowledge graphs" began to gain more and more traction in the academic literature [55, 70, 46]. Questions began to arise regarding how knowledge graphs relate to existing concepts such as graph databases,

International Research School in Artificial Intelligence in Bergen (AIB 2022).
Editors: Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza; Article No. 1; pp. 1:1–1:21
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

RDF graphs, ontologies, semantic networks, etc.; questions also began to arise about what, specifically, a "knowledge graph" even was [19, 14, 10, 11]. Ambivalent to such concerns, knowledge graphs continued to firmly establishing themselves in both industry and academia.

Some authors prefer to define a knowledge graph as "*a graph-structured knowledge base*" [50, 62]. Other authors go further, and assert additional necessary conditions that a knowledge graph must satisfy to be called as such, including the presence of a schema [53], the description of multiple domains [53], the use of an ontology and a reasoner to derive new knowledge [19], the presence of inference rules [10], etc. But perhaps the most accepted modern definition of a knowledge graph is as a "*graph whose nodes represent entities and whose edges represent relationships between those entities*" [70, 46, 51], sometimes adding the qualifier that the "*graph intends to collect and convey knowledge*" [33]. This latter definition is inclusive enough to embrace the diverse ways in which knowledge graphs have been used and studied, and is also the definition we adopt for the purposes of these lecture notes.

Why, then, have knowledge graphs gained so much traction and attention in practice, and in the literature? In practice, knowledge graphs have proven to be a useful abstraction for representing, integrating, managing and exploiting diverse data at large scale. Typical use-cases for knowledge graphs involve the extraction and accumulation of data from diverse sources within a unified graph-based representation. As Noy et al. [51] put it, "*knowledge graphs and similar structures usually provide a shared substrate of knowledge within an organization, allowing different products and applications to use similar vocabulary and to reuse definitions and descriptions that others create. Furthermore, they usually provide a compact formal representation that developers can use to infer new facts and build up the knowledge*" [51]. Such use-cases have given rise to the announcement not only of a broad range of enterprise knowledge graphs [65, 16, 43, 54, 51, 18, 28, 63, 25], internal to particular companies, but also open knowledge graphs [45, 13, 68, 31] available to the public.

In the academic setting, the use of graphs to represent both data and knowledge is far from a novel idea [24]; however, Knowledge Graphs, as an area, is remarkable in the degree to which it has brought together researchers from distinct communities that had previously been exploring graph-structured data, knowledge and analytics in isolation. In particular, the Database community has long explored *graph query languages and databases* as a way to query data stored as graphs; the Semantic Web community has long explored *graph data* and *ontologies* as a way to represent, integrate, interlink and reason over data on the Web; the Graph Theory and Algorithms community has long explored *fundamental properties of graphs*, and how different types of *graph analytics* can be used to understand and extract patterns from graphs (or networks); the Natural Language Processing and Information Extraction communities have long used *text graphs*, *semantic networks*, and other graph-based representations to induce structure from natural language. Notably, however, the area of Knowledge Graphs has also gained attention in communities where graphs have not traditionally been a mainstream topic; for example, within the Machine Learning community, the topics of *graph embeddings* and *graph neural networks* have recently gained significant traction, yielding novel machine learning techniques that can be applied natively over graphs. All these communities are now working on interrelated topics under the common umbrella of "knowledge graphs", which gives rise to a number of open questions regarding how such diverse techniques relate to or can complement each other [32].

Much has been written about knowledge graphs in the past decade, and in particular in the last five years, including a variety of books (e.g., [52, 56, 20, 40, 33]) and surveys of specific aspects of knowledge graphs (e.g., [53, 69]). Indeed the author of these lecture notes has recently co-authored a book [33] (among other contributions [32, 34]) on the topic.

The volume of literature published in recent years has become somewhat overwhelming, in fact. Rather than providing yet another introduction to knowledge graphs, the goal of these lecture notes is to provide a roadmap of the existing literature on knowledge graphs and related topics, guiding the reader towards literature where they can learn more about the area in general, or about specific sub-topics of interest to them.

Thus our goal herein is to collate and summarise the secondary and tertiary literature that provides introductions, primers, surveys, perspectives, etc., regarding knowledge graphs and related topics. Specifically, we consider the following types of literature:

Books with global authorship (excludes edited collections).
Collections of edited book chapters on a specific topic (excludes proceedings).
Book chapters published in a collection not otherwise mentioned.
Articles published in journals.
Miscellaneous publications in conferences, workshops, or in preprint or online form.

Within each topic and category, we order papers by year and subsequently by author. We use the icon 🔓 to indicate an open access (OA) publication, and 🔓 to indicate a publication with an open access preprint or alternative version in a persistent repository (e.g., on arXiv). In the digital version of these notes, one can click on these icons to directly access the OA version. For the purposes of formatting, we may also abbreviate long titles, where we refer the reader to the bibliographic citation for the full details of the publication.

In terms of topics, we divide the literature into:

General: covering the broader topic of knowledge graphs.
Data: covering graph data models, databases and query languages.
Semantics: covering graph schemata, rules and ontologies.
Analytics: covering graph theory, algorithms, measures and frameworks.
Learning: covering knowledge graph embeddings, graph neural networks, etc.
Lifecycle: covering knowledge graph completion, refinement and publication.

The topics tend to become more technical as the lecture notes progress, so we recommend newcomers to check out some of the literature in the "General" category before proceeding. In the case of publications that cover multiple topics, we assign them to the topic that it places the most focus on. We include a range of different types of literature to suit different tastes, ranging from hands-on walkthroughs to theoretical treatments, from industry use-cases to academic conceptualisations, from high-level discussion to detailed definitions, etc.

Let's begin.

2 General: Knowledge Graphs

Here we cover literature providing a broad overview of the emerging area of Knowledge Graphs, including discussion of data, semantics, analytics, learning, and the knowledge graph lifecycle. The literature here provides a good starting point for newcomers.

Books

The Knowledge Graph Cook Book: Recipes That Work [12] (Blumauer & Nagy, 2020) introduces knowledge graphs in terms of data models, core concepts, and application scenarios. The book delves into organisational issues regarding enterprise knowledge graphs, before discussing specific aspects of (RDF) graph data, taxonomies and ontologies, and how knowledge graphs can be constructed. It concludes with discussion of enterprise knowledge graph architectures and services, set of interviews from industry leaders, and discussion of the future for knowledge graphs. The book is largely enterprise focussed.

Knowledge Graphs: Methodology, Tools & Selected Use Cases [20] (Fensel et al., 2020) first introduces knowledge graphs from a conceptual and practical viewpoint (contrasting open and enterprise/proprietary knowledge graphs), and then discusses how they can be constructed (knowledge graph creation, hosting, curation, and deployment) and used (AI use-cases, semantics, and dialogue systems). The final part of the book is dedicated to applications, and discussion of domains in which knowledge graphs can – and have been – deployed. The focus of the book is on building and maintaining knowledge graphs.

Knowledge Graphs: Data in Context for Responsive Businesses [8] (Barrasa et al., 2021) provides a practical introduction to knowledge graphs written by authors from Neo4j, targeted towards chief data officers. After motivating and defining knowledge graphs, the authors discuss how they can be modelled, and the role that taxonomies and ontologies play. The authors then compare actioning knowledge graphs vs. decisioning knowledge graphs, where the key focus is on data management and data analytics, respectively. The importance of context (for AI) is discussed, before the book concludes with an outlook. The book thus introduces and motivates knowledge graphs from an enterprise perspective.

Knowledge Graphs [33] (Hogan et al., 2021) provides a broad, conceptual introduction to knowledge graphs, covering graph data models and query languages, different forms of graph schema, contextual representations, ontologies and rules, and graph learning; it further discusses the creation, enrichment, quality, refinement, and publication of knowledge graphs. A brief survey of specific open and enterprise knowledge graphs, and their applications, is also provided. An appendix delves into the historical setting that gives rise to knowledge graphs. The presentation is mostly example-based. Formal definitions are provided, but can be skipped by the uninterested reader.

Knowledge Graphs: Fundamentals, Techniques & Applications [40] (Kejriwal et al., 2021) provides a broad introduction to knowledge graphs, covering popular graph-based data models, knowledge graph construction, knowledge graph completion, methods for querying and reasoning over knowledge graphs, and areas in which knowledge graphs have been successfully deployed. Aside from more general aspects, a major focus of the book is on creating knowledge graphs from text and other semi-structured sources, and subsequently refining the knowledge graph using (in particular) learning techniques.

Knowledge Graph [57] (Qi et al., 2022) is – at the time of writing – an upcoming book that is not yet available for download. According to the book's synopsis, it provides a systematic and comprehensive overview of knowledge graphs, their theoretical foundations, key techniques, methodologies, and applications. A key focus of the book is on the construction and management of knowledge graphs, including information extraction from text, as well as key techniques for knowledge fusion and reasoning.

Collections

Exploiting Linked Data and Knowledge Graphs in Large Organizations [52] (Pan et al., eds., 2017) is a collection of chapters, starting with an introduction to knowledge graphs, covering related standards, architectures for enterprise knowledge graphs, knowledge graph construction, ways in which knowledge graphs can be summarised and explored, question answering, and more besides. The book concludes with a discussion of applications and future directions. Overall the book is largely focussed on the Semantic Web.

Knowledge Graphs: New Directions for KR on the Semantic Web [14] (Bonatti et al., eds., 2018) collects together a number of short reports resulting from the discussions at Dagstuhl Seminar 18371. These reports summarise discussions among participants on a wide range of topics relating to knowledge graphs, including evolution and dynamics, scholarly knowledge graphs, logic and learning, knowledge graph creation and management, symbolic reasoning, multilingualism, privacy and access control, graph analytics, etc.

Knowledge Graphs and Big Data Processing [35] (Janev et al., eds., 2020) is an open access collection of chapters that provide a general introduction to knowledge graphs, and their application for processing and managing data at large scale. Various authors have contributed chapters on both Knowledge Graphs and Big Data, covering, in the former case, an introduction to graph-based knowledge representation, the creation of knowledge graphs, data exchange using knowledge graphs, knowledge graph embeddings and their applications, and more. A key focus of the book is on how knowledge graphs can be leveraged for the purposes of Big Data applications.

Knowledge Graphs for eXplainable Artificial Intelligence [67] (Tiddi et al., eds., 2020) is a collection of chapters that introduce knowledge graphs more from an AI perspective. Earlier chapters cover knowledge graphs on the Web, embeddings, explainability in the context of knowledge graphs, and benchmarks. Chapters on applications include recommender systems, natural language processing, context understanding, explanations, transfer learning, and predictive analytics. The book concludes with an outlook to the future, as well as ethical and social issues surrounding knowledge graphs and explainable AI. The book thus largely focuses on knowledge graphs in the context of learning and AI.

Book chapters

Knowledge Graphs: Research Directions [32] (Hogan, 2020) is a book chapter providing a more technical introduction to knowledge graphs, including formal definitions of concepts such as graph models, queries, ontologies, rules, context, embeddings, and graph neural networks. A key aim of the chapter is to synthesise a set of research problems that arise from how these concepts potentially relate and complement each other, presenting a list of nine research topics that intersect different areas.

Articles

Knowledge Graphs [24] (Gutierrez and Sequeda, 2021) is an article discussing the history of knowledge graphs, and the phenomena that influenced them and led to their popularisation. The paper traces knowledge graphs back to ancient traditions of representing knowledge in diagrammatic form, taking a journey through the advent of logic, information retrieval, semantic networks, knowledge representation, the Web, and finally knowledge graphs. The focus of the article is on setting knowledge graphs in a broader historical perspective relating to data and knowledge.

Knowledge Graphs [34] (Hogan et al., 2021) is an article providing a tutorial on some of the key concepts and techniques underlying knowledge graphs. The article first motivates knowledge graphs, and then discusses graph data models, graph query languages, shapes for validation, and contextual representations. An introduction to ontologies and reasoning is then followed by discussion of graph analytics, knowledge graph embeddings, graph neural networks, and symbolic learning. The article concludes with future directions.[1]

[1] This article was extended into the book of the same name, mentioned previously [33].

Knowledge Graphs 2021: A Data Odyssey [71] (Weikum, 2021) is a position paper that begins with a brief overview of open and enterprise knowledge graphs, their applications, and the challenges they pose. The paper then poses a number of positions, namely that knowledge graphs are more than simple graphs (often they consider higher-arity relations, provenance, constraints, etc.), and that knowledge graphs should prioritise precision (correctness) over recall (coverage), and thus should be constructed from select sources and incremental processes. The paper concludes with a list of open research challenges and problems that could be addressed with techniques from the area of Databases.

Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases [72] (Weikum et al., 2021) is a comprehensive article (comprising 380 pages) discussing the creation and curation, more generally, of knowledge bases. However, much of the content relates both directly and indirectly to knowledge graphs, which are also explicitly mentioned in various sections. The article covers key concepts relating to knowledge bases, methods for knowledge integration, techniques for knowledge base construction and curation, for schema construction, and for resolving entities. The article concludes with discussion of open and enterprise knowledge graphs, and an outlook to the future.

Miscellaneous

Towards a Definition of Knowledge Graphs [19] (Ehrlinger and Wöß, 2016) investigates the concept of knowledge graphs and the various – and sometimes incompatible or even contradictory – definitions attributed to them. They highlight that the differences between knowledge bases, knowledge graphs and ontologies remain unclear. Applying a terminological analysis, they arrive at the definition that "*A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge*" [19].[2]

3 Data: Models and Query Languages

This section pertains to literature that discusses graph data models and graph query languages. Much of the literature comes from the (Graph) Database and Semantic Web communities, which have traditionally been hot spots for research on graph data management techniques.

Books

Graph Databases [58] (Robinson et al., 2015) provides a practical introduction to graph data models and graph databases. The book walks through various examples using the Neo4j graph database, and the Cypher query language for graphs. The authors discuss the creation of applications on top of a graph database, and real-world use-cases where such databases are deployed. Techniques implemented by graph databases (specifically Neo4j) are also discussed. The final chapter concludes with a discussion of graph algorithms, focusing particularly on graph search (depth- and breadth-first search, Dijkstra, A*, etc.). The book has a practical focus using concrete examples from Neo4j/Cypher.

Querying Graphs [15] (Bonifati et al., 2018) provides a technical introduction and overview of the state-of-art with respect to graph databases and graph query languages. The book focuses on the property graph model and its variants. Different formal fragments

[2] This definition was contested by later authors [34], given that it would exclude many initiatives surrounding knowledge graphs not involving ontologies, such as works in the machine learning community on graph representation, and (seemingly) the Google Knowledge Graph itself, which gave origin to the modern use of the phrase; in these settings, ontologies are not (always) used.

of query languages are introduced, including regular path queries, unions of conjunctive queries, relational queries, regular queries, etc. Graph constraints are introduced in terms of functional and entity dependencies. The book then discusses query specification, which incorporates the paradigms of query-by-example and reverse-engineering of queries. The latter part of the book turns to implementation issues, including representation, compression, indexing, query processing, physical operators, and cardinality estimation. It concludes with some open research challenges. The book is targeted at a more academic audience, providing formal definitions and theoretical discussion throughout.

Graph Databases in Action [9] (Bechberger and Perryman, 2020) gives a practical guide to graph databases with concrete examples provided in the Gremlin graph traversal language. The book first discusses graph data modelling from a practical viewpoint. It then discusses how to run graph traversals, insertions, deletions, paths, filters, subgraph extraction and graph analytics over graphs using Gremlin. Practical issues, such as application development and performance pitfalls, are further discussed. The book has a practical focus and is principally targeted at software developers.

Collections

Graph Data Management: Fundamental Issues & Recent Developments [21] (Fletcher et al., eds., 2018) collates six chapters relating to graph data management. The first chapter provides a general introduction to graph data management. Next, techniques for graph visualisation are discussed, including planarisation, energy-based approaches, and approaches for large graphs. The third chapter discusses methods for discovering motifs (frequently-occurring subgraph patterns) in graphs. The following chapter describes query relaxation and approximation in the context of flexible query processing for graphs. The fifth chapter provides an overview of parallel processing frameworks for graphs. The final chapter concludes with a survey of benchmarks for graph processing systems.

Book chapters

Storing and Querying Semantic Data in the Cloud [36] (Janke and Staab, 2018) offers lecture notes on techniques for storing and querying RDF graphs in local, distributed and cloud environments. After some preliminary definitions on RDF graphs and SPARQL queries, the lecture notes describe different architectures for RDF stores in local, cloud-based, distributed, peer-to-peer and federated settings. The notes describe partitioning and replication strategies for RDF graphs, popular indexing techniques, distributed query processing strategies, fault tolerance, and available (RDF/SPARQL) benchmarks.

Articles

Survey of Graph Database Models [5] (Angles et al., 2008) is a survey of a variety of graph-based data models that have been proposed in the literature down through the years. After a general introduction to the history, main concepts and applications driving graph-based data modelling, the paper enumerates different graph database models, ranging from straightforward models with simple nodes and edges, to more complex models that support hypernodes (with nested elements), attributes on nodes and edges, relations viewed as entities, derivation and inheritance, nested relations, constraints, schema, etc. The survey also covers query languages proposed for such models.

Query Languages for Graph Databases [73] (Wood, 2012) provides an overview and formal definition of the different primitives – specifically conjunctive queries, regular path queries, conjunctive regular path queries, and extended conjunctive regular path queries – underlying graph query languages. A brief survey is provided of graph query languages and the features they support, including also path comparisons, aggregation operators, node creation, approximate matching, ranking, etc. The article concludes with a discussion of the expressive power of different fragments of graph query languages.

Foundations of Modern Query Languages for Graph Databases [4] (Angles et al., 2017) covers popular data models and graph query languages from a conceptual point of view. The article specifically covers the directed edge-labelled graph (e.g., RDF) and property graph models. It then delves into the core primitives and semantics underlying graph query languages, including basic graph patterns, path queries, and relational algebra. Examples in concrete query languages – specifically SPARQL, Cypher and Gremlin – are presented. More advanced querying primitives, featuring recursion, are also covered.

RDF Data Storage and Query Processing Schemes: A Survey [75] (Wylot et al., 2018) surveys data management techniques for RDF graphs, including techniques for indexing and processing queries over large-scale RDF graphs. The survey covers the storage and indexing of RDF graphs both on individual machines, as well as in distributed settings over multiple machines (using NoSQL, Hadoop, Spark, etc.). Federated query processing is also discussed. The paper concludes with a survey of different SPARQL-based benchmarks.

A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs [2] (Ali et al., 2021) is a detailed survey on systems for querying RDF graphs. After some preliminaries on RDF/SPARQL, the survey discusses different storage schemes and indexing techniques that enable efficient access over RDF graphs. Thereafter, both traditional and more modern join processing techniques are described, followed by query processing techniques involving other relational operators and paths. Different partitioning strategies are studied and compared. An appendix includes a survey of over one hundred distributed RDF stores and SPARQL query engines, and the techniques they use.

Miscellaneous

Querying in the Age of Graph Databases & Knowledge Graphs [6] (Arenas et al., 2021) provides notes for a tutorial on querying graphs in the era of knowledge graphs. The paper begins by studying the growth in popularity surrounding knowledge graphs, finding fewer than one hundred papers on the topic in DBLP in 2015, which grows to almost one thousand in 2020. The paper asks: what is new about knowledge graphs and how do they relate to graph databases? A brief discussion is provided on data vs. information vs. knowledge, graph data and querying, and traditions in which graphs have been used to model knowledge (semantic networks, graph databases, semantic web). The authors outline their perspective on knowledge graphs, characterised by knowledge representation, integration and production using graphs. The paper then formalises different graph data models and query language features, and concludes that the popularity of graphs for representing knowledge is due to their being a "*simple, flexible and extensible data structure*", while also being a "*a deep-rooted form of representing human knowledge*".

4 Semantics: Schemata, Rules and Ontologies

Explicit representations of semantics play an important role in many knowledge graphs. Here we provide pointers to literature covering the topics of graph schemata (in various forms), rules, and ontologies. Much of this literature comes from the Knowledge Representation and Semantic Web communities, who have long studied and debated these topics.

Books

Foundations of Semantic Web Technologies [30] (Hitzler et al., 2010) introduces key Semantic Web standards and concepts using a mix of formal definitions and examples in concrete syntax. The book begins with a general motivation and historical perspective with respect to semantics, knowledge and reasoning. The book then describes the RDF data model, the RDFS schema language, and their formal semantics. The book continues with an in-depth treatment of OWL, covering its syntax, features, and formal semantics. Rules are introduced and their relation with ontologies discussed. Query languages for RDF (including SPARQL) are presented. The book concludes with discussion on ontology engineering topics and applications of Semantic Web concepts and standards. The book uses a mix of formal definitions and examples in concrete syntax, separating introductory material from more advanced concepts. It includes exercises for students.

Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL [3] (Allemang and Hendler, 2011) provides a pragmatic introduction to the Semantic Web; the importance of semantics; various key standards including RDF, RDFS, OWL, SKOS, and SPARQL, and how they can be used on the Web. A particular focus of the book is on how these standards can be used to model data and semantics, and how they enable reasoning and querying, both in local and decentralised (Web) settings. The book also touches on the importance of semantic modelling, in terms of human communication, explanation, prediction, and integrating heterogeneous data. The focus is on the use of lightweight ontology features for semantic modelling. The discussion, though often conceptual, is largely example-based, using concrete syntaxes. Thus the book is suitable both for an academic audience, as well as practitioners interested in semantic modelling.

An Introduction to Description Logic [7] (Baader et al., 2017) provides a comprehensive introduction to the area of Description Logics, which studies decidable fragments of first order logic that form the basis of ontology languages like OWL, and can be used to define the semantics of knowledge graphs. The book first introduces basic description logics, and then describes how to define their semantics with model theory. Reasoning algorithms for expressive description logics based on tableau are introduced, and the complexity of reasoning problems is studied. The latter part of the book is dedicated to more practical aspects of description logics, including tractable profiles, query answering, concrete ontology languages, and applications. The book is primarily aimed at an academic audience, and uses formal definitions alongside examples to introduce concepts.

Validating RDF Data [22] (Labra Gayo et al., 2017) discusses the use of shapes and shape languages (including SHACL and ShEx) for validating RDF graphs. Such languages allow for specifying constraints over a graph, for encapsulating and combining multiple constraints as a "shape", for targeting specific nodes of the graph with particular shapes, and for validating graphs with respect to the shapes defined. Thus, shape languages can be seen as a form of validating schema for graphs. The book begins by introducing RDF graphs, issues relating to data quality, and the key concepts underlying shape languages. It then covers the ShEx and SHACL languages. The book concludes by discussing applications for shapes, and comparing the ShEx and SHACL languages. The discussion is example driven, using concrete syntax, and suitable for a broad audience.

An Introduction to Ontology Engineering [38] (Keet, 2018) provides a general introduction to the titular area. The book begins by discussing the notion of an ontology in Computer Science, and how ontologies are used. It then describes the logical foundation of ontologies, covering first-order logic, description logics and the OWL 2 ontology language. The next section of the book is devoted to ontology development, covering methodologies, tools, top-down approaches and bottom-up approaches. The book concludes with discussion of modern ontology engineering topics, including ontology-based data access, natural language in ontologies, contextual frameworks for representing uncertainty and temporal validity, and finally ontology modularisation. The book uses a mix of formal and example-driven presentation, with exercises provided for students.

Ontology Engineering [41] (Kendall and McGuinness, 2019) provides a pragmatic introduction to ontology engineering, with an emphasis on modelling. After an introduction to foundational aspects of ontologies, the book provides an overview of some key concepts, such as domain analysis, levels of abstraction, ontology evaluation, ontology design patterns, etc. The book then describes how to collect requirements and analyse use-cases. The following chapter introduces the importance of terminology for ontology engineering, and how to collect and curate domain terms. The book wraps up with discussion on conceptual modelling, including ontology reuse, naming conventions, metadata, etc. The book has a practical focus, relying on a more informal, didactic presentation.

The Web of Data [32] (Hogan, 2020) offers a comprehensive discussion of the Semantic Web standard, the Linked Data principles, and how they come together to realise a Web of Data. The book offers a general motivation for the Web of Data, before providing a detailed discussion of how graphs (RDF), semantic schemata (RDFS), ontologies (OWL), query languages (SPARQL), shapes (SHACL), and publishing principles (Linked Data) combine to enable data, and not just documents, to be interlinked on the Web. Each chapter provides a general motivation, examples in concrete syntax, as well as formal definitions, such that it can serve as both an introductory text book, and a reference book. Examples are included throughout for students to test their learning.

Collections

Ontology Engineering in a Networked World [66] (Suárez-Figueroa et al., eds., 2012) is a collection of book chapters relating to the topic of ontology engineering. The chapters broach a wide range of topics, including concrete methodologies for ontology engineering, ontology design patterns, requirements specification, ontology localisation, modularisation of ontologies, ontology evolution, and ontology matching. The book concludes with some more practice-oriented chapters based on the NeOn ontology engineering toolkit.

Applications and Practices in Ontology Design, Extraction, and Reasoning [17] (Cota et al., eds., 2020) collects together thirteen chapters relating to ontologies and rules. The diverse topics covered include ontology modularisation and reuse, FAIR principles, knowledge creation through mapping languages, probabilistic and preferential description logics, axiom pinpointing, defeasible reasoning, querying and reasoning via rules, as well as applications of ontologies within the humanities, the scholarly domain, and music.

Book Chapters

Foundations of Description Logics [59] (Rudolph, 2011) provides lecture notes on the theoretical foundations of description logics. The notes begin with a general introduction to description logics and the Semantic Web. The semantics of description logics are formally defined through model theory. Different description logics, and the features they support, are introduced. A number of semantic relations are introduced, namely concept equivalence, ontology equivalence and emulation. A primer is provided on modelling with description logics, showing how transitivity, cardinality constraints, etc., can be axiomatised; the open and closed world assumptions are also discussed. Reasoning tasks relating to satisfiability and entailment are described, and reasoning algorithms are sketched. Finally the concrete OWL language is introduced and mapped to description logics. The book offers a mostly formal treatment (with examples) of the topic.

5 Analytics: Theory, Algorithms, Measures & Frameworks

Graphs have long been used to conceptualise networks and interactions in a variety of domains. Analytics such as centrality, community detection, spectral analysis, etc., can then glean important insights about the respective domains from such graphs. Here we discuss literature regarding the graph theory, algorithms and measures that underlie such analytics, and the frameworks used to compute them. The literature here mainly stems from the Graph Theory and (various) Network Analysis communities, with graph processing frameworks studied by the Big Data, Database, Distributed Systems, and other related communities.

Books

Graphs, Algorithms and Optimization [42] (Kocay and Kreher, 2005) provides a general introduction to graphs, graph algorithms, and optimisation techniques. After some preliminaries on graphs, the book introduces paths, walks, and algorithms to find shortest (weighted) paths. Special classes of graphs are introduced, including bipartite graphs, line graphs, Moore graphs, and Euler tours. The book discusses trees and cycles, including algorithms for spanning trees, tree encodings, etc. Next, connectivity is discussed, including the notion of blocks, and algorithms to detect them. The book continues by introducing concepts relating to alternating paths, matchings, network flows, Hamilton cycles, digraphs, graph colourings, planar graphs, and graph embeddings on surfaces. Optimisation techniques are introduced in the form of linear programming, discrete linear programming, and related algorithms. The book thus provides a quite formal and technical introduction to graph theory, algorithms and optimisation.

Systems for Big Graph Analytics [76] (Yan et al., 2017) discusses parallel frameworks for distributed processing and analytics over large-scale graphs. The book introduces three main computational models for processing graphs in a distributed or parallel setting: vertex-based, block-based, subgraph-based and matrix-based. Vertex-based computation involves message passing between vertices of the graph, which then perform computations on the messages received. Block-based computation partitions vertices into densely-connected blocks, and defines separate communication and computation within and between blocks. In subgraph-based computation, (possibly overlapping) subgraphs are grown dynamically from seed vertices during the computation. Matrix-based computation views the graph as an adjacency or incidence matrix, and applies computation in terms of linear algebra operations. The book discusses technical concepts relating to communication and load balancing, out-of-core computation, fault tolerance, on-demand querying, shared memory abstractions, partitioning, and more besides. Systems such as Pregel, Giraph, GraphX, GraphLab, BigGraph@CUHK, Blogel, G-Thinker, PEGASUS, GBASE, SystemML, etc., are discussed throughout in relation to these concepts.

Graph Algorithms [49] (Needham and Hodler, 2019) features concrete examples of how to implement a variety of graph algorithms in Apache Spark and Neo4j. The book first contrasts transactional (OLTP) and analytical (OLAP) workloads, and describes use-cases for graph analytics. A primer on graph theory is provided, including properties of graphs (random, small-world, scale-free, etc.), different graph models, etc. Graph processing frameworks are introduced. The book then delves into specific graph algorithms and measures (and their implementation) for graph search/path finding, centrality measures and community detection. Practical use-cases are presented. The book concludes by discussing the relation between graph analytics and machine learning techniques. The book adopts a hands-on approach to introducing different graph algorithms.

The Practitioner's Guide to Graph Data [23] (Gosnell and Broecheler, 2020) provides a hands-on guide to modelling data as graphs and implementing various types of analytics and algorithms over those graphs using Gremlin, Cassandra, among other tools. After an introduction to graph data modelling, and an example use-case, the book delves into traversals using Gremlin, the use of Cassandra to index graphs, as well as techniques for navigating hierarchical data modelled as trees, graph search and path finding algorithms, collaborative filtering for recommendations, and entity resolution. The book concludes with discussion of graph algorithms, distributed graphs, graph theory and network theory. The book thus blends practical examples with more conceptual discussion.

Collections

Managing and Mining Graph Data [1] (Aggarwal and Wang, eds., 2010) is a collection of book chapters on graph data management and mining. The chapters of the book cover a variety of topics including graph data models, formal properties of graphs, graph generators, graph query languages, graph indexing, path queries and graph pattern matching, graph matching, keyword search on graphs, clustering algorithms, dense subgraph discovery, graph classification through kernels and boosting, frequent subgraph mining, streaming graphs, privacy for graphs, etc. The latter part of the book focuses on graph mining for use-cases in specific domains including the Web, social networks, software, biology, and chemistry. The collection largely targets an academic audience, with formal definitions and technical discussion found throughout its different chapters.

6 Learning: Embeddings and Architectures

One of the key sub-areas of Knowledge Graphs has been representation learning for graphs. Key techniques here include graph embeddings, which involve learning numerical representations for nodes (entities), edge labels (relations), and graphs themselves; and graph neural networks, which layer learned functions and message passing over the topology of the graph. The literature here is predominantly from the Machine Learning community.

Books

Graph Representation Learning [26] (Hamilton, 2020) is a book covering a broad range of topics relating to learning over graphs. The book begins by introducing different graphs models, and different abstract machine learning tasks one can consider over graphs. Next, different graph measures and algorithms are presented, along with spectral graph theory. The book then delves into the technical details of node embeddings and graph neural networks. It concludes with discussion of generative graph models.

Deep Learning on Graphs [47] (Ma and Tang, 2021) focuses on deep learning techniques for graphs. After a general introduction to and motivation for the topic, the authors discuss graph models, metrics, computational frameworks, and spectral graph theory. A similar introduction is provided for deep learning, including feedforward networks, convolutional and recurrent neural networks, autoencoders, and training methodologies. The book then focuses on graph embeddings, graph neural networks, and other deep learning models applied over graphs. It concludes with discussion of applications of graph neural networks for natural language processing, computer vision, data mining, and biochemistry. The book concludes with advanced topics, such as the expressivity of graph neural networks, combinatorial optimisation on graphs, etc.

Articles

Representation Learning on Graphs: Methods and Applications [27] (Hamilton et al., 2017) is an article reviewing techniques and applications relating to learning over graphs. The review primarily focuses on node and subgraph embeddings. The survey covers techniques for node embeddings based on factorisation, random walks, encoder–decoder architectures, and more besides; applications for node embeddings, such as pattern discovery, community detection, node classification and link prediction, are discussed. Techniques surveyed for subgraph embeddings include convolutional approaches, graph-coarsening, and graph neural networks; applications discussed for subgraph embeddings include subgraph classification, drug discovery, molecular classification, image classification, computer programme verification, and logical reasoning.

Knowledge Graph Embedding: A Survey of Approaches and Applications [69] (Wang et al., 2017) provides a comprehensive survey on knowledge graph embeddings, divided into translational models (TransE and related embeddings, Gaussian embeddings, etc.), and semantic matching models (RESCAL and related embeddings, neural-based embeddings, etc.). Within each category, specific embeddings are defined in terms of their entity embeddings, relation embeddings, plausibility scoring function, and constraints. The survey then discusses model training, and how negative examples can be extracted from knowledge graphs for training (under an open/closed world assumption). Embeddings are compared in terms of time and space complexity. The survey discusses how models can encode additional information, such as entity types, paths, text, rules, attributes, temporal metadata, and graph structures. The paper wraps-up with discussion of applications internal (link prediction, triple/entity classification, entity resolution) and external (relation extraction, question answering, recommendations) to the knowledge graph.

A Comprehensive Survey on Graph Neural Networks [74] (Wu et al., 2021) provides a detailed survey of advances on graph neural networks. After a general introduction, brief history, and preliminary definitions, the authors present a taxonomy of different graph neural networks that includes recurrent graph neural networks, convolutional graph neural networks, graph autoencoders, and spatio-temporal graph neural networks. They also identify tasks at different levels – node-level, edge-level and graph-level – and different learning paradigms – semi-supervised, supervised and unsupervised (embeddings). Within each category, the survey lists the specific graph neural networks that have been proposed, their inputs, their pooling and readout operations, and their time complexity. Theoretical aspects, such as VC dimension, expressivity with respect to graph isomorphism tests, universal approximation, etc., are also briefly discussed. The survey concludes with discussion of datasets, tasks, applications, and future directions.

A Survey on Knowledge Graphs: Representation, Acquisition, and Applications [37] (Ji et al., 2022) is a survey article that focuses primarily on representation learning and knowledge acquisition in the context of knowledge graphs. The survey first provides a technical introduction to representational concepts such as knowledge graph embeddings, plausibility scoring functions, encoding models, and contextual embeddings. In terms of knowledge acquisition, techniques for knowledge graph completion, entity discovery, and relation extraction are discussed. Temporal aspects are considered, along with applications relating to language models, question answering, and recommender systems.

Miscellaneous

Graph Neural Networks Meet Neural-Symbolic Computing: Survey & Perspective [44] (Lamb et al., 2020) provides a concise overview of graph neural networks in the broader context of neural-symbolic computing, and how the two interrelate. The paper first recaps different classifications of neural-symbolic computing frameworks, classifying graph neural networks as a TYPE 1 system, i.e., based on standard deep learning techniques (arguably not even neural-symbolic given the lack of symbolic representations). The paper discusses developments leading up to graph convolutional networks, graph neural networks, message-passing neural networks, and graph attention networks. Thereafter, the paper highlights the promise of combining graph neural networks with neural-symbolic reasoning through applications relating to relational learning and reasoning, combinatorial optimisation, and constraint satisfaction problems.

7 Lifecycle: Construction, Refinement, Publication

The final collection of literature that we consider relates to what we broadly call the "lifecycle" of knowledge graphs, encompassing knowledge graph construction, refinement, and publication. The literature discussed here comes primarily from the Database, Information Extraction, Machine Learning, and Semantic Web communities.

Books

Linked Data: Evolving the Web into a Global Data Space [29] (Heath and Bizer, 2011) describes how RDF graphs and Linked Data principles can combine to form a Web of Data that can be conceived of as a decentralised knowledge graph that spans the Web. The core concept is to use RDF as a graph-structured data model in which Web identifiers – URIs or IRIs – form the nodes and edge labels of the graph. Performing a HTTP lookup on these identifiers resolves – or *dereferences* – to a potentially remote RDF graph providing further data about the entity or relation (type) identified. Though they pre-date the modern popularisation of knowledge graphs, these principles have been used to publish a variety of important open knowledge graphs on the Web, including DBpedia, Freebase, Wikidata, YAGO, amongst (many) others. The book thus introduces principles and best practices for publishing graph-structured data on the Web.

Domain-Specific Knowledge Graph Construction [39] (Kejriwal, 2019) provides a brief general introduction to knowledge graphs along with some domain-specific examples. The book then focuses on a number of specific topics relating to the construction and refinement of knowledge graphs, specifically information extraction techniques, entity resolution, and knowledge graph completion. The book concludes with a discussion of ecosystems in which knowledge graphs have been broadly deployed, such as Linked Data, Google Knowledge Graphs, and Schema.org.

Designing and Building Enterprise Knowledge Graphs [61] (Sequeda & Lassila, 2021) gives a general introduction to different types of data that can be found in an enterprise setting, and the benefits of mapping data to knowledge graphs. A core focus of the book is on mapping relational databases to knowledge graphs, where it covers relevant mapping languages and mapping design patterns. The book further discusses the people, processes and tools involved in building and maintaining an enterprise knowledge graph.

Web Data APIs for Knowledge Graphs [48] (Meroño-Peñuela et al., 2021) discusses techniques and standards by which the content of (open) knowledge graphs can be published on the Web and accessed through APIs. After introducing knowledge graphs, the Linked

Data principles, RDF, SPARQL and GraphQL, the book discusses how knowledge graphs can be accessed using HTTP requests, REST APIs, and SPARQL services, further discussing a number of tools to define REST APIs based on SPARQL queries, and to transform the response for SPARQL queries. The book is thus of particular interest to readers interested in publishing knowledge graphs on the Web.

Articles

Quality Assessment for Linked Data: A Survey [77] (Zaveri et al., 2016) collates a set of quality dimensions for Linked Datasets collected from the literature. Given that Linked Data refers to a set of principles for publishing graph-structured (RDF) data on the Web, many (though not all) dimensions apply for knowledge graphs [34]. The dimensions are organised into four high-level categorisations. Accessibility encompasses dimensions relating to how data can be accessed such as availability, licensing, interlinking, security, and performance. Intrinsic dimensions refer to potential issues within the data, such as syntactic validity, semantic accuracy, consistency, conciseness, and completeness. Contextual dimensions depend on a particular purpose, and include relevancy, trustworthiness, understandability, and timeliness. Representational dimensions deal with how data are represented, and include conciseness, interoperability, interpretability, and versatility. The paper includes discussion of measures for dimensions, as well as tools to detect issues.

Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods [53] (Paulheim, 2017) provides a comprehensive overview of techniques for refining knowledge graphs. The article first introduces the notion of a knowledge graph in the context of the Semantic Web, surveying prominent open and enterprise knowledge graphs. A categorisation of knowledge graph refinement approaches are then presented, based on whether they aim to complete or detect errors in the knowledge graph, what elements of the knowledge graph they target, and whether or not they depend on external sources. A categorisation of evaluation methods is also presented, including partial goal standards, using the knowledge graph itself as a silver standard, retrospective evaluation, and performance evaluation. Under techniques for knowledge graph completion, the survey considers methods that complete type assertions, and that predict relations. Techniques considered for error detection include binary classification (correct/incorrect), statistical techniques, reasoning, etc. The survey concludes with a summary of its findings.

Miscellaneous

Knowledge Graph Lifecycle: Building & Maintaining Knowledge Graphs [64] (Simsek et al., 2021) defines a lifecycle for knowledge graphs involving four stages: knowledge creation, knowledge hosting, knowledge curation (incorporating assessment, cleaning and enrichment), and knowledge deployment. Knowledge creation can involve manual processes or mappings from legacy sources. Knowledge hosting primarily refers to storing the graph in a database, enabling it to be queried. Knowledge curation involves three sub-phases: assessment aims to gather information about the quality of the knowledge graph; cleaning aims to pinpoint and address specific quality issues; enrichment aims to improve the completeness of the knowledge graph. Finally, knowledge deployment refers to the use of the knowledge graph for (e.g., end-user) applications. The paper concludes with lessons learnt from practical deployments of knowledge graphs.

8 Conclusions

Knowledge graphs have been the subject of a vast amount of publications in recent years. Our goal in these lecture notes has been to provide the reader with some orientation on how to approach this literature: on where to learn more about knowledge graphs in general, or about topics relating to data, semantics, analytics, learning, or the knowledge graph lifecycle. Our hope is that by the time you have read this far, you will have deviated from these notes in order to go out and explore some of the literature mentioned.

Much of the literature discussed herein deals with state-of-the-art techniques, tools, languages, etc., relating to general or specific aspects of knowledge graphs. But what about the future of Knowledge Graphs as a research area? As discussed in the introduction to these lecture notes, we expect Knowledge Graphs to establish itself as a research area in its own right, complete with its own conferences, journals, etc. This area will ideally serve as a confluence point for researchers coming from different constituent communities to pursue novel ideas relating to the use of a graph abstraction to collate and deploy knowledge at large scale. In this sense, the future of Knowledge Graphs should hopefully see combinations of techniques from different areas, such as graph representation learning that can consider semantics expressed as ontologies or rules, or combinations of graph querying and analytics in the form of hybrid languages and query processing tools, etc. For more discussion on possible future research directions for knowledge graphs, we refer the interested reader to the lecture notes "*Knowledge Graphs: Research Directions*" [32].

References

1 Charu C. Aggarwal and Haixun Wang, editors. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer, 2010. doi:10.1007/978-1-4419-6045-0.

2 Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs. *VLDB Journal*, 2021. doi:10.1007/s00778-021-00711-3.

3 Dean Allemang and James A. Hendler. *Semantic Web for the Working Ontologist - Effective Modeling in RDFS and OWL, Second Edition*. Morgan Kaufmann, 2011. URL: http://www.elsevierdirect.com/product.jsp?isbn=9780123859655.

4 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017. doi:10.1145/3104031.

5 Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Computing Surveys*, 40(1):1:1–1:39, 2008. doi:10.1145/1322432.1322433.

6 Marcelo Arenas, Claudio Gutiérrez, and Juan F. Sequeda. Querying in the Age of Graph Databases and Knowledge Graphs. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 2821–2828. ACM, 2021. doi:10.1145/3448016.3457545.

7 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, Cambridge, United Kingdom, 2017. doi:10.1017/9781139025355.

8 Jesus Barrasa, Amy E. Hodler, and Jim Webber. *Knowledge Graphs: Data in Context for Responsive Businesses*. O'Reilly Media, 2021.

9 Dave Bechberger and Josh Perryman. *Graph Databases in Action*. Manning, 2020.

10 Luigi Bellomarini, Daniele Fakhoury, Georg Gottlob, and Emanuel Sallinger. Knowledge Graphs and Enterprise AI: The Promise of an Enabling Technology. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 26–37. IEEE Computer Society, 2019. doi:10.1109/icde.2019.00011.

11 Michael K. Bergman. A Common Sense View of Knowledge Graphs. Adaptive Information, Adaptive Innovation, Adaptive Infrastructure Blog, July 2019. URL: `http://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/`.

12 Andreas Blumauer and Helmut Nagy. *The Knowledge Graph Cook Book: Recipes That Work*. monochrom, 2020.

13 Kurt Bollacker, Patrick Tufts, Tomi Pierce, and Robert Cook. A platform for scalable, collaborative, structured information integration. In Ullas Nambiar and Zaiqing Nie, editors, *Intl. Workshop on Information Integration on the Web (IIWeb'07)*, 2007. URL: `https://www.aaai.org/Papers/Workshops/2007/WS-07-14/WS07-14-004.pdf`.

14 Piero Andrea Bonatti, Stefan Decker, Axel Polleres, and Valentina Presutti. Knowledge Graphs: New Directions for Knowledge Representation on the Semantic Web (Dagstuhl Seminar 18371). *Dagstuhl Reports*, 8(9):29–111, 2018. URL: `https://drops.dagstuhl.de/opus/volltexte/2019/10328/pdf/dagrep_v008_i009_p029_18371.pdf`.

15 Angela Bonifati, George H. L. Fletcher, Hannes Voigt, and Nikolay Yakovets. *Querying Graphs*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018. `doi:10.2200/S00873ED1V01Y201808DTM051`.

16 Spencer Chang. Scaling Knowledge Access and Retrieval at Airbnb. AirBnB Medium Blog, September 2018. URL: `https://medium.com/airbnb-engineering/scaling-knowledge-access-and-retrieval-at-airbnb-665b6ba21e95`.

17 Giuseppe Cota, Marilena Daquino, and Gian Luca Pozzato, editors. *Applications and Practices in Ontology Design, Extraction, and Reasoning*, volume 49 of *Studies on the Semantic Web*. IOS Press, 2020. `doi:10.3233/SSW49`.

18 Deepika Devarajan. Happy Birthday Watson Discovery. IBM Cloud Blog, December 2017. URL: `https://www.ibm.com/blogs/bluemix/2017/12/happy-birthday-watson-discovery/`.

19 Lisa Ehrlinger and Wolfram Wöß. Towards a Definition of Knowledge Graphs. In Michael Martin, Martí Cuquet, and Erwin Folmer, editors, *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*, volume 1695 of *CEUR Workshop Proceedings*. Sun SITE Central Europe (CEUR), September 2016. URL: `http://ceur-ws.org/Vol-1695/paper4.pdf`.

20 Dieter Fensel, Umutcan Simsek, Kevin Angele, Elwin Huaman, Elias Kärle, Oleksandra Panasiuk, Ioan Toma, Jürgen Umbrich, and Alexander Wahler. *Knowledge Graphs: Methodology, Tools and Selected Use Cases*. Springer, 2020. `doi:10.1007/978-3-030-37439-6`.

21 George H. L. Fletcher, Jan Hidders, and Josep Lluís Larriba-Pey, editors. *Graph Data Management: Fundamental Issues and Recent Developments*. Data-Centric Systems and Applications. Springer, 2018. `doi:10.1007/978-3-319-96193-4`.

22 José Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitris Kontokostas. *Validating RDF Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2017. `doi:10.2200/S00786ED1V01Y201707WBE016`.

23 Denise Gosnell and Matthias Broecheler. *The Practitioner's Guide to Graph Data*. O'Reilly Media, 2020.

24 Claudio Gutiérrez and Juan F. Sequeda. Knowledge graphs. *Commun. ACM*, 64(3):96–104, 2021. `doi:10.1145/3418294`.

25 Ferras Hamad, Isaac Liu, and Xian Xing Zhang. Food Discovery with Uber Eats: Building a Query Understanding Engine. Uber Engineering Blog, June 2018. URL: `https://eng.uber.com/uber-eats-query-understanding/`.

26 William L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020. `doi:10.2200/S01045ED1V01Y202009AIM046`.

27 William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017. URL: `http://sites.computer.org/debull/A17sept/p52.pdf`.

28 Qi He, Bee-Chung Chen, and Deepak Agarwal. Building The LinkedIn Knowledge Graph. LinkedIn Blog, October 2016. URL: `https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph`.

29 Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space (1st Edition)*, volume 1 of *Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool, 2011.

30 Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Chapman and Hall/CRC Press, 2010.

31 Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. YAGO2: Exploring and querying world knowledge in time, space, context, and many languages. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011 (Companion Volume)*, pages 229–232. ACM Press, March 2011.

32 Aidan Hogan. Knowledge Graphs: Research Directions. In Marco Manna and Andreas Pieris, editors, *Reasoning Web. Declarative Artificial Intelligence - 16th International Summer School 2020, Oslo, Norway, June 24-26, 2020, Tutorial Lectures*, volume 12258 of *Lecture Notes in Computer Science*, pages 223–253. Springer, 2020.

33 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers, 2021. `doi:10.2200/S01125ED1V01Y202109DSK022`.

34 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge Graphs. *ACM Computing Surveys*, 54(4):71:1–71:37, 2021. `doi:10.1145/3447772`.

35 Valentina Janev, Damien Graux, Hajira Jabeen, and Emanuel Sallinger, editors. *Knowledge Graphs and Big Data Processing*, volume 12072 of *Lecture Notes in Computer Science*. Springer, 2020. `doi:10.1007/978-3-030-53199-7`.

36 Daniel Janke and Steffen Staab. Storing and Querying Semantic Data in the Cloud. In Claudia d'Amato and Martin Theobald, editors, *Reasoning Web. Learning, Uncertainty, Streaming, and Scalability - 14th International Summer School 2018, Esch-sur-Alzette, Luxembourg, September 22-26, 2018, Tutorial Lectures*, volume 11078 of *Lecture Notes in Computer Science*, pages 173–222. Springer, 2018. `doi:10.1007/978-3-030-00338-8`.

37 Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2021. `doi:10.1109/TNNLS.2021.3070843`.

38 C. Maria Keet. *An Introduction to Ontology Engineering*. College Publications, 2018. URL: `https://open.umn.edu/opentextbooks/textbooks/590`.

39 Mayank Kejriwal. *Domain-Specific Knowledge Graph Construction*. Springer Briefs in Computer Science. Springer, 2019. `doi:10.1007/978-3-030-12375-8`.

40 Mayank Kejriwal, Craig A. Knoblock, and Pedro Szekely, editors. *Knowledge Graphs: Fundamentals, Techniques, and Applications*. The MIT Press, 2021.

41 Elisa F. Kendall and Deborah L. McGuinness. *Ontology Engineering*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2019. `doi:10.2200/S00834ED1V01Y201802WBE018`.

42 William L. Kocay and Donald L. Kreher. *Graphs, algorithms and optimization*. Chapman&Hall/CRC Press, 2005.

43 Arun Krishnan. Making search easier: How Amazon's Product Graph is helping customers find products more easily. Amazon Blog, August 2018. URL: `https://blog.aboutamazon.com/innovation/making-search-easier`.

44 Luís C. Lamb, Artur S. d'Avila Garcez, Marco Gori, Marcelo O. R. Prates, Pedro H. C. Avelar, and Moshe Y. Vardi. Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4877–4884. ijcai.org, 2020. `doi:10.24963/ijcai.2020/679`.

45 Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web Journal*, 6(2):167–195, 2015.

46 Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2181–2187. AAAI Press, August 2015.

47 Yao Ma and Jiliang Tang. *Deep Learning on Graphs*. Cambridge University Press, 2021. URL: `https://web.njit.edu/~ym329/dlg_book/`.

48 Albert Meroño-Peñuela, Pasquale Lisena, and Carlos Martinez-Ortiz. *Web Data APIs for Knowledge Graphs: Easing Access to Semantic Data for Application Developers*. Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers, 2021. `doi:10.2200/S01114ED1V01Y202107DSK021`.

49 Mark Needham and Amy E. Hodler. *Graph Algorithms*. O'Reilly Media, 2019.

50 Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

51 Natasha F. Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale Knowledge Graphs: Lessons and Challenges. *ACM Queue*, 17(2):20, 2019.

52 Jeff Z. Pan, Guido Vetere, José Manuél Gómez-Pérez, and Honghan Wu, editors. *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. Springer, 2017. `doi:10.1007/978-3-319-45654-6`.

53 Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web Journal*, 8(3):489–508, 2017. `doi:10.3233/SW-160218`.

54 R. J. Pittman, Amit Srivastava, Sanjika Hewavitharana, Ajinkya Kale, and Saab Mansour. Cracking the Code on Conversational Commerce. eBay Blog, April 2017. URL: `https://www.ebayinc.com/stories/news/cracking-the-code-on-conversational-commerce/`.

55 Jay Pujara, Hui Miao, Lise Getoor, and William W. Cohen. Knowledge graph identification. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, , Josian Xavier Parreira, Lora Aroyo, Natasha Fridman Noy, Christopher A. Welty, and Krzysztof Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 542–557. Springer, October 2013. `doi:10.1007/978-3-642-41335-3_34`.

56 Guilin Qi, Huajun Chen, Kang Liu, Haofen Wang, Qiu Ji, and Tianxing Wu. *Knowledge Graph*. Springer, 2020. (to appear).

57 Guilin Qi, Huajun Chen, Kang Liu, Haofen Wang, Qiu Ji, and Tianxing Wu. *Knowledge Graph*. Springer Singapore, 2022.

58 Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases, 2nd Edition*. O'Reilly Media, 2015.

59 Sebastian Rudolph. Foundations of Description Logics. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter F. Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, volume 6848 of *Lecture Notes in Computer Science*, pages 76–136. Springer, August 2011.

60 Edward W. Schneider. Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis. In *Association for the Development of Instructional Systems (ADIS), Chicago, Illinois, April 1972*, 1973.

61 Juan Sequeda and Ora Lassila. *Designing and Building Enterprise Knowledge Graphs.* Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers, 2021. `doi:10.2200/S01105ED1V01Y202105DSK020`.

62 Stephan Seufert, Patrick Ernst, Srikanta J. Bedathur, Sarath Kumar Kondreddi, Klaus Berberich, and Gerhard Weikum. Instant Espresso: Interactive Analysis of Relationships in Knowledge Graphs. In Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, pages 251–254. ACM Press, April 2016.

63 Saurabh Shrivastava. Bring rich knowledge of people, places, things and local businesses to your apps. Bing Blogs, July 2017. URL: `https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps`.

64 Umutcan Simsek, Kevin Angele, Elias Kärle, Juliette Opdenplatz, Dennis Sommer, Jürgen Umbrich, and Dieter Fensel. Knowledge Graph Lifecycle: Building and Maintaining Knowledge Graphs. In David Chaves-Fraga, Anastasia Dimou, Pieter Heyvaert, Freddy Priyatna, and Juan F. Sequeda, editors, *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Online, June 6, 2021*, volume 2873 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021. URL: `http://ceur-ws.org/Vol-2873/paper12.pdf`.

65 Amit Singhal. Introducing the Knowledge Graph: things, not strings. Google Blog, May 2012. URL: `https://www.blog.google/products/search/introducing-knowledge-graph-things-not/`.

66 Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, Enrico Motta, and Aldo Gangemi, editors. *Ontology Engineering in a Networked World.* Springer, 2012. `doi:10.1007/978-3-642-24794-1`.

67 Ilaria Tiddi, Freddy Lécué, and Pascal Hitzler, editors. *Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges*, volume 47 of *Studies on the Semantic Web*. IOS Press, 2020. URL: `http://ebooks.iospress.nl/volume/knowledge-graphs-for-explainable-artificial-intelligence-foundations-applications-and-challenges`.

68 Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

69 Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, December 2017. `doi:10.1109/TKDE.2017.2754499`.

70 Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph Embedding by Translating on Hyperplanes. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1112–1119. AAAI Press, July 2014.

71 Gerhard Weikum. Knowledge Graphs 2021: A Data Odyssey. *Proc. VLDB Endow.*, 14(12):3233–3238, 2021. URL: `http://www.vldb.org/pvldb/vol14/p3233-weikum.pdf`.

72 Gerhard Weikum, Xin Luna Dong, Simon Razniewski, and Fabian M. Suchanek. Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. *Found. Trends Databases*, 10(2-4):108–490, 2021. doi:10.1561/1900000064.

73 Peter T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, 2012. doi:10.1145/2206869.2206879.

74 Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021. doi:10.1109/TNNLS.2020.2978386.

75 Marcin Wylot, Manfred Hauswirth, Philippe Cudré-Mauroux, and Sherif Sakr. RDF Data Storage and Query Processing Schemes: A Survey. *ACM Computing Surveys*, 51(4):84:1–84:36, 2018. doi:10.1145/3177850.

76 Da Yan, Yuanyuan Tian, and James Cheng. *Systems for Big Graph Analytics*. Springer Briefs in Computer Science. Springer, 2017. doi:10.1007/978-3-319-58217-7.

77 Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for Linked Data: A Survey. *Semantic Web Journal*, 7(1):63–93, 2016.

Reasoning in Knowledge Graphs

Ricardo Guimarães
University of Bergen, Norway

Ana Ozaki
University of Bergen, Norway

Abstract

Knowledge Graphs (KGs) are becoming increasingly popular in the industry and academia. They can be represented as labelled graphs conveying structured knowledge in a domain of interest, where nodes and edges are enriched with metaknowledge such as time validity, provenance, language, among others. Once the data is structured as a labelled graph one can apply reasoning techniques to extract relevant and insightful information. We provide an overview of deductive and inductive reasoning approaches for reasoning in KGs.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Knowledge Graphs, Description Logics, Knowledge Graph Embeddings

Digital Object Identifier 10.4230/OASIcs.AIB.2022.2

Category Invited Paper

Funding *Ricardo Guimarães*: The first author is supported by the ERC project "Lossy Preprocessing" (LOPRE), grant number 819416, led by Prof. Saket Saurabh.
Ana Ozaki: The second author is supported by the NFR project "Learning Description Logic Ontologies", grant number 316022.

Acknowledgements Part of this work has been done in the context of CEDAS (Center for Data Science, University of Bergen, Norway).

1 Introduction

Knowledge Graphs (KGs) [55, 61, 111, 119] are becoming increasingly popular in the industry and academia. They can be represented as labelled graphs conveying structured knowledge in a domain of interest, where nodes and edges are enriched with metaknowledge. Provenance and time validity are the most common kinds of metaknowledge. Since facts in KGs usually come from multiple datasources, it is important to record the provenance information, which is often in the format of an URL (or multiple URLs). Facts may happen multiple times and, therefore, a temporal dimension with time validity is important to record such information. Other kinds of metaknowledge include contextual information and language.

One of the most popular methods to model KGs consists in representing them as directed edge-labelled graphs [55, 61]. In this model, each entity in the domain of interest (people, places) are represented as nodes, while the different relations between pairs of these entities are expressed as as directed edges with a label that specifies the type of the relationship. Figure 1 depicts an example of such graph using entities such as *Artur Ávila* and *Brazil*, and relationships such as *citizenship*.

In addition to data models, there are concrete languages and systems which implement KGs. The Resource Description Framework (RDF) [79] is one of the most prominent of these languages. In fact, RDF is the W3C standard for writing KGs. A KG written in RDF, that is an *RDF graph*, is a collection of triples $(subject, predicate, object)$ that indicate that a relationship (given by the predicate) holds between the entities given as

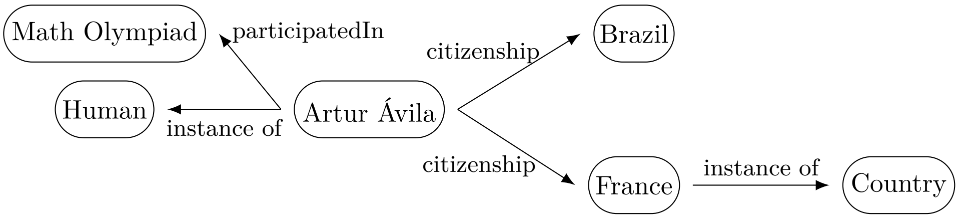


Figure 1 Example of directed edge-labelled graph.

subject and object. Example 1 depicts the RDF graph corresponding to the directed edge-labelled graph from Figure 1. Here, we will also refer to triples using a prefixed notation: $predicate(subject, object)$, which is closer to the logical formalisations discussed in Section 2.

▶ **Example 1.** We present below the RDF graph using Terse RDF Syntax, known as "Turtle", each line corresponds to a triple. The relationships (predicates) with prefix `rdf` and `rdfs` are imported from existing vocabularies that define the intended meaning of the terms.

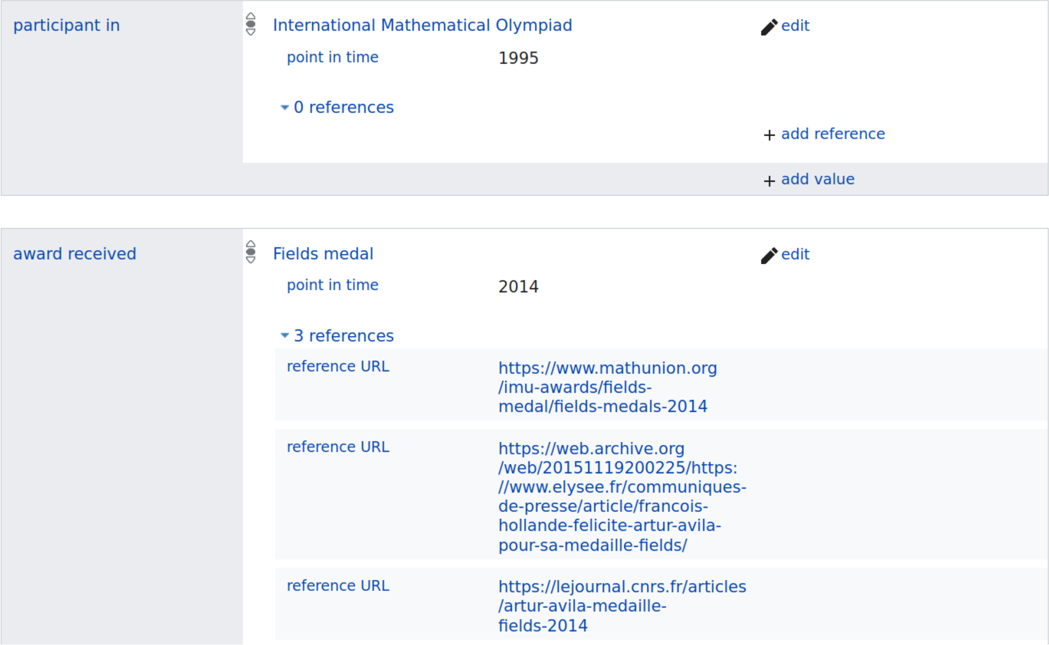
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <http://www.example.org/> .

# "Artur Avila" "is a" "Human"
:ArturAvila rdf:type :Human .
# "Artur Avila" "has country of citizenship" "France"
:ArturAvila :citizenship  :France .
# "Artur Avila" "has country of citizenship" "Brazil"
:ArturAvila :citizenship :Brazil .
# "France" "is a" "Country"
:France rdf:type :Country .
# "Artur Avila" "participated in" "Math Olympiad"
:ArturAvila :participatedIn :MathOlympiad .
```

As example of well-known KGs, we have Wikidata [112], a KG which contains much of the information displayed on Wikipedia. Example 2 discusses an excerpt of the Wikidata about Artur Avila.

▶ **Example 2.** Figure 2 contains some information about the mathematician Artur Ávila. Binary relations, such as `participantIn`, serve as labels to edges connecting nodes in the graph. In this example, the nodes would be the mathematician, the International Mathematical Olympiad, and the Fields medal. The fact that he participated in this event can be represented with the triple `participantIn`(`ArturAvila`, `MathOlympiad`). There is temporal metaknowledge associated with this fact which is the year of the participation, 1995. As we can see in Figure 2, not every fact is annotated with all the relevant metaknowledge, such as provenance information. The fact that Artur Ávila won the Fields medal is annotated with both temporal information and provenance (in the format of URLs).

Once the data is structured as a labelled graph one can apply reasoning techniques to extract relevant and insightful information. The three main classical types of reasoning are deduction, induction, and abduction:

- in deduction, we assume that a proposition or a formula α holds and, if $\alpha \to \beta$ is valid, we can deduce β;



participant in

International Mathematical Olympiad ✎ edit

point in time 1995

▾ 0 references

+ add reference

+ add value

award received

Fields medal ✎ edit

point in time 2014

▾ 3 references

reference URL https://www.mathunion.org/imu-awards/fields-medal/fields-medals-2014

reference URL https://web.archive.org/web/20151119200225/https://www.elysee.fr/communiques-de-presse/article/francois-hollande-felicite-artur-avila-pour-sa-medaille-fields/

reference URL https://lejournal.cnrs.fr/articles/artur-avila-medaille-fields-2014

Figure 2 Excerpt of the Wikidata page of Artur Ávila (Q715043).

- in induction, we have that α and β holds and attempt to generalize the facts and generate a *rule*, which is an expression of the form $\alpha \to \beta$ (or $\beta \to \alpha$);
- finally, in abduction, we assume that a proposition or a formula β holds and, if $\alpha \to \beta$ is valid, we attempt to find an explanation α for β.

▶ **Example 3.** Consider the rule "if someone is a participant of an event then this person attended the event", which can be expressed in First Order logic (FOL) with the sentence $\forall x, y(\texttt{participantIn}(x, y) \to \texttt{attendant}(x, y))$ (and in Description Logic with the role inclusion $\texttt{participantIn} \sqsubseteq \texttt{attendant}$). Given the triple in Example 2 and the rule, we can deduce that Artur Ávila attended the event, in symbols, $\texttt{attendant}(\texttt{ArturAvila}, \texttt{MathOlympiad})$. If, instead of the rule, we have $\texttt{attendant}(\texttt{ArturAvila}, \texttt{MathOlympiad})$ and the triple in Example 2, then an inductive procedure could attempt to generalize the fact and generate the rule $\forall x, y(\texttt{participantIn}(x, y) \to \texttt{attendant}(x, y))$ (or $\forall x, y(\texttt{attendant}(x, y) \to \texttt{participantIn}(x, y))$, which can happen since inductive procedures can make wrong generalizations). Finally, given the rule and the fact $\texttt{attendant}(\texttt{ArturAvila}, \texttt{MathOlympiad})$, an abductive procedure could attempt to find the triple in Example 2 as an explanation for the fact $\texttt{attendant}(\texttt{ArturAvila}, \texttt{MathOlympiad})$.

While reasoning can often empower information retrieval [14, 63] and reasoners can work as query mechanisms, our focus is on reasoning rather than data retrieval. Pure database-like retrieval is strictly more constrained, as nothing that is not asserted can be derived (except in the case that the missing data is treated immediately as false). Yet, the line between the two is sometimes blurry [61]. Here, we focus on the main approaches commonly regarded as reasoning. We provide an overview of deductive (Section 2) and inductive (Section 3) reasoning approaches for reasoning in KGs. We do not cover query languages such as SPARQL [90], even though one can enrich SPARQL queries with reasoning capabilities. We conclude in Section 5.

2 Deductive Reasoning

Deductive reasoning in KGs is commonly performed by mapping its contents to a logic-based formalism. With this transformation in place, users and designers can apply reasoners for extracting logical consequences from the information specified in a KG. In this section, we discuss three of the most popular formalisms for deductive reasoning with KGs: Description Logics, Datalog and SHACL. Description Logics underpin the Web Ontology Language, OWL 2 [84], which is the current W3C standard for ontologies. Datalog corresponds to a family of languages inspired by the logical programming language Prolog, whose computational properties makes it attractive in many use-cases. Finally, SHACL is a more recent W3C recommendation designed to validate constraints in RDF graphs.

2.1 Description Logics

Description logics (DLs) are a family of knowledge representation formalisms [7]. Each DL has its own language with its own expressivity and, thus, different computational costs for different tasks. As already mentioned, DLs underpin OWL (including the current version OWL 2). Thus, the field of DL flourished together with the popularisation of ontologies as a way of sharing knowledge in disciplines such as Medicine and Biology. While the use-cases of ontologies and KGs diverge, the two are, nevertheless, related. KGs such as DBPedia incorporate ontologies [71] that help users and developers to understand the data.

In the following, whenever we refer to a DL ontology or a knowledge base, we mean a finite set of formulas (or axioms) in a DL. Despite their differences, these languages are very similar in the way which they are used to describe knowledge. Moreover, most DLs have decidable reasoning problems and are tailored for specific applications making DL ontologies valuable tools for deductive reasoning with KGs, even if DLs cannot capture everything that a KG can represent. In all DLs, the main ideas of the domain of interest are described via concept descriptions. These concept descriptions essentially create classes to which one can assign the elements of the domain. Then, a DL ontology will contain statements (formulas) that, among other things, determine how these concept descriptions relate to each other. We will discuss more about representing knowledge with DLs next.

Given a domain of interest, the first step is to determine the key notions to be described, which will then form a set of terms. These terms can be either concept names, role names or individual names. Concept names determine the basic groups to which elements of the domain may belong, role names the basic relationships between these elements and individual names refer to (some) elements of the domain. The set of terms in an ontology is called *signature*, composed of three pairwise disjoint sets: *concept names* ($\mathsf{N_C}$), *role names* ($\mathsf{N_R}$) and *individual names* ($\mathsf{N_I}$).

▶ **Example 4.** If we are modelling knowledge on notable scientists, following Figure 2, we could have among the atomic concepts in our signature `Scientist`, `Award` and `University`. We could also have roles such as `receivedAward` and `participantIn`. Finally, we would need individual names to refer to particular elements of the domain, such as `ArturAvila`, `FieldsMedal` and `MathOlympiad`.

These are just the fundamental building blocks, as it is possible to build complex concept descriptions (and sometimes even complex roles) by using a set of constructors, which varies according to the DL selected. Many DLs allow the ontology engineer to represent the conjunction (intuitively, the intersection) of two concepts, for instance, $\texttt{Scientist} \sqcap \texttt{Brazilian}$ to refer to Brazilian scientists. Table 1 lists the concept constructors allowed in the DL $\mathcal{ALC}$.

Table 1 Complex concepts in $\mathcal{ALC}$. $\mathtt{C}$, $\mathtt{D}$ are concept expressions and $\mathtt{r} \in \mathsf{N_R}$.

Name	Syntax	Semantic
Conjunction	$\mathtt{C} \sqcap \mathtt{D}$	$\mathtt{C}^{\mathcal{I}} \cap \mathtt{D}^{\mathcal{I}}$
Disjunction	$\mathtt{C} \sqcup \mathtt{D}$	$\mathtt{C}^{\mathcal{I}} \cup \mathtt{C}^{\mathcal{I}}$
Negation	$\neg\mathtt{C}$	$\Delta^{\mathcal{I}} \setminus \mathtt{C}^{\mathcal{I}}$
Existential Restriction	$\exists\mathtt{r}.\mathtt{C}$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.(x,y) \in \mathtt{r}^{\mathcal{I}} \wedge y \in \mathtt{C}^{\mathcal{I}}\}$
Value Restriction	$\forall\mathtt{r}.\mathtt{C}$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y.(x,y) \in \mathtt{r}^{\mathcal{I}} \rightarrow y \in \mathtt{C}^{\mathcal{I}}\}$

Using concepts, roles and individuals, one can write formulas which express the constraints about the domain of interest. These formulas, or axioms, also vary according to the DL at hand. Regardless of the DL, ontologies are often split into two parts, the TBox and the ABox. The TBox contains the terminological knowledge, that is, the relationships between concepts and between roles, while the ABox contains the assertions, which concern characteristics of individuals. Table 2 lists the types of axioms allowed in $\mathcal{ALC}$. *Concept inclusions* and *equalities* are TBox axioms, while concept and role assertions are ABox axioms.

Table 2 Axioms in $\mathcal{ALC}$, $\mathtt{a}, \mathtt{b} \in \mathsf{N_I}$.

Name	Syntax	Semantics
Concept inclusion	$\mathtt{C} \sqsubseteq \mathtt{D}$	$\mathtt{C}^{\mathcal{I}} \subseteq \mathtt{D}^{\mathcal{I}}$
Concept equality	$\mathtt{C} \equiv \mathtt{D}$	$\mathtt{C}^{\mathcal{I}} = \mathtt{D}^{\mathcal{I}}$
Concept assertion	$\mathtt{C}(\mathtt{a})$	$\mathtt{a}^{\mathcal{I}} \in \mathtt{C}^{\mathcal{I}}$
Role assertion	$\mathtt{r}(\mathtt{a},\mathtt{b})$	$(\mathtt{a}^{\mathcal{I}}, \mathtt{b}^{\mathcal{I}}) \in \mathtt{r}^{\mathcal{I}}$

Now, we make additional notes about the relationship between KGs and DL ontologies. As we mentioned in the beginning of this section, KGs may have ontologies associated to them. In this context, the "ontology" usually refers to the part of KG that corresponds to a TBox, while the remaining part of the KG corresponds to an ABox [61]. There are also components in KGs in general, even when considering RDF alone, that cannot be mapped into DL axioms (such as metadata, as discussed in Section 1). Example 5 continues our running example, with a simple $\mathcal{ALC}$ ontology which describes some constraints about the domain.

▶ **Example 5.**

$$\begin{aligned}
\mathtt{Scientist} &\sqsubseteq \mathtt{Human}\\
\mathtt{Human} &\sqsubseteq \neg\mathtt{University}\\
\mathtt{PhD} &\sqsubseteq \exists\mathtt{educatedAt}.\mathtt{University}\\
&\mathtt{Scientist}(\mathtt{ArturAvila})\\
&\mathtt{participantIn}(\mathtt{ArturAvila}, \mathtt{MathOlympiad})
\end{aligned}$$

In the ontology above, the first axiom states that every scientist is a human, and the second says that no human is a university. The third axiom says that everyone with a PhD title should have been educated at some university. The last two are assertions: one states that Artur Ávila is a scientist and the other that he participated in the International Mathematical Olympiad.

In DLs, interpretations act as the most popular form of semantics. An interpretation is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is an arbitrary set of elements and $\cdot^{\mathcal{I}}$ is a function which maps each concept to a subset of $\Delta^{\mathcal{I}}$, each role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual name to an element of $\Delta^{\mathcal{I}}$. Each axiom in an ontology places a new constraint on interpretations that satisfy the ontology. Tables 1 and 2 describe how each axiom constrains the possible interpretations of an ontology. If an interpretation $\mathcal{I}$ complies with every requirement specified by an ontology $\mathcal{O}$, we say that $\mathcal{I}$ satisfies $\mathcal{O}$, in symbols $\mathcal{I} \models \mathcal{O}$. Example 6 clarifies this notion with an interpretation for the ontology in Example 5.

▶ **Example 6.** Consider the following interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ such that

$$\begin{aligned}
\Delta^{\mathcal{J}} &= \{\texttt{ArturAvila}, \texttt{Bob}, \texttt{IMO1995}, \texttt{UFRJ}\};\\
\texttt{Human}^{\mathcal{J}} &= \{\texttt{ArturAvila}, \texttt{Bob}\};\\
\texttt{Scientist}^{\mathcal{J}} &= \{\texttt{ArturAvila}\};\\
\texttt{PhD}^{\mathcal{J}} &= \{\texttt{ArturAvila}\};\\
\texttt{University}^{\mathcal{J}} &= \{\texttt{UFRJ}\};\\
\texttt{educatedAt}^{\mathcal{J}} &= \{(\texttt{ArturAvila}, \texttt{UFRJ})\};\\
\texttt{participantIn}^{\mathcal{J}} &= \{(\texttt{ArturAvila}, \texttt{IMO1995})\};\\
\texttt{ArturAvila}^{\mathcal{J}} &= \texttt{ArturAvila}; \text{ and}\\
\texttt{MathOlympiad}^{\mathcal{J}} &= \texttt{IMO1995}.
\end{aligned}$$

The interpretation $\mathcal{J}$ satisfies the ontology in Example 5.

2.1.1 Attributed DLs

While DLs provide an important advantage when reasoning with OWL ontologies, there are important features seen in real-world KGs that cannot be captured in a useful way in traditional DLs. OWL and RDF were designed using a directed edge-labelled model, which enforces every piece of data to be either a node (an entity or literal) or a relation. However, not only there are RDF graphs (and even OWL features) that cannot be expressed in DLs, but there are other graph data models which allow nodes and edges to be enriched with annotations. In a property graph (used for instance in Neo4J), each node and relation may have a map from keys to values as annotations. The Wikidata model is even more flexible, as whole statements (already more complex than assertions) may have different annotations (called qualifiers), with multiple values for the same type of qualification. For example, in Figure 2 the year 1995 is the value for the qualifier "point in time" for the statement "Artur Ávila participated in the International Mathematical Olympiad".

Attributed DLs [21, 22, 65, 68, 85] add non-functional attribute-value pairs to DL axioms (not only assertions) with the goal of representing annotations. This extension presupposes a set of variables $\mathsf{N_V}$ which can be used to build *specifiers.* These are expressions representing sets of annotations.

▶ **Definition 7.** *The set of specifiers* $\mathbf{S}$ *is the smallest set containing the following expressions:*
variables: X*;*
closed specifiers: $[a_1 : v_1, \ldots, a_n : v_n]$*; and*
open specifiers: $\lfloor a_1 : v_1, \ldots, a_n : v_n \rfloor$*;*
where $X \in \mathsf{N_V}$ *is a variable,* $n \in \mathbb{N}$*,* $a_i \in \mathsf{N_I}$ *and* $v_i \in \mathsf{N_I} \cup \{+\} \cup X.c$ *with* $c \in \mathsf{N_I}$.

The symbol "+" has the meaning "at least one". A closed specifier is satisfied iff the set of annotations matches exactly the specification, while an open specifier requires only that the annotations appear (others might occur as well).

▶ **Example 8.** Consider the statements in Figure 2. Using attributed DLs, we can express specifically the statement about Ávila's participation as follows:

$$\mathtt{participantIn}(\mathtt{ArturAvila}, \mathtt{mathOlympiad})@[\mathtt{pointInTime} : 1995].$$

However, to express the Fields medal award, an assertion with an open specifier would be preferred, to account for other annotations (e.g. provenance):

$$\mathtt{awardReceived}(\mathtt{ArturAvila}, \mathtt{FieldsMedal})@\lfloor \mathtt{pointInTime} : 2014 \rfloor.$$

Given a classical, non-attributed, DL $\mathcal{L}$ (e.g. $\mathcal{ALC}$), its attributed version, $\mathcal{L}_{@+}$ (e.g. $\mathcal{ALC}_{@+}$), has essentially the same concept expressions as $\mathcal{L}$ except that concept and role names are associated with a specifier. For instance, since in $\mathcal{ALC}$ $A \sqcap B$ is a valid concept expression, we have that $A@S \sqcap B@S'$ is a valid concept expression in $\mathcal{ALC}_{@+}$, in which S, S' are specifiers as in Definition 7. Note that we can use an empty open specifier $\lfloor\rfloor$ if we do not want to constrain annotations.

▶ **Example 9.** The following are valid concept and role expressions in $\mathcal{ALC}_{@+}$:

- Brazilians: $\mathtt{Brazilian}@\lfloor\rfloor$
- Doctors of Philosophy whose only annotation is start time 1995: $\mathtt{PhD}@[\mathtt{startTime} : 1995]$
- The "membership" relation with some starting time $\mathtt{memberOf}@\lfloor \mathtt{startTime} : + \rfloor$

In Example 9, we deliberately omitted specifiers of the form $\mathtt{X.c}$ as they are more intricate. We will clarify their meaning later when we discuss axioms in attributed DLs. Regarding concept and role expressions, given a standard DL $\mathcal{L}$, its attributed extension $\mathcal{L}_{@+}$ allows the following axioms:

- $\mathcal{L}_{@+}$ concept assertions are $A(a)@S$;
- $\mathcal{L}_{@+}$ role assertions are $r(a, b)@S$; and
- $\mathcal{L}_{@+}$ concept inclusions are $X_1 : S_1, \ldots, X_n : S_n(C \sqsubseteq D)$;

where $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, $a, b \in \mathsf{N_I}$, C and D are $\mathcal{L}_{@+}$ concept expressions, S is a specifier that is not a set variable, $X_1, \ldots, X_n \in \mathsf{N_V}$ and $S_1, \ldots, S_n$ are specifiers.

▶ **Example 10.** If we consider the DL $\mathcal{ALC}$, and its attributed version $\mathcal{ALC}_{@+}$ we can express the following statements:

- $\mathtt{PhD}(\mathtt{ArturAvila})@\lfloor \mathtt{reference} : + \rfloor$
- $\mathtt{memberOf}(\mathtt{ArturAvila}, \mathtt{USNAS})@[\mathtt{startTime} : 2019]$
- $\mathtt{X} : \lfloor \mathtt{reference} : + \rfloor(\mathtt{PhD}@[\mathtt{reference} : \mathtt{X.reference}] \sqsubseteq$
  $\exists \mathtt{educatedAt}@\lfloor \mathtt{reference} : \mathtt{X.reference} \rfloor.\mathtt{University})$

The semantics is also similar to standard DLs and it is given by interpretations, but modified to include annotation sets. Hence, terms are interpreted as follows:

- $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \mathcal{P}_f(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$;
- $r^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \times \mathcal{P}_f(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$;
- $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$;

where $\mathcal{P}_f(\mathtt{S})$ is the set of all finite subsets of a set $\mathtt{S}$. Example 11 clarifies the meaning of interpreting terms alone (that is, without considering complex concepts nor specifiers).

▶ **Example 11.**

$$\begin{aligned}
&\texttt{ArturAvila}^{\mathcal{J}} = \texttt{ArturAvila}\\
&\texttt{CarlosChagas}^{\mathcal{J}} = \texttt{CarlosChagas}\\
&\texttt{PhD}^{\mathcal{J}} = \{(\texttt{ArturAvila}, \{(\texttt{pointInTime}, 2001)\}), (\texttt{CarlosChagas}, \emptyset)\}\\
&\texttt{memberOf}^{\mathcal{J}} = \{((\texttt{ArturAvila}, \texttt{USNAS}), \{(\texttt{startTime}, 2019),\\
&\qquad\qquad\qquad\qquad\qquad\qquad (\texttt{subjectRole}, \texttt{ForeignAssociate})\})\}\\
&\texttt{Brazilian}^{\mathcal{J}} = \{(\texttt{ArturAvila}, \{(\texttt{reference}, \texttt{cv})\}), (\texttt{CarlosChagas}, \emptyset)\}\\
&\texttt{French}^{\mathcal{J}} = \{(\texttt{ArturAvila}, \{(\texttt{reference}, \texttt{cv})\})\}
\end{aligned}$$

The semantics of specifiers and other expressions containing free variables is defined using a *variable assignment*. A variable assignment is a function, $\mathcal{Z} : \mathsf{N_V} \mapsto \mathcal{P}_f(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}})$

$$\begin{aligned}
X^{\mathcal{I},\mathcal{Z}} &:= \{\mathcal{Z}(X)\};\\
[a : b]^{\mathcal{I},\mathcal{Z}} &:= \{\{(a^{\mathcal{I}}, b^{\mathcal{I}})\}\};\\
[a : X.b]^{\mathcal{I},\mathcal{Z}} &:= \{\{(a^{\mathcal{I}}, \delta^{\mathcal{I}}) \mid \exists \delta \in \Delta^{\mathcal{I}} \, : \, (b^{\mathcal{I}}, \delta) \in \mathcal{Z}(X)\}\};\\
[a : +]^{\mathcal{I},\mathcal{Z}} &:= \{\{(a^{\mathcal{I}}, \delta_1), \ldots, (a^{\mathcal{I}}, \delta_\ell)\} \mid \ell \leq 1 \text{ and } \delta_i \in \Delta^{\mathcal{I}}\}\};\\
[a_1 : v_1, \ldots, a_n : v_n]^{\mathcal{I},\mathcal{Z}} &:= \{\cup_{i=1}^{n} \psi_i \mid \psi_i \in [a_i : v_i]^{\mathcal{I},\mathcal{Z}}, 1 \leq i \leq n\};\\
\lfloor a_1 : v_1, \ldots, a_n : v_n \rfloor^{\mathcal{I},\mathcal{Z}} &:= \{\psi \in \mathcal{P}_f(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \mid \psi \supseteq \phi \text{ for some } \phi \in [a_i : v_i]^{\mathcal{I},\mathcal{Z}}\}.
\end{aligned}$$

Given the semantics of terms and specifiers, concept names and role names with non-empty specifiers are interpreted as follows:

$$\begin{aligned}
A@S^{\mathcal{I},\mathcal{Z}} &:= \{\delta \in \Delta^{\mathcal{I}} \mid (\delta, \psi) \in A^{\mathcal{I}} \text{ for some } \psi \in S^{\mathcal{I},\mathcal{Z}}\};\\
r@S^{\mathcal{I},\mathcal{Z}} &:= \{(\delta_1, \delta_2) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (\delta_1, \delta_2, \psi) \in \texttt{r}^{\mathcal{I}} \text{ for some } \psi \in S^{\mathcal{I},\mathcal{Z}}\}.
\end{aligned}$$

The concept constructors in an attributed DL are inherited from its standard counterpart, with the semantics extended to accommodate the annotations. Intuitively, a specifier will reduce the extension of a concept or role, that is, if $\texttt{t} \in \mathsf{N_C} \cup \mathsf{N_R}$ and S is a specifier, then, for any interpretation $\mathcal{I}$: $\texttt{t@S}^{\mathcal{I}} \subseteq \texttt{t@}\lfloor\rfloor^{\mathcal{I}}$.

▶ **Example 12.** Consider the interpretation $\mathcal{J}$ from Example 11. Below we list the interpretation of the concept and role expressions from Example 9 in $\mathcal{J}$ using $\mathcal{Z}(X) = \{(\texttt{startTime} : \mathsf{2000}, \texttt{startTime} : \mathsf{2019}, \texttt{reference} : \texttt{cv})\}$.

$$\begin{aligned}
\texttt{Brazilian@}\lfloor\rfloor^{\mathcal{J},\mathcal{Z}} &= \{\texttt{ArturAvila}\}\\
\texttt{PhD@}[\texttt{startTime} : \mathsf{1995}]^{\mathcal{J},\mathcal{Z}} &= \emptyset\\
\texttt{memberOf@}\lfloor\texttt{startTime} : +\rfloor^{\mathcal{J},\mathcal{Z}} &= \{(\texttt{ArturAvila}, \mathsf{USNAS})\}\\
\texttt{French@}[\texttt{reference} : \texttt{X.reference}]^{\mathcal{J},\mathcal{Z}} &= \{\texttt{ArturAvila}\}
\end{aligned}$$

The main drawback of attributed DLs is that even $\mathcal{ALC}_{@+}$ is undecidable, preventing the development of sound and complete reasoning mechanisms. Krötzsch et al. [68] proved that $\mathcal{ALCH}_{@}$ is still decidable, although its satisfiability problem lies in 2ExpTime. Bourgaux and Ozaki [21] proved that reasoning in a more restricted DL, called DL-Lite$^{\mathcal{R}}_{@}$, is in PSpace.

2.1.2 Description Logic Reasoners

Reasoning in DLs refers to obtaining implicit knowledge from explicit data. The formal semantics discussed before allows us to derive new information which respects the constraints defined in a DL ontology. For example, if we specify that "PhDs are humans" by adding to an ontology the axiom $\mathtt{PhD} \sqsubseteq \mathtt{Human}$, then from the assertion $\mathtt{PhD}(\mathtt{ArturAvila})$ we can conclude $\mathtt{Human}(\mathtt{ArturAvila})$, that is, that since Artur Ávila has a PhD title, he is a human.

Reasoning with DL ontologies is a mature and active field of study. There are already many well-known reasoners such as ELK [59], HermiT [100], Pellet [101], FaCT++ [107], Konclude [102] and RDFox [82] used in academia and in the industry. Some of these reasoners are benchmarked in standard reasoning tasks so that both users and developers can have an overview of the empirical state-of-art [89, 97]. There are also many prototypical implementations, which are tailored to specific extensions of DLs which still do not have a widespread adoption.

The DL reasoners differ in a number of factors, for example, ELK can only handle OWL 2 EL ontologies, which are restricted to the DL $\mathcal{EL}$++. Besides differing according to their target language (e.g. $\mathcal{EL}$++), they may also be based on different reasoning techniques (e.g. tableaux, hyper-tableaux, consequence-based reasoning) [7]. Other reasoners specialise in performance, for instance, by giving approximate results such as the case of TrOWL [105].

2.2 Datalog

Datalog is a query language derived from the logical programming language Prolog [30]. It was actively studied in the late 80s an early 90s, in the database community as an implementation of recursive queries. More recently, its interest has been revived for applications for querying graph-like data structures in ontology-based data access [14] and query answering [3, 27]. Besides the ability to express recursive queries, Datalog has polynomial complexity for many reasoning tasks which relate to queries and motivate its use today for querying databases, ontologies, and KGs. There are many extensions of Datalog [26, 27, 44, 94] but here we will focus on "pure" Datalog and its relevant modifications for handling KGs and ontologies.

2.2.1 Syntax

We begin the formalisation of Datalog with the signature. We assume three pairwise disjoint sets of symbols: a set of variables $\mathbf{V}$, a set of constants $\mathbf{C}$, and a set of predicates $\mathbf{P}$. Each predicate is represented as P/n where P is its name and n its arity. A *term* in Datalog is either a variable or a constant. An *atom* is an expression of the form $P(x_1, \ldots, x_n)$ where $P/n \in \mathbf{P}$ and $x_i \in \mathbf{C} \cup \mathbf{V}$ for $1 \leq i \leq n$. Using atoms we can write *rules*. A rule r is an expression of the form $H \leftarrow B_1, \ldots, B_m$ where H is the *head* atom and $B_1, \ldots, B_n$ are *body* atoms or *subgoals* [2, 108]. In Datalog, these rules must be *safe*, that is, each variable that appears in the head of the rule appears in its body. A finite collection of safe rules constitutes a *Datalog program*.

▶ **Example 13.** The following rule says that if someone teaches at a university then he/she is a lecturer:

$$\mathtt{Lecturer}(\mathtt{x}) \leftarrow \mathtt{TeachesAt}(\mathtt{x}, \mathtt{y}), \mathtt{University}(\mathtt{y}).$$

The rule above is safe because the variable x appears both in the head and in the body. As an example of an unsafe rule, consider a rule which says that a worker is someone who works at some place:

$$\mathtt{WorksAt}(\mathtt{x}, \mathtt{y}) \leftarrow \mathtt{Worker}(\mathtt{x}).$$

If an atom has no variables we call it a *fact*, for instance, we can state that Artur Ávila is a PhD with `PhD(ArturAvila)`. A Datalog *database instance* is a finite collection of facts. The safety constraint for rules and the requirement that every database contains only facts ensures that the logical derivations in a Datalog system are finite. Meaning that, in a finite number of applications of the rules over the database instance, one can derive every possible fact that is entailed.

▶ **Example 14.** In this example, we want to use Datalog to know if Artur Ávila has an Erdős number. The following rules are useful for knowing if someone has an Erdős number

$$\begin{aligned} \texttt{HasErdosNumber(x)} &\leftarrow \texttt{HasErdosNumber(y)}, \texttt{CoAuthor(x,y)}. \\ \texttt{CoAuthor(x,y)} &\leftarrow \texttt{CoAuthor(y,x)}. \end{aligned}$$

And then, we can state some basic facts

$$\begin{aligned} &\texttt{HasErdosNumber(PaulErdos)}. \\ &\texttt{CoAuthor(ArturAvila,BarrySimon)}. \\ &\texttt{CoAuthor(VilmosTotik,BarrySimon)}. \\ &\texttt{CoAuthor(VilmosTotik,PaulErdos)}. \\ &\texttt{CoAuthor(ArturAvila,WellingtonDeMelo)}. \end{aligned}$$

Example 14 shows one interesting feature of Datalog: recursion. In the first rule, for example, the predicate `HasErdosNumber/1` appears both in the head and in the body of the same rule. In pure Datalog, recursion does not complicate the semantics much, and in many cases a recursive program can be rewritten without it [2]. However, some complications occur in Datalog extensions (as some mentioned later in this section).

2.2.2 Semantics

Semantics in Datalog can be defined in many ways: with a model-theoretic approach, via fixpoint semantics, or based on proofs. Here, we introduce the first of these, as it relates more closely to the semantics of Description Logics introduced in Section 2.1. To understand the semantics, we must first map each rule to a sentence in FOL. Given a Datalog rule $H \leftarrow B_1, \ldots, B_n$, its corresponding FOL sentence is:

$$\forall x_1, \ldots, x_m \left((B_1(\mathbf{u}_1) \wedge \cdots \wedge B_n(\mathbf{u}_n)) \rightarrow H(\mathbf{u}_h) \right)$$

where $\mathbf{u}_h$ is the sequence of variables that appear in H, $\mathbf{u}_i$ is the sequence of variables that appear in B_i, and $x_1, \ldots, x_m$ are the variables occurring in the rule. Moreover, each fact corresponds to an atomic formula in FOL.

Then, we will need to consider only *Herbrand interpretations.* A Herbrand interpretation maps each constant to its own name and each n-ary predicate to a subset of $\mathbf{C}^n$. In this way, a Herbrand interpretation can be identified with the set of facts that it satisfies. An interpretation $\mathcal{I}$ satisfies a database instance I, if $I \subseteq \mathcal{I}$. An interpretation $\mathcal{I}$ satisfies a rule $H \leftarrow B_1, \ldots, B_n$ if for every variable substitution by constants θ, $\{B_1\theta, \ldots, B_n\theta\} \subseteq \mathcal{I}$ implies $H\theta \in \mathcal{I}$. Finally, an interpretation satisfies a program P if it satisfies every rule. We denote the satisfaction relation with an infix operator $\models$, where $\mathcal{I} \models X$ indicates that the interpretation $\mathcal{I}$ satisfies X, where X can be a fact, a database instance, a rule, a program, or a set combining those.

▶ **Example 15.** Consider the two following Herbrand interpretations:

$$\begin{aligned}\mathcal{I}_1 = \{\,&\texttt{PhD}(\texttt{ArturAvila}),\\ &\texttt{Scientist}(\texttt{ArturAvila}),\\ &\texttt{educatedAt}(\texttt{ArturAvila}, \texttt{UFRJ}),\\ &\texttt{participantIn}(\texttt{ArturAvila}, \texttt{MathOlympiad})\,\}\end{aligned}$$

$$\mathcal{I}_2 = (\mathcal{I}_1 \cup \{\texttt{University}(\texttt{UFRJ})\}) \setminus \{\texttt{Scientist}(\texttt{ArturAvila})\}.$$

The interpretation $\mathcal{I}_1$ satisfies the fact $\texttt{Scientist}(\texttt{ArturAvila})$, but it does not satisfy $\texttt{University}(\texttt{UFRJ})$, which is only satisfied by $\mathcal{I}_2$. Moreover, the rule

$$\texttt{University}(\texttt{y}) \leftarrow \texttt{educatedAt}(\texttt{x}, \texttt{y}) \wedge \texttt{PhD}(\texttt{x}),$$

is only satisfied by $\mathcal{I}_2$ as $\texttt{University}(\texttt{UFRJ}) \notin \mathcal{I}_1$. The rule $\texttt{PhD}(\texttt{x}) \leftarrow \texttt{Scientist}(\texttt{x})$ is satisfied by both interpretations. $\mathcal{I}_1$ satisfies both $\texttt{PhD}(\texttt{ArturAvila})$ and $\texttt{Scientist}(\texttt{ArturAvila})$, and $\mathcal{I}_2$ satisfies the rule vacuously because $\texttt{Scientist}(\texttt{ArturAvila}) \notin \mathcal{I}_2$.

Example 15 shows that we can have multiple interpretations for a given Datalog program and database instance. We can associate each pair of Datalog program and database instance (P, I) with the set of all Herbrand interpretations that satisfy them $HI(P, I)$. Interestingly, we can take the intersection of these models (even if there are infinitely many) and obtain the minimal Herbrand interpretation for (P, I) [30]. This allows to assign the following meaning to the consequences ($cons$) of a program P with database instance I in terms of all possible Herbrand models as follows

$$cons(P, I) = \bigcap_{\mathcal{I} \in HI(P,I)} \mathcal{I}$$

The semantics as presented here confer Datalog an interesting property: the unique name assumption (UNA). The name of each constant identifies it uniquely. In DLs, the convention is that there is no UNA, which can cause confusion regarding the possible interpretations of an ontology. Arguably, UNA matches the intended meaning in database applications [2].

2.2.3 Datalog Extensions

Pure Datalog, as we have presented, often lacks expressive power to represent more complex rules (or queries in a database point of view). To cover these gaps, many extensions of Datalog have been developed [2, 26, 44].

Many Datalog extensions allow rule atoms to appear negated, which gives the possibility to express interesting relationships. However, these Datalog variants have to deal with two issues that arise from negation. The first concerns groundings, replacements of variables by constants, on infinite domains. The second regards the actual semantics of Datalog programs and happens because the intersection of Herbrand interpretations ceases to characterise the consequences of Datalog with negation in a meaningful way. There are many different approaches to circumvent both issues [2], sometimes using semantics that differ considerably from the one presented here for pure Datalog. In other cases, the issues are solved by employing a similar semantics but placing syntactic restrictions on the usage of negation [2, 30]. A complete categorisation of the different possibilities (and thus, of Datalog variants) is beyond the scope of this work, and we refer the reader to classical references on Datalog for the details [2, 108].

Many Datalog implementations also include built-in predicates which represent, for example, arithmetic operations (e.g. sum, subtraction) and numeric comparisons (such as less than, $<$). These elements may also induce infinite groundings, so their syntactic use is usually constrained in a similar way as it happens with negation.

There are also Datalog extensions which focus on temporal reasoning [23,34,51]. Nowadays, temporal extensions of Datalog, and associated reasoners, often address the *Stream Reasoning* paradigm [38, 113]. In Stream Reasoning, the goal is to perform inferences of a (usually) rapid stream of data. In [11], for example, the authors employ DatalogMTL to formalise reasoning tasks that must account for time. Such tasks involve calculating or counting values over a period of time, for instance, to compute the revenue per year of a company [11].

2.2.4 Datalog and Description Logics

Datalog and Description Logics are not completely disjoint fields. There have been many studies on how these two formalisms relate [43, 67, 94, 95], in particular because Datalog materialisation, a form of inferring new facts, is very efficient and can be applied to reasoning problems involving ontologies [27, 28] and KGs [110].

Datalog$^{\pm}$, for instance, is a family of Datalog variants designed to capture the expressivity of Description Logics in the *DL-Lite* family. *DL-Lite* is composed mostly of lightweight DLs tailored for querying large collections of assertional data, while the concept inclusions are kept relatively simple. In fact, Datalog$^{\pm}$ is strictly more expressive than *DL-Lite* [27]. Krötzsch, Rudolph and Schmitt [67] investigated fragments of Description Logics (such as $\mathcal{ALC}$ and $\mathcal{SROIQ}$) that can be captured in Datalog, while Rosati [94] proposes a framework for integrating DLs with Datalog$^{\neg\vee}$, that is, Datalog extended with negation in the body and boolean disjunction in the head.

The *Semantic Web Rule Language (SWRL)* [57] closely relates DLs and Datalog. More specifically, this language is based both on OWL DL (a fragment of the first OWL proposal) and RuleML (which is essentially a fragment of Datalog. The resulting language is very expressive, so much that reasoning with SWRL with very expressive DLs such as $\mathcal{SROIQ}$ is undecidable. There are, however, subsets of SWRL that retain decidability, even when combined with $\mathcal{SROIQ}$, the so called *DL rules* [66].

2.2.5 Reasoning with Datalog

There are numerous implementations of Datalog embedded in database management systems and inference engines (e.g. RDFox [82], Apache JENA[1]) or available as libraries (e.g. pyDatalog[2], Datalog in Racket[3]). The implementations vary regarding the extensions implemented, the semantics adopted, and the actual syntax of the Datalog rules. Next, we discuss some of the most recent reasoning systems to perform inferences in Datalog rules and KGs.

RDFox [82] is a column store which employs a parallel materialisation method for Datalog reasoning. Materialisation is a popular strategy to save time in reasoning systems. Every entailment computed is stored instead of being discarded after each query, saving costs in subsequent calls. RDFox is designed to manage large volumes of RDF data and provide reasoning services using Datalog extensions tailored for this use case.

[1] `https://jena.apache.org/`
[2] `https://sites.google.com/site/pydatalog/`
[3] `https://docs.racket-lang.org/datalog/`

Another notable Datalog system built with KGs as its primary use case is Vadalog [12]. It focusses on Warded Datalog$^{\pm}$, a variant of Datalog with the three highly desirable properties: (1) ability to express recursive patterns, (2) enough expressivity to capture queries over KGs in fragments of *DL-Lite* (more precisely, SPARQL queries with OWL 2 QL entailment), and (3) polynomial data complexity.

Carral et al. [29] devised a rule engine with reasoning services based on Datalog materialisation. The engine is designed for an extension of Datalog similar to Datalog$^{\pm}$ and targets KGs as its main use-cases. One of the aspects that distinguishes this tool from other approaches mentioned here is the ability to easily integrate with data sources in different formats such as OWL ontologies, SPARQL endpoints, and RDF stores.

As we mentioned in the previous sections, KGs often include uncertain information and time-sensitive data. Chekol et al. [32] devised a framework combining Markov Logic Networks (MLNs) [92] and Datalog extended with inequalities to facilitate time-aware maintenance of KGs. MLNs are a Statistical Relational Learning approach which combines First-Order Logic and probabilities to represent dependencies between events, while they still remain uncertain. In a similar vein, Bellomarini et al. [10] extend the Vadalog system to express probabilistic rules. Recently, Wang et al. [114] developed a DatalogMTL reasoner by combining traditional reasoning methods for Datalog (materialisation) and automata-based methods.

Leone et al. [73] adapted the DLV2 answer set programming system, which already had Datalog capabilities, to handle large KGs (in the paper, the authors focus on DBPedia). These improvements concern mostly scalability, reducing processing time and memory consumption.

2.3 SHACL

The Shape Constraint Language (SHACL) is a W3C recommendation [62] whose purpose is to validate RDF graphs. Each SHACL constraint is called a *shape* which specifies to which nodes it applies and what conditions such nodes must satisfy. Given a KG in RDF, one can use different tools to check that the KG complies with a set of shapes. KGs generally lack a schema when compared with a database. Therefore, constraints are crucial for quality assessment and maintenance of large KGs [88]. SHACL and similar languages, such as ShEx [17], are tailored for the particular task of constraint validation, whereas OWL is tailored for modelling domains of knowledge and for reasoning. However, as we discuss later in this section, one can define reasoning problems and perform inference using SHACL. Next, we illustrate the capabilities of SHACL with an example.

▶ **Example 16.** The RDF graph below, written in terse triple notation, describes a single shape using SHACL (we omit the prefixes' declarations here).

```
:HumanShape rdf:type sh:NodeShape;
    sh:targetClass  :Human; # applies to all humans
    sh:property [
        sh:path :birthDate ; #  predicate for the date of birth
        sh:maxCount 1 ; # must have at most 1
        sh:minCount 1 ; # must have at least 1
        sh:datatype xsd:date ; # must be a date
    ];
    sh:property [
        sh:path :citizenship ; # citizenship predicate
        sh:minCount 1 ; # must have at least 1
        sh:node [ a sh:NodeShape;
                  sh:class :Country];  # must be a country
    ] .
```

The first defines the name of the shape (`:HumanShape`). The shape *targets* the nodes whose type (`rdf:type`) is the node `:Human` (Line 2). It also specifies that each such node must have exactly one birth date and it must be a valid date (lines 3 to 8). Lines 9 to 14 state that the target nodes must (1) be related via `:citizenship` to at least one node and (2) be related by `:citizenship` only to objects of type `:Country` (this specification is done by using an anonymous shape in lines 12 and 13).

Programs called *SHACL processors* check whether an RDF graph complies with a set of shapes. These programs generate a validation report for a given graph and set of shapes, indicating which nodes and constraints are violated, if any. Example 17 depicts an example of validation report.

▶ **Example 17.** A SPARQL processor receiving shapes from Example 16 and the RDF graph from Example 1 could produce the validation report below[4].

```
 1  [
 2      a sh:ValidationResult;
 3      sh:resultSeverity sh:Violation;
 4      sh:sourceConstraintComponent sh:MinCountConstraintComponent;
 5      sh:sourceShape _:n51;
 6      sh:focusNode <http://www.example.org/ArturAvila>;
 7      sh:resultPath <http://www.example.org/birthDate>;
 8      sh:resultMessage "Less than 1 values";
 9  ] .
10  [
11      a sh:ValidationResult;
12      sh:resultSeverity sh:Violation;
13      sh:sourceConstraintComponent sh:NodeConstraintComponent;
14      sh:sourceShape _:n52;
15      sh:focusNode <http://www.example.org/ArturAvila>;
16      sh:value <http://www.example.org/Brazil>;
17      sh:resultPath <http://www.example.org/citizenship>;
18      sh:resultMessage "Value does not have shape Blank node _:n53";
19  ] .
```

The report specifies two violations: one says that the RDF graph does not include a birth date for `:ArturAvila` (lines 1 to 9) and the other (lines 10 to 19) indicates that the graph does not guarantee that `:Brazil` is a country.

2.3.1 DL-like Syntax for SHACL

Since we are interested in SHACL from the point of view of reasoning, we will look at the SHACL formalisation by Jakubowski and Van den Bussche [16] which adapts the one by Corman, Reutter and Savković [35]. This alternative syntax is much closer to that presented for Description Logics in Section 2.1, and it will aid us in describing SHACL's semantics and its connections to the DLs.

First, we need three pairwise disjoint sets of node names ($\mathsf{N_N}$), shape names ($\mathsf{N_S}$) and property names ($\mathsf{N_P}$). A *signature* Σ is be subset of $\mathsf{N_N} \cup \mathsf{N_S} \cup \mathsf{N_P}$. Using these terms, one can define path expressions which are either property names or built as in Table 3 (these path expressions essentially come from SPARQL [35, 62]). Finally, we can consider shape expressions: terms in $\mathsf{N_S}$ or expressions using the constructors in Table 4.

[4] The validation report was generated at `https://shacl.org/playground/`.

Table 3 Syntax and semantics of abstract SHACL path expressions [16].

Name	Syntax	Semantics
Inverse	$\mathtt{p}^-$	$\{(a,b) \mid (b,a) \in \mathtt{p}^{\mathcal{I}}\}$
Union	$\mathtt{E}_1 \cup \mathtt{E}_2$	${\mathtt{E}_1}^{\mathcal{I}} \cup {\mathtt{E}_2}^{\mathcal{I}}$
Composition	$\mathtt{E}_1 \circ \mathtt{E}_2$	$\{(a,b) \mid \exists c.(a,c) \in {\mathtt{E}_1}^{\mathcal{I}} \wedge (c,b) \in {\mathtt{E}_2}^{\mathcal{I}}\}$
Reflexive closure	$\mathtt{E}?$	$\mathtt{E}^{\mathcal{I}} \cup \{(a,a) \mid a \in \Delta^{\mathcal{I}}\}$
Reflexive-transitive Closure	$\mathtt{E}^*$	The $\subseteq$-minimal S with $\mathtt{E}?^{\mathcal{I}} \subseteq S$ and $(a,b),(b,c) \in S$ implies $(a,c) \in S$

Table 4 Syntax and semantics of abstract SHACL shape expressions [16]. $\mathcal{I}$ is an interpretation defined over the signature Σ, $\mathtt{E}$ is a path expression, $\mathtt{p} \in \mathsf{N_P}$, $\mathtt{s} \in \mathsf{N_S}$, $\mathtt{c} \in \mathsf{N_N}$ $n \in \mathbb{N}$, $Q \subseteq \mathsf{N_P}$ and ϕ_1, ϕ_2 are shape expressions. Also, $R(a)$ denotes the set $\{b \mid (a,b) \in R\}$, where R is a binary relation.

Name	Syntax	Semantics
Top	$\top$	$\Delta^{\mathcal{I}}$
Atomic Shape	$\mathtt{s}$	$\mathtt{s}^{\mathcal{I}}$
Nominal	$\{\mathtt{c}\}$	$\mathtt{c}^{\mathcal{I}}$
Conjunction	$\phi_1 \wedge \phi_2$	$\phi_1^{\mathcal{I}} \cap \phi_2^{\mathcal{I}}$
Disjunction	$\phi_1 \vee \phi_2$	$\phi_1^{\mathcal{I}} \cup \phi_2^{\mathcal{I}}$
Complement	$\neg\phi_1$	$\Delta^{\mathcal{I}} \setminus \phi_1^{\mathcal{I}}$
Minimum cardinality	$\geq n\mathtt{E}.\phi_1$	$\{a \in \Delta^{\mathcal{I}} \mid (\#(\phi_1^{\mathcal{I}} \cap \mathtt{E}^{\mathcal{I}}(a))) \geq n\}$
Equality	$eq(\mathtt{p},\mathtt{E})$	$\{a \in \Delta^{\mathcal{I}} \mid \mathtt{p}^{\mathcal{I}}(a) = \mathtt{E}^{\mathcal{I}}(a)\}$
Disjointness	$disj(\mathtt{p},\mathtt{E})$	$\{a \in \Delta^{\mathcal{I}} \mid \mathtt{p}^{\mathcal{I}}(a) \cap \mathtt{E}^{\mathcal{I}}(a) = \emptyset\}$
Closed shape	$closed(Q)$	$\{a \in \Delta^{\mathcal{I}} \mid \mathtt{p}^{\mathcal{I}}(a) = \emptyset$ for every $\mathtt{p} \in \Sigma \setminus Q\}$

We can regard a set of shapes, as the one in Example 16, as a set of statements relating shape expressions. These statements can be of two types *shape definitions* and *shape constraints*. Shape definitions are expressions of the form $\mathtt{s} \equiv \phi_{\mathtt{s}}$ with $\mathtt{s} \in \mathsf{N_S}$ and $\phi_{\mathtt{s}}$ a shape expression. Shape constraints are statements of the form $\phi_1 \sqsubseteq \phi_2$. A set of shapes can be represented as a pair of sets (D,T) called *shape schema* [6], in which D is a set of shape definitions and T a set of shape constraints. The shape schema must satisfy some syntactical requirements to ensure that it represents a valid SHACL document, we refer the reader to [6, 16] for more details.

▶ **Example 18.** This is a translation of the SHACL constraints over the citizenship relation in Example 16.

$$\neg(\geq 1.\mathtt{citizenship}.\neg(\{\mathtt{country}\})) \wedge (\geq 1.\mathtt{citizenship}.\top)$$

The first conjunct states that the target can only be related via `citizenship` to countries `Country`. The second specifies that there must be at least one object to which the target relates to via citizenship.

2.3.2 Semantics

When considering SHACL semantics, we have to make distinctions similar to the Datalog case. However, in this case the distinction concerns recursion. A shape is recursive if it refers to itself directly or indirectly, otherwise, it is non-recursive. The W3C recommendation [62] describes the language precisely enough to specify the meaning of validation against *non-recursive* shapes, however, the semantics of validation containing recursive shapes is open. This gap in SHACL's semantics sparked a number of distinct approaches attempting to clarify the validation of recursive SHACL [6,15,35].

For the purposes of this work, it will be enough to consider the semantics for non-recursive SHACL alone. Here, we will also focus on the formalisation proposed by Bogaerts, Jakubowski and Van den Bussche [16]. Thus, we will present the semantics using interpretations, which is also similar to the ones seen in Section 2.1. However, before we proceed, we remark two key differences between standard DL (and OWL) reasoning and SHACL reasoning. The first is the presence of the UNA in SHACL [62]. This means that every term in $\mathsf{N_N} \cup \mathsf{N_P}$ must be mapped to a different element of the domain in SHACL [16], similarly to the case of many Datalog semantics [2]. Furthermore, even blank nodes in a RDF graph (i.e. those who do not have an IRI) must be mapped into distinct elements of the domain [35].

The second main difference is that relations in SHACL are closed, meaning that if the graph does not have a triple (subject, predicate, object), then it is assumed to be false. More precisely, each triple can be seen as a fact (similar to the meaning in Datalog) or an assertion (as in DL ABoxes). For example, the triple (Artur Ávila, memberOf, USNAS) represents the fact that Artur Ávila is member of the United States' National Academy of Science (USNAS). If such triple would not be present in the Wikidata's KG, then SHACL would assume that Artur Ávila is not a member of the USNAS when validating the KG against a set of shapes. This is not the case in standard OWL and DL reasoning which follows the Open World Assumption (OWA). With the OWA, unless there is a statement or its negation (either asserted or inferred), the status of any triple is assumed to be unknown. As Bogaerts, Jakubowski and Van den Bussche [16] remark, this does not mean that SHACL presupposes the Closed World Assumption (CWA) completely. Triples about unknown entities could still be true or false. For instance, if the KG did not include any reference to Artur Ávila, then the status of the triple (Artur Ávila, memberOf, USNAS) would be "unknown", thus there could be an interpretation for that KG in which the triple is true, while another interpretation would consider it false.

Now, for the actual semantics, we consider that a KG G is a finite set of triples (subject, predicate, object), or equivalently, facts of the form $\mathit{predicate}(\mathit{subject}, \mathit{object})$. Given a KG G, we write N_G to denote the set of nodes (entities that appear as subject or object) that occur in G, and $\mathtt{p}^G$ to represent the set $\{(\mathtt{a}, \mathtt{b}) \mid (\mathtt{a}, \mathtt{p}, \mathtt{b}) \in G\}$. One can build an interpretation $\mathcal{I}(G) = (\Delta^{\mathcal{I}(G)}, \cdot^{\mathcal{I}(G)})$ over $\mathsf{N_N} \cup \mathsf{N_P}$ based on the KG G that preserves the intended meaning of the triples and shapes as follows [16]

- $\Delta^{\mathcal{I}(G)} = \mathsf{N_N}$;
- $\mathtt{c}^{\mathcal{I}(G)} = \mathtt{c}$, for all $\mathtt{c} \in \mathsf{N_N}$; and
- $\mathtt{p}^{\mathcal{I}(G)} = \mathtt{p}^G$, for all $\mathtt{p} \in \mathsf{N_P}$.

The semantics for complex path expressions follows the rules stated in Table 3. Shapes are mapped to subsets of $\Delta^{\mathcal{I}(G)}$ using the rules described in Table 4.

2.3.3 Reasoning with SHACL

The main reasoning task in SHACL is validation of constraints, that is, given an RDF graph which acts as an interpretation, one must decide whether it satisfies a set of constraints defined by shapes. As mentioned earlier, there are programs called SHACL processors which are either standalone programs [47], or embedded in larger systems for managing KGs, such as RDFox [82]. We list the most prominent of these reasoning tasks below (see also [5,72,87,88]).

Shape containement: given two shapes ϕ_1 and ϕ_2, decide whether every target node that complies with ϕ_1 will also comply with ϕ_2. If so then we say that ϕ_1 is contained in ϕ_2.

SHACL satisfiability: given a set of shapes S, decide whether there exists G that complies with all shapes in S.

Explanation for Violations: Given a set of shapes S and a KG G, decide whether (B, A) with $B \subseteq G$ and $A \cap G = \emptyset$ is such that $(G \setminus B) \cup A$ complies with S.

Leinberger et al. [72] studies the decision problem of shape containment through DLs using the abstract formalisation of SHACL proposed by Corman, Reutter and Savković [35] that we presented here. More precisely, they map the problem of shape containment to concept subsumption, a well known decision problem in DLs [7]. The problem of shape containment was later generalised by Pareti et al. [88]. The authors studied the containment problem between sets of shapes and also proposed and investigated SHACL satisfiability and a more restricted version, constraint satisfiability. The tasks studied were restricted to non-recursive SHACL, but they are also being extended to consider recursive shapes [87].

While explanation finding and associated tasks are classified as abductive reasoning, we find it convenient to mention here the work due to Ahmetaj et al. [5]. The authors define and investigate the computational complexity of many reasoning problems related to explanations in SHACL, as the one we mentioned before. Their theoretical framework rely on a primarily deductive approach which is also underpinned by the DL abstraction of SHACL.

3 Inductive Reasoning

Inductive reasoning in KGs can be performed with different approaches. One of the main goals of this task is to “complete” the KGs. KG completion is the task of inferring new facts which are plausible based on patterns already present in a given KG. In this section, we discuss classical approaches for KG completion, namely, knowledge graph embeddings [20] (see [55] for some approaches using graph neural networks).

3.1 Knowledge Graph Embeddings

An embedding in Representation Learning (a subarea of Machine Learning) corresponds to a mapping from a collection of objects to a vector space model, often low-dimensional ones. The first approaches for knowledge graph embeddings mapped both objects (corresponding to nodes in the KG) and relations to vectors into a low-dimensional latent space encoding regularities in the KG [20]. These methods also have a score function that determines if a given triple is likely to be true or not, given the embeddings learned. Here we will assume, without loss of generality, that the higher the score, the more likely the triple is to be true. The score function is also the main component of the loss function that should be minimised during the training phase.

▶ **Example 19.** Consider a KG G that contains the entities $ArturAvila$, $CarlosChagas$, $NiedeGuidon$, $archelogist$, $biologist$ and $mathematician$; and the relation $occupation$. Ideally, an embedding method should learn vector representations (embeddings) for entity and relations that allows us to infer new and correct triples. For instance, suppose that G contains $(ArturAvila, occupation, mathematician)$ and $(CarlosChagas, occupation, biologist)$, but does not include $(NiedeGuidon, occupation, archeologist)$. If the KG also includes $NiedeGuidon$ in other triples, then we would expect that we could derive the missing triple by looking at the embeddings for $NiedeGuidon$, $occupation$ and $archeologist$. For instance, if we use TransE [20] and it generates the embeddings $(0.5, 0.2)$ for $NiedeGuidon$, $(0.25, 0.5)$

for ($archeologist$), and $(-0.3, -0.2)$ for the relation *occupation*, then we could expect to infer the triple ($NiedeGuidon, occupation, archeologist$), as TransE gives high scores for a triple (s, p, o) if $||(\mathbf{s} + \mathbf{p}) - \mathbf{o}||_\ell \approx 0$, where $\mathbf{s}$, $\mathbf{p}$ and $\mathbf{o}$ are, respectively, the embeddings learned for s, p and o, and $||.||_\ell$ denotes the ℓ-norm.

Even today, this is the standard approach when designing KG embedding models, despite the variety of strategies: some rely on geometric-based mappings [20,103], others on expressing the graph as products of tensors [106], and more recent methods rely on neural networks [40].

This representation is shown to be not suitable for encoding rules, in particular, concept and role inclusions present in ontologies [53, 58, 118]. For example, we could have a concept inclusion that states that every mathematician is a scientist, but traditional embedding methods would not be able to use this information when learning embeddings, nor would guarantee that such constraint would be respected. They also do not take into account the time dimension, which may be central in some KGs that contain inconsistencies if we disregard the temporal annotations. Since there is already a fairly recent survey on KG embedding methods in general due to Dai et al. [36], we will focus on the approaches which propose embedding models which focus on compliance with ontological constraints or on representing the time dimension.

3.1.1 KG Embeddings and Ontological Constraints

There are different approaches to incorporate ontological constraints in KG embeddings. The goal is to improve the accuracy of these methods in KG completion tasks and also reduce the amount of data required to achieve a reasonable performance during training, particularly when the KG has sparse relations [46]. Moreover, compliance with ontological constraints might also lead to models that capture the semantics of the relations and entities included.

Fatemi, Ravanbakhsh and Poole [46] modify the SimplE embedding model so that it complies with additional constraints which express concept and role inclusions by enforcing inequality constraints on the score function of affected triples. More precisely, to express the role inclusion $r \sqsubseteq s$, the model enforces that the score function f is such that $\mathrm{f}(x, r, y) \leq \mathrm{f}(x, s, y)$ while also requiring every coordinate of every embedding to be non-negative. The strategy for concept inclusions is analogous, but involves transforming the pertinence relation to each class (i.e. the relation `rdf:type`) into a new relation. The resulting variant, named Simple$^+$ performs similarly to SimplE, and with faster convergence when taxonomic information is available. It also retains the same theoretical qualities of the original model.

In [37], the authors introduce two KG embedding models, TransOWL and TransROWL, which employs a *knowledge injection* approach. The two models proposed consist in adaptions of TransE [20] and TransR [77] in which the loss function is enriched with terms expressing ontological constraints expressing class and property hierarchies and equivalences, and inverse properties. The empirical evaluation shown reductions on the false positive rates, but also that there is still room for improvement.

BoxE [1] is a geometric KG embedding model designed to cope with patterns between relations such as symmetry, inversion and composition, as well as one-to-many and many-to-many relations. The ability to expresses those complex relationship patterns is a common short-coming in translation-based embedding models [1, 103], which can damper the benefits of their comparatively more explainable formal framework. In addition, BoxE employs knowledge injection to guarantee that the outcome always satisfies a rule. The authors prove that BoxE can be injected with rules in a language that combines the union, symmetry, hierarchy and intersection constraints. There were previous approaches to rule injection, but with more limited languages [41, 93].

There are also other approaches which rely directly on existing reasoners. Wiharja et al. [118] investigate iterative strategies for KG completion using a combination of methods that involves deductive reasoning (classical reasoners) and inductive reasoning (KG embeddings). In addition to showing that traditional KG embeddings do not comply with ontological constraints specified in OWL or in SHACL. In their iterative approach they employ an incomplete, but highly efficient reasoning procedure to detect triples that violate the constraints. The wrong triples are deleted, and the correct triples are added to the current KG, and the process is repeated until a fixpoint or other convergence condition is reached. ReasonKGE is a similar iterative approach proposed by Jain et al. [58] that relies on the use of deductive reasoning as an intermediate step. The main difference is that the wrong triples (that is, the ones that violate the ontological constraints) are not discarded, they are added to the set of negative samples for training the internal embedding model.

Gutiérrez-Basulto and Schockaert [53] introduce geometric models, where relations are mapped to convex regions, rather than vectors. This principled solution is suitable for a large fragment of the first-order Horn rule language, called quasi-chained. This language also corresponds to the chain-Datalog variant, which also places constraints on the negation (in particular, in combination with recursion) [109]. In this approach using convex regions [53], any triple predicted by that embedding model will not only be consistent with the quasi-chained ontology at hand, but it will also be a logical consequence of that same ontology. However, the geometric model based on convex-region, as introduced in the mentioned work, does not capture negation beyond the ability of expressing disjointness [86]. This is investigated in a new semantics for representing relations which allows full negation, called cone semantics [86]. In cone semantics, an interpretation maps each concept name to an axis-aligned, convex, cone (henceforth, simply cone). The authors then focus on the DL $\mathcal{ALC}$ and extend the semantics to all of its concept expressions (i.e. those built using the concept constructors in Table 1) so that those are also mapped to cones.

To conclude this discussion on KG embeddings and ontological constraints, we remark that there are still open questions regarding the ability of embedding models in capturing the semantics of entities and relations in a way that is logically consistent. The main argument for focussing on these questions remains to be able to verify that the embeddings capture connections between entities, concepts and relations. Therefore, there is still work to be done if we consider different languages and other forms of building defining interpretations using embeddings, as even the most principled approaches in this direction still have drawbacks [53, 58, 86].

3.1.2 KG Embeddings and Temporal Information

There are two prominent aspects in KGs that invoke time-awareness. First, as discussed earlier, KGs often include facts annotated with time validity [39]. For instance, the fact that Artur Ávila is member of the USNAS is only valid from 2019 onwards. The second aspect is the dynamic nature of KGs [60, 99]. KGs can be edited both by humans and programs, receiving constant updates, which also require constant repair. Hence, the KG itself changes with time. When performing reasoning, for instance via KG completion with embeddings, it might be crucial to take into account time validity and changes over time. These needs fostered KG embedding proposals that give a particular attention to temporal information. This is still a very recent and active area of research, and as such, we only briefly summarise some approaches in this field while presenting some of the characteristics that differentiate these methods from usual (atemporal) KG embedding strategies. We refer the reader to the survey due to Kazemi et al. [60] for a more detailed and thorough overview of temporal KG embedding approaches (and other time-aware Representation Learning methods for KGs).

Many approaches for temporal embeddings depart from atemporal KG embedding methods and modify the models to accommodate time [98] or devise frameworks in which traditional KG embedding methods are seen as components [120, 125]. Moreover, the strategies for creating this models often fall into the same three categories as traditional embedding models: geometric models [96, 115, 116, 122], tensor factorisation models [31, 50, 76, 98] and models based on neural networks [74]. HyTE [39], for example, was one of the first embedding models to encode time into the embeddings. HyTE represents the dynamic KG as a series of *snapshots*, each snapshot being a static KG at a specific point in time. In this model, translation-based embedding models, such as TransE, are modified with the specification of hyperplanes that indicate the time component of a fact. Wang and Li [116] employed the same hyperplane-based approach and extended TransD.

SEDE [127] also departs from TransE and uses the same snapshot-modelling intuition as HyTE. However the adaption for accommodating temporal information is inspired by strategies used in semantic word evolution. QCHyTE is another model based on HyTE which view the changes over time as gradual rather than abrupt, using a probabilistic approach [33]. Other methods that also model changes as gradual processes using probabilities include the frameworks due to Liao et al. [75] and ATiSE model [123]. TKGFrame [126] quantifies and models the temporal dependencies between relations over the same individual. TKGFrame is able to quantify, for example, that a transition from *bornIn* to *marriedTo* requires more "time" than the transition from *bornIn* to *diedIn* [126].

The existing models for temporal KG embeddings also differ regarding on whether they focus on events (individual time points), such as DE-SimplE [50], or if they are able to handle intervals, as is the case of TeRo [122]. Granularity of time intervals and precision of time points is also relevant to time-sensitive applications and it has been explored in the context of KG embeddings [70, 83, 124]. The TDG2E model [104], for instance, extends TransE with temporal information and uses a recurrent neural network to encode the dependencies between different snapshots and address issues related to the sparsity of temporal annotations. Li et al. [74] also use recurrent neural networks to represent temporal dependencies, but without using a geometric model underneath. The strategy relies primarily on an application of recurrence structures with graph convolutional networks [74]. The TIME2BOX model [25] is particularly expressive as it can deal with facts with unspecified time validity, open intervals and closed intervals (including left/right-open).

We note that some challenges that remain ubiquitous to KG embedding models in general are also being addressed in the temporal subfield. As usual, computational performance, in terms of resource usage, is an eminent concern [121], as well KG completion quality under adverse conditions, such as relations that occur in few facts in a KG [81]. Ensuring that the semantics of entities and relations is captured adequately (and ideally, with formal guarantees) is also a major concern and an active research problem in this field [78]. Models such as RETRA [117] and the approach due to Lie et al. [78] focus on contexts of facts defined by sharing of entities and relations.

Bourgaux, Ozaki and Pan [22]'s proposal also addresses the problem of capturing temporal aspects and ontological constraints. More precisely, the authors extended the geometric models as proposed by Gutiérrez-Basulto and Schockaert [53] from the classical DL setting to attributed DLs (as presented in Section 2.1.1). Then, they include the time dimension by enriching the attributed DL obtained with temporal operators. These operators allow a user to specify temporal dependencies such as "before", "until" and "during". Furthermore, the authors identify a fragments of the resulting language that ensure that a convex geometric model exists capturing its semantics.

4 Extracting Rules

Association Rule Mining (ARM) [4] and Formal Concept Analysis (FCA) [49] are two approaches for extracting rules from data. Initially, these techniques were applied for data structured in the format of databases. More recently, there has been increased interest in applying ARM and FCA for extracting rules from KGs [18, 19, 48, 52, 54, 69].

ARM is a practical and highly scalable approach for extracting rules from KGs. One of the most prominent tools to extract rules from KGs based on ARM is AMIE [48]. Current versions of this tool can extract rules from large KGs in a few minutes [69]. This approach uses two measures, called *support* and *confidence*, to guide the search of rules and to identify significant patterns to be extracted. In this context, the support of a rule is the number of true predictions of a rule in a KG. The authors of AMIE point out that for many real world KGs the support measure drastically decreases for the vast majority of the rules with more than 4 predicates in the body of the rule. Since low support for a rule means that the rule would only be applicable to few objects, this implies that, in most cases, long and difficult to interpret rules (with more than 4 predicates) can be discarded. Moreover, many combinations of 2 or 3 predicates already results in low suport.

▶ **Example 20.** Consider a rule with the binary predicate "teachesAt" in its body. The first argument of this predicate can only refer to educators while the second can only refer to educational institutions, which already significantly limits the possible combinations of predicates that can result in a rule with high support.

So the search space can be drastically reduced by applying the support measure to guide the search for significant patterns in the data. The main difficulty with the ARM approach is that, in order to achieve high scalability, the structure of the rules can be radically limited and that it does not provide theoretical guarantees regarding the extracted rules. In particular, this approach does not guarantee that all relevant rules in a certain rule language are extracted. It is also the case that the number of rules extracted can be very large (even though the size of each rule is limited).

The FCA approach addresses the difficulties of ARM, in particular, it gives theoretical guarantees for the rules extracted. The classical setting focuses on propositional Horn theory. In the DL context, there are e.g. works applying FCA to compute all concept inclusions of the $\mathcal{EL}$ ontology language (this language corresponds to the $\mathcal{ALC}$ description logic without negation, disjunction, and value restriction, see Table 1) that hold in a dataset [42, 52]. FCA also focus on succinctness, in particular, on constructing a set of rules that is minimal, called the *Duquenne-Guigues base* or *stem base*. The main challenges in applying FCA for extracting rules from KGs are (1) scalability, (2) the presence of a significant portion of erroneus information, and (3) the fact that rules generated with FCA can be complex and difficult to interpret. One way of dealing with challenges (1) and (2) is to combine FCA and ARM. A simple way of dealing with challenge (3) is by limiting the size of rules extracted, as it is the case in ARM.

5 Conclusion

In this work, we discussed different forms of inductive and deductive reasonings on Knowledge Graphs. Deductive and inductive reasoning have different strengths and shortcomings. Combining those to provide efficient and explainable reasoning is one of the current challenges in Artificial Intelligence. Besides deductive and inductive reasoning, there is also a third modality: abductive reasoning, which we briefly discuss next.

This third form of reasoning also aims at inferring facts, but from another perspective. Instead of focusing on new consequences that can be discovered, abductive reasoning concerns finding explanations for entailments already known or the lack of expected entailments. For example, suppose that we find out using inductive reasoning that the Wikidata KG implies that Artur Ávila is European. An abductive reasoning procedure could then be used to identify, among the numerous facts in Wikidata, which are strictly necessary to reach this conclusion. That is, we would like to find an explanation for that conclusion. Similarly, we might be interested in adding facts, axioms or rules to our KG or reasoning system to reach a particular conclusion without explicitly stating it.

Abduction in DLs, for instance, is a rich area of research with many different applications [13, 45, 64, 91]. One of them is debug and repair of DL ontologies via explanations and "explanation-like" objects, such as justifications [56], MinAs [9] and others [80]. The main goal is to find the explanations for undesired consequences and transform the ontology (or KG) so that it does not entail those anymore. Also, in the context of repair, we might be interested in finding the largest portions of the KG that do not entail a certain fact or present a particular behaviour. Such sets, and similar constructions, appear under many names in the Ontology Repair literature such as MaNAs [9], repairs [8], among others [80]. Some studies relate abductive reasoning and Datalog, either solving this problem for Datalog programs [24] or using Datalog to perform abduction in other languages. Additionally, as we mentioned in Section 2.3, there are also works on explanations for SHACL violations [5].

To conclude, we note that the approaches presented and mentioned here differ regarding the treatment of the KG and the reasoning problems they aim to solve. Understanding their advantages and disadvantages is key to devising better solutions for the existing challenges in reasoning with KGs.

References

1 Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. Boxe: A box embedding model for knowledge base completion. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: `https://proceedings.neurips.cc/paper/2020/hash/6dbbe6abe5f14af882ff977fc3f35501-Abstract.html`.

2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley, 1994.

3 Serge Abiteboul and Victor Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.

4 Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *Special Interest Group on Management Of Data SIGMOD*, 22(2):207–216, June 1993.

5 Shqiponja Ahmetaj, Robert David, Magdalena Ortiz, Axel Polleres, Bojken Shehu, and Mantas Šimkus. Reasoning about explanations for non-validation in SHACL. In *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning.* International Joint Conferences on Artificial Intelligence Organization, September 2021. `doi:10.24963/kr.2021/2`.

6 Medina Andresel, Julien Corman, Magdalena Ortiz, Juan L. Reutter, Ognjen Savkovic, and Mantas Simkus. Stable Model Semantics for Recursive SHACL. In *Proceedings of The Web Conference 2020.* ACM, April 2020. `doi:10.1145/3366423.3380229`.

7 Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic.* Cambridge University Press, 2017.

8 Franz Baader, Francesco Kriegel, Adrian Nuradiansyah, and Rafael Peñaloza. Making repairs in description logics more gentle. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, pages 319–328. AAAI Press, 2018. URL: `https://aaai.org/ocs/index.php/KR/KR18/paper/view/18056`.

9 Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *J. Log. Comput.*, 20(1):5–34, 2010. `doi:10.1093/logcom/exn058`.

10 Luigi Bellomarini, Eleonora Laurenza, Emanuel Sallinger, and Evgeny Sherkhonov. Reasoning under uncertainty in knowledge graphs. In *Rules and Reasoning*, pages 131–139. Springer International Publishing, 2020. `doi:10.1007/978-3-030-57977-7_9`.

11 Luigi Bellomarini, Markus Nissl, and Emanuel Sallinger. Monotonic aggregation for temporal datalog. In Ahmet Soylu, Alireza Tamaddoni-Nezhad, Nikolay Nikolov, Ioan Toma, Anna Fensel, and Joost Vennekens, editors, *Proceedings of the 15th International Rule Challenge, 7th Industry Track, and 5th Doctoral Consortium @ RuleML+RR 2021 co-located with 17th Reasoning Web Summer School (RW 2021) and 13th DecisionCAMP 2021 as part of Declarative AI 2021, Leuven, Belgium (virtual due to Covid-19 pandemic), 8 - 15 September, 2021*, volume 2956 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021. URL: `http://ceur-ws.org/Vol-2956/paper30.pdf`.

12 Luigi Bellomarini, Emanuel Sallinger, and Georg Gottlob. The vadalog system. *Proceedings of the VLDB Endowment*, 11(9):975–987, May 2018. `doi:10.14778/3213880.3213888`.

13 Meghyn Bienvenu. Complexity of abduction in the EL family of lightweight description logics. In Gerhard Brewka and Jérôme Lang, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, pages 220–230. AAAI Press, 2008. URL: `http://www.aaai.org/Library/KR/2008/kr08-022.php`.

14 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS'13)*, pages 213–224. ACM, 2013.

15 Bart Bogaerts and Maxime Jakubowski. Fixpoint Semantics for Recursive SHACL. *Electronic Proceedings in Theoretical Computer Science*, 345:41–47, September 2021. `doi:10.4204/eptcs.345.14`.

16 Bart Bogaerts, Maxime Jakubowski, and Jan Van den Bussche. SHACL: A Description Logic in Disguise. In *Proceedings of the 33rd Benelux Conference on Artificial Intelligence and the 30th Belgian Dutch Conference on Machine Learning (BNAIC/BENELEARN 2021)*, August 2021. `arXiv:2108.06096`.

17 Iovka Boneva, José Emilio Labra Gayo, and Eric G. Prud'hommeaux. Semantics and validation of shapes schemas for RDF. In Claudia d'Amato, Miriam Fernández, Valentina A. M. Tamma, Freddy Lécué, Philippe Cudré-Mauroux, Juan F. Sequeda, Christoph Lange, and Jeff Heflin, editors, *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, volume 10587 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2017. `doi:10.1007/978-3-319-68288-4_7`.

18 Daniel Borchmann. *Learning terminological knowledge with high confidence from erroneous data*. PhD thesis, Higher School of Economics, 2014.

19 Daniel Borchmann and Felix Distel. Mining of $\mathcal{EL}$-GCIs. In *The 11th IEEE International Conference on Data Mining Workshops*, Vancouver, Canada, 2011.

20 Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

21 Camille Bourgaux and Ana Ozaki. Querying attributed dl-lite ontologies using provenance semirings. In *AAAI*, pages 2719–2726. AAAI Press, 2019.

22 Camille Bourgaux, Ana Ozaki, and Jeff Z. Pan. Geometric models for (temporally) attributed description logics. In Martin Homola, Vladislav Ryzhikov, and Renate A. Schmidt, editors, *Proceedings of the 34th International Workshop on Description Logics (DL 2021) part of Bratislava Knowledge September (BAKS 2021), Bratislava, Slovakia, September 19th to 22nd, 2021*, volume 2954 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021. URL: `http://ceur-ws.org/Vol-2954/paper-7.pdf`.

23 Sebastian Brandt, Elem Güzel Kalayci, Roman Kontchakov, Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. Ontology-based data access with a horn fragment of metric temporal logic. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1070–1076. AAAI Press, 2017. URL: `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14881`.

24 F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000. `doi:10.1109/69.877512`.

25 Ling Cai, Krzysztof Janowicz, Bo Yan, Rui Zhu, and Gengchen Mai. Time in a Box. In *Proceedings of the 11th on Knowledge Capture Conference*. ACM, December 2021. `doi:10.1145/3460210.3493566`.

26 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog±. In *Proceedings of the 12th International Conference on Database Theory - ICDT '09*. ACM Press, 2009. `doi:10.1145/1514894.1514897`.

27 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, 14:57–83, July 2012. `doi:10.1016/j.websem.2012.03.001`.

28 Andrea Calì, Georg Gottlob, and Andreas Pieris. Query answering under non-guarded rules in Datalog+/-. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Proceedings of the 4th International Conference on Web Reasoning and Rule Systems (RR 2010)*, volume 6333 of *LNCS*, pages 1–17. Springer, 2010.

29 David Carral, Irina Dragoste, Larry González, Ceriel Jacobs, Markus Krötzsch, and Jacopo Urbani. VLog: A rule engine for knowledge graphs. In *Lecture Notes in Computer Science*, pages 19–35. Springer International Publishing, 2019. `doi:10.1007/978-3-030-30796-7_2`.

30 S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, March 1989. `doi:10.1109/69.43410`.

31 Melisachew Wudage Chekol. Tensor decomposition for link prediction in temporal knowledge graphs. In *Proceedings of the 11th on Knowledge Capture Conference*. ACM, December 2021. `doi:10.1145/3460210.3493558`.

32 Melisachew Wudage Chekol, Giuseppe Pirrò, Joerg Schoenfisch, and Heiner Stuckenschmidt. Marrying uncertainty and time in knowledge graphs. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 88–94. AAAI Press, 2017. URL: `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14730`.

33 Shuo Chen, Lin Qiao, Biqi Liu, Jue Bo, Yuanning Cui, and Jing Li. Knowledge Graph Embedding Based on Hyperplane and Quantitative Credibility. In *Machine Learning and Intelligent Communications*, pages 583–594. Springer International Publishing, 2019. `doi:10.1007/978-3-030-32388-2_50`.

34 Jan Chomicki and Tomasz Imielinski. Temporal deductive databases and infinite objects. In Chris Edmondson-Yurkanan and Mihalis Yannakakis, editors, *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas, USA*, pages 61–73. ACM, 1988. `doi:10.1145/308386.308416`.

35 Julien Corman, Juan L. Reutter, and Ognjen Savković. Semantics and validation of recursive SHACL. In *Lecture Notes in Computer Science*, pages 318–336. Springer International Publishing, 2018. doi:10.1007/978-3-030-00671-6_19.

36 Yuanfei Dai, Shiping Wang, Neal N. Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, May 2020. doi:10.3390/electronics9050750.

37 Claudia d'Amato, Nicola Flavio Quatraro, and Nicola Fanizzi. Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs. In *The Semantic Web*, pages 441–457. Springer International Publishing, 2021. doi:10.1007/978-3-030-77385-4_26.

38 Ariyam Das, Sahil M. Gandhi, and Carlo Zaniolo. ASTRO: A datalog system for advanced stream reasoning. In Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang, editors, *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 1863–1866. ACM, 2018. doi:10.1145/3269206.3269223.

39 Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha P. Talukdar. HyTE: Hyperplane-based Temporally aware Knowledge Graph Embedding. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2001–2011. Association for Computational Linguistics, 2018. doi:10.18653/v1/d18-1225.

40 Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1811–1818. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366.

41 Boyang Ding, Quan Wang, Bin Wang, and Li Guo. Improving knowledge graph embedding using simple constraints. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 110–121. Association for Computational Linguistics, 2018. doi:10.18653/v1/P18-1011.

42 Felix Distel. *Learning description logic knowledge bases from data using methods from formal concept analysis.* PhD thesis, Dresden University of Technology, 2011.

43 Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. $\mathcal{AL}$-log: Integrating datalog and description logics. *Journal of Intelligent and Cooperative Information Systems*, 10(3):227–252, 1998.

44 Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997. doi:10.1145/261124.261126.

45 Corinna Elsenbroich, Oliver Kutz, and Ulrike Sattler. A case for abductive reasoning over ontologies. In Bernardo Cuenca Grau, Pascal Hitzler, Conor Shankey, and Evan Wallace, editors, *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, Georgia, USA, November 10-11, 2006*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006. URL: http://ceur-ws.org/Vol-216/submission_25.pdf.

46 Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. Improved knowledge graph embedding using background taxonomic information. In *AAAI*, pages 3526–3533. AAAI Press, 2019.

47 Mónica Figuera, Philipp D. Rohde, and Maria-Esther Vidal. Trav-SHACL: Efficiently validating networks of SHACL constraints. In *Proceedings of the Web Conference 2021*. ACM, April 2021. doi:10.1145/3442381.3449877.

48 Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6):707–730, 2015.

49 Bernhard Ganter and Rudolph Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1997.

50 Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. Diachronic Embedding for Temporal Knowledge Graph Completion. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3988–3995, April 2020. doi:10.1609/aaai.v34i04.5815.

51 Georg Gottlob, Erich Grädel, and Helmut Veith. Datalog LITE: a deductive query language with linear time model checking. *ACM Trans. Comput. Log.*, 3(1):42–79, 2002. doi:10.1145/504077.504079.

52 Ricardo Guimarães, Ana Ozaki, Cosimo Persia, and Baris Sertkaya. Mining EL bases with adaptable role depth. In *AAAI*, pages 6367–6374. AAAI Press, 2021.

53 Víctor Gutiérrez-Basulto and Steven Schockaert. From knowledge graph embedding to ontology embedding? an analysis of the compatibility between vector space representations and rules. In *Proceedings of KR*, 2018.

54 Tom Hanika, Maximilian Marx, and Gerd Stumme. Discovering implicational knowledge in wikidata. In Diana Cristea, Florence Le Ber, and Baris Sertkaya, editors, *ICFCA*, volume 11511 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2019.

55 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2021.

56 Matthew Horridge. *Justification based explanation in ontologies.* PhD thesis, University of Manchester, 2011.

57 Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin N. Grosof, and Mike Dean. *SWRL: A Semantic Web Rule Language.* W3C Member Submission, 21 May 2004. Available at http://www.w3.org/Submission/SWRL/.

58 Nitisha Jain, Trung-Kien Tran, Mohamed H. Gad-Elrab, and Daria Stepanova. Improving Knowledge Graph Embeddings with Ontological Reasoning. In Andreas Hotho, Eva Blomqvist, Stefan Dietze, Achille Fokoue, Ying Ding, Payam M. Barnaghi, Armin Haller, Mauro Dragoni, and Harith Alani, editors, *ISWC*, volume 12922 of *Lecture Notes in Computer Science*, pages 410–426. Springer, 2021.

59 Yevgeny Kazakov, Markus Krötzsch, and František Simančík. ELK reasoner: Architecture and evaluation. In Ian Horrocks, Mikalai Yatskevich, and Ernesto Jimenez-Ruiz, editors, *Proceedings of the OWL Reasoner Evaluation Workshop 2012 (ORE'12)*, volume 858 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

60 Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *J. Mach. Learn. Res.*, 21:70:1–70:73, 2020. URL: http://jmlr.org/papers/v21/19-447.html.

61 Mayank Kejriwal, Craig A. Knoblock, and Pedro Szekely. *Knowledge Graphs.* The MIT Press, March 2021. URL: https://www.ebook.de/de/product/39993807/mayank_kejriwal_craig_a_knoblock_pedro_szekely_knowledge_graphs.html.

62 Holger Knublauch and Dimitris Kontokostas, editors. *Shapes Constraint Language (SHACL).* W3C Recommendation, 20 July 2017. Available at http://www.w3.org/TR/shacl/.

63 Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to ontology-based data access. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2656–2661. AAAI Press/IJCAI, 2011.

64 Patrick Koopmann, Warren Del-Pinto, Sophie Tourret, and Renate A. Schmidt. Signature-based abduction for expressive description logics. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 592–602, 2020. doi:10.24963/kr.2020/59.

65 Markus Krötzsch, Maximilian Marx, Ana Ozaki, and Veronika Thost. Attributed description logics: Reasoning on knowledge graphs. In Jérôme Lang, editor, *IJCAI*, pages 5309–5313. ijcai.org, 2018.

66 Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. Description logic rules. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikos Avouris, editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 80–84. IOS Press, 2008.

67 Markus Krötzsch, Sebastian Rudolph, and Peter H. Schmitt. On the semantic relationship between datalog and description logics. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Proceedings of the 4th International Conference on Web Reasoning and Rule Systems (RR 2010)*, volume 6333 of *LNCS*, pages 88–102. Springer, 2010.

68 Markus Krötzsch, Maximilian Marx, Ana Ozaki, and Veronika Thost. Attributed description logics: Ontologies for knowledge graphs. In *Lecture Notes in Computer Science*, pages 418–435. Springer International Publishing, 2017. `doi:10.1007/978-3-319-68288-4_25`.

69 Jonathan Lajus, Luis Galárraga, and Fabian M. Suchanek. Fast and exact rule mining with AMIE 3. In Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, Peter Haase, and Michael Cochez, editors, *ESWC*, volume 12123 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2020.

70 Julien Leblay, Melisachew Wudage Chekol, and Xin Liu. Towards Temporal Knowledge Graph Embeddings with Arbitrary Time Precision. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. ACM, October 2020. `doi:10.1145/3340531.3412028`.

71 Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6:167–195, 2015. `doi:10.3233/SW-140134`.

72 Martin Leinberger, Philipp Seifer, Tjitze Rienstra, Ralf Lämmel, and Steffen Staab. Deciding SHACL shape containment through description logics reasoning. In *Lecture Notes in Computer Science*, pages 366–383. Springer International Publishing, 2020. `doi:10.1007/978-3-030-62419-4_21`.

73 Nicola Leone, Carlo Allocca, Mario Alviano, Francesco Calimeri, Cristina Civili, Roberta Costabile, Alessio Fiorentino, Davide Fuscà, Stefano Germano, Giovanni Laboccetta, Bernardo Cuteri, Marco Manna, Simona Perri, Kristian Reale, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. Enhancing DLV for large-scale reasoning. In *Logic Programming and Nonmonotonic Reasoning*, pages 312–325. Springer International Publishing, 2019. `doi:10.1007/978-3-030-20528-7_23`.

74 Zixuan Li, Xiaolong Jin, Wei Li, Saiping Guan, Jiafeng Guo, Huawei Shen, Yuanzhuo Wang, and Xueqi Cheng. Temporal Knowledge Graph Reasoning Based on Evolutional Representation Learning. In Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai, editors, *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 408–417. ACM, 2021. `doi:10.1145/3404835.3462963`.

75 Siyuan Liao, Shangsong Liang, Zaiqiao Meng, and Qiang Zhang. Learning Dynamic Embeddings for Temporal Knowledge Graphs. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. ACM, March 2021. `doi:10.1145/3437963.3441741`.

76 Lifan Lin and Kun She. Tensor decomposition-based temporal knowledge graph embedding. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, November 2020. `doi:10.1109/ictai50040.2020.00151`.

77 Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2181–2187. AAAI Press, 2015. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571`.

78 Yu Liu, Wen Hua, Jianfeng Qu, Kexuan Xin, and Xiaofang Zhou. Temporal knowledge completion with context-aware embeddings. *World Wide Web*, 24(2):675–695, March 2021. doi:10.1007/s11280-021-00867-6.

79 Frank Manola and Eric Miller, editors. *Resource Description Framework (RDF): Primer.* W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/rdf-primer/.

80 Vinícius Bitencourt Matos, Ricardo Ferreira Guimarães, Yuri David Santos, and Renata Wassermann. Pseudo-contractions as gentle repairs. In Carsten Lutz, Uli Sattler, Cesare Tinelli, Anni-Yasmin Turhan, and Frank Wolter, editors, *Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, volume 11560 of *Lecture Notes in Computer Science*, pages 385–403. Springer, 2019. doi:10.1007/978-3-030-22102-7_18.

81 Mehrnoosh Mirtaheri, Mohammad Rostami, Xiang Ren, Fred Morstatter, and Aram Galstyan. One-shot learning for temporal knowledge graphs. In Danqi Chen, Jonathan Berant, Andrew McCallum, and Sameer Singh, editors, *3rd Conference on Automated Knowledge Base Construction, AKBC 2021, Virtual, October 4-8, 2021*, 2021. doi:10.24432/C55K56.

82 Yavor Nenov, Robert Piro, Boris Motik, Ian Horrocks, Zhe Wu, and Jay Banerjee. RDFox: A highly-scalable RDF store. In *The Semantic Web - ISWC 2015*, pages 3–20. Springer International Publishing, 2015. doi:10.1007/978-3-319-25010-6_1.

83 Runyu Ni, Zhonggui Ma, Kaihang Yu, and Xiaohan Xu. Specific Time Embedding for Temporal Knowledge Graph Completion. In *2020 IEEE 19th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC).* IEEE, September 2020. doi:10.1109/iccicc50026.2020.9450214.

84 W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview.* W3C Recommendation, 27 october 2009. Available at http://www.w3.org/TR/owl2-overview/.

85 Ana Ozaki, Markus Krötzsch, and Sebastian Rudolph. Temporally attributed description logics. In Carsten Lutz, Uli Sattler, Cesare Tinelli, Anni-Yasmin Turhan, and Frank Wolter, editors, *Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, volume 11560 of *Lecture Notes in Computer Science*, pages 441–474. Springer, 2019.

86 Özgür Lütfü Özçep, Mena Leemhuis, and Diedrich Wolter. Cone semantics for logics with negation. In Christian Bessiere, editor, *IJCAI*, pages 1820–1826. ijcai.org, 2020.

87 Paolo Pareti, George Konstantinidis, and Fabio Mogavero. Satisfiability and containment of recursive SHACL. *CoRR*, abs/2108.13063, 2021. arXiv:2108.13063.

88 Paolo Pareti, George Konstantinidis, Fabio Mogavero, and Timothy J. Norman. SHACL satisfiability and containment. In *Lecture Notes in Computer Science*, pages 474–493. Springer International Publishing, 2020. doi:10.1007/978-3-030-62419-4_27.

89 Bijan Parsia, Nicolas Matentzoglu, Rafael S. Gonçalves, Birte Glimm, and Andreas Steigmiller. The OWL reasoner evaluation (ORE) 2015 competition report. *Journal of Automated Reasoning*, 59(4):455–482, February 2017. doi:10.1007/s10817-017-9406-8.

90 Eric Prud'hommeaux and Andy Seaborne, editors. *SPARQL Query Language for RDF.* W3C Recommendation, 15 January 2008. Available at http://www.w3.org/TR/rdf-sparql-query/.

91 Júlia Pukancová and Martin Homola. Abductive Reasoning with Description Logics: Use Case in Medical Diagnosis. In Diego Calvanese and Boris Konev, editors, *Proceedings of the 28th International Workshop on Description Logics, Athens,Greece, June 7-10, 2015*, volume 1350 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1350/paper-60.pdf.

92 Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, January 2006. doi:10.1007/s10994-006-5833-1.

93 Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1119–1129. The Association for Computational Linguistics, 2015. doi:10.3115/v1/n15-1118.

94 Riccardo Rosati. $\mathcal{DL}$+*log*: A tight integration of description logics and disjunctive datalog. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 68–78. AAAI Press, 2006.

95 Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Description logic reasoning with decision diagrams: Compiling $\mathcal{SHIQ}$ to disjunctive datalog. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *Proceedings of the 7th International Semantic Web Conference (ISWC'08)*, volume 5318 of *LNCS*, pages 435–450. Springer, 2008.

96 Ali Sadeghian, Mohammadreza Armandpour, Anthony Colas, and Daisy Zhe Wang. ChronoR: Rotation Based Temporal Knowledge Graph Embedding. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 6471–6479. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16802.

97 Floriano Scioscia, Michele Ruta, Ivano Bilenchi, Filippo Gramegna, Eugenio Di Sciascio, and Davide Loconte. Owl reasoner evaluation results, 2021. doi:10.5281/ZENODO.5013799.

98 Pengpeng Shao, Dawei Zhang, Guohua Yang, Jianhua Tao, Feihu Che, and Tong Liu. Tucker decomposition-based temporal knowledge graph completion. *Knowledge-Based Systems*, 238:107841, February 2022. doi:10.1016/j.knosys.2021.107841.

99 Umang Sharan and Jennifer Neville. Temporal-relational classifiers for prediction in evolving domains. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, December 2008. doi:10.1109/icdm.2008.125.

100 Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A Highly-Efficient OWL Reasoner. In Catherine Dolbear, Alan Ruttenberg, and Ulrike Sattler, editors, *OWLED*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

101 Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

102 Andreas Steigmiller, Thorsten Liebig, and Birte Glimm. Konclude: System description. *Journal of Web Semantics*, 27-28:78–85, August 2014. doi:10.1016/j.websem.2014.06.003.

103 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HkgEQnRqYQ.

104 Xiaoli Tang, Rui Yuan, Qianyu Li, Tengyun Wang, Haizhi Yang, Yundong Cai, and Hengjie Song. Timespan-Aware Dynamic Knowledge Graph Embedding by Incorporating Temporal Evolution. *IEEE Access*, 8:6849–6860, 2020. doi:10.1109/access.2020.2964028.

105 Edward Thomas, Jeff Z. Pan, and Yuan Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *Lecture Notes in Computer Science*, pages 431–435. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13489-0_38.

106 Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016. URL: http://proceedings.mlr.press/v48/trouillon16.html.

107 Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNCS*, pages 292–297. Springer, 2006.

108 Jeffrey Ullman. *Principles of database and knowledge-base systems*. Computer Science Press, Rockville, Md, 1988.

109 Jeffrey D. Ullman and Allen Van Gelder. Parallel complexity of logical query programs. *Algorithmica*, 3(1-4):5–42, November 1988. `doi:10.1007/bf01762108`.

110 Jacopo Urbani, Ceriel Jacobs, and Markus Krötzsch. Column-oriented datalog materialization for large knowledge graphs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 2016. To appear.

111 Boris Villazón-Terrazas, Nuria García-Santa, Yuan Ren, Alessandro Faraotti, Honghan Wu, Yuting Zhao, Guido Vetere, and Jeff Z. Pan. Knowledge graph foundations. In Jeff Z. Pan, Guido Vetere, José Manuél Gómez-Pérez, and Honghan Wu, editors, *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, pages 17–55. Springer, 2017.

112 Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10), 2014.

113 Przemyslaw Andrzej Walega, Mark Kaminski, and Bernardo Cuenca Grau. Reasoning over streaming data in metric temporal datalog. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3092–3099. AAAI Press, 2019. `doi:10.1609/aaai.v33i01.33013092`.

114 Dingmin Wang, Pan Hu, Przemyslaw Andrzej Walega, and Bernardo Cuenca Grau. Meteor: Practical reasoning in datalog with metric temporal operators. *CoRR*, abs/2201.04596, 2022. `arXiv:2201.04596`.

115 Jingbin Wang, Wang Zhang, Xinyuan Chen, Jing Lei, and Xiaolian Lai. 3DRTE: 3D Rotation Embedding in Temporal Knowledge Graph. *IEEE Access*, 8:207515–207523, 2020. `doi:10.1109/access.2020.3036897`.

116 Zhihao Wang and Xin Li. Hybrid-TE: Hybrid Translation-Based Temporal Knowledge Graph Embedding. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, November 2019. `doi:10.1109/ictai.2019.00205`.

117 Simon Werner, Achim Rettinger, Lavdim Halilaj, and Jürgen Lüttin. RETRA: Recurrent transformers for learning temporally contextualized knowledge graph embeddings. In *The Semantic Web*, pages 425–440. Springer International Publishing, 2021. `doi:10.1007/978-3-030-77385-4_25`.

118 Kemas Wiharja, Jeff Z. Pan, Martin J. Kollingbaum, and Yu Deng. Schema aware iterative knowledge graph completion. *Journal of Web Semantics*, 65:100616, December 2020. `doi:10.1016/j.websem.2020.100616`.

119 Honghan Wu, Ronald Denaux, Panos Alexopoulos, Yuan Ren, and Jeff Z. Pan. Understanding knowledge graphs. In Jeff Z. Pan, Guido Vetere, José Manuél Gómez-Pérez, and Honghan Wu, editors, *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, pages 147–180. Springer, 2017.

120 Jiapeng Wu, Yishi Xu, Yingxue Zhang, Chen Ma, Mark Coates, and Jackie Chi Kit Cheung. TIE: A framework for embedding-based incremental temporal knowledge graph completion. In Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai, editors, *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 428–437. ACM, 2021. `doi:10.1145/3404835.3462961`.

121 Tianxing Wu, Arijit Khan, Huan Gao, and Cheng Li. Efficiently embedding dynamic knowledge graphs. *CoRR*, abs/1910.06708, 2019. `arXiv:1910.06708`.

122 Chengjin Xu, Mojtaba Nayyeri, Fouad Alkhoury, Hamed Shariat Yazdi, and Jens Lehmann. TeRo: A time-aware knowledge graph embedding via temporal rotation. In *Proceedings of the 28th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, 2020. `doi:10.18653/v1/2020.coling-main.139`.

123 Chenjin Xu, Mojtaba Nayyeri, Fouad Alkhoury, Hamed Yazdi, and Jens Lehmann. Temporal Knowledge Graph Completion Based on Time Series Gaussian Embedding. In *Lecture Notes in Computer Science*, pages 654–671. Springer International Publishing, 2020. `doi:10.1007/978-3-030-62419-4_37`.

124 Yonghui Xu, Shengjie Sun, Huiguo Zhang, Chang'an Yi, Yuan Miao, Dong Yang, Xiaonan Meng, Yi Hu, Ke Wang, Huaqing Min, Hengjie Song, and Chuanyan Miao. Time-Aware Graph Embedding: A Temporal Smoothness and Task-Oriented Approach. *ACM Transactions on Knowledge Discovery from Data*, 16(3):1–23, June 2022. doi:10.1145/3480243.

125 Youri Xu, Haihong E, Meina Song, Wenyu Song, Xiaodong Lv, Haotian Wang, and Jinrui Yang. RTFE: A Recursive Temporal Fact Embedding Framework for Temporal Knowledge Graph Completion. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5671–5681. Association for Computational Linguistics, 2021. doi:10.18653/v1/2021.naacl-main.451.

126 Jiasheng Zhang, Yongpan Sheng, Zheng Wang, and Jie Shao. TKGFrame: A Two-Phase Framework for Temporal-Aware Knowledge Graph Completion. In *Web and Big Data*, pages 196–211. Springer International Publishing, 2020. doi:10.1007/978-3-030-60259-8_16.

127 Yujing Zhou, Jia Peng, Lei Wang, Daren Zha, and Nan Mu. SEDE: semantic evolution-based dynamic knowledge graph embedding. *Aust. J. Intell. Inf. Process. Syst.*, 16(4):64–73, 2019. URL: http://ajiips.com.au/papers/V16.4/v16n4_68-77.pdf.

Integrating Ontologies and Vector Space Embeddings Using Conceptual Spaces

Zied Bouraoui
CRIL Laboratory, Université d'Artois, Arras, France

Víctor Gutiérrez-Basulto
School of Computer Science & Informatics, Cardiff University, UK

Steven Schockaert
School of Computer Science & Informatics, Cardiff University, UK

Abstract

Ontologies and vector space embeddings are among the most popular frameworks for encoding conceptual knowledge. Ontologies excel at capturing the logical dependencies between concepts in a precise and clearly defined way. Vector space embeddings excel at modelling similarity and analogy. Given these complementary strengths, there is a clear need for frameworks that can combine the best of both worlds. In this paper, we present an overview of our recent work in this area. We first discuss the theory of conceptual spaces, which was proposed in the 1990s by Gärdenfors as an intermediate representation layer in between embeddings and symbolic knowledge bases. We particularly focus on a number of recent strategies for learning conceptual space representations from data. Next, building on the idea of conceptual spaces, we discuss approaches where relational knowledge is modelled in terms of geometric constraints. Such approaches aim at a tight integration of symbolic and geometric representations, which unfortunately comes with a number of limitations. For this reason, we finally also discuss methods in which similarity, and other forms of conceptual relatedness, are derived from vector space embeddings and subsequently used to support flexible forms of reasoning with ontologies, thus enabling a looser integration between embeddings and symbolic knowledge.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning

Keywords and phrases Conceptual Spaces, Ontologies, Vector Space Embeddings, Learning and Reasoning

Digital Object Identifier 10.4230/OASIcs.AIB.2022.3

Category Invited Paper

Funding This work has been supported by a grant from the Leverhulme Trust (project RPG-2021-140).

1 Introduction

In Artificial Intelligence (AI), the traditional approach for encoding knowledge about concepts has been to use logic-based representations, typically in the form of a rule base. Such a rule base is often called an ontology in this context.

International Research School in Artificial Intelligence in Bergen (AIB 2022).
Editors: Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza; Article No. 3; pp. 3:1–3:30
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Example 1.** Consider the following rules:

$$\begin{aligned}
\mathsf{expertInAI}(X) &\leftarrow \mathsf{authorOf}(X, Y), \mathsf{hasTopic}(Y, \mathsf{artificialIntelligence})\\
\mathsf{hasTopic}(X, \mathsf{artificialIntelligence}) &\leftarrow \mathsf{hasTopic}(X, \mathsf{knowledgeRepresentation})\\
\mathsf{hasTopic}(X, \mathsf{artificialIntelligence}) &\leftarrow \mathsf{hasTopic}(X, \mathsf{machineLearning})\\
\mathsf{hasTopic}(X, \mathsf{artificialIntelligence}) &\leftarrow \mathsf{hasTopic}(X, \mathsf{multiAgentSystems})\\
\mathsf{hasTopic}(X, \mathsf{artificialIntelligence}) &\leftarrow \mathsf{hasTopic}(X, \mathsf{naturalLanguageProcessing})
\end{aligned}$$

Here we have used the notational conventions from logic programming, where the conclusion of the rule is shown on the left-hand side and "," denotes conjunction. The first rule intuitively asserts that somebody who has published a paper on an AI topic is an expert in AI. The remaining rules encode that knowledge representation, machine learning, multi-agent systems and natural language processing are sub-fields of AI. Along with the ontology, we are usually given a set of facts, e.g.:

$$\{\mathsf{authorOf}(\mathsf{bob}, p), \mathsf{hasTopic}(p, \mathsf{knowledgeRepresentation})\}$$

Given this set of facts, together with the aforementioned rules, we can conclude that $\mathsf{hasTopic}(p, \mathsf{artificialIntelligence})$ holds and thus also that $\mathsf{expertInAI}(\mathsf{bob})$ holds.

Using ontologies for encoding conceptual knowledge has at least two key advantages. First, the formal semantics of the underlying logic ensures that knowledge can be encoded in a precise and unambiguous way. This, in turn, ensures that different applications can rely on a shared understanding of the meaning of the concepts involved. Second, ontologies enable us to capture knowledge in a transparent and interpretable way[1], which makes it relatively straightforward to update knowledge and to support decisions with meaningful explanations. But ontologies, and symbolic approaches to knowledge representation more generally, also have important drawbacks. A first issue stems from the fact that the knowledge which is captured in an ontology is rarely complete. For instance, consider the following set of facts:

$$\{\mathsf{authorOf}(\mathsf{alice}, q), \mathsf{hasTopic}(q, \mathsf{planning})\}$$

As none of the available rules express that planning is a sub-field of AI, we cannot infer that $\mathsf{expertInAI}(\mathsf{alice})$ holds. Nonetheless, to a human observer, it seems clear that this would be a valid inference, even without a precise understanding of what the predicate $\mathsf{expertInAI}$ is supposed to capture. Essentially, standard frameworks for modelling ontologies lack a mechanism for inductive reasoning [28]. This is not something which can be easily addressed, as inductive arguments rely on graded notions such as similarity and typicality [58, 50, 66, 51]. Another issue is that many concepts are difficult to characterise in a satisfactory way using logical rules. For instance, somebody with a single published paper in AI would not normally be considered to be an AI expert, except perhaps if the work was particularly influential or groundbreaking, but formalising such notions using rules is challenging. Probabilistic extensions of standard ontology languages [36, 15] may alleviate some of the aforementioned issues, but such frameworks still do not allow us to model similarity, or aspects that are a matter of degree (e.g. being an expert in AI).

[1] It should be noted, however, that the extent to which a given ontology is interpretable will depend on its size and the way it has been encoded. Symbolic rules that have been learned from data can often be difficult to interpret, for instance.

The most common alternative to ontologies is to encode conceptual knowledge using vector space representations. Most work on vector representations of conceptual knowledge has focused on knowledge graphs (KGs), which are sets of triples of the form (e, r, f), where e and f are entities and r is a binary relation. Note that both individuals and attribute values are typically regarded as entities in this context. As an example, we may consider the following knowledge graph:

$$\begin{aligned} K = \{&(\mathsf{bob}, \mathsf{authorOf}, \mathsf{p}), (p, \mathsf{hasTopic}, \mathsf{knowledgeRepresentation}),\\ &(p, \mathsf{hasTopic}, \mathsf{artificialIntelligence}), (\mathsf{bob}, \mathsf{hasProperty}, \mathsf{expertInAI})\} \end{aligned}$$

Approaches for Knowledge graph embedding (KGE) learn a vector representation $\mathbf{e} \in \mathbb{R}^n$ for each entity e and a scoring function $\phi_r : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ for each relation type r, such that $\phi_r(\mathbf{e}, \mathbf{f})$ captures the plausibility of the triple (e, r, f), i.e. the plausibility that the relation r holds between the entities e and f [14, 75, 70, 69]. The vector $\mathbf{e}$ is called the *embedding* of entity e. The purpose of KGE is at least two-fold. First, it is hoped that this embedding will uncover some of the underlying semantic dependencies in the KG, and that as a result, we will be able to identify plausible triples that are missing from the given KG. Second, by encoding the information that is captured in the knowledge graph using vectors, it becomes easier to exploit this information in neural network models.

Figure 1 shows a vector encoding of the paper p and some of the considered subject areas. For this example, we assume that the dot product between p and a subject area indicates how relevant that subject area is to p, i.e. we have $\phi_{\mathsf{hasTopic}}(\mathbf{e}, \mathbf{f}) = \mathbf{e} \cdot \mathbf{f}$. Let us write $\mathbf{v_{ML}}$, $\mathbf{v_{AI}}$, $\mathbf{v_{NLP}}$ and $\mathbf{v_{KR}}$ for the vector representations of the different subject areas, and $\mathbf{p}$ for the representation of p. According to this embedding, we have $\mathbf{p} \cdot \mathbf{v_{ML}} \approx \mathbf{p} \cdot \mathbf{v_{NLP}} > \mathbf{p} \cdot \mathbf{v_{KR}}$, which captures the knowledge that p is more closely related to machine learning and natural language processing than to knowledge representation. Moreover, note how the norm of $\mathbf{v_{AI}}$ is larger than the norms of $\mathbf{v_{ML}}$, $\mathbf{v_{NLP}}$ and $\mathbf{v_{KR}}$. This intuitively captures the knowledge that the term artificial intelligence is broader in meaning. For instance, we can encode the knowledge that machine learning is a sub-discipline of AI by ensuring that for every vector $\mathbf{x} \in \mathbb{R}^2$ it holds that:

$$\mathbf{v_{ML}} \cdot \mathbf{x} < \mathbf{v_{AI}} \cdot \mathbf{x}$$

Note that in this example, we have only focused on one relation (i.e. hasTopic). In general, we can model multiple relations by using higher-dimensional vectors, together with scoring functions that depend on relation-specific parameters (see Section 2.3 for more details). When it comes to modelling conceptual knowledge, an important advantage of KGE is that it naturally supports inductive inferences. Moreover, such representations are better suited for modelling graded notions such as similarity than symbolic representations. However, the extent to which "rule-like" knowledge can be captured is limited. As we saw in the aforementioned example, we can model the fact that one concept is subsumed by another, but it is not clear how more complex rules can be encoded using vector space embeddings. Moreover, KGE models lack the transparency of symbolic representations, which makes it harder to generate meaningful explanations or to update representations (e.g. to correct mistakes, add new knowledge, or take account of changes in the world).

It is thus clear that ontologies and vector space embeddings have complementary strengths and weaknesses when it comes to modelling conceptual knowledge. Accordingly, various authors have proposed strategies for combining these two paradigms. For instance, rules are sometimes used to regularise neural networks [24, 74, 43], to generate supplementary training data [7], or to determine the structure of a neural network [59, 67]. Other approaches use rules

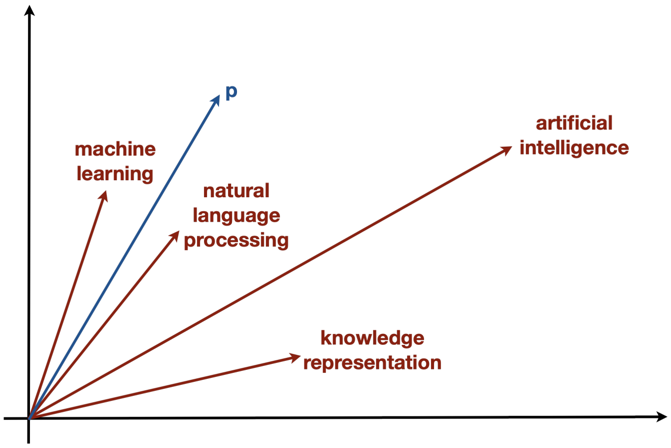


Figure 1 Illustration of a simple knowledge graph embedding, in which the dot product between p and a subject area indicates how relevant that subject area is to p.

to reason about the predictions of neural networks [44, 77], or treat rules as latent variables which are inferred by a neural network [56]. Note, however, how in the aforementioned research lines, rules and vector representation are treated as fundamentally distinct. Rules are either used as a supervision signal for learning neural networks (or vector space embeddings) or they are used for reasoning in a way that is largely decoupled from the neural networks or vector space embeddings themselves. Another observation is that rules essentially play a supportive role, to help overcome the limitations of some neural network model.

The first question we address in this paper is whether a tighter integration of rules and vector representations is possible. The main idea is to view symbolic knowledge as qualitative constraints on some underlying geometric model. This idea was developed in the 1990s by Gärdenfors in his theory of conceptual spaces [27]. The key characteristic of conceptual spaces is that concepts are represented as regions, rather than vectors. A rule $A(x) \leftarrow B(x), C(x)$ can then be viewed as the constraint that the intersection of the regions representing B and C should be included in the region representing A. While the theory of conceptual spaces offers an elegant solution to the question of how symbolic and vector representation could be integrated, it has two limitations that have hampered its adoption within AI:

- In practice, it is often difficult to learn region-based representations of concepts from data.
- Conceptual space representations cannot be used for modelling relational knowledge, e.g. rules involving binary predicates.

These two limitations, and strategies for addressing them, are discussed in Sections 3 and 4.

The second question we discuss is how vector space representations can be used in a supportive role, to help overcome some of the limitations of symbolic reasoning with ontologies. Here, the starting point is that some of the aforementioned shortcomings can be alleviated within a purely symbolic setting, for instance by relying on default reasoning [42, 20, 32], analogical reasoning [31, 54, 61], or qualitative versions of similarity based reasoning [65, 63]. The main problem with implementing such strategies in practice comes from the fact that they often rely on types of background knowledge which is not usually available in symbolic form (e.g. qualitative similarity relations). However, in some cases, this background knowledge can be obtained from vector space embeddings. In this case, we still have a loose integration between vector representations and rules, but rather than trying to improve neural network learning, as in the works described above, now the focus is on making symbolic reasoning

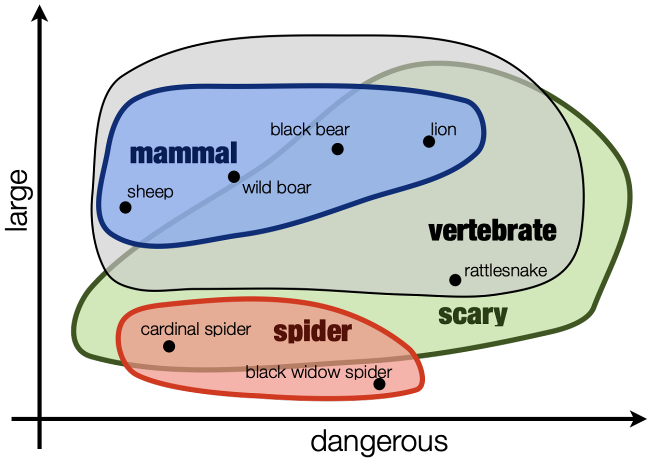


Figure 2 Illustration of a conceptual space of animals.

more flexible and adding some kind of inductive reasoning capability. For instance, in the setting from Example 1, if we know that the vector representation of planning is highly similar to the vector representation of knowledgeRepresentation, we can plausibly infer the following rule:

$$\mathsf{hasTopic}(X, \mathsf{artificialIntelligence}) \leftarrow \mathsf{hasTopic}(X, \mathsf{planning})$$

In Section 5, we discuss a number of strategies that build on this idea, focusing on how such plausible inferences can be integrated with standard deductive reasoning.

2 Background

In this section, we briefly introduce the main concepts that we will build on in the following sections. First, Section 2.1 discusses the theory of conceptual spaces. In Section 2.2 we then cover two standard formalisms for encoding ontological rules: existential rules and the $\mathcal{EL}$-family of description logics. Finally, Section 2.3 provides an introduction into Knowledge Graph Embedding.

2.1 Conceptual Spaces

Similar to vector-space embeddings, conceptual spaces [27] are geometric representations of the entities from a given domain of discourse. However, conceptual spaces differ from standard embeddings in two important ways: (i) properties and concepts are represented as regions and (ii) the dimensions of a conceptual space correspond to semantically meaningful features. These two differences enable conceptual spaces to act as an interface between neural representations, on the one hand, and symbolic knowledge, on the other hand. This is illustrated in Figure 2, which shows a conceptual space of animals. Specific animals are represented as points in this space. Concepts such as mammal and properties such as scary are represented as regions. The dimensions capture the ordinal features dangerous and large. In this representation, the region modelling mammal is included in the region modelling vertebrate, which intuitively captures the rule $\mathsf{vertebrate}(X) \leftarrow \mathsf{mammal}(X)$, i.e. all mammals are vertebrates. Note how this representation can also capture semantic dependencies that are harder to encode using rules, e.g. the fact that large spiders are scary.

While it is convenient to think about conceptual spaces as vector space embeddings with some added structure, conceptual spaces do not necessarily have the structure of a vector space. A conceptual space is defined from a set of *quality dimensions* $Q_1, ..., Q_n$. Each of these quality dimensions captures a primitive feature. As a standard example, the conceptual space of colours is built from three quality dimensions, representing hue, saturation and intensity. A distinction is made between *integral* and *separable* quality dimensions. Intuitively, separable quality dimensions are those that have a meaning on their own. For instance, *size* could be seen as a separable dimension. On the other hand, *hue* is not separable, as we cannot imagine the hue of a colour without also specifying its saturation and intensity. This distinction between integral and separable dimensions plays an important role in cognitive theories, as it affects how similarity is perceived. For instance, Euclidean distance is normally used when integral dimensions need to be combined, whereas Manhattan distance is used when separable dimensions need to be combined [49, 27]. Quality dimensions are partitioned into so-called *domains*, where dimensions that belong to the same domain are assumed to be integral, while dimensions from different domains are assumed to be separable. For instance, a conceptual space of physical objects may be composed of three domains: the colour domain (containing the hue, saturation and intensity quality dimensions), the size domain (containing only a single quality dimension) and the shape domain (containing several dimensions).

We can view domains as Cartesian products of quality dimensions. For instance, if D_i is composed of the quality dimensions $Q_1, ..., Q_k$ then the elements of D_i are tuples $(x_1, ..., x_k) \in Q_1 \times ... \times Q_k$. We can thus intuitively think of domains as vector spaces, although in general it is not required that domains satisfy the axioms of a vector space. An individual (e.g. a specific apple) is represented as an element $(x_1, ..., x_k)$ of a given domain, whereas we can think of properties (e.g. green) as regions. One of the central assumptions in the theory of conceptual spaces is that each *natural* property corresponds to a *convex* region in some domain. A *concept* is characterised in terms of a set of natural properties, along with information about how these properties are correlated. To define this notion of convexity, we have to assume that each domain D_i is equipped with a ternary betweenness relation $\mathsf{bet}_i \subseteq D_i \times D_i \times D_i$. A region $R \subseteq D_i$ is then said to be *convex* iff

$$\forall a, b, c \in D_i \,.\, a \in D_i \wedge c \in D_i \wedge \mathsf{bet}_i(a, b, c) \Rightarrow b \in D_i$$

In this paper, our focus will be on learning conceptual spaces from data. In this case, we will only consider domains that correspond to Euclidean spaces, where the notion of convexity can be interpreted in the standard way. Our focus will be on (i) learning region based representations of properties and concepts (ii) identifying quality-dimensions and (iii) grouping these quality-dimensions into domains.

2.2 Ontology Languages

We next look at two of the most popular Horn-like formalisms to encode ontologies, namely existential rules [10, 35] and the $\mathcal{EL}$-family of description logics [8]. Informally, an existential rule is a datalog-like rule (i.e. a logic programming rule of the kind we used in Example 1) with existentially quantified variables in the head, i.e. it extends traditional datalog with *value invention*. As a consequence, existential rules describe not only constraints on the currently available knowledge or data, but also *intensional* knowledge about the domain of discourse. Likewise, the $\mathcal{EL}$-family of description logics can be used for modelling intentional knowledge. In fact, some expressive members of the $\mathcal{EL}$-family are restrictions of existential rules to unary and binary relations.

Existential Rules

Syntax. Let $\mathbf{C}, \mathbf{N}$ and $\mathbf{V}$ be infinite disjoint sets of *constants, (labelled) nulls* and *variables*, respectively. A *term* t is an element in $\mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$; an *atom* α is an expression of the form $R(t_1, \ldots, t_n)$, where R is a *relation name (or predicate)* with *arity* n and terms t_i. An *existential rule* σ is an expression of the form

$$\exists X_1, \ldots, X_j . H_1 \land \ldots \land H_k \leftarrow B_1 \land \ldots \land B_n, \tag{1}$$

where $n \geq 0$, $k \geq 1$, $B_1, \ldots B_n$ and $H_1, \ldots, H_k$ are atoms with terms in $\mathbf{C} \cup \mathbf{V}$, and $X_1, ..., X_j \in \mathbf{V}$. From here on, we assume w.l.o.g. that $k = 1$ [21] and we omit the subscript in H_1. We further allow *negative constraints* (also simply called *constraints*), which are expressions of the form $\bot \leftarrow B_1 \land \ldots \land B_n$, where the B_is are as above and $\bot$ denotes the truth constant *false*. A finite set Σ of existential rules and constraints is called an *ontology*. Let $\mathfrak{R}$ be a set of relation names. A *database* D is a finite set of *facts* over $\mathfrak{R}$, i.e. atoms with terms in $\mathbf{C}$. A *knowledge base (KB)* $\mathcal{K}$ is a pair (Σ, D) with Σ an ontology and D a database.

Semantics. An *interpretation* $\mathcal{I}$ *over* $\mathfrak{R}$ is a (possibly infinite) set of atoms over $\mathfrak{R}$ with terms in $\mathbf{C} \cup \mathbf{N}$. An interpretation $\mathcal{I}$ is a *model of* Σ if it satisfies all rules and constraints: $\{B_1, \ldots, B_n\} \subseteq \mathcal{I}$ implies $\{H\} \subseteq \mathcal{I}$ for every existential rule σ in Σ, where existential variables can be witnessed by constants or labelled nulls, and $\{B_1, \ldots, B_n\} \not\subseteq \mathcal{I}$ for all constraints defined as above in Σ; it is a *model of a database* D if $D \subseteq \mathcal{I}$; it is a model of a KB $\mathcal{K} = (\Sigma, D)$, written $\mathcal{I} \models \mathcal{K}$, if it is a model of Σ and D. We say that a KB $\mathcal{K}$ is satisfiable if it has a model. We refer to elements in $\mathbf{C} \cup \mathbf{N}$ simply as *objects*, call atoms α containing only objects as terms *ground*, and denote with $\mathfrak{O}(\mathcal{I})$ the set of all objects occurring in $\mathcal{I}$.

▶ **Example 2.** Let $D = \{\mathsf{wife}(\mathsf{anna}), \mathsf{wife}(\mathsf{marie})\}$ be a database and Σ an ontology composed by the following existential rules:

$$\mathsf{husband}(Y) \leftarrow \mathsf{wife}(X) \land \mathsf{married}(X, Y) \tag{2}$$

$$\exists X . \mathsf{husband}(X) \land \mathsf{married}(X, Y) \leftarrow \mathsf{wife}(Y) \tag{3}$$

$$\bot \leftarrow \mathsf{husband}(X) \land \mathsf{wife}(X) \tag{4}$$

Then, an example of a model of $\mathcal{K} = (\Sigma, D)$ is the set of atoms

$$D \cup \{\mathsf{husband}(o_1), \mathsf{husband}(o_2), \mathsf{married}(o_1, \mathsf{anna}), \mathsf{married}(o_2, \mathsf{marie})\}$$

where o_i are labelled nulls. Note that e.g. $\{\mathsf{married}(\mathsf{anna}, \mathsf{marie}), \mathsf{husband}(\mathsf{marie})\}$ is not included in any model of $\mathcal{K}$ due to (4).

$\mathcal{EL}$-family

We introduce some basic notions about description logics, focusing on $\mathcal{EL}_\bot$, one of the most commonly used logics from the $\mathcal{EL}$-family. The interested reader can find more details on description logics in [9].

Syntax. Consider countably infinite but disjoint sets of *concept names* $\mathsf{N_C}$ and *role names* $\mathsf{N_R}$. These concept and role names are combined to $\mathcal{EL}_\bot$ *concepts*, in accordance with the following grammar, where $A \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$:

$$C, D := \top \mid \bot \mid A \mid C \sqcap D \mid \exists r.C$$

For instance, $A \sqcap (\exists r.(B \sqcap C))$ is an example of a well-formed $\mathcal{EL}_\bot$ concept, assuming $A, B, C \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$. The fragment of $\mathcal{EL}_\bot$ in which $\bot$ is not used is known as $\mathcal{EL}$. An $\mathcal{EL}_\bot$ *TBox (ontology)* $\mathcal{T}$ is a finite set of *concept inclusions (CIs)* of the form $C \sqsubseteq D$, where C, D are $\mathcal{EL}_\bot$ concepts.

▶ **Example 3.** The ontology in Example 2 can be expressed using the following $\mathcal{EL}$ concept inclusions

$$\exists\mathsf{married}.\mathsf{Wife} \sqsubseteq \mathsf{Husband} \tag{5}$$

$$\mathsf{Wife} \sqsubseteq \exists\mathsf{married}.\mathsf{Husband} \tag{6}$$

$$\mathsf{Husband} \sqcap \mathsf{Wife} \sqsubseteq \bot \tag{7}$$

Semantics. The semantics of description logics are usually given in terms of first-order interpretations $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$. Such interpretations consist of a nonempty *domain* $\Delta^\mathcal{I}$ and an *interpretation function* $\cdot^\mathcal{I}$, which maps each concept name A to a subset $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and each role name r to a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The interpretation function $\cdot^\mathcal{I}$ is extended to complex concepts as follows:

$$(\top)^\mathcal{I} = \Delta^\mathcal{I}, \qquad (\bot^\mathcal{I}) = \emptyset \qquad (C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I},$$
$$(\exists r.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \exists d' \in C^\mathcal{I}, (d, d') \in r^\mathcal{I}\}.$$

We now introduce two classical reasoning tasks. An interpretation $\mathcal{I}$ *satisfies* a concept inclusion $C \sqsubseteq D$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$; it is a model of a concept C if $C^\mathcal{I} \neq \emptyset$; it is a *model* of a TBox $\mathcal{T}$ if it satisfies all CIs in $\mathcal{T}$. A concept C *subsumes a concept D relative to a TBox $\mathcal{T}$* if every model $\mathcal{I}$ of $\mathcal{T}$ satisfies $C \sqsubseteq D$. We denote this by writing $\mathcal{T} \models C \sqsubseteq D$. A concept C is *satisfiable w.r.t. $\mathcal{T}$* if there is a common model of C and $\mathcal{T}$.

2.3 Knowledge Graph Embedding

Let a set of entities $\mathcal{E}$ and a set of binary relations $\mathcal{R}$ be given. A knowledge graph (KG) is a subset of $\mathcal{E} \times \mathcal{R} \times \mathcal{E}$. In other words, a knowledge graph is a set of triples of the form (e, r, f). These triples encode the fact that the relation r holds between the entities e and f. For instance, we may have a triple such as $(\mathsf{london}, \mathsf{capitalOf}, \mathsf{uk})$, encoding that London is the capital of the UK. A knowledge graph is thus essentially a set of relational facts, with the limitation that all relations are binary. Note, however, that the set of entities $\mathcal{E}$ typically includes both individuals (i.e. constants referring to specific objects, e.g. london) and attribute values, which allow us to encode unary predicates. For instance, the relational fact $\mathsf{scary}(\mathsf{lion})$ Could be encoded as the KG triple $(\mathsf{lion}, \mathsf{hasAttribute}, \mathsf{scary})$.

The aim of Knowledge Graph Embedding (KGE) is to learn a vector encoding $\mathbf{e} \in \mathbb{R}^n$ for each $e \in \mathcal{E}$ and a scoring function $\phi_r : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ for each $\in \mathcal{R}$. The vector $\mathbf{e}$ is usually referred to as the *embedding* of e. The scoring function is designed such that $\phi_r(\mathbf{e}, \mathbf{f})$ indicates how likely it is that (e, r, f) is a valid triple, i.e. that the relational fact $r(e, f)$ is true. We may assume, for instance, that for each $r \in \mathcal{R}$ we also have a threshold λ_r such that (e, r, f) is considered to be valid iff $\phi_r(\mathbf{e}, \mathbf{f}) \geq \lambda_r$. A comprehensive overview of knowledge graph embedding models is beyond the scope of this paper; please refer to [72, 60] for more complete introductions. To illustrate the main concepts, we discuss a number of popular models. TransE [14] was one of the first KGE models. Relations in this model are viewed as translations. In particular, each relation $r \in \mathcal{R}$ is represented by a vector $\mathbf{r} \in \mathbb{R}^n$. The corresponding scoring function ϕ_r is given by:

$$\phi_r(e, f) = -d(\mathbf{e} + \mathbf{r}, \mathbf{f})$$

with d either Euclidean or Manhattan distance. Another popular choice is to use a bilinear scoring function. In this case, r is parametrised by a matrix $\mathbf{M_r}$ and we have:

$$\phi_r(e, f) = \mathbf{e}^T \, \mathbf{M_r} \, \mathbf{f}$$

Different models differ in which constraints they put on the matrix $\mathbf{M_r}$. For instance, in the RESCAL model [47] this matrix is unconstrained, whereas DistMult [76] only allows diagonal matrices. In recent years, several authors have focused on designing models that make it easier to capture certain relational structures. For instance, embeddings based on hyperbolic geometry have been used to make it easier to model hierarchical structures, such as *is-a* and *part-of* hierarchies [48]. Region-based models, e.g. representing entities as boxes or cones, have been used for their ability to model both hierarchies and intersections [1, 52, 79]. In [68] a model is proposed in which relations are viewed as rotations, to facilitate modelling relational composition, as well as properties such as symmetry. It should be noted, however, that while these models can capture certain relational dependencies to some extent, in most models there is no explicit link between a given knowledge graph embedding and the relational dependencies it captures. Moreover, relatively little is known about which kinds of dependencies different models are capable of capturing (or, more generally, which sets of dependencies can be jointly captured). Of course, this first requires us to formalise what it means for an embedding to capture a relational dependency. We will return to this question in Section 4.

3 Learning Conceptual Space Representations

If we want to use conceptual spaces as an interface between symbolic ontologies and vectors space embeddings, a crucial question is whether it is possible to learn conceptual spaces from data. What matters in this context is (i) whether we can learn region-based representations of concepts and (ii) whether we can learn vector representations in which dimensions are meaningful and organised into domains. These two issues are discussed in Sections 3.1 and 3.2 respectively.

3.1 Modelling Concepts as Regions

Learning Gaussian Representations. In learned vector space embeddings, the objects from some domain of interest are represented as points or vectors, as in conceptual spaces. Most embedding models do not learn region-based representations of concepts. However, if we have access to a number of instances $c_1, ..., c_n$ of a given concept C, we can aim to learn a region-based representation of C from embeddings of these instances. The potential of this strategy stems from the fact that in many embedding models, these instances can be expected to appear clustered together in the vector space. To illustrate this, consider Figure 3, which shows the first two principal components of a 300-dimensional embedding of BabelNet concepts [46] using NASARI vectors[2], which have been learned from Wikipedia and are linked to BabelNet [22]. In the figure, the red points correspond to entities that are instances of the concept Artist, while the blue points correspond to entities that are instances of Painter. For instance, the embeddings of *Edouard Manet*, *Vanessa Bell* and *Claude Monet* appear close to the centre of the blue point cloud. As can be seen, painters appear as a distinct cluster in this vector space embedding.

[2] Downloaded from `http://lcl.uniroma1.it/nasari/`.

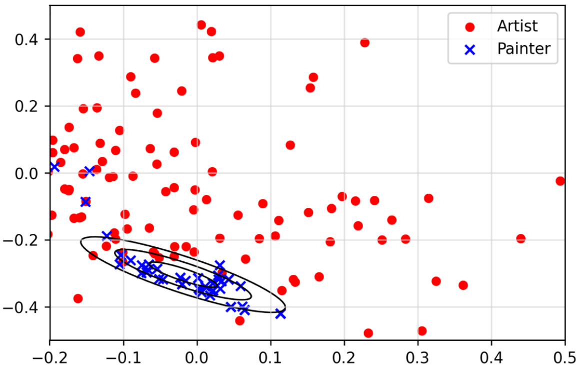


Figure 3 First two principal components of a vector space embedding of BabelNet entities, where blue points correspond to instances of the concept Artist and red points correspond to instances of the concept Painter, according to Wikidata.

When attempting to learn a region-based concept representation, we are faced with two challenges: (i) we typically only have access to positive examples and (ii) the number of available instances is often much smaller than the number of dimensions in the vector space. This means that we inevitably have to make some simplifying assumptions to make learning possible. A natural choice is to represent concepts as Gaussians. This has the advantage that concept representations can be learned in a principled way, as the problem of estimating Gaussians from observations, either with or without prior knowledge, has been well-studied. Representing concepts using probability distributions, rather than hard regions, also fits well with the view that concept boundaries tend to be fuzzy and ill-defined more often than not. Note that in neural models, concepts are typically represented as vectors, with concept membership determined in terms of dot products, e.g. $\sigma(\mathbf{e} \cdot \mathbf{c})$ is often used to estimate the probability that the entity e (with embedding $\mathbf{e}$) is an instance of concept C (with embedding $\mathbf{c}$), with σ the sigmoid function. This choice effectively means that concepts are represented as spherical regions in the vector space. When using Gaussians, we relax this modelling choice, allowing concepts to be represented using ellipsoidal regions instead.

To deal with the (typically) small number of instances that are available for learning a concept, [17] only considered Gaussians with diagonal covariance matrices. In this case, the problem simplifies to learning a number of univariate Gaussians, i.e. one per dimension. Moreover, a Bayesian formulation with a flat prior was used for estimating the Gaussians. As a consequence, concepts are actually represented using Student t-distributions. The practical implication is that slightly wider ellipsoidal regions are learned than those that would be obtained when using maximum likelihood estimates. Some contours of the learned distribution for the concept Painter are shown in Figure 3.

Bayesian learning with prior knowledge. As mentioned above, [17] used a Bayesian formulation for learning Gaussian concept representations. While a flat (i.e. non-informative) prior was used in that paper, the same formulation can be used with informative priors, which offers a natural strategy for incorporating prior knowledge about the concept C being modelled. Such prior knowledge is particularly important when the number of available instances of C is very small (or, in an extreme case, when no instances of C are given at all). This idea was developed in [18], where two sources of prior knowledge were used: ontologies and vector space embeddings of the concept names. In both cases, the prior knowledge allows us to relate the target concept C to other concepts. However, in practice we typically do not yet have a representation of these other concepts, i.e. we are trying to jointly learn a representation of all concepts of interest. This creates circular dependencies, e.g. the representation of concept A provides us with a prior on the representation of concept B, but the representation of concept B also provides us with a prior on the representation of A. This can be addressed using Gibbs sampling; we refer to [18] for the details.

Priors on Mean. Suppose we have concept inclusions of the form $(C \sqsubseteq D_1)$,...,$(C \sqsubseteq D_k)$, and suppose we have a Gaussian representation of the concepts $D_1, ..., D_k$. Then we can induce a prior on the mean of the Gaussian representing C based on the idea that the mean of C should have a high probability in the Gaussians modelling $D_1, ..., D_k$. This can be achieved efficiently by taking advantage of the fact that the product of k Gaussians is proportional to another Gaussian. In addition to ontologies, we can also use vector space embeddings of the (names of the) concepts $C, D_1, ..., D_k$. Specifically, [18] proposed a strategy based on the view that there should be a fixed vector offset between the embedding of a concept C and the mean of the Gaussian that represents it.

Priors on Variance. To obtain a prior on the variance of the Gaussian representing C, we take the view that this variance should be similar to that of the concepts that are most similar to C. To find such concepts, we could take the siblings of C in an ontology, the concepts whose vector space embedding is most similar to the embedding of C, or we could use a hybrid strategy where we select the siblings whose embedding is most similar to that of C. We again refer to [18] for details.

Exploiting contrast sets. A common strategy for learning conceptual space representations is to associate each concept with a single point, which intuitively represents its prototype [30]. The region representing a given concept C then consists of all points that are closer to the prototype of C than to the prototype of any other concept, i.e. concept regions are obtained as the Voronoi tessellation of a set of prototype points. This strategy is appealing, because it allows us to learn concept regions with a much wider extension than when learning Gaussians, especially in cases where we only have a few instances per concept. The main idea is illustrated in Figure 4, where we are interested in learning a region for the concept C. When using Gaussians, we would end up with ellipsoidal regions (contours) similar to the ones displayed in the figure. As a result, most points of the space are not assigned to any of the concepts. In contrast, if we construct prototypes by averaging the embeddings of the instances of a concept, and compute the resulting Voronoi tessellation, we essentially carve up the space, as also illustrated in the figure. To see why this can be beneficial in practice, Figure 5 shows the vector representations of the instances of three concepts: Songbook, Brochure and Guidebook. Now consider the left-most test instance of Songbook. If we are only given the training instances of this concept, this test instance is unlikely to be covered by the resulting representation. In contrast, if we instead attempt to carve up the space into regions corresponding to Songbook, Brochure and Guidebook, then this test instance would be classified correctly. The problem with implementing the aforementioned idea is that it only works if we are given a set of concepts that form a *contrast set* [33], i.e. a set of mutually exclusive natural categories that exhaustively cover some sub-domain. For example, the set of all common color names, the set $\{\mathsf{Fruit}, \mathsf{Vegetable}\}$ and the set $\{\mathsf{NLP}, \mathsf{IR}, \mathsf{ML}\}$ can intuitively be thought of as contrast sets. We say that two concepts are conceptual neighbours if they belong to the same contrast set and compete for coverage (i.e. are adjacent in the resulting Voronoi tessellation).

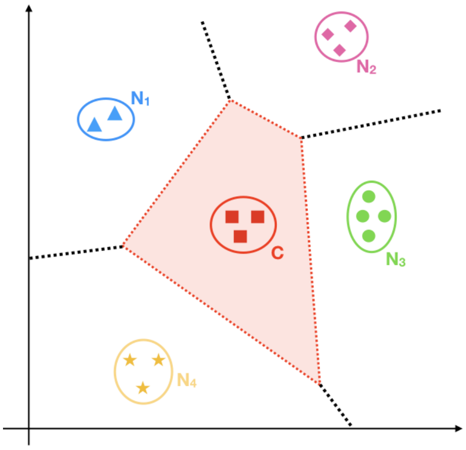


Figure 4 Estimating concept regions based on conceptual neighbourhood.

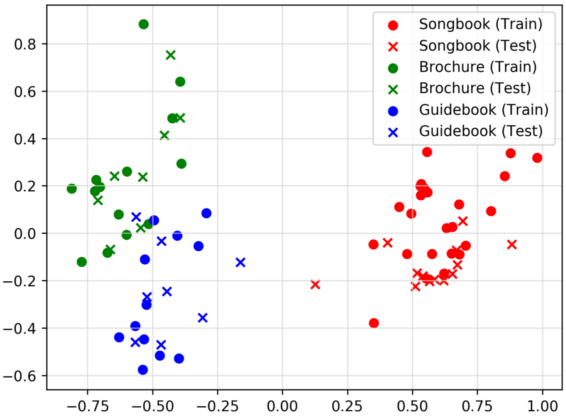


Figure 5 Instances of three BabelNet categories which intuitively can be seen as conceptual neighbors. The figure shows the first two principal components of the NASARI vectors.

Existing ontologies do not typically describe contrast sets or conceptual neighbourhood. To deal with this, [16] introduced a strategy for learning conceptual neighbourhood from data, i.e. for discovering pairs of concepts that are conceptual neighbours. Note that they focus on conceptual neighbourhood rather than contrast sets, as the need for contrast sets to be exhaustive is difficult to guarantee. The method then relies on the simplifying assumption that the target concept C, along with its known conceptual neighbours $N_1, ..., N_k$ forms a contrast set. To represent the concept C, first a Gaussian is learned by pooling the instances of $C, N_1, ..., N_k$ together. The ellipsoidal contours of this Gaussian are then carved up into sub-regions for $C, N_1, ..., N_k$ by learning logistic regression classifiers. Specifically, the region representing C is obtained by training logistic regression classifiers that separate the instances of C and N_i, for each $i \in \{1, ..., k\}$. To learn conceptual neighbourhood from data, the first step of the strategy from [16] consists in generating weakly supervised training examples. To this end, they start with two concepts A and B that are siblings in a given taxonomy (i.e. concepts that have the same parent) and for which a sufficiently large number of instances is given. They then compare the performance of the following two types of concept representations:

Table 1 Selected examples of siblings A–B which are predicted to be conceptual neighbours with high and medium confidence.

High confidence	Medium confidence
Actor – Comedian	Cruise ship – Ocean liner
Journal – Newspaper	Synagogue – Temple
Club – Company	Mountain range – Ridge
Novel – Short story	Child – Man
Tutor – Professor	Monastery – Palace
Museum – Public aquarium	Fairy tale – Short story
Lake – River	Guitarist – Harpsichordist

1. Learn a Gaussian representation of A and B from their given instances.
2. Learn a Gaussian representation from the combined instances of A and B, and then split the resulting region by training a logistic regression classifier that separates A-instances from B-instances.

If the second representations perform (much) better at classifying held-out instances, we can assume that A and B are conceptual neighbours. If the second representations perform much worse, then we can assume that A and B are not conceptual neighbours. In case the performance is similar, then the pair A, B is disregarded when constructing the weakly labelled training set. Table 1 shows some examples of pairs of concepts A, B that were predicted to be conceptual neighbours using this process. Given the resulting training set, we can then train a standard text classifier on sentences that mention both A and B from some text corpus. Consider, for instance, the concepts *Hamlet* and *Village*, and the following sentence [3]:

In British geography, a hamlet is considered smaller than a village and ...

The sentence suggests that *hamlet* and *village* are conceptual neighbors as it makes clear that these concepts are closely related but different. Once a classifier is trained, based on the weakly supervised training set, we can then apply it to other concepts. To learn the representation of a given target concept C (e.g. a concept with only few known instances), we can then use the text classifier to identify which of its siblings, in a given taxonomy, are most likely to be conceptual neighbours, and determine the representation of C accordingly. Tables 2 and 3 show some examples of the top conceptual neighbor predicted by the text classifier, for different target concepts. In particular, Table 3 shows examples where the resulting concept representations (i.e. the representations of the target concepts obtained by exploiting the predicted conceptual neighbourhood) were of high quality, as measured in terms of F1 score for held-out entities. Similarly, Table 2 shows examples where the resulting concept representations were of low quality. As can be seen, the predicted conceptual neighbours in Table 3 are clearly more meaningful than the predicted neighbours in Table 2. This illustrates how the quality of the concept representations is closely linked to our ability to find meaningful conceptual neighbours. Overall, the experiments in [16] showed that using predicted conceptual neighbourhood, on average, led to much better concept representations than when estimating Gaussians from the known instances of the target concept.

[3] `https://en.wikipedia.org/wiki/Hamlet_(place)`

Table 2 Top conceptual neighbors selected for categories associated with a low F1 score.

Concept	Top neighbor	F1
Bachelor's degree	Undergraduate degree	34
Episodic video game	Multiplayer gamer	34
501(c) organization	Not-for-profit arts organization	29
Heavy bomber	Triplane	41
Ministry	United States government	33

Table 3 Top conceptual neighbors selected for categories associated with a high F1 score.

Concept	Top neighbor	F1
Amphitheater	Velodrome	67
Proxy server	Application server	61
Ketch	Cutter	74
Quintet	Brass band	67
Sand dune	Drumlin	71

3.2 Learning Quality Dimensions

The dimensions of learned vector spaces do not normally correspond to semantically meaningful properties. This is an important difference with conceptual spaces, which severely limits the interpretability of learned vector space representations. In this section, we review work that has focused on mitigating this issue, by identifying interpretable directions in learned vector spaces. These interpretable directions can then play the role of quality dimensions. This is illustrated in Figure 6, which shows a two-dimensional projection of an embedding of movies from [25]. Along with the embedding of the movies themselves, the figure also shows two directions that have been identified: one direction which ranks the movies from least to most *scary*, and another direction which ranks the movies from least to most *romantic*. Formally, we say that the direction of some vector $\mathbf{v}$ models a property P, such as scary, if for entities e_1 and e_2, with embeddings $\mathbf{e_1}$ and $\mathbf{e_2}$, we have $\mathbf{e_1} \cdot \mathbf{v} > \mathbf{e_2} \cdot \mathbf{v}$ if entity e_1 has the property P to a greater extent than entity e_2.

Identifying quality dimensions. Assume that a set of entities $\mathcal{E}$ is given, together with a vector space embedding $\mathbf{e} \in \mathbb{R}^n$ for each entity $e \in \mathcal{E}$. To find interpretable directions, [25] proposed a simple strategy which relies on the assumption that a text description D_e is available for each entity e. Let V be the set of all words (or common multi-word expressions such as “New York”) that appear in these descriptions D_e. For $v \in V$, we say that the word v is relevant for the entity e if v appears at least once in the description D_e. It was proposed in [25] to learn a linear classifier in the embedding space, for each $v \in V$, separating the entities for which v is relevant from those for which this is not the case. If this classifier is able to separate these entities well, the assumption is that the word v must be important, i.e. that it describes an aspect that is captured by the embedding space. In this case, the normal vector $\mathbf{v}$ of the hyperplane that was learned by the classifier is treated as a candidate direction. These candidate directions are then clustered, and the each cluster is treated as a quality dimension. This clustering step has the advantage that quality dimensions become easier to interpret, as we have a set of words to describe them, rather than a single word, and it ensures that different quality dimensions are sufficiently different. We refer to [2] for an extensive evaluation of the resulting quality dimensions. We illustrate the main findings with some examples. First, some of the clusters that are found closely correspond to the intuition of quality dimensions. For instance, the following clusters were found in [25], starting from a vector space embedding of movies:

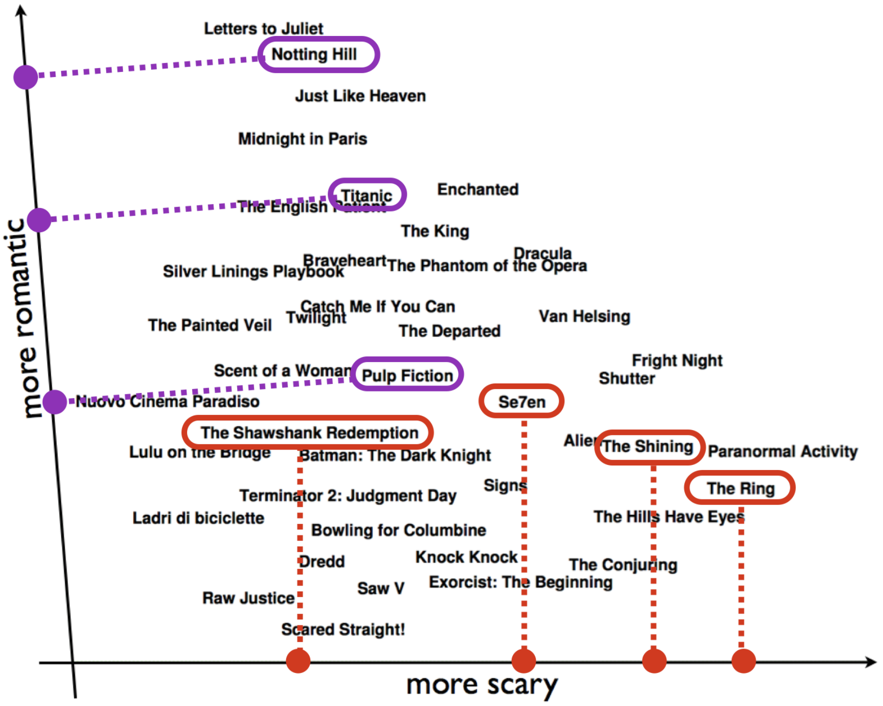


Figure 6 Interpretable directions within a vector space embedding of movies.

- touching, inspirational, warmth, dignity, sadness, heartwarming, ...
- clever, schemes, satire, smart, witty dialogue, ingenious, ...
- bizarre, odd, twisted, peculiar, lunacy, surrealism, obscure, ...
- predictable, forgettable, unoriginal, formulaic, implausible, contrived, ...
- tragic, anguish, sorrow, fatal, misery, bitter, heartbreaking, ...
- romantic, lovers, romance, the chemistry, kisses, true love, ...
- eerie, paranoid, spooky, impending doom, dread, ominous, ...
- scary, shivers, chills, creeps, frightening, the dark, goosebumps, ...
- cheesy, camp, corny, tacky, laughable, a guilty pleasure, ...
- hilarious, humorous, really funny, a very funny movie, amusing, ...
- wonderful, fabulous, a joy, gem, delighted, happy, perfect, great, ...

Arguably, all these directions correspond to clear and salient semantic attributes of movies. On the other hand, many other clusters rather corresponded to movie themes, e.g.:

- horror movies, zombie, much gore, slashers, vampires, scary monsters, ...
- killer, stabbings, a psychopath, serial killer, ...
- supernatural, a witch, ghost stories, mysticism, a demon, the afterlife, ...
- scientist, experiment, the virus, radiation, the mad scientist, ...
- criminal, the mafia, robbers, parole, the thieves, the mastermind, ...

While these directions express semantically meaningful properties, it would be more natural to represent such properties as regions than as quality dimensions. The fact that such thematic properties cannot be distinguished from the semantic attributes mentioned above is clearly a limitation of the method from [25]. In [2], it was found that the nature of the clusters, i.e. whether they intuitively correspond to quality dimensions rather than thematic properties, to some extent depends on the scoring function that is used for evaluating the linear classifiers. However, regardless of the scoring function that is used, a mixture of different types of properties is found. One possible solution could be to require that clusters which correspond to quality dimensions should contain a sufficient proportion of adjectives, as clusters consisting mostly of nouns are more likely to be thematic properties. On the other hand, it is not clear that having thematic "quality dimensions" is necessarily problematic. While it makes the resulting representation different from a conceptual space, it still allows us to "disentangle" the vector representation into different aspects (e.g. genre, sentiment, emotion). Furthermore, a cluster of terms related to horror movies could still be viewed as a quality dimension if we view it as ranking movies based on how "horror-like" they are.

A number of improvements to the basic method from [25] have been explored. In [3] a fine-tuning strategy is introduced, which modifies the initial vector space based on the discovered quality dimensions, while [6] suggests to learn quality dimensions in a hierarchical fashion, with the top-level dimensions essentially partitioning the vector space into thematic domains, and the lower-level dimensions intuitively corresponding to quality dimensions within each of these thematic domains. In terms of how the resulting quality dimensions could be useful, the main focus has so far been on their ability to support interpretable classifiers, with [25] introducing a rule based classifier, which compares entities with training examples along a small number of quality dimensions, and [3, 6] using the quality dimensions as features for low-depth decision trees.

Organising quality dimensions into domains. The quality dimensions of a conceptual space are organised into domains. Accordingly, as we have seen in the previous section, the quality dimensions that can be identified in learned vector spaces also intuitively belong to different kinds. It would be of interest to group quality dimensions of the same kind together, to learn a structure which is akin to conceptual space domains. For instance, in the movies domain, we could imagine one group of quality dimensions about the emotion a movie evokes, as well as groups about the genre, the cinematographic style, etc. We will refer to these groups of learned quality dimensions as *facets*, rather than domains, to avoid confusion (e.g. domain can also refer to the domain-of-discourse, such as movies, or to the domain of a description logic interpretation) and to highlight the fact that there are still important differences between these facets and conceptual space domains. In addition to grouping quality dimensions that are concerned with the same aspect of meaning, we also want to learn a corresponding lower-dimensional vector space for each facet. In other words, the central aim is to decompose the given vector space into a number of lower-dimensional spaces, each of which captures a different aspect of meaning.

Note that we cannot learn these facets by simply clustering the quality dimensions. For instance, *thriller* and *scary* may be represented by similar directions in the vector space, but they should be assigned to different facets. In contrast, *romance* and *horror* would be represented by dissimilar directions but nonetheless belong to the same facet. The key solution, which was developed in [5] and [4], is to rely on word embeddings to identify words that describe properties of the same kind. For instance, the word embeddings of different movie genres tend to be similar, because such words tend to appear in similar contexts. In the same way, different adjectives describing emotions tend to be represented using similar word vectors. This suggests a simple strategy for learning facets: (i) cluster the word vectors of the words associated with the quality dimensions that were identified in the given vector space; and (ii) represent the facet by the vector space that is spanned by quality dimensions that are assigned to it. Unfortunately, this strategy was found to perform poorly in [5]. The main reason is that in many areas there is one dominant facet, such as the genre in the case of movies. When applying the aforementioned strategy, what happens is that each of the resulting facet-specific vector spaces mostly models the dominant facet. To address this issue, [5] proposed an iterative strategy, in which the dominant facet is first identified and then explicitly disregarded when determining the second facet, etc. Another practical challenge is that the overall method is computationally demanding, especially the fact that a linear classifier has to be learned for each word from the vocabulary, to identify the interpretable directions (in the overall space and in each of the lower-dimensional facet-specific spaces). To address this issue, [4] introduced a model that directly learns facet-specific vector spaces from bag-of-words representations of the entities, using a mixture-of-experts model to generalize the GloVe [53] word embedding model. Using this approach, facet-specific vector spaces can be learned much more efficiently, and moreover the resulting embeddings tend to be of a higher quality. The main limitation, however, is that this model assumes that suitable vector spaces can be learned from bag-of-words representations (rather than being agnostic to how the initial vector space embedding is learned) and that GloVe is a suitable embedding model for learning these vector spaces.

The resulting facet-specific embeddings can be used in a number of different ways. Perhaps the most immediate application of such representations is that they facilitate concept learning. For instance, suppose we want to represent each concept as a Gaussian. Furthermore, suppose that only one of the facet-specific vector spaces is relevant for modelling the considered concept. If we learn a Gaussian in each of the factor-specific vector spaces, we should end up with Gaussian with a large variance for the irrelevant facets, and a Gaussian with a much lower variance in the vector space corresponding to the relevant facet. This advantage of facet-specific vector spaces was empirically confirmed in [4]. Moreover, they found that even strategies that only rely on the resulting quality dimensions, e.g. learning low-depth decision trees, were benefiting from learning facet-specific vector spaces, as the lower-dimensional nature of each vector space acts as a regulariser.

4 Modelling Relations with Conceptual Spaces

Conceptual spaces act as an interface between vector space embeddings and symbolic knowledge. However, because conceptual spaces do not capture relational knowledge, they are essentially limited to capturing Horn rules with unary predicates. In this section, we explore whether the framework of conceptual spaces can be generalised to encode rules with binary and higher arity relations. We focus on the analysis presented in [37] but use a construction that is somewhat more intuitive than the one used in the latter paper. The main idea is to view a k-ary relation as a convex region in the Cartesian product of k conceptual spaces. For simplicity, in this section we will assume that conceptual spaces correspond to Euclidean spaces. Each individual a is then represented as a vector $\mathbf{a} \in \mathbb{R}^n$. A tuple $(a_1, ..., a_k)$ is represented as the concatenation of the vectors representing $a_1, ..., a_k$, i.e. $(a_1, ..., a_k)$ is represented as the $n \cdot k$-dimensional vector $\mathbf{a_1} \oplus ... \oplus \mathbf{a_k}$, where we write $\oplus$ for vector concatenation.

The main idea is illustrated in Figure 7. In this toy example, we assume that individuals are represented in a one-dimensional conceptual space. Unary predicates such as herbivore then correspond to intervals, while binary predicates such as eats correspond to convex regions in $\mathbb{R}^2$. In this figure, the tuple $(\mathsf{lion}, \mathsf{zebra})$ corresponds to a point in the region encoding the eats predicate. This captures the knowledge that lions eat zebras. Moreover, we can now also model dependencies between unary and binary predicates. For instance, the representation captures the following rule:

$$\mathsf{carnivore}(X) \leftarrow \mathsf{eats}(X, Y), \mathsf{animal}(Y)$$

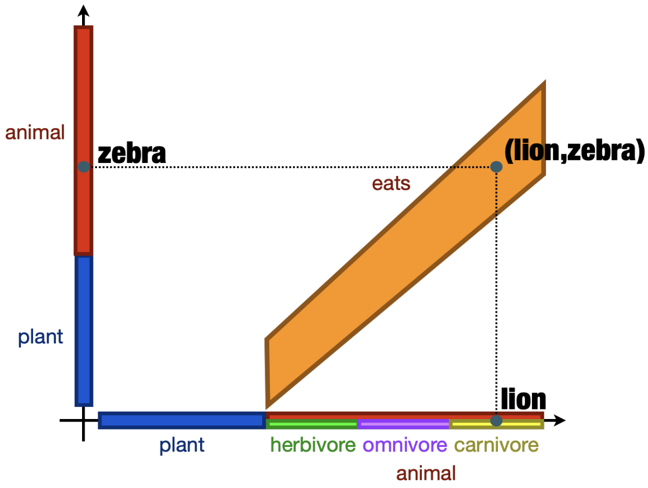


Figure 7 Illustration of a relational conceptual space.

This can be seen as follows. Consider a point $\mathbf{p} \in \mathbb{R}^2$ in the region representing eats, such that its projection on the Y-axis lies in the interval representing animal. For each such a point $\mathbf{p}$, it holds that its projection on the X-axis lies in the interval representing $\mathsf{carnivore}$. We can think of each point $\mathbf{p}$ as the representation of a possible instantiation of the tuple (X, Y). The aforementioned observation about $\mathbf{p}$ then corresponds to the view that every tuple satisfying the body of the rule also satisfies its head. In a similar way, we can also model rules with existential quantifiers, e.g.:

$$\exists Y.\mathsf{eats}(X, Y) \wedge \mathsf{animal}(Y) \leftarrow \mathsf{carnivore}(X)$$

To see why this rule is satisfied for the configuration depicted in Figure 7, consider a value $x \in \mathbb{R}$ which lies in the interval representing $\mathsf{carnivore}$. Then we can always find a coordinate $y \in \mathbb{R}$ such that the point $\mathbf{p} = (x, y)$ lies in the region for eats and such that y lies in the interval modelling animal. In Section 4.1 we discuss these intuitions in more detail. We also provide a characterisation about the kinds of relational rules that can be modelled using convex regions. Subsequently, in Section 4.2 we discuss the relationship with knowledge graph embedding models.

4.1 Geometric Models of Relational Rules

We consider geometric interpretations η, which map each individual a to a point $\eta(a) \in \mathbb{R}^n$ and each k-ary relation r to a convex region $\eta(r)$ in $\mathbb{R}^{k \cdot n}$. These geometric interpretations can intuitively be seen as defining a relational counterpart to conceptual spaces. We now discuss what it means for a geometric interpretation η to satisfy different kinds of relational knowledge. First, a relational fact of the form $r(a_1, ..., a_k)$ is satisfied if the representation of the tuple $(a_1, ..., a_k)$ lies in the region representing r, i.e.:

$$\eta(a_1) \oplus ... \oplus \eta(a_k) \in \eta(r)$$

Now we consider a basic relational entailment of the following form:

$$r(X_1, ..., X_k) \leftarrow s(X_1, ..., X_k)$$

This rule is satisfied if the region modelling s is included in the region modelling r, i.e. it corresponds to the following geometric constraint:

$$\eta(s) \subseteq \eta(r)$$

Conjunctions in the body of a rule can be modelled using intersections. For instance, consider the following rule:

$$r(X_1, ..., X_k) \leftarrow s(X_1, ..., X_k), t(X_1, ..., X_k) \tag{8}$$

The corresponding geometric constraint is as follows:

$$\eta(s) \cap \eta(t) \subseteq \eta(r)$$

This simple geometric characterisation only works because each relation is applied to the same tuple $(X_1, ..., X_k)$. To see how we can model more general rules, let us consider a rule of the following form:

$$r(X, Z) \leftarrow s(X, Y), t(Y, Z) \tag{9}$$

The main idea is to view this rule as a special case of (8). In particular, let us consider ternary relations r^*, s^* and t^* defined as follows: $r^*(X, Y, Z) \equiv r(X, Z)$, $s^*(X, Y, Z) \equiv s(X, Y)$ and $t^*(X, Y, Z) \equiv t(Y, Z)$. Then clearly (9) is equivalent to:

$$r^*(X, Y, Z) \leftarrow s^*(X, Y, Z), t^*(X, Y, Z)$$

whose geometric characterisation is given by $\eta(s^*) \cap \eta(t^*) \subseteq \eta(r^*)$. This is illustrated in Figure 8, where the relationship between the two-dimensional regions $\eta(r)$, $\eta(s)$, $\eta(t)$ and the three-dimensional regions $\eta(r^*)$, $\eta(s^*)$, $\eta(t^*)$ is shown. To explain how the regions $\eta(r^*)$, $\eta(s^*)$, $\eta(t^*)$ relate to $\eta(r)$, $\eta(s)$, $\eta(t)$ more formally, we have to introduce some notations. Let $I = \{i_1, ..., i_l\} \subseteq \{1, ..., k\}$ be a set of indices. For a point $(x_1, ..., x_{k \cdot n}) \in \mathbb{R}^{k \cdot n}$, we define its *restriction to* I as follows

$$(x_1, ..., x_{k \cdot n}) \downarrow I = \bigoplus_{i \in I} (x_{n \cdot i+1}, ..., x_{n \cdot i+n})$$

For instance if $n = 2$, $k = 4$ and $I = \{1, 4\}$ we have $(x_1, ..., x_8) \downarrow I = (x_1, x_2, x_7, x_8)$. In particular, note that when $(x_1, ..., x_{k \cdot n})$ is the representation of a tuple $(a_1, ..., a_k)$, and $(b_1, ..., b_l)$ is obtained from $(a_1, ..., a_k)$ be only keeping the arguments at the positions in I, then $\eta(b_1, ..., b_l) = \eta(a_1, ..., a_k) \downarrow I$. We define the notion of *cylindrical extension* as follows. Let R be a region in $\mathbb{R}^{l \cdot n}$ with $l < k$ and let $I = \{i_1, ..., i_l\} \subseteq \{1, ..., k\}$ Then we define:

$$\mathsf{ext}_I^k(R) = \{\mathbf{x} \in \mathbb{R}^{k \cdot n} \,|\, \mathbf{x} \downarrow I \in R\}$$

Let us now return to the problem of modelling the rule (9). We have $\eta(r^*) = \mathsf{ext}^3_{\{1,3\}}(\eta(r))$, $\eta(s^*) = \mathsf{ext}^3_{\{1,2\}}(\eta(s))$ and $\eta(t^*) = \mathsf{ext}^3_{\{2,3\}}(\eta(t))$. We thus find that the rule (9) corresponds to the following geometric constraint:

$$\mathsf{ext}^3_{\{1,2\}}\big(\eta(s)\big) \cap \mathsf{ext}^3_{\{2,3\}}\big(\eta(t)\big) \subseteq \mathsf{ext}^3_{\{1,3\}}\big(\eta(r)\big)$$

While the rule (9) only involves binary relations, clearly we can apply the same strategy to rules involving relations of other arities, and to rules with more than two atoms in the body.

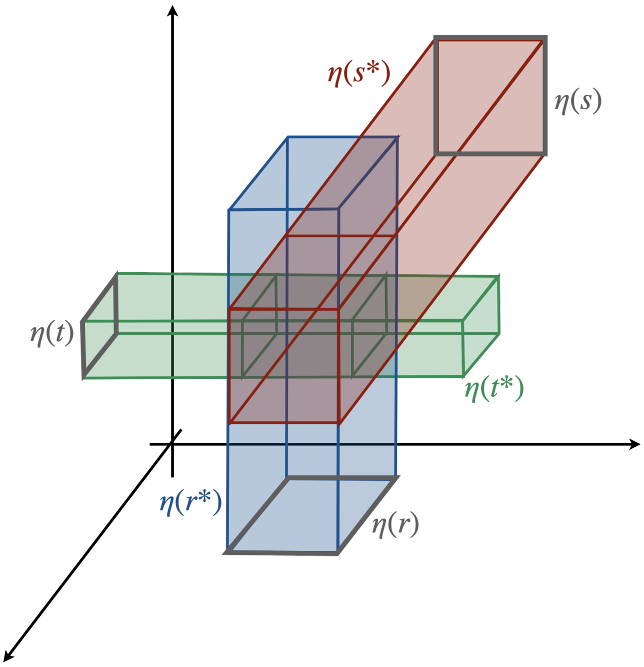


Figure 8 Illustration of the constraint $\eta(s^*) \cap \eta(t^*) \subseteq \eta(r^*)$.

Finally, we discuss how rules with existential quantifiers can be modelled. Let us consider the following example:

$$\exists Y \,.\, r(X,Y) \wedge s(Y,Z) \leftarrow t(X,Z) \tag{10}$$

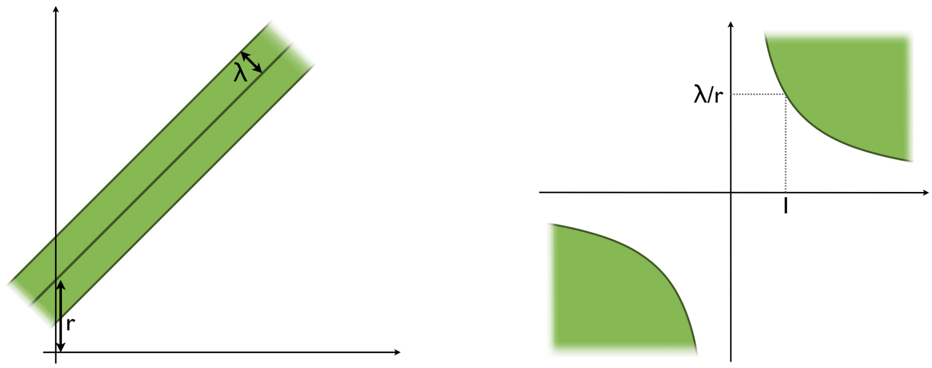
The key challenge is to characterise the region that models the head of this rule. Note that, as before, $r(X,Y) \wedge s(Y,Z)$ can be modelled by treating r and s as ternary relations. Relying again on the cylindrical extension, we find that this conjunction can be modelled as $\mathsf{ext}^3_{\{1,2\}}(\eta(r)) \cap \mathsf{ext}^3_{\{2,3\}}(\eta(s))$. To model the existential quantifier, we can then simply remove the coordinates pertaining to the variable Y. In other words, the rule (10) corresponds to the following geometric constraint:

$$\eta(t) \subseteq \left(\mathsf{ext}^3_{\{1,2\}}\big(\eta(r)\big) \cap \mathsf{ext}^3_{\{2,3\}}\big(\eta(s)\big)\right) \downarrow \{1,3\}$$

In this way, using a combination of cylindrical extensions and projections, any relational rule can be translated into a corresponding geometric constraint. It is worth pointing out that a similar treatment of rules was already proposed by Zadeh [78] in his theory of approximate reasoning. The main difference with the aforementioned approach is that relations in the latter case are modelled as fuzzy sets.

A central question is which kinds of rules can be faithfully[4] modelled in terms of the aforementioned geometric constraints. The answer depends on which kinds of regions we allow as the geometric interpretation $\eta(r)$ of a relation r. Without any restrictions, arbitrary sets of relational rules can be modelled correctly. However, in practice, it makes sense to require $\eta(r)$ to be convex. While the cognitive plausibility of this assumption is unclear, in

[4] Note that we use this notion of faithfulness informally here; we refer to [37] for a formal treatment of geometric models.



(a) TransE.

(b) DistMult.

Figure 9 Region based view of knowledge graph embedding models.

practice we can only hope to learn region-based representations in high-dimensional spaces by making drastic simplifying assumptions, as we also saw in Section 3. For this reason, most strategies for modelling relational knowledge end up learning convex representations; this will be discussed in more detail in Section 4.2. With this convexity assumption, however, clearly some sets of rules cannot be jointly modelled. For instance we cannot model the rule $\bot \leftarrow r_1(X,Y), r_2(X,Y)$, capturing that relations r_1 and r_2 are disjoint, together with the following facts: $r_1(a,a)$, $r_1(b,b)$, $r_2(a,b)$, $r_2(b,a)$. Indeed, if $\eta(r_1)$ and $\eta(r_2)$ are convex, from $\eta(a) \oplus \eta(a) \in \eta(r_1)$, $\eta(b) \oplus \eta(b) \in \eta(r_1)$, $\eta(a) \oplus \eta(b) \in \eta(r_2)$ and $\eta(b) \oplus \eta(a) \in \eta(r_2)$, we find:

$$\frac{(\eta(a)+\eta(b))}{2} \oplus \frac{(\eta(a)+\eta(b))}{2} \in \eta(r_1) \cap \eta(r_2)$$

and thus r_1 and r_2 are not disjoint in the geometric interpretation η. However, in [37] it was shown that many sets of relational rules can still be faithfully captured by geometric models. In particular, consider a relational rule of the following form:

$$\exists Y_1, ..., Y_r . H_1 \wedge ... \wedge H_s \leftarrow B_1, ..., B_t$$

where $H_1, ..., H_s, B_1, ..., B_t$ are atoms. We say that such a rule is quasi-chained, if every atom B_i appearing in the body shares at most 1 variable with the atoms $B_1, ..., B_{i-1}$. It can be shown that any set of quasi-chained rules with a finite model can be faithfully captured by a geometric model in which every relation is represented as a convex region [37]. Some open questions remain, however, including the following:

- Is there a larger fragment of existential rules that can be faithfully modelled using geometric interpretations with convex regions?
- Is there a way to relax the convexity assumption, such that arbitrary existential rules can be captured, while keeping representations simple enough to be learnable?

Finally, it should be noted that the restriction to arbitrary convex regions means that negation and disjunction cannot easily be modelled. Some authors have proposed geometric models that were specifically designed with such logical connectives in mind, including the use of axis aligned cones [52]. Recently, the ability of convex regions to model temporally attributed description logics has also been studied [19].

4.2 Link with Knowledge Graph Embedding

Thus far, we have not discussed how region-based representations of relations may be learned from data. In the last few years, there has been an increasing interest in region based representations, as already mentioned in Section 3.1. Most approaches, however, only use regions for modelling concepts, and deal with relations in an ad hoc way. For instance, the approach from [79] represents entities using cones, but uses a feed-forward neural network for modelling relations. Similarly, [52] propose a cone based model for embedding $\mathcal{ALC}$ ontologies, but they refrain from modelling roles in the same way. However, in [1] a knowledge graph embedding model is proposed in which relations are explicitly modelled as hyperboxes. More generally, many of the standard knowledge graph embedding models can be interpreted as region based models. In particular, for a relation r with scoring function f_r we can consider the following region:

$$\eta(r) = \{\mathbf{e} \oplus \mathbf{f} \,|\, f_r(e, f) \geq \lambda_r\}$$

with λ_r some threshold. Figure 9 illustrates how TransE and DistMult can be viewed as region-based models in this way. However, viewed as region based models, TransE and bilinear models such as DistMult are severely limited in which kinds of existential rules they can capture; we refer to [37] for more details.

5 Plausible Symbolic Reasoning using Vector Space Embeddings

Leaving aside the difficulties of tightly integrating geometric and symbolic representations, it is highly relevant for the development of robust AI systems to understand how symbolic approaches to AI can be made more flexible by equipping them with inductive capabilities, i.e. making it possible to infer likely concept inclusions (or rules) by using the knowledge of the ontology in combination with the additional background knowledge provided by vector representations. In other words, one would like symbolic systems to incorporate mechanisms to use predictions made by neural approaches, informing about plausible situations witnessed in the data, in a principled way. In the rest of this section we will discuss ways in which this idea can be implemented.

One of the most natural solutions is to use vector representations to implement a form of similarity based reasoning [23, 13]. For instance, we could have a KB with factual knowledge stating that strawberries are instances of the concept berries, $\mathsf{Berry}(\mathsf{strawberry})$, and ontological knowledge stating that berries are healthy, $\mathsf{Berry} \sqsubseteq \mathsf{Healthy}$. Clearly, this KB entails that strawberries are healthy. Further, using a standard word embedding [45], we can find out that $\mathsf{strawberry}$ and $\mathsf{raspberry}$ are highly similar. Now, using the KB and the additional similarity information, we can infer that it is plausible that raspberries are berries and, therefore, healthy. This same idea could be lifted to find the similarity between concept names (classes) and find plausible rules. For instance, assume that strawberries and raspberries are concept names and that our ontology specifies that strawberries are healthy, i.e. $\mathsf{Strawberry} \sqsubseteq \mathsf{Healthy}$. Using the similarity between strawberries and raspberries, we could then infer that the concept inclusion $\mathsf{Raspberry} \sqsubseteq \mathsf{Healthy}$ is plausible. However, implementing this strategy in a principled way is difficult, because it is unclear how we can formally relate degrees of similarity to the plausibility of the inferred rules, i.e. if we can infer using standard deduction that $C_1 \sqsubseteq X$, how similar does concept C_2 needs to be to C_1 to accept the plausible inference $C_2 \sqsubseteq X$? For this reason, rather than focusing on similarity based reasoning, it has been proposed to focus on *interpolative reasoning* instead [64]. The main difference is that instead of focusing on the similarity between two entities, we focus on how one entity relates to a group of entities. For instance, we say that the concept $\mathsf{Raspberry}$ is *conceptually between* the concepts $\mathsf{Strawberry}$, $\mathsf{Blackberry}$ and Cherry. Intuitively, this means that we accept that any (natural) property that holds for each of the concepts $\mathsf{Strawberry}$, $\mathsf{Blackberry}$, Cherry is likely to hold for $\mathsf{Raspberry}$ as well. In addition to using similarity based strategies, humans also rely on analogies for inferring plausible knowledge. Analogical reasoning can be particularly powerful, as it allow us to make predictions about concepts that may themselves not be similar to any other concepts. Recent models from the field of Natural Language Processing make it possible to discover analogies with a high level of accuracy [71]. It is thus of interest to explore whether analogy based reasoning processes could be used as another mechanism for exploiting knowledge from neural representations for symbolic reasoning. We now discuss in more detail how interpolative and analogical reasoning can be formalised in the context of description logics.

5.1 Interpolative Reasoning

We start by illustrating how the interpolation pattern works [26, 64]. Assume that we have the following knowledge about some concept C:

$$\mathsf{Strawberry} \sqsubseteq C \qquad \mathsf{Orange} \sqsubseteq C$$

Intuitively, even if we know nothing else about C, we could still make the following inductive inference:

$$\mathsf{Raspberry} \sqsubseteq C \tag{11}$$

This conclusion relies on background knowledge about strawberries, oranges and raspberries, in particular the fact that raspberries are expected to have all the *natural* properties that strawberries and oranges have in common (e.g. being high in vitamin C). In such a case, we say that raspberries are *conceptually between* strawberries and oranges. Importantly, knowledge about conceptual betweenness can be derived from data-driven representations. For instance, [25] found that geometric betweenness closely corresponds to conceptual betweenness in vector spaces learned with multi-dimensional scaling.

The notion of *naturalness* plays a central role, as it is clear that the conclusion in (11) can only be justified by making certain assumptions on the concept C. If C could be an arbitrary concept, e.g. a concept representing the union of Orange and $\mathsf{Strawberry}$, there is no reason to believe that the inference is valid, but for natural properties the inference seems intuitively plausible. This idea that only some properties admit inductive inferences has been extensively studied in philosophy [34, 57, 27]. In the context of conceptual spaces, "natural properties" are those which are modelled as convex regions, as explained in Section 2.1. To determine which concepts, in a given ontology, are likely to be natural, a useful heuristic is to consider the concept name: concepts that correspond to standard natural language terms are normally assumed to be natural [29].

The extension $\mathcal{EL}^{\bowtie}$ of $\mathcal{EL}$ was designed based on the above intuitions, with the aim of enabling reasoning about conceptual betweenness and natural concepts, and thus supporting interpolative reasoning. Syntactically, $\mathcal{EL}$ is extended with the in-between constructor, which allows us to describe the set of objects that are between two concepts: we write $C \bowtie D$ to denote all objects that are between the concepts C and D. We further assume that $\mathsf{N_C}$ contains a distinguished infinite set of *natural concept names* $\mathsf{N_C^{Nat}}$. The syntax of $\mathcal{EL}^{\bowtie}$ *concepts* C, D is thus defined by the following grammar, where $A \in \mathsf{N_C}$, $A' \in \mathsf{N_C^{Nat}}$ and $r \in \mathsf{N_R}$:

$$C, D := \top \mid A \mid C \sqcap D \mid \exists r.C \mid N \qquad N, N' := A' \mid N \sqcap N' \mid N \bowtie N'$$

Concepts of the form N, N' are called *natural concepts.*

▶ **Example 4.** Using the following $\mathcal{EL}^{\bowtie}$ TBox $\mathcal{T}$, we can now model the situation described above:

$$\mathsf{Strawberry} \sqsubseteq \mathsf{Healthy} \tag{12}$$

$$\mathsf{Orange} \sqsubseteq \mathsf{Healthy} \tag{13}$$

$$\mathsf{Raspberry} \sqsubseteq \mathsf{Strawberry} \bowtie \mathsf{Orange} \tag{14}$$

$$\mathsf{Healthy} \sqsubseteq \exists \mathsf{improves}.\mathsf{QualityOfLife} \tag{15}$$

such that $\mathsf{Strawberry}, \mathsf{Orange}, \mathsf{Raspberry}, \mathsf{Healthy} \in \mathsf{N}_\mathsf{C}^\mathsf{Nat}$.

The semantics of $\mathcal{EL}^{\bowtie}$ needs to adequately characterise natural concepts and concept betweenness, and thus support interpolation, i.e.: such that from $A \sqsubseteq B_1 \bowtie B_2$, $B_1 \sqsubseteq C$ and $B_2 \sqsubseteq C$, we can derive $A \sqsubseteq C$, provided that C is *natural.* To this end, Ibáñez-García et al. [41] proposed two semantics: a feature-enriched semantics inspired by formal concept analysis [73] and a geometric semantics inspired by conceptual spaces. In the former, at the semantic level a set of features is associated with each concept. Note that these features are semantic constructs, which have no direct counterpart at the syntactic level. A concept is then natural if it is completely characterized by these features, while B is between A and C if the set of features associated with B contains the intersection of the sets associated with A and C. In the second semantics, concepts are interpreted as regions from a vector space. A concept is then natural if it is interpreted as a convex region, while B is between A and C if the region corresponding to B is geometrically between the regions corresponding to A and C (i.e. in the convex hull of their union). We refrain from giving the full technical details, but invite the interested reader to look at [41]. Ibáñez-García et al. [41] also investigate the complexity of reasoning with interpolation, and show that under both semantics the concept subsumption problem becomes computationally more costly than in pure $\mathcal{EL}$: coNP-complete under the feature semantics and PSpace-hard under the geometric semantics.

One of the main drawbacks of the feature semantics is that it is too restrictive and cannot support interpolation in an adequate way if the $\bot$ construct is present. To address this shortcoming, Schockaert et al. [62] recently introduced a new semantics based on an abstract ternary betweenness relation bet over elements of the domain, such that that $\mathsf{bet}(a, b, c)$ if b is between a and c. We then have that $A \sqsubseteq B_1 \bowtie B_2$ is satisfied in an interpretation $\mathcal{I}$ if every element in $A^\mathcal{I}$ is between some individual from $B_1^\mathcal{I}$ and some element from $B_2^\mathcal{I}$. A central result from [62] shows that the feature-enriched semantics from [41] can essentially be seen as a special case, where the betweenness relation bet fulfills certain properties. The results in [62] are preliminary, leaving open for example, the complexity of reasoning under this new semantics.

The logic $\mathcal{EL}^{\bowtie}$ is built on the idea of conceptual betweenness. This ensures that the semantics remains close to cognitive models of category based induction, and information about conceptual betweenness can moreover be readily obtained from embeddings. However, an important open question is whether it is possible to develop meaningful forms of rule interpolation that go beyond this idea of conceptual betweenness. For instance, consider the following rules:

$$\begin{aligned}
&\mathsf{burglary}(L,T) \leftarrow \mathsf{burglary}(L,T-2), \mathsf{burglary}(L,T-1)\\
&\mathsf{burglary}(L,T) \leftarrow \mathsf{burglary}(L,T-1), \mathsf{burglary}(L_1,T-1), \mathsf{burglary}(L_2,T-1), \mathsf{n}(L,L_1),\\
&\qquad\qquad\qquad\quad \mathsf{n}(L,L_2), L_1 \neq L_2\\
&\mathsf{burglary}(L,T) \leftarrow \mathsf{burglary}(L,T-2), \mathsf{burglary}(L_1,T-1), \mathsf{burglary}(L_2,T-1), \mathsf{n}(L,L_1),\\
&\qquad\qquad\qquad\quad \mathsf{n}(L,L_2), L_1 \neq L_2
\end{aligned}$$

Intuitively, these rules partially characterise the spatio-temporal diffusion pattern of crime hotspots. For instance, the first rule asserts that if there has been a burglary at time points $T-1$ and $T-2$ at a given location (e.g. during the two previous days), then it is likely that there will be a burglary at time point T in the same location. The other two rules include the predicate n to encode information about neighbouring locations. Given these rules, the following rule also seems plausible:

$$\begin{aligned}
&\mathsf{burglary}(L,T) \leftarrow \mathsf{burglary}(L_1,T-2), \mathsf{burglary}(L_2,T-2), \mathsf{burglary}(L,T-1), \mathsf{n}(L,L_1),\\
&\qquad\qquad\qquad\quad \mathsf{n}(L,L_2), L_1 \neq L_2
\end{aligned}$$

However, it is unclear how the underlying principle could be formalised, and how the associated background information could be obtained.

5.2 Analogical Reasoning

Reasoning by analogy has been extensively studied in cognitive science, philosophy, and artificial intelligence [31, 38, 39, 12, 55, 11]. In the context of AI, the formalisation of analogical reasoning typically builds on analogical proportions, i.e. statements of the form *"A is to B what C is to D"* [12, 55, 11]. For instance, a notable result in this area has been the development of analogical classifiers, which are based on the principle that whenever the features of four examples are in an analogical proportion, then their class labels should be in an analogical proportion as well [12, 40]. Somewhat surprisingly, analogical reasoning was only recently considered for completing ontologies [61]. Schockaert et al. [61] took inspiration from analogical classifiers to infer plausible concept inclusions. The resulting inference pattern is called *rule extrapolation*; it is illustrated in the next example.

▶ **Example 5** ([61], Rule Extrapolation). Suppose we have an ontology with the following concept inclusions:

$$\mathsf{Young} \sqcap \mathsf{Cat} \sqsubseteq \mathsf{Cute} \tag{16}$$

$$\mathsf{Adult} \sqcap \mathsf{WildCat} \sqsubseteq \mathsf{Dangerous} \tag{17}$$

$$\mathsf{Young} \sqcap \mathsf{Dog} \sqsubseteq \mathsf{Cute} \tag{18}$$

Suppose we are furthermore given that "Cat is to WildCat what Dog is to Wolf". Trivially, we also have that "Young is to Adult what Young is to Adult" and "Cute is to Dangerous what Cute is to Dangerous". Using rule extrapolation, we can then infer the following:

$$\mathsf{Adult} \sqcap \mathsf{Wolf} \sqsubseteq \mathsf{Dangerous} \tag{19}$$

The knowledge inferred by analogical reasoning could also be used to transfer knowledge from one domain to another:

▶ **Example 6** ([61], Rule translation)**.** Suppose we are given the following knowledge:

$$\mathsf{Program} \sqsubseteq \exists \mathsf{specifies}.\mathsf{Software} \tag{20}$$

and the fact that "Program is to Plan what Software is to Building". Then we can plausibly infer:

$$\mathsf{Plan} \sqsubseteq \exists \mathsf{specifies}.\mathsf{Building} \tag{21}$$

Rule translation is useful as ontologies are often developed using "templates" to encode knowledge from different domains (e.g. knowledge about different professions). The strategy from Example 6 then allows us to complete the ontology by introducing additional domains.

As in the case of interpolative reasoning, the main objective of Schockaert et al. [61] was to establish the principles for incorporating analogical reasoning and, in particular, to develop a model-theoretic semantics. To this end, the description logic $\mathcal{EL}_{\bot}^{\mathsf{ana}}$ is introduced, which extends $\mathcal{EL}_{\bot}^{\bowtie}$ with so-called analogy assertions. Formally, $\mathcal{EL}_{\bot}^{\mathsf{ana}}$ concepts C, D are defined by the following grammar, where $A \in \mathsf{N_C}$, $A' \in \mathsf{N_C^{Nat}}$, $r \in \mathsf{N_R}$ and $r' \in \mathsf{N_R^{Int}}$:

$$\begin{aligned} C, D &:= \top \mid \bot \mid A \mid C \sqcap D \mid \exists r.C \mid N \\ N, N' &:= A' \mid N \sqcap N' \mid N \bowtie N' \mid \exists r'.N \end{aligned}$$

Note how $\mathcal{EL}_{\bot}^{\mathsf{ana}}$ concepts extend $\mathcal{EL}_{\bot}^{\bowtie}$ concepts by allowing existential restrictions over so-called intra-domain roles, i.e. roles from the designated set $\mathsf{N_R^{Int}}$, as natural concepts. An $\mathcal{EL}_{\bot}^{\mathsf{ana}}$ TBox is a finite set containing two types of assertions: (i) $\mathcal{EL}_{\bot}^{\mathsf{ana}}$ concept inclusions, and (ii) *analogy assertions* of the form $C_1 \triangleright D_1 :: C_2 \triangleright D_2$, where C_1, C_2, D_1, D_2 are natural $\mathcal{EL}_{\bot}^{\mathsf{ana}}$ concepts.

The semantics of $\mathcal{EL}_{\bot}^{\mathsf{ana}}$ builds on the feature-enriched semantics of $\mathcal{EL}_{\bot}^{\bowtie}$. Recall that analogies involve transferring knowledge from one application domain to another domain, e.g. from software engineering to architecture. Hence, at the semantic level these domains will be associated with subsets of features $\mathcal{F}$. In particular, interpretations will specify a partition $[\mathcal{F}_1, ..., \mathcal{F}_k]$ of $\mathcal{F}$, defining the different domains of interest. To capture the intuition of analogies, some of the partition classes will be viewed as being analogous, in the sense that there is some kind of structure-preserving mapping between them. We again refrain from giving the full technical details. We point out that Schockaert et al. [61] formally show that the analogical patterns exemplified above are supported under the proposed semantics.

The investigation by Schockaert et al. [61] leaves open several interesting questions such as establishing the computational complexity of reasoning in $\mathcal{EL}_{\bot}^{\mathsf{ana}}$. For the practical uptake of $\mathcal{EL}_{\bot}^{\mathsf{ana}}$, it would be also important to consider nonmonotonic extensions, as analogical assertions might introduce conflicts with the existing ontological knowledge.

6 Conclusions

Combining symbolic reasoning with sub-symbolic learning is an important and widely studied challenge for AI research. To enable such a combination in a principled way, a key question is how we can unify the two rather distinct types of representations that are involved, i.e. symbols and vectors. In this paper, we discussed a number of strategies that are inspired by the theory of conceptual spaces. First, we looked at the possibility of achieving a tight integration between symbolic and vector representations based on the idea that concepts can be viewed as regions in vector space embeddings. Moreover, we also explored the idea that meaningful "quality dimensions" can be identified in learned embeddings, adding more structure and a degree of interpretability to the vector representations themselves. However, we also argued that the use of region based representations has some inherent limitations when it comes to modelling relational knowledge. For this reason, we finally discussed a number of settings in which vectors and symbols are combined in a looser way. Essentially, the underlying idea is to exploit the similarity structure captured by the vector space to identify symbolic knowledge that plausibly, but not deductively, follows from a given knowledge base.

References

1 Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. BoxE: A box embedding model for knowledge base completion. In *NeurIPS*, 2020.

2 Thomas Ager. *Disentangling low-dimensional vector space representations of text documents*. PhD thesis, Cardiff University, 2021.

3 Thomas Ager, Ondrej Kuzelka, and Steven Schockaert. Modelling salient features as directions in fine-tuned semantic spaces. In *CoNLL*, pages 530–540, 2018.

4 Rana Alshaikh, Zied Bouraoui, Shelan S. Jeawak, and Steven Schockaert. A mixture-of-experts model for learning multi-facet entity embeddings. In *COLING*, pages 5124–5135, 2020.

5 Rana Alshaikh, Zied Bouraoui, and Steven Schockaert. Learning conceptual spaces with disentangled facets. In *CoNLL*, pages 131–139, 2019.

6 Rana Alshaikh, Zied Bouraoui, and Steven Schockaert. Hierarchical linear disentanglement of data-driven conceptual spaces. In *IJCAI*, pages 3573–3579, 2020.

7 Abhijeet Awasthi, Sabyasachi Ghosh, Rasna Goyal, and Sunita Sarawagi. Learning from rules generalizing labeled exemplars. In *ICLR*, 2020.

8 Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *IJCAI*, pages 364–369, 2005.

9 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

10 Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.

11 Nelly Barbot, Laurent Miclet, and Henri Prade. Analogy between concepts. *Artificial Intelligence*, 275:487–539, 2019.

12 Sabri Bayoudh, Laurent Miclet, and Arnaud Delhay. Learning by analogy: A classification rule for binary and nominal data. In *IJCAI*, pages 678–683, 2007.

13 Islam Beltagy, Cuong Chau, Gemma Boleda, Dan Garrette, Katrin Erk, and Raymond J. Mooney. Montague meets Markov: Deep semantics with probabilistic logical form. In **SEM*, pages 11–21, 2013.

14 Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.

15 Stefan Borgwardt, İsmail İlkan Ceylan, and Thomas Lukasiewicz. Recent advances in querying probabilistic knowledge bases. In *IJCAI*, pages 5420–5426, 2018.

16 Zied Bouraoui, José Camacho-Collados, Luis Espinosa Anke, and Steven Schockaert. Modelling semantic categories using conceptual neighborhood. In *AAAI*, pages 7448–7455. AAAI Press, 2020.

17 Zied Bouraoui, Shoaib Jameel, and Steven Schockaert. Inductive reasoning about ontologies using conceptual spaces. In *AAAI*, pages 4364–4370, 2017.

18 Zied Bouraoui and Steven Schockaert. Learning conceptual space representations of interrelated concepts. In Jérôme Lang, editor, *IJCAI*, pages 1760–1766, 2018.

19 Camille Bourgaux, Ana Ozaki, and Jeff Z. Pan. Geometric models for (temporally) attributed description logics. In Martin Homola, Vladislav Ryzhikov, and Renate A. Schmidt, editors, *DL*, 2021.

20 Katarina Britz, Thomas Meyer, and Ivan Varzinczak. Semantic foundation for preferential description logics. In *Australasian Joint Conference on Artificial Intelligence*, pages 491–500. Springer, 2011.

21 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.

22 José Camacho-Collados, Mohammad Taher Pilehvar, and Roberto Navigli. Nasari: Integrating explicit knowledge and corpus statistics for a multilingual representation of concepts and entities. *Artificial Intelligence*, 240:36–64, 2016.

23 Claudia d'Amato, Nicola Fanizzi, Bettina Fazzinga, Georg Gottlob, and Thomas Lukasiewicz. Ontology-based semantic search on the web and its combination with the power of inductive reasoning. *Ann. Math. Artif. Intell.*, 65(2-3):83–121, 2012.

24 Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. In *EMNLP*, pages 1389–1399, 2016.

25 Joaquín Derrac and Steven Schockaert. Inducing semantic relations from conceptual spaces: A data-driven approach to plausible reasoning. *Artif. Intell.*, 228:66–94, 2015.

26 Didier Dubois, Henri Prade, Francesc Esteva, Pere Garcia, and Lluis Godo. A logical approach to interpolation based on similarity relations. *International Journal of Approximate Reasoning*, 17(1):1–36, 1997.

27 Peter Gärdenfors. *Conceptual spaces: The geometry of thought.* MIT press, 2000.

28 Peter Gärdenfors. How to make the semantic web more semantic. In A.C. Varzi and L. Vieu, editors, *Formal Ontology in Information Systems*, pages 19–36. IOS Press, 2004.

29 Peter Gärdenfors. *The geometry of meaning: Semantics based on conceptual spaces.* MIT press, 2014.

30 Peter Gärdenfors and Mary-Anne Williams. Reasoning about categories in conceptual spaces. In *IJCAI*, pages 385–392, 2001.

31 Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive science*, 7(2):155–170, 1983.

32 Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. Semantic characterization of rational closure: From propositional logic to description logics. *Artificial Intelligence*, 226:1–33, 2015.

33 Robert L Goldstone. Isolated and interrelated concepts. *Memory & Cognition*, 24(5):608–628, 1996.

34 Nelson Goodman. *Fact, fiction, and forecast.* Harvard University Press, 1955.

35 Georg Gottlob, Michael Morak, and Andreas Pieris. Recent advances in datalog$^{\pm}$. In *Reasoning Web*, volume 9203 of *Lecture Notes in Computer Science*, pages 193–217, 2015.

36 Víctor Gutiérrez-Basulto, Jean Christoph Jung, Carsten Lutz, and Lutz Schröder. Probabilistic description logics for subjective uncertainty. *J. Artif. Intell. Res.*, 58:1–66, 2017.

37 Víctor Gutiérrez-Basulto and Steven Schockaert. From knowledge graph embedding to ontology embedding? an analysis of the compatibility between vector space representations and rules. In *KR*, pages 379–388, 2018.

38 Douglas R Hofstadter, Melanie Mitchell, et al. The copycat project: A model of mental fluidity and analogy-making. *Advances in Connectionist and Neural Computation Theory*, 2:205–267, 1995.

39 Keith J Holyoak and Paul Thagard. The analogical mind. *American psychologist*, 52(1):35–44, 1997.

40 Nicolas Hug, Henri Prade, Gilles Richard, and Mathieu Serrurier. Analogical classifiers: A theoretical perspective. In *ECAI*, pages 689–697, 2016.

41 Yazmín Ibáñez-García, Víctor Gutiérrez-Basulto, and Steven Schockaert. Plausible reasoning about el-ontologies using concept interpolation. In *KR*, pages 506–516, 2020.

42 Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial intelligence*, 44(1-2):167–207, 1990.

43 Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In *ACL*, pages 292–302, 2019.

44 Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In *NeurIPS*, pages 3753–3763, 2018.

45 Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *NAACL-HLT*, pages 746–751, 2013.

46 Roberto Navigli and Simone Paolo Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

47 Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816, 2011.

48 Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. *NIPS*, 30:6338–6347, 2017.

49 Robert M Nosofsky. Choice, similarity, and the context theory of classification. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 10(1):104–114, 1984.

50 Daniel N Osherson, Edward E Smith, Ormond Wilkie, Alejandro Lopez, and Eldar Shafir. Category-based induction. *Psychological Review*, 97(2):185–200, 1990.

51 Matías Osta-Vélez and Peter Gärdenfors. Category-based induction in conceptual spaces. *Journal of Mathematical Psychology*, 96, 2020.

52 Özgür Lütfü Özçep, Mena Leemhuis, and Diedrich Wolter. Cone semantics for logics with negation. In *IJCAI*, pages 1820–1826, 2020.

53 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

54 Henri Prade and Gilles Richard, editors. *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548 of *Studies in Computational Intelligence*. Springer, 2014.

55 Henri Prade and Gilles Richard. From analogical proportion to logical proportions: A survey. In *Computational Approaches to Analogical Reasoning: Current Trends*, pages 217–244. Springer, 2014.

56 Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *ICLR*, 2020.

57 W.V. Quine. *From a Logical Point of View*. Harvard University Press, 1953.

58 Lance J Rips. Inductive judgments about natural categories. *Journal of Verbal Learning and Verbal Behavior*, 14(6):665–681, 1975.

59 Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NIPS*, pages 3788–3800, 2017.

60 Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data*, 15(2):1–49, 2021.

61 Steven Schockaert, Yazmín Ibáñez-García, and Víctor Gutiérrez-Basulto. A description logic for analogical reasoning. In *IJCAI*, pages 2040–2046. ijcai.org, 2021.

62 Steven Schockaert, Yazmín Angélica Ibáñez-García, and Víctor Gutiérrez-Basulto. Modelling concept interpolation in description logics using abstract betweenness relations. In *DL*, 2021.

63 Steven Schockaert and Henri Prade. Solving conflicts in information merging by a flexible interpretation of atomic propositions. *Artif. Intell.*, 175(11):1815–1855, 2011.

64 Steven Schockaert and Henri Prade. Interpolative and extrapolative reasoning in propositional theories using qualitative knowledge about conceptual spaces. *Artif. Intell.*, 202:86–131, 2013.

65 Mikhail Sheremet, Dmitry Tishkovsky, Frank Wolter, and Michael Zakharyaschev. A logic for concepts and similarity. *Journal of Logic and Computation*, 17(3):415–452, 2007.

66 Steven A Sloman. Feature-based induction. *Cognitive Psychology*, 25(2):231–280, 1993.

67 Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezný, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *J. Artif. Intell. Res.*, 62:69–100, 2018.

68 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2018.

69 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2019.

70 Théo Trouillon, Christopher R. Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *J. Mach. Learn. Res.*, 18:130:1–130:38, 2017.

71 Asahi Ushio, José Camacho-Collados, and Steven Schockaert. Distilling relation embeddings from pretrained language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *EMNLP*, pages 9044–9062, 2021.

72 Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

73 Rudolf Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In *Ordered Sets*, pages 445–470. Springer, 1982.

74 Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, pages 5498–5507, 2018.

75 Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

76 Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

77 Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *IJCAI*, pages 1755–1762, 2020.

78 LA Zadeh. Calculus of fuzzy restrictions. In *Fuzzy Sets and Their Applications to Cognitive and Decision Processes: Proceedings of the US–Japan Seminar on Fuzzy Sets and Their Applications, Held at the University of California, Berkeley, California, July 1-4, 1974*, pages 1–39, 1975.

79 Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. ConE: Cone embeddings for multi-hop reasoning over knowledge graphs. *NeurIPS*, 2021.

Combining Embeddings and Rules for Fact Prediction

Armand Boschin ✉
Télécom Paris, Institut Polytechnique de Paris, France

Nitisha Jain ✉
Hasso Plattner Institute, University of Potsdam, Germany

Gurami Keretchashvili ✉
Télécom Paris, Institut Polytechnique de Paris, France

Fabian Suchanek ✉ 🏠
Télécom Paris, Institut Polytechnique de Paris, France

Abstract

Knowledge bases are typically incomplete, meaning that they are missing information that we would expect to be there. Recent years have seen two main approaches to guess missing facts: Rule Mining and Knowledge Graph Embeddings. The first approach is symbolic, and finds rules such as "If two people are married, they most likely live in the same city". These rules can then be used to predict missing statements. Knowledge Graph Embeddings, on the other hand, are trained to predict missing facts for a knowledge base by mapping entities to a vector space. Each of these approaches has their strengths and weaknesses, and this article provides a survey of neuro-symbolic works that combine embeddings and rule mining approaches for fact prediction.

2012 ACM Subject Classification Information systems → Information systems applications

Keywords and phrases Rule Mining, Embeddings, Knowledge Bases, Deep Learning

Digital Object Identifier 10.4230/OASIcs.AIB.2022.4

Category Invited Paper

Funding This work was partially funded by ANR-20-CHIA-0012-01 ("NoRDF").

1 Introduction

A knowledge base (KB) is a computer-processable collection of knowledge about the world. KBs typically contain real-world entities (such as organizations, people, movies, or locations) and their relationships (who was born where, which movie plays where, etc.). Thousands of such KBs are publicly available, including, e.g., Wikidata [60], DBpedia [4], and YAGO [53]. These KBs contain millions of entities and relationships between them, saying, e.g., who was born in which city, which actor acted in which movie, or which city is located in which country. Such KBs are used for question answering, Web search, text understanding, personal assistants, and other AI applications [66].

KBs are usually never complete; there are always facts that are missing from the KB. This is due to the way in which KBs are constructed: Some of them are constructed automatically by extracting facts from Web sources. Such an extraction may fail to extract all information, and the underlying sources can be incomplete themeselves. Other KBs are fed by a community, and may be incomplete simply because not all facts have yet been added. *Fact prediction* is the task of predicting facts that are true in the real world, but missing in the KB. Although

International Research School in Artificial Intelligence in Bergen (AIB 2022).
Editors: Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza; Article No. 4; pp. 4:1–4:30
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

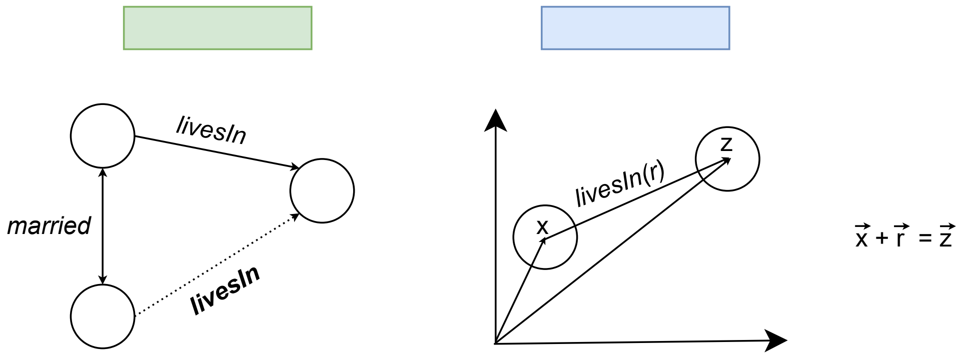


Figure 1 Rule Mining and Embeddings.

this may never make the KB complete, it will at least add facts that were missing. There are two major approaches to this end: Rule Mining and Knowledge Graph Embeddings. *Rule mining* is a symbolic approach. It finds rules such as the following in a KB:

$$\textit{married}(x,y) \wedge \textit{livesIn}(x,z) \Rightarrow \textit{livesIn}(y,z)$$

This rule means that if some person x is married to some person y, and x lives in a city z, then y also lives in that city. Such rules are usually not true in all instances, and typically come with a confidence score. Modern systems [30, 34, 40] can find such rules automatically on KBs of millions of entities. These rules can then be used to predict missing facts: If we know that some person lives in some city, but we do not know the place of residence of their spouse, we can use the rule to predict that, with high likelihood, the spouse lives in the same city.

The other methods to predict missing facts are *embedding-based methods*. These methods are a gift of the renaissance of neural networks in the 2010's. They project the entities and facts of a KB into a vector space. In its simplest variant, an entity x is mapped to its embedding, the vector $\vec{x}$. A relationship r, likewise, is mapped to a vector $\vec{r}$. These embeddings have the following property: If $\vec{r}$ is the vector for the *livesIn* relationship, then we can walk from the embedding $\vec{x}$ of a person x to the embedding $\vec{z}$ of their place of residence z by computing $\vec{z} = \vec{x} + \vec{r}$. This gives us another way of guessing the place of residence for some person y: We just find the city whose embedding is closest to $\vec{y} + \vec{r}$.

Each of these methods has its advantages and disadvantages: While rules are easy to understand for humans (and embeddings are less intuitively accessible), embeddings can take into account signals from all facts in which an entity occurs (and not just the ones mentioned in the rule, which are typically few). Therefore, recent years have seen fruitful endeavors to combine neural methods with symbolic methods. Both rule mining techniques and embedding techniques have been surveyed in recent articles [62, 9, 46, 73], among which is our own previous tutorial article [54]. Hence, in this tutorial, we survey approaches that combine both techniques.

The article is structured as follows: Section 2 introduces knowledge bases, rule mining techniques, and embedding techniques, following largely [54]. Section 3 discusses embeddings in more detail. Section 4 discusses embedding techniques that use rule mining techniques. Section 5, vice versa, discusses rule mining techniques that use embedding techniques. We conclude in Section 6.

2 Preliminaries

2.1 Knowledge Bases

Knowledge Bases. To define a knowledge base [54], we need a set $\mathcal{I}$ of *entities*. An entity is anything that can be an object of thought [67]. General-purpose KBs are typically concerned with entities such as places (e.g., *Paris*, or *India*), people (such as politicians, scientists, or actors), organizations (such as companies or associations), or artworks (such as movies, books, etc.). But knowledge bases can also be concerned with biomedical entities, geological formations, scientific articles, or any other type of entities.

In what follows, we assume a set $\mathcal{R}$ of binary *relation names* (also called *relations*, *relationships*, or *predicates*). For example, the relation *locatedIn* holds between a city and a country; the relation *actedIn* holds between an actor and a movie; and the relation *presidentOf* holds between a person and a country. Finally, we need a set $\mathcal{L}$ of *literals*. These are strings or numbers. A *fact* (or an *assertion*, *triple*, or *statement*) is then of the form $\langle s,\ r,\ o\rangle$ with a *subject* $s \in \mathcal{I}$, a *relation* $r \in \mathcal{R}$ and an *object* $o \in \mathcal{I} \cup \mathcal{L}$ [30][1]. An example of a fact is $\langle$*Paris*, *locatedIn*, *France*$\rangle$. The *inverse* of a relation r is a relation r^-, so that $\langle x, r, y\rangle$ holds if and only if $\langle y, r^-, x\rangle$ holds. For example, the inverse of *hasNationality* is *hasCitizen*. A *knowledge base* $\mathcal{K}$ over the sets $\mathcal{I}, \mathcal{R}, \mathcal{L}$ is then a set of facts over these sets. Whenever $\mathcal{K}$ is clear from the context, we write $\langle s, r, o\rangle$ to mean $\langle s, r, o\rangle \in \mathcal{K}$.

Taxonomies. Knowledge bases typically also define *classes*. Intuitively, a class can be understood as a set of entities, its *instances*. For example, the class of capital cities contains the city of Paris, the city of Beijing, etc. Many formalisms use unary predicates to express class membership, stating, e.g., *city(Paris)*. If every instance of some class y is also an instance of some class y', then y is called a *subclass* of y'. For example, the class *capitalCity* is a subclass of the class *city*, which is itself a subclass of *geographicLocation*. This gives us a hierarchy of classes – the *taxonomy*. Figure 2 shows an example of a taxonomy of classes.

Many KBs express the taxonomy by binary relations. To say that an entity x belongs to a class y, the KB adds the triple $\langle x,\ \textit{type},\ y\rangle$. To say that a class y is a subclass of a class y', we add $\langle y,\ \textit{subclassOf},\ y\text{'}\rangle$. However, a taxonomy has an inherent semantics that is different from other facts that hold between entities, and therefore, one is usually ill-advised to treat the link $\langle$*Paris*, *type*, *city*$\rangle$ in the same way as $\langle$*Paris*, *locatedIn*, *France*$\rangle$.

Axioms. KBs typically come with a set of logical constraints. For example, we can impose that if x is an instance of a class y, and if y is a subclass of the class y', then x must also be an instance of y':

$$\langle x, \textit{type}, y\rangle \wedge \langle y, \textit{subclassOf}, y'\rangle \Rightarrow \langle x, \textit{type}, y'\rangle$$

Typical axioms are the following:

- **Domain and Range Constraints** say that the subject (resp. object) of a relation must belong to a certain class, as in “People are born in places (and not, say, in organizations)”.
- **Cardinality Constraints** say that the number of objects per subject for a certain relation is restricted, as in “People can have at most one birth place”.
- **Symmetry, transitivity, and inverse constraints** say that a relation is symmetric, transitive, or the inverse of another relationship.
- **Disjointness constraints** say that two classes cannot have instances in common, e.g., places and people.

[1] For our purpose, in line with the other works [17, 18, 40], we do not consider blank nodes.

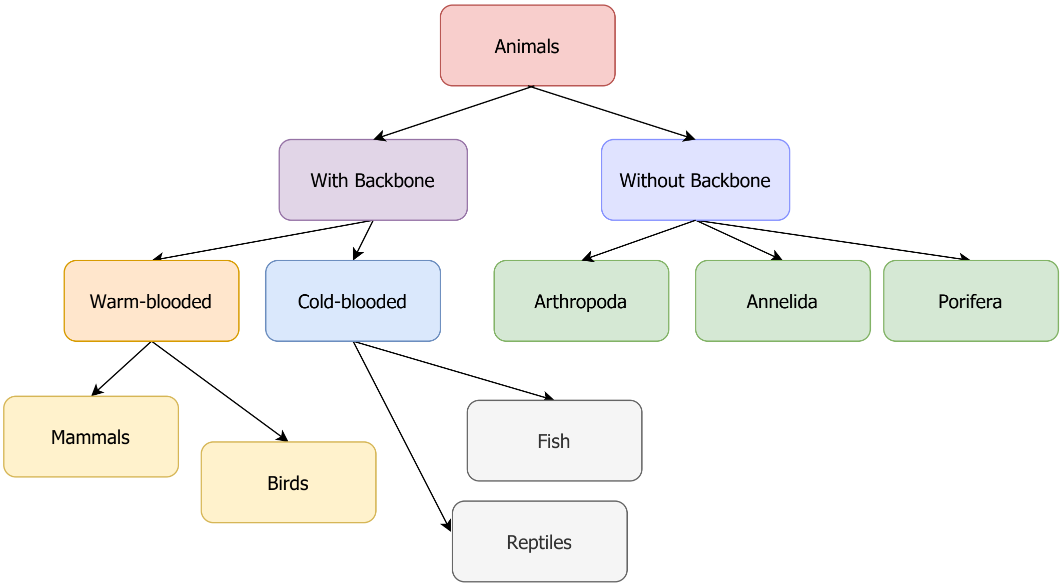


Figure 2 Taxonomy Example.

Such axioms exist in packages of different complexity: The Resource Description Framework Schema *RDFS* is a system of basic axioms that are concerned mainly with class membership. The axioms are so basic that they cannot result in contradictions. The Web Ontology Language *OWL* is a system of axioms that exists in several flavors – from the simple to the undecidable [54]. Such packages of axioms, together with the taxonomy, are sometimes called *ontology* or *schema*. Automated reasoners can be used to (1) predict facts that follow logically from these axioms and (2) determine whether a KB is inconsistent with respect to these axioms.

Fact Prediction. In what follows, we will assume an ideal knowledge base $\mathcal{K}^*$, which contains all facts of the real world (see [44] for a discussion of such a KB). One typically assumes that all facts in some given KB $\mathcal{K}$ are also true in the real world, i.e., $\mathcal{K} \subseteq \mathcal{K}^*$. However, the KBs are typically incomplete, i.e., there are facts in the real world that are not in the KB (i.e., $\mathcal{K} \subsetneq \mathcal{K}^*$). Predicting a fact f that is true in the real world, but not yet in the KB, is called the problem of *fact prediction*.

World Assumptions. Fact prediction is complicated by the fact that the KBs typically do not store negative information [43]. That is: while a KB may store that Elvis Presley has sung the song “All Shook Up”, it will not store the fact that he did not sing the song “The Winner Takes It All”. This raises the question what we should do if the KB does not contain certain statements (e.g., the KB does not contain the fact that Elvis sang “Always on my mind”, which is true in the real world). In a database, one would assume that any fact that does not appear in our data is not true in the real world – an assumption known as the *Closed World Assumption*. This assumption, however, is usually false for KBs, as KBs are highly incomplete and miss many facts from the real world. Hence, it is more appropriate to make the *Open World Assumption*, which says that if an assertion is not in the KB, it may or may not be true in the real world. Thus, in our example, if the KB does not contain the assertion that Elvis sang “Always on my mind”, we would not be entitled to conclude that this assertion would be false in the real world (which it is indeed not).

Negative assertions. A negative assertion is a statement that is known to be false. Such statements are essential as counter-examples in rule mining and fact prediction, so as to avoid an over-generalization. For example, Woody Allen married his step-daughter. If we find 10 other people who married their step-daughter, and no person who is *not* married to their step-daughter, we would conclude that people in general marry their step-daughters. The problem is now that KBs do not contain negative assertions. No KB tells us that Elvis Presley was *not* married to his step-daughter. And the Open World Assumption prevents us from assuming this negative assertion from the facts that are in the KB. This means that we have, in theory, no way to generate counter-examples for rule mining and fact prediction. Hence, we could mine the rule "Everybody is married to their step-daughter" without any obstruction.

Several remedies have been proposed. One is the *Partial Completeness Assumption*, or *Local Closed World Assumption* [17]. It says that if a KB contains the facts $\langle s, r, o_1 \rangle$, ..., $\langle s, r, o_n \rangle$, then any fact $\langle s, r, o' \rangle$ with $o' \notin \{o_1, ..., o_n\}$ must be false in the real world. The rationale is that if some contributor made the effort to add the objects $o_1, ..., o_n$, they would for sure also have added any remaining object o'. It can be shown that this assumption is generally true for relations that have few objects, such as *hasBirthDate* or *hasNationality* [18]. Indeed, in most KBs, the relations are designed in such a way that the average number of objects per subject is lower than the average number of subjects per object [18]. For example, a KB is more likely to contain the relation *hasNationality* (one person has few nationalities) rather than *hasCitizen* (one country has millions of citizens). A relation that has a higher average number of objects per subject than subjects per object can simply be replaced by its inverse [18]. With this, the PCA works generally well.

The method can be used as follows to generate a large number of negative examples: take any fact $\langle s, r, o \rangle$ from the KB, replace o by a randomly chosen object o' such that $\langle s, r, o' \rangle$ is not in the KB, and assume that $\langle s, r, o' \rangle$ is a negative assertion. The assertion $\langle s, r, o' \rangle$ is called a *corrupted* variant of $\langle s, r, o \rangle$. The method is also often applied in the same way to the subjects of the triples. This, however, creates a problem: Since relations generally have more subjects per object than vice versa, the PCA is much less plausible in this setting. For example, while it is, under the PCA, safe to assume that if some person Mary is American, she is not French, it is not safe to assume there are no more Americans than those in the KB. This is why the original PCA is applied only to the objects.

2.2 Rule Mining

Rules and Axioms. We have already seen that KBs can come with axioms, such as the symmetry of a relation. These axioms are usually defined manually, and they allow no exceptions. In what follows, we will be concerned with *rules.* These also express constraints on the data, but different from axioms, they are not imposed on the data, but automatically mined from the data. As such, they also allow for exceptions. For example, we can find that *marriedTo* is "usually" symmetric in the data of a given KB, meaning that for most couples, the *marriedTo* fact holds in both directions – although there are some couples for which the relation holds only in one direction, presumably because of missing data. This is why such rules are also called *soft rules* (as opposed to the "hard" axioms). Let us now make this idea more formal.

Atoms and Rules. An *atom* is an expression of the form $\langle \alpha, r, \beta \rangle$, where r is a relation and α, β are either entities or variables [30] (we write variables in lower case, and entities in upper case). For example, $\langle x, livesIn, Berlin \rangle$ is an atom with one variable, x. An atom is *instantiated* if at least one of its arguments is an entity. If both arguments are entities, the atom is *grounded* and tantamount to a fact. A *conjunction* of atoms $B_1, ..., B_n$ is of the form $B_1 \wedge ... \wedge B_n$. For example, we can build the conjunction $\langle x, livesIn, Paris \rangle \wedge \langle x, wasBornIn, Berlin \rangle$, which, intuitively, designates all people x who were born in Berlin and live in Paris. To make this intuition more formal, we need the notion of a substitution. A *substitution* σ is a partial mapping from variables to entities. Substitutions can be straightforwardly extended to atoms and conjunctions. For example, the substitution $\sigma = \{x \rightarrow Mary\}$ can be applied to our conjunction above, and it yields $\langle Mary, livesIn, Paris \rangle \wedge \langle Mary, wasBornIn, Berlin \rangle$.

A (Horn) *rule* is a formula of the form $B_1 \wedge ... \wedge B_n \Rightarrow H$, where the $B_1 \wedge ... \wedge B_n$ is a conjunction of *body atoms*, and H is the *head atom*. An example for a rule is

$$\langle x, married, y \rangle \wedge \langle x, livesIn, z \rangle \Rightarrow \langle y, livesIn, z \rangle$$

Let us call this rule R^* in what follows. Two atoms A, A' are *connected* if they have common variables. It is common [17, 18, 40] to impose that all atoms in a rule are transitively connected and that rules are closed. A rule is *closed* if every variable in the head appears in at least one atom in the body. A rule is *grounded* if all of its atoms are grounded.

Predictions. Given a rule $R = B_1 \wedge ... \wedge B_n \Rightarrow H$ and a substitution σ, we can apply σ to both the body and the head of R, and obtain an *instantiation* of R, which we denote by $\sigma(R)$. In our example, we could instantiate the above rule R^* by $\sigma = \{x \rightarrow Mary, y \rightarrow Bob, z \rightarrow Paris\}$, and obtain $\sigma(R^*)$ as

$$\langle Mary, married, Bob \rangle \wedge \langle Mary, livesIn, Paris \rangle \Rightarrow \langle Bob, livesIn, Paris \rangle$$

If $\sigma(B_i) \in \mathcal{K}\ \forall i \in \{1, ..., n\}$, we call $\sigma(H)$ a *prediction* of R from $\mathcal{K}$, and we write $\mathcal{K} \wedge R \models \sigma(H)$. Suppose, e.g., that we have a KB $\mathcal{K} = \{\langle Paris, locatedIn, France \rangle, \langle Mary, married, Bob \rangle, \langle Mary, livesIn, Paris \rangle\}$. Here, our example rule R^* can be instantiated as before by $\sigma = \{x \rightarrow Mary, y \rightarrow Bob, z \rightarrow Paris\}$. Then, all body atoms of the instantiated rule $\sigma(R^*)$ appear in $\mathcal{K}$. Hence, the rule predicts the head atom of $\sigma(R^*)$, which is $\langle Bob, livesIn, Paris \rangle$. Hence, we write $\mathcal{K} \wedge R^* \models \langle Bob, livesIn, Paris \rangle$.

Mining Rules. Inductive Logic Programming (ILP) is the task of finding rules automatically [54]. Typically, one provides a set of *positive examples* (i.e., facts that the rules shall predict), and a set of *negative examples* (facts that the rules must not predict). In the context of KBs, ILP faces several challenges: First, KBs usually do not provide negative examples. We have discussed a method to generate negative examples above, the Partial Completeness Assumption (Section 2.1). Another challenge is that a strict application of the definition of ILP to rule mining would find only rules that are true in all instantiations. However, in real-world KBs, there can be exceptions to rules, e.g., due to faulty or missing data. Hence, rule mining typically aims for rules that have a high *support* (the number of positive examples predicted by the rule), and a high *confidence* (the proportion of examples it predicts that are positive). In this way, the methods can find rules even if they do not apply in all instances, such as "If two people are married, then the children of one of them are also the children of the other".

AMIE [17] was one of the first rule mining systems for large KBs under the Open World Assumption. It starts with the most general rules (such as "everybody is married with each other"), and refines them until their confidence is high enough (e.g., "if two people are parents of the same children, they are most likely married”). This relies on the observation that the support of a rule decreases monotonically when a rule is made more specific. The RuDiK system [40] can mine logical rules like AMIE, but brings a number of improvements: First, RuDiK can also mine negative rules, such as “If two people are siblings, they are not married”. Second, RuDiK can mine relations between literals, such as “Someone’s birth date is always before someone’s death date”. Finally, RuDiK removes facts that have been covered by a rule, so that subsequent rules are forced to predict facts that have not already been predicted. This allows not just for some optimizations of the mining algorithm, but also to mine rules that predict more unknown facts correctly.

The AnyBURL system [34] is a bottom-up rule mining system: It starts with path rules that are specific to one instance, and generalizes them to achieve good support. A particular advantage of the system is that the user can trade running time for rule quality, i.e., get better rules by waiting longer.

The DRUM system [49] is a linear formulation of the rule mining problem using one-hot-encoding vectors for entities and adjacency matrices for relations. As it is linear, the problem is fully differentiable and can then be solved using gradient descent techniques. This solving appraoch proved to be very good for predictions involving previously unseen entities or relations.

Let us now turn to the second family of methods that can be used to predict missing facts: Knowledge Graph Embeddings.

2.3 Embeddings

Embeddings. An embedding for a group of objects (e.g. words, relations, or entities) is an injective function that maps each object to a real-valued vector, so that the intrinsic relations between the objects are maintained [54]. In the case of KBs, we are looking to embed entities and relations. In particular, given a KB, we would want the entities that are semantically similar in the KB to be mapped to vectors that are close to each other in the vector space.

The most basic embeddings [7] are designed so that, for a fact $\langle s, r, o \rangle$, we have $\vec{s} + \vec{r} \approx \vec{o}$, where $\vec{\cdot}$ is the embedding vector of the underlying entity or relation. For example, if we know $\langle$*Elvis*, *marriedTo*, *Priscilla*$\rangle$, then we would want the vector $\overrightarrow{Elvis} + \overrightarrow{marriedTo}$ to be close to the vector $\overrightarrow{Priscilla}$. An embedding with these properties has several advantages: First, the embedding allows us to feed entities and relations into machine learning methods that work on vectors (e.g., classification algorithms). The vectors are typically low in dimension (e.g., a few hundred), which makes them particularly suited for such applications. Second, the embedding provides a natural way of grouping together similar entities, so that given one entity, we can find its peers by scanning the vector space. In our example, we would expect Elvis to be close in the vector space to other singers. Finally, the embeddings allow for link prediction: If we do not know the spouse of Elvis, we can just compute the vector $\overrightarrow{Elvis} + \overrightarrow{marriedTo}$ and propose that the person that we find there is the spouse. If the embedding is well designed, that would actually work.

Terminology. In the literature about KB embeddings, the KB is often called a *knowledge graph* (KG) instead of a *knowledge base*. This is because embedding approaches typically project away literals and facts with literals. Consequently, fact prediction is known as *link prediction* in this scenario. Furthermore, the approaches typically do not deal with classes, taxonomies, or axioms. What remains is then a graph where the nodes are entities, and the edges are relations. In this scenario, facts are usually called *triples*, the subject is called the *head* of the triple, and the object is called the *tail*.

Link prediction with embeddings. Knowledge graph embeddings are created by trainable machine-learning models, typically neural networks. We will discuss these methods in detail in Section 3. All of these models take as input a fact $\langle h,\ r,\ t\rangle$, and output a score of its likelihood of being true: the higher the score, the more likely the model believes the fact to be. This score is typically denoted by $f(\langle h, r, t\rangle)$ or $f_{\vec{r}}(\vec{h}, \vec{t})$. To train such a model, we need a KB of true facts. We train the model to give a high score to these facts. To avoid over-generalization, we also have to train the model with counter-examples. These are typically generated by corrupting the facts from the KB (Section 2.1), i.e., by taking a fact $\langle h, r, t\rangle$ from the KB and replacing the tail by a random entity t'. The model is then trained to give the true triples from the input KB a higher score than the corrupted triples.

We can then use the models for link prediction as follows: We take a partially-filled triple for which we would like to know the head or tail entity, e.g., $\langle$*Elvis*, *marriedTo*, *?*$\rangle$. We try out all possible tail entities from the KB, and score the resulting triple using the scoring function. The predicted entity is intuitively the one with the highest resulting score. All entities can be sorted according to the scores of their triple. Each entity is then associated to its *prediction rank*, i.e., to the position that it has in the ranked list of predictions.

In the supervised setting, we often know the true answer (*Priscilla*), and we can compute its prediction rank $PR_{\langle Elvis, marriedTo, ?\rangle}(Priscilla)$. Several metrics are computed from the prediction ranks of head and tail entities. If $\mathcal{T}$ the set of known true facts, the metrics are the following:

- Mean Rank (MR): the average prediction ranks of the correct entities

$$MR = \frac{1}{2\,|\mathcal{T}|}\left(\sum_{(h,r,t)\in\mathcal{T}} PR_{\langle ?,r,t\rangle}(h) + PR_{\langle h,r,?\rangle}(t)\right)$$

- Mean Reciprocal Rank (MRR): the average of the inverse of the prediction ranks

$$MRR = \frac{1}{2\,|\mathcal{T}|}\left(\sum_{(h,r,t)\in\mathcal{T}} \frac{1}{PR_{\langle ?,r,t\rangle}(h)} + \frac{1}{PR_{\langle h,r,?\rangle}(t)}\right)$$

- Hit at k (Hit@k): proportion of the tests in which the prediction rank is better than k (typical values for k are 1, 3 and 10)

$$Hit@k = \frac{1}{2\,|\mathcal{T}|}\left(\sum_{(h,r,t)\in\mathcal{T}} \mathbb{1}\{PR_{\langle ?,r,t\rangle}(h) \leq k\} + \mathbb{1}\{PR_{\langle h,r,?\rangle}(t) \leq k\}\right)$$

Both MRR and Hit@k have values between 0 and 1, higher values indicate better results. In some cases, multiple entities can be correct answers (e.g. for 1-N relations) and the model should not be penalized for predicting another true answer that is simply more likely than the one at hand. Those metrics are usually computed in a *filtered* setting in which prediction ranks are computed by removing the other true entities ranked better than the one at hand.

3 Embedding Models

In the last decade, numerous methods for computing knowledge graph embeddings have been proposed. The methods differ from one another in terms of how they relate the entities and relations of the KG in the latent space. The existing models can be categorized as geometric, tensor-based or convolutional. In this section, we introduce and discuss a few popular models from each category.

In the following, let us consider a KG with n entities $\mathcal{E} = \{e_1, \ldots e_n\}$ and m relations $\mathcal{R} = \{r_1 \ldots r_m\}$ that is to be embedded in a d-dimensional vector space. $\mathbb{R}$ (resp. $\mathbb{C}$) is the field of real (resp. complex) numbers.

3.1 Geometric models

Geometric models interpret relations as geometric operations in the vector space. The earliest of these models is TransE, which we now describe in detail.

TransE [7] is a *translation-based model*, i.e., it uses a geometric distance to measure the similarity of the entities. Given a fact $\langle h, r, t\rangle$, its goal is to find vectors $\vec{h}, \vec{r}, \vec{t}$, so that $\vec{h} + \vec{r} \approx \vec{t}$.

One way to do that is to design a neural network [54]. We first create a *vocabulary*, i.e., an ordered list of all entities in the KG. Then we create, for each entity, its *one-hot encoding*. This is simply a vector that has as many dimensions as there are entities. Every element of the vector is set to zero, and only the i^{th} element is set to one, where i is the position of the entity in the vocabulary. The same is done for the relations. Then we design a network as follows: The input is the one-hot encoding of the head, the one-hot encoding of the relation, and the the one-hot encoding of the tail of a given fact from the KB. That is, if n is the number of entities, and m is the number of relations, the network has $m + 2 \times n$ input neurons. The first hidden layer of the network then maps each of these vectors to a d-dimensional real vector in $\mathbb{R}^d$. An entity e is mapped to a vector $\vec{e}$, and a relation r is mapped to $\vec{r}$. The further layers then reduce these vectors to a single output that scores the input assertion. More precisely, the network computes, for an input fact $\langle h, r, t\rangle$ from the KB, the function $f_{\vec{r}}(\vec{h}, \vec{t}) = -||\vec{h} + \vec{r} - \vec{t}||$ (where $|| \cdot ||$ is either the 1-norm or the 2-norm). The network is then trained with facts from the KB to maximize this score for these facts. It is trained with negative assertions to minimize this score. This leads to embeddings that verify the simple arithmetic equation $\vec{h} + \vec{r} \approx \vec{t}$ [54]. The important thing here is that the later hidden layers take their decision based solely on the output of the first hidden layer. The vectors computed by the first layer thus contain all the necessary information to assess the truth value of an assertion – and this is what we want from a good embedding. Thus, we will use the vectors that the first layer outputs as the embeddings of the input entities.

One limitation of TransE is the inability to model symmetric relationships [65]: if r is symmetric (i.e. $\langle h, r, t\rangle$ true implies $\langle t, r, h\rangle$ to be true as well), then r tends to have an embedding vector close to $\vec{0}$ because minimizing both $||\vec{h} + \vec{r} - \vec{t}||_2$ and $||\vec{t} + \vec{r} - \vec{h}||_2$ simultaneously happens if and only if $\vec{r} = \vec{0}$. Another problem appears with one-to-many relations. Consider for example the facts ⟨*ElonMusk*, *founderOf*, *SpaceX*⟩ and ⟨*ElonMusk*, *founderOf*, *Tesla*⟩. TransE would give very similar embeddings to both *SpaceX* and *Tesla*, and thus fail to differentiate between the two companies. TransE also has problems modeling many-to-one, reflexive, and transitive relations, and to capture multiple semantics of a relation.

TransH [65] tries to alleviate some limitations of TransE by allowing an entity to have different representations in the embedding space depending on the relation it is involved with. Each relation r is represented not only by a vector $\vec{r}$, but also by an hyperplane (i.e. a sub-space of one dimension less than the embedding space). Algebraically an hyperplane can be defined by a single vector, namely the vector that is orthogonal to it. Thus, each relation r is associated with a set of two vectors: $\vec{r}$ for the relation itself, and $\vec{h_r}$ for its hyperplane.

To compute the score of a triple $\langle h, r, t\rangle$, the embeddings $\vec{h}$, $\vec{t}$ of the entities are first projected onto the hyperplane defined by $\vec{h_r}$, and they are then connected by the translation vector $\vec{r}$ of the relation. Given a relation r, let p_r be the linear orthogonal projection on the hyperplane defined by $\vec{h_r}$. Then the loss function of TransH can be written as $f(\langle h, r, t\rangle) = f_{\vec{r}}(\vec{h}, \vec{t}) = -||p_r(\vec{h}) + \vec{r} - p_r(\vec{t})||_2^2$.

This is designed to solve the limitations of TransE: a reflexive relation r can have a translation vector $\vec{r}$ close to $\vec{0}$, since all information is contained in $\vec{h_r}$. For relations with several objects, likewise, the objects can be embedded in the same place in the hyperplane only for that specific relation.

TransR [31] extends the idea of sub-space projection of TransH by proposing that the projection step is now done on any sub-space of a given dimension. Let d be the dimension of the embedding space and d' the dimension of the relation-specific sub-spaces. Algebraically a linear projection from a vector space of dimension d into one of its sub-spaces of dimension d' is simply represented by a matrix of dimension $d \times d'$. Each relation r is then represented by a vector $\vec{r}$ and a projection matrix M_r. Thus, TransR is simply an evolution of TransH that increases the expressiveness of the model by increasing the number of parameters. Intuitively, this should allow the model to *learn* a greater amount of useful information from the known facts it is trained on. CTransR [31] is an extension of TransR, which operates by clustering diverse head-tail entity pairs into groups and learning distinct relation vectors for each group.

TransD [25] in turn proposes to keep the idea of projecting on any possible sub-space but reduces the number of parameters compared to TransD in order to limit the risk of overfitting. This is done by allowing only the sub-space projections that are defined by a *low-rank* matrix: that is a matrix that can be decomposed as a product of vectors.

Several other improvements have also been proposed in the direction of translation embedding methods, including TransG [68], TransF [16], and KG2E [21]. Other geometric models perform rotation-like transformations in the vector space instead of pure translations. Its most prominent examples are RotatE [55] and HAKE [72].

RotatE [55] aims to be particularly suited for relations that are symmetric, anti-symmetric, inverses of each other, and compositions of each other, which are typical for KGs. For instance, the relation *marriedTo* is a symmetric relation: $\langle x$, *marriedTo*, $y\rangle$ implies $\langle y$, *marriedTo*, $x\rangle$. Further, many relations such as familial relations are compositional. For example, $\langle x$, *hasParent*, $y\rangle$ and $\langle y$, *hasParent*, $z\rangle$ imply $\langle x$, *hasGrandParent*, $z\rangle$. RotatE captures these relation patterns by defining each relation as a rotation from the head entity to the tail entity in the vector space. Specifically entities and relations are now embedded in $\mathbb{C}^d$ and for any relation r, the modulus of each component $\vec{r_i}$ is 1. For a triple $\langle x, r, y\rangle$, the model then tries to achieve $\vec{y} \approx \vec{x} \circ \vec{r}$, where $\circ$ is the element-wise product. Intuitively, a relation r applies a coordinate-wise rotation on the head entity so as to come close to the tail entity. The score function is then $||\vec{x} \circ \vec{r} - \vec{y}||$. A relation is symmetric if and only if its embedding belongs to $\{-1, +1\}^d$ (i.e. coordinate-wise rotations of 0 or π radians), r_1 and r_2 are are symmetric if and only if their embeddings are complex conjugates, and a relation r_3 is the composition of two relations r_1 and r_2 if and only if $\vec{r_3} = \vec{r_1} \circ \vec{r_2}$ (i.e., the coordinate-wise rotations of r_3 are the successive rotations of r_1 and r_2).

HAKE [72] extends the RotatE embeddings by taking into account and preserving the semantic hierarchies of the entities in the KGs. For example, the entity *Paris* is part of *France*, which is a part of the EU. Such hierarchies between entities are quite common in most KGs such as Yago and Freebase. To model these relations between entities, HAKE represents an entity e (and a relation r) in the vector space in two parts: as $\vec{e}_m$ and $\vec{r}_m$

in the modulus part and as $\vec{e}_p$ and $\vec{r}_p$ in the phase part. The modulus part is aimed at differentiating entities at different hierarchies from each other, such as *Paris* from *France*, while the phase part distinguishes the different entities at the same hierarchy level, e.g. *Paris* and *Lyon*. In this manner, HAKE is able to represent the semantic hierarchies associated with KGs, and outperform other techniques by learning better embeddings.

3.2 Semantic Matching models

Another common category of embedding methods compares the vector of the subject and the vector of the object directly in order to assess how likely the fact is to be true.

RESCAL [38] is the simplest model in this category. Entities are represented as vectors and relations become bilinear functions (simply represented as square matrices). A triple $\langle h, r, t\rangle$ is then scored by the application of the relation-specific bilinear function to the entity embeddings: $f(\langle h, r, t\rangle) = \vec{h}^t \cdot M_r \cdot \vec{t}$, where $\vec{h}$ (resp. $\vec{t}$) is the embedding of h (resp. t) and $M_r \in \mathbb{R}^{d\times d}$ is the representing matrix of r. Intuitively, this bi-linear scoring function can be interpreted as some sort of *scalar product* between the entities in some relation-specific distortion of the embedding space. This is simply an intuition as no sufficient constraints are imposed on the relation matrices to make them scalar products. Precisely, they are not forced to be symmetric nor positive definite.

DistMult [69] is a variation of the RESCAL models where the relation matrices are all forced to be diagonal. This simplifies the computations, and reduces the parameter space. As a drawback, DistMult gives the same score for the triples $\langle h, r, t\rangle$ and $\langle t, r, h\rangle$. Thus, it is unable to model asymmetric relations such as *sonOf*, *actedIn* etc. Despite these limitations, DistMult has been recently shown to perform as well as many recently proposed models, presumably due to its simplicity and scalability [48].

ComplEx [58] improves upon the DistMult model by using the same diagonal constraint, but with complex-valued embedding vectors. Entities and relations are then simply represented as vectors in $\mathbb{C}^d$ and the Hermitian product is used instead of the bi-linear product in the scoring function. This allows the approach to take into account asymmetric relations in the KGs, as in the triple $\langle$*Paris*, *capitalOf*, *France*$\rangle$ (where France is not the capital of Paris). The scoring function is defined as $f(\langle h, r, t\rangle) = Re(\vec{h} \cdot M_R \cdot \bar{\vec{t}})$ where $Re(c)$ is the real part of $c \in \mathbb{C}$ and M_r is the diagonal matrix with $\vec{r}$ on its diagonal. The fact that the Hermitian product is not commutative solves the problem of representing asymmetric relations and switching to complex vector space doubles the number of parameters thus increasing the expressiveness of the model.

SimplE [27] proposes to extend one of the most generic multiplicative methods: Canonical Polyadic (CP) decomposition [22]. This method is used for decomposing tensors into a sum of products. It can be applied to KG embeddings because a KG with n entities and m relations is simply represented as a 3-dimensional adjacency tensor $\mathcal{T} \in \{0,1\}^{n\times n\times m}$: $\mathcal{T}[i, j, k] = 1$ if $\langle e_i, r_k, e_j\rangle$ is true and 0 else. As explained in [27], CP decomposition represents entities e with two vectors $(\vec{h_e}, \vec{t_e}) \in (\mathbb{R}^d)^2$ and relations r with a vector $\vec{r} \in \mathbb{R}^d$ where d is the dimension of the embedding. These vectors are learned in order to be able to reconstruct the tensor $\mathcal{T}$ by estimating $\hat{\mathcal{T}}[i, j, k] = \langle \vec{h_{e_i}}, \vec{t_{e_j}}, \vec{r_k}\rangle = \sum_{\ell=1..d} \vec{h_{e_i}}[l] \times \vec{t_{e_j}}[l] \times \vec{r_k}[l]$. This estimation is used in the case of KG embeddings as a scoring function of triples. SimplE just proposes to represent relations r with two vectors $\vec{r}$ and $\vec{r^{-1}}$, the scoring function being now $f(e_i, r, e_j) = \frac{1}{2}(\langle \vec{h_{e_i}}, \vec{t_{e_j}}, \vec{r}\rangle + \langle \vec{h_{e_j}}, \vec{t_{e_i}}, \vec{r^{-1}}\rangle)$. The authors show that their model is fully expressive, meaning that if given enough embedding dimensions it can exactly represent any KG. It is then argued that simple logical constraints can be implemented in the model by applying constraints on the relation embeddings. We will later see one such application in Section 4.3.

3.3 Deep Models

Deeper neural architectures have also been introduced for KB embeddings, with the hope that hidden layers can capture more complex interaction patterns between entities and relations (and then estimate more complex scoring functions). In such models, the first part of the network (which, in shallow networks, just maps facts to their embeddings or their projections) now adds additional layers (possibly numerous) that receive as inputs the embeddings, and produce as outputs some extracted features. The second part of the network now computes the scoring function from the features extracted by the first part of the network, and not directly from the embedding (or its projection) as in shallow models. The scoring function also becomes a parameter of the model (to be trained) and is not defined a priori anymore. This often entails that we lose the interpretability of the scoring function [54]. There are many deep neural network based models that have been proposed over the years, early examples of such models are SME, NTN and MLP [6, 52, 13].

NTN [52] was introduced by Socher et al. as a generalization of the RESCAL model. It employs a combination of linear transformations and nonlinear activation functions to obtain head and tail embeddings. As such, while this is a more expressive model, it is also quite complex with a large number of parameters that are harder to train. Better and lightweight architectures have been since proposed, such as MLP, where the paramaters are shared among all the relations.

ConvE and ConvKB [11, 10] are popular examples of models that are based on convolutional neural networks (CNN). These can learn complex nonlinear features of the entities and relations with fewer parameters by using 2D convolutions over embeddings. ConvE has been shown to be particularly effective for complex graph with nodes having a high number of incoming edges. The model introduced the 1-N scoring scheme where for a given triple $\langle h, r, t\rangle$ where t is to be predicted, the matching is performed with all the tail entities at the same time, leading to speedier training. ConvE has proven to be a competitive embedding model and a popular baseline for more recent deep learning approaches.

Graph Convolutional Networks (GCNs) have recently gained popularity for performing link prediction in knowledge graphs in tandem with standard embedding techniques. GCNs are a form of message passing multi-layer neural networks, first introduced by [28] for semi-supervised node classification on graph structured datasets. One layer of GCN encodes information about the immediate neighbours of a node in feature vector, and k layers stacked on top of each other can encode the information of the neighbourhood k hops away. GCNs can overcome the limitations of knowledge graph embedding models in terms of neglecting the attributes of the entities and ignoring the graph structure by encoding the entities based on their neighbours in the graph. Several extensions of GCNs have been suggested for multi-relational knowledge graphs.

Relational GCNs (R-GCNs) [50] are GCNs for graphs with a large number of relations, which makes them particularly suitable for knowledge graphs. Link prediction is essentially an auto-encoder framework: An encoder creates the feature representations for the entities from its neighbours (these features are generated from relation-specific transformations that are dependent on the type and direction of the relations). A decoder (in this case, DistMult factorization) is a scoring function to predict the labelled edges. R-GCNs show improvements compared to DistMult, HolE and ComplEx for the link prediction task. While R-GCN extended the GCN models on knowledge graphs by including the different types of relations during the generation of entity representations, they do not represent relations themselves.

VR-GCN [70] is an extension of the R-GCN model that generates both entity and relation embeddings explicitly. It ensures relation representation and different entity roles (head or tail in different triples), and it conforms to the translation representation from translational embeddings where $\vec{h} + \vec{r} \approx \vec{t}$. While the primary goal of this technique is to enable graph alignment, the performance of VR-GCN is also discussed in terms of the link prediction task, with VR-GCN acting as the encoder and DistMult as the decoder for scoring the triples.

SACN [51] leverages a variant of the existing knowledge graph embeddings ConvE and TransE as decoder along with a variant of GCN (weighted GCN) as encoder. The weighted GCN encoder learns representations for the entities in the graph by utilizing the graph structure, node attributes and the associated relations while weighing different relations differently and learning these weights during the training. The entity representations are given to the decoder, which is a combination of ConvE and TransE (based on ConvE but having the translational property of TransE model) that performs better than the ConvE model. The encoder and decoder are trained jointly to learn the entity representations and score triples to verify and improve the representations.

CompGCN [59] generalizes previous GCN methods by jointly learning the representation of the nodes and the relations in the multi-relational KGs while leveraging composition functions from embedding approaches. CompGCN is able to scale well with the increasing number of relations and outperforms several previous models including TransE, DistMult, ComplEx, R-GCN and SACN.

3.4 Evaluation of Embedding Methods

3.4.1 Evaluation Protocol

Evaluation. Rule methods are typically evaluated under the *open world assumption*, i.e., any fact that is predicted is manually evaluated to see whether it holds in the real world or not. Thus, even a fact that does not appear in the KB can be counted as correct. This evaluation is obviously very labor-intensive, but it targets what rule mining is interested in: the prediction of yet-unknown facts. KB embedding models, in contrast, are typically evaluated under the closed world assumption: Given a KB, one removes a certain portion of it to obtain a training KB. One then trains the embeddings on this reduced KB, and uses the embeddings to predict facts. If these facts appear in the original KB, they count as correct, otherwise they count as incorrect.

Datasets. Three very common KBs for evaluating embedding approaches are FB15k [7], WN18 [6] and Yago3-10 [11]. FB15k and WN18 were both proposed by Bordes et al. respectively in 2013 and 2014. FB15k is an extraction from Freebase where entities were selected based on the number of citations in the original KB. WN18 is a subset of Wordnet in which entities are *synsets*, that is semantic senses of words (selected on their popularity in the KB) and predicates are lexical relations between those senses. Yago3-10 was proposed by Dettmers et al. in 2018 as a subset of Yago3 [33] in which most of the facts describe people (e.g., by citizenship, gender, and profession). The three KBs have been initially randomly split into training, validation and test subsets and those splits always stay the same. Table 1 shows some statistics about these datasets.

Table 1 Details on the various KBs used for embedding evaluation.

Dataset	Number of entities	Number of relations	Number of training facts	Number of evaluation facts	Number of test facts
FB15k	14,951	1,345	483,142	50,000	59,071
FB15k-237	14,541	237	272,115	17,535	20,466
WN18	40,943	18	141,442	5,000	5,000
WN18RR	40,943	11	86,835	3,034	3,134
Yago3-10	123,182	37	1,079,040	5,000	5,000

3.4.2 Shortcomings of Benchmarks

While embedding models have gained popularity for the link prediction task and obtained state-of-the-art results, several studies have recently taken a critical look at the performance and evaluation aspects of these models. The benchmark datasets on which the embedding models are trained have also been scrutinized.

Toutanova et al. [57] were the first to find data leakage issues in the FB15k dataset. More precisely, the authors noted that, for certain relations r, the inverse relation r^- was also present in the data. This makes the prediction of a fact $\langle x, r, y\rangle$ trivial if the fact $\langle y, r^-, x\rangle$ is already there. As a remedy, the authors constructed the dataset FB15k-237 by removing the inverse triples and keeping only one relation out of the reverse relations. Dettmers et al. [11] similarly found issues with the WN18 dataset and created the WN18RR dataset. Table 1 shows the statistics about these datasets. With the introduction of these new datasets and their adoption for the evaluation of newer embedding models, it could be ensured that the models are not just learning trivial entailment, but learning to correctly predict non-trivial facts that require actual inference. However, most papers still showed the results for the evaluation of new models on both the old and new version of the datasets.

Akrami et al. [2] conducted a further detailed study questioning the performance of embedding models in the presence of data leakage and data redundancy. The study found a sizeable percentage of inverse, duplicate, and Cartesian product relations in the popular datasets FB15k, WNRR and Yago3-10. Duplicate relations are relations with different names that share the same facts (e.g., *hasCitizenship* and *hasNationality*). Cartesian product relations are relations that hold between all instances of a class (e.g., *sameSpeciesAs*). Such relations can be predicted trivially. Hence, the authors argued, the performance of these models would be significantly worse for link prediction on actual unseen data in realistic settings. Their experiments analysed various popular embeddings models including TransE, TransH, TransR, TransD, DistMult, ComplEx, ConvE, Tucker, and RotatE and showed substantial drops in performance with different datasets after removing the unrealistic triples, so much so that simple rule based techniques could achieve better accuracy than complex embedding techniques. The authors therefore strongly advocated the need to re-evaluate existing embedding approaches to find an effective solution for the link prediction task.

Rossi et al. [47] take a critical look at the properties of the entities in the benchmark datasets that are used to evaluate link prediction performance of embedding models. They focused on the Freebase and Wordnet datasets and performed a detailed experimental analysis of the features of these datasets and their limiting effect on the performance of embedding models. For instance, the authors showed that embedding models perform artificially better for the most frequent entities in the dataset. In FB15k, the entity *United States* appears in a lot of triples, and therefore, the TransE and DistMult models show better scores while predicting this entity as the missing entity. If the most frequent entities were removed from these datasets, the model performance (counter-intuitively) improved, indicating the over-fitting of the models on the most representative entities. Therefore, the authors advocated that better benchmarking practices and metrics are needed to determine the capability and fairness of the models.

Pujara et al. [42] performed an interesting study on the effect of sparsity and unreliable data on the performance of embeddings. Existing curated KGs like Wordnet and Freebase were modified in different experiments to introduce sparsity (in terms of relations or entities) and unreliable or corrupted triples, so that they resemble real-world KGs derived from text (such as NELL [8]). The authors found that performance is closely linked with sparsity, i.e. embeddings work well for relations and entities that have a dense representation and sparsity adversely affects their performance. Experiments showed that unreliable triples also degraded the performance. However, the authors made an interesting conclusion, namely that corrupted triples still improved embeddings marginally, therefore it is better to have a large noisy KG rather than a small set of very high quality triples.

These studies helped in bringing into focus the flaws that are inherent in all the popular benchmark datasets due to which global metrics for the evaluation of embedding models are proved to be insufficient and misleading. Thus, there is a need for careful and fine-grained evaluation of the performance of embedding models for their application in realistic use cases.

3.4.3 Shortcomings of the protocol

Several works have studied the shortcomings of the evaluation protocol for KB embeddings.

Pezeshkpour et al. [41] focused on the evaluation metrics and pointed out the need and importance of calibration of the embedding models before they can be deployed in real-world scenarios. For example, if the model says with 0.5 confidence that a triple is true, then the actual probability of the triples with this confidence should also be 0.5. In particular, they found that the model calibration as well as the ranking metrics were highly susceptible to the choice of negative sampling during training, with random replacement of subject or object entity *(Random-N)* leading to worst results. In order to improve the evaluation techniques, the authors proposed the *CarefulN* method to select negative samples. Here, the highest scoring negative sample having an entity type which is different from the target entity type is selected as a negative sample. E.g. given a triple $\langle$*Barack Obama*, *presidentOf*, *USA*$\rangle$, if *USA* is the target entity to be predicted, and the ranked list of predicted entities is *(USA, Hawaii, United Nations, Michelle Obama, . . .)*, then we choose $\langle$*Barack Obama*, *presidentOf*, *Michelle Obama*$\rangle$ as the negative sample since the type for *Michelle Obama* is different from *USA*. This technique explicitly ensures that the negative sample being generated is a true negative. Following this technique, they derived a new benchmark dataset Yago3-TC for evaluating KG completion that consists of both true and false facts for facilitating the correct measurement of triple classification accuracy.

Sun et al. [56] looked into the very specific issue of the recent neural-networks based embedding models showing inconsistent performance gains across different datasets such as FB15k-237 and WNRR18. They investigated in detail the models ConvKB [10] and CapsE [61] and found an unusual score distribution to be the reason for this discrepancy. For instance, many negatively sampled triples were given the same score as the correct triple. To break ties for the triples with the same score, they proposed a RANDOM evaluation protocol, i.e. if multiple triples have the same score, one of them is chosen randomly. Experiments demonstrated that recent deep models such as ConvKB, and CapsE were indeed affected by different evaluation protocols (unlike other models like ConvE and RotatE) and this could be detected with the proposed RANDOM protocol.

Kadlec et al. [26] were among the first to question the performance gains achieved by the newly proposed model architectures. The authors were able to perform suitable fine tuning the hyper-parameters for DistMult, one of the first embedding models proposed, and outperform several new models. This raised concerns on the performance gains by the newer models, advocating for a closer inspection of the training practices and objectives.

Ruffinelli et al. [48] re-implemented several existing models and performed extensive analysis of the performance of these models to compare them on a common ground. Going beyond previous works such as [36] (which studied the loss functions) and [29] (which looked into the negative sampling strategies), this paper performed a comprehensive and empirical evaluation of the effect of different training strategies and parameters such as the regularizer, optimizer and loss functions on a number of new and old embedding models. Their analysis indicated that the training parameters play a major role in the embedding performance. With a systematic fine tuning of these parameters, even the older model architectures such as RESCAL can match or outperform the recently introduced improved models. This work makes a strong point of the need for re-assessing the individual benefits claimed by recent and newer embedding models in light of the older models. The authors caution that the performance gain reported by newer models could be mitigated by merely fine tuning of the training strategies and therefore, this warrants close inspection.

Jain et al. [23] raised questions regarding the very semantic representation learned by the embeddings models in the first place. They performed classification and clustering experiments on the embeddings in order to analyse their semantic capability. The authors challenge the common notion that entities having similar meaning (i.e. belonging to the same class or type) such as *politicians*, *actors* etc. would be represented by similar vectors. They constructed a dataset with entities belonging to different levels of the taxonomy for Yago3-10 and DBpedia datasets, such as from *person* to *artist* to *painter*. Detailed experiments demonstrated that both clustering and classification showed poor results for entities having fine-grained classes. This means that embeddings are unable to capture the semantics of entities beyond the top level classes (*person*, *organization*, *places* in Yago). These surprising results indicated that though embeddings might show good performance on the link prediction task, their utility for other semantic tasks such as reasoning etc. should be carefully examined.

Wang et al. [64] inspected the evaluation protocol of the embedding techniques for the KB completion task. They argue that the Entity Ranking (ER) protocol, where the missing head or tail entity is predicted for a triple, is more suitable for evaluating a question answering task, but not for the KB completion task. This is due to the fact that the context of the missing information in terms of a head or tail entity would not be available when attempting to find new missing triples of the form $\langle ?, r, ? \rangle$. With the ER protocol, the models may not be penalized for ranking certain incorrect triples higher since they are not encountered at all. The paper instead proposes a Pairwise Ranking (PR) protocol where all possible entity pairs are considered and ranked with respect to a particular relation. Extensive experiments show that popular embedding models provide worse performance with PR protocol than with the ER protocol, even on seemingly easy datasets.

These studies have emphasized the need for better evaluation protocols and a critical look at the training strategies of the embedding models for the task of KB completion in realistic settings.

4 Embedding Methods with Logical Components

4.1 Rationale

Rule mining methods and embedding methods are complementary for the purpose of link prediction:

- Rule mining produces patterns that can be understood by humans. Thus, their predictions can be explained and justified.
- Rule mining can, in principle, work together with the schema of the KB, axioms, and other types of logical constraints.
- Rule mining methods can deal with literals and numerical values, while embedding methods typically project these away. Rule mining can find, e.g., that the death date is always later than the birth date.
- Rule mining is typically evaluated under the open world assumption: it is explicitly designed to predict facts that are not yet in the KB. Embedding methods, in turn, are typically tuned to predict what is already there.
- On the flip-side, rule mining methods typically predict based on a single rule; it is harder to predict facts with several rules that reinforce or contradict each other.
- Rule mining methods do not have a holistic view on an entity, with all its relations; they are restricted to the relations that appear in the rules.
- Rule mining methods often generate rules with a low confidence, i.e., with a high rate of false predictions.

For these reasons, several works have taken to combine symbolic and embedding methods. Here, the symbolic component does not necessarily come from Rule Mining, but can also come from the logical axioms of the ontology of the input KB. The existing approaches fall into three classes:

- **Adding simple axioms.** Some approaches constrain the embeddings by simple axioms, which concern inverse, symmetric, or equivalent relations. We discuss them in Sections 4.2 [12], 4.3 [15], and 4.4 [35].
- **Complex constraints.** Other approaches support more general and complex logical constraints on the embeddings. We discuss them in Section 4.5 [23], 4.6 [14], and 4.7 [63].
- **Joint learning.** Finally, a number of approaches jointly embeddings and confidences for rules. We discuss them in Section 4.8 [19] and 4.9 [20, 71].

4.2 Improving Knowledge Graph Embeddings Using Simple Constraints

A first simple combination of logical rules and embeddings is presented by Ding et al. [12]. The authors focus on relation entailments, i.e., rules of the form $\langle x, r, y\rangle \Rightarrow \langle x, r', y\rangle$ that can also be denoted as $r \Rightarrow r'$. For example, if two people are married, then they also know each other: $\mathit{marriedTo} \Rightarrow \mathit{knows}$. Such entailments can either be axioms from the ontology, or they can be soft rules, i.e., rules with a confidence score that do not hold in all instantiations. For example, a soft rule can be: If a person is born in a country, then the person probably has the citizenship of that country. This is very often the case though not always. Such soft rules can be mined by a rule mining system, and from now on, a set of such entailments is assumed to be available.

If it is known that a relation r entails a relation r' and that $\langle x, r, y\rangle$ holds for some entities x, y, then $\langle x, r', y\rangle$ holds. Thus, the score that an embedding model gives to the fact $\langle x, r, y\rangle$ should not be larger than the score it gives to $\langle x, r', y\rangle$:

$$f(\langle x, r, y\rangle) \leq f(\langle x, r', y\rangle) \tag{1}$$

The authors enforce this condition on the ComplEx model (see Section 3) by imposing that for a given entity x, all the real parts of the components of the embedding vector $\vec{x} \in \mathbb{C}^d$ have to be non-negative, and all the imaginary parts have to be smaller than or equal to one. Given an entity x, d the embedding dimension, the constraints are formalized in Equation 2 where $Re(\cdot)$ (resp. $Im(\cdot)$) returns the real (resp. imaginary) part of a complex number and $\vec{x}_i$ is the i^{th} component of the vector $\vec{x}$.

$$\forall i \in \{1, \ldots, d\} : Re(\vec{x}_i) \geq 0 \wedge Im(\vec{x}_i) \leq 1 \tag{2}$$

This constraint can be intuitively justified by seeing each component of $\vec{x}$ as a feature, whose value is zero if the feature does not apply to the entity x, and greater than 0 if the feature applies to x, but never below zero. The constraint on the imaginary component serves as a kind of normalization. With this non-negativity constraint, the desideratum of Equation 1 can be achieved by requiring:

$$\forall i \in \{1, \ldots, d\} : Re(\vec{r}_i) \leq Re(\vec{r'}_i) \wedge Im(\vec{r}_i) = Im(\vec{r'}_i) \tag{3}$$

If the entailment does not hold strictly, but only with a certain confidence, the condition can be relaxed by introducing a real-valued confidence level λ and vector slack variables $\vec{\alpha}, \vec{\beta}$, which turn Equation 3 into

$$\forall i \in \{1, \ldots, d\} : \lambda \times (Re(\vec{r}_i) - Re(\vec{r'}_i)) \leq \vec{\alpha}_i, \lambda \times (Im(\vec{r}_i) - Im(\vec{r'}_i))^2 \leq \vec{\beta}_i \tag{4}$$

The larger the confidence level λ, and the smaller the slack variables $\vec{\alpha}$ and $\vec{\beta}$, the more Equation 4 resembles the hard constraint of Equation 3.

When these constraints are imposed on the ComplEx model, then the model is forced to give a high score to facts that are logically entailed by other facts to which it gave a high score. The authors then show that this improves the performance of link prediction over the original model.

4.3 Improved Knowledge Graph Embedding Using Background Taxonomic Information

Fatemi et al [15] introduce another way to improve knowledge graph embeddings, which uses the taxonomy of the KB. For example, a knowledge base might contain the information that *Emmanuel Macron* is a *president*, but it does not contain information that he is a *mammal*, because it is implied by taxonomical knowledge. With this knowledge, if we know that mammals are warm-blooded, we can conclude that *Emmanuel Macron* is warm-blooded as well, without having explicit facts about this relation in the KG. Going one step further than relation entailment, this work leverages the subsumption property of the relations as well as the classes in KG. For example, the relation *presidentOf* is a sub-property of *managerOf*, which in turn is a sub-property of *employedBy*. Formally, if a relation r_1 is a sub-property of a relation r_2, then $\forall x, y : \langle x, r_1, y\rangle \Rightarrow \langle x, r_2, y\rangle$. To represent class subsumption, the authors model the entities as the characteristic functions of the class they belong to. This means that if entity e is in class C i.e. $\langle e, type, C\rangle$, then the characteristic function between e and C is true – written as $\langle e, C, true\rangle$. Hence, class subsumption can be expressed as a special case of relation subsumption. For instance, if *president* is subclass of *mammal* in the taxonomy, then $\langle EmmanuelMacron, president, true\rangle \Rightarrow \langle EmmanuelMacron, mammal, true\rangle$.

The proposed framework is a modification of the SimplE [27] embedding model (see Section 3.2), which makes use of these axioms. SimplE considers two embeddings for each relation: one embedding r^+ for relation itself and another r^- for its inverse relation. Similarly, there are two embeddings for each entity: one as a head entity e^+, and another as the tail entity e^-. These embeddings are concatenated to obtain the final embedding for a relation or entity. The proposed modification of this model is restricting the entity embeddings to be element-wise non-negative.

In order to enforce the axiom that a relation r is a sub-relation of a relation s ($\forall x, y : \langle x, r, y\rangle \Rightarrow \langle x, s, y\rangle$), the model adds an equality constraint as $\vec{r} = \vec{s} - \vec{\delta_r}$ where $\vec{\delta_r}$ is a non-negative vector, which expresses how r is different from s. This vector is learned during training. With this, the function μ (that maps embeddings to the probability of a triple) obeys the constraint $\mu\langle x, s, y\rangle \;\geq\; \mu\langle x, r, y\rangle$.

Thus, the resulting $SimpleE^+$ model is able to enforce subsumption properties for entities and relations and therefore, incorporate taxonomic knowledge in the embeddings to learn more interpretable representation for words [37]. The experimental evaluation shows that the proposed model is able to outperform traditional SimplE for the KG completion task and also has a faster convergence rate when taxonomic information is available.

4.4 Regularizing Knowledge Graph Embeddings via Equivalence and Inversion Axioms

We have so far seen approaches that concentrate on subproperty axioms. We shall now look into two other types of axioms [35]: Given two relations r_1 and r_2, an equivalence axiom $r_1 \equiv r_2$ means that r_1 and r_2 are semantically equivalent though distinct in the KB (e.g., *part of* and *component of*). An inverse axiom $r_2 \equiv \bar{r_2}$ means that r_1 is the inverse predicate of r_2 (e.g., *part of* and *has part*). The approach assumes that these axioms are defined in the ontology of the input KB.

Given the two sets of equivalence and inversion axioms, constraints are enforced in the training of the models. Let $r_1 \equiv r_2$ be an equivalence (resp. inversion) axiom. This means that relations r_1 and r_2 are equivalent (resp. inverse) and then the scoring function f of an embedding model should verify $f(\langle h, r, t\rangle) = f(\langle h, r_2, t\rangle)$ (resp. $f(\langle h, r, t\rangle) = f(\langle t, r_2, h\rangle)$) given any entities h and t.

In the case of equivalence, this is simply implemented by forcing the embeddings of r_1 and r_2 to be the identical. In the case of an inversion axiom, the constraint has to be specified for each model in the form of a model-dependent function Φ such that the constraint $\vec{r_2} = \Phi(\vec{r_1})$ results in $f(\langle h, r_1, t\rangle) = f(\langle t, r_2, h\rangle)$. For example, in the case of TransE [7], using $\Phi : \vec{r_2} \mapsto -\vec{r_1}$, one gets $f(\langle h, r_1, t\rangle) = ||\vec{h} + \vec{r_1} - \vec{t}|| = ||-\vec{h} - \vec{r_1} + \vec{t}||$ (by homogeneity of the norm) and then $f(\langle h, r_1, t\rangle) = ||\vec{t} + \Phi(\vec{r_1}) - \vec{h}|| = f(\langle t, r_2, h\rangle)$. Note that Φ needs to be an involution, i.e., $\forall r, \Phi(\Phi(r)) = r$.

These constraints are called *hard constraints* because they entirely determine some embeddings. Another possibility is to use soft constraints in order to enforce axioms that are not entirely true. For example *married with* and *partner of* are not entirely semantic equivalents but their embeddings are similar to one another. Intuitively the objective of soft constraints is to nudge the model to adopt some desired properties rather than enforce hard-coded requirements. This is done by adding two weighted terms to the usual training loss: $\hat{\mathcal{L}} = \mathcal{L} + \lambda \left[\sum_{r_1 \equiv r_2} ||\vec{r_1} - \vec{r_2}||_2^2 + \sum_{r_1 \equiv \bar{r_2}} ||\vec{r_2} - \Phi(\vec{r_1})||_2^2\right]$ where λ is an hyper-parameter that needs to be determined during training.

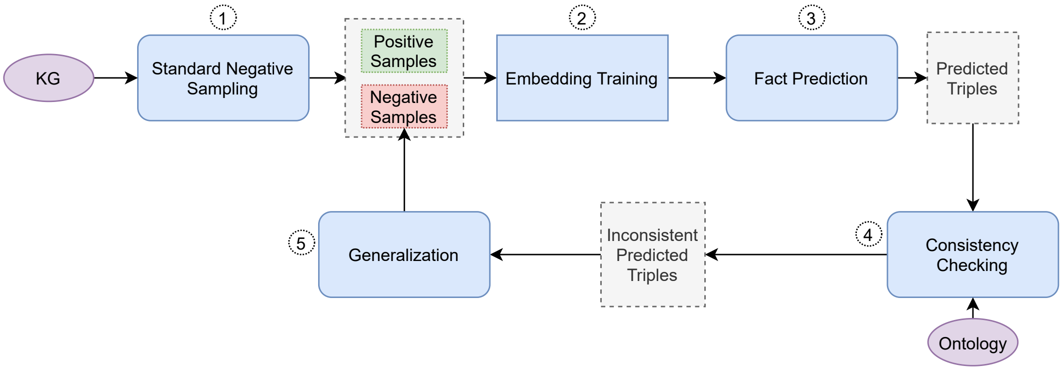


Figure 3 *ReasonKGE* Framework.

4.5 Improving Knowledge Graph Embeddings with Ontological Reasoning

Until now, we have concentrated mainly on very simple types of axioms to improve embeddings. ReasonKGE [24] is a method that can use complex constraints as well. The idea is to use symbolic reasoning to find predictions by the model that are logically inconsistent, and to feed these as negative samples into a retraining step. Traditionally, embedding methods generate negative triples by randomly replacing the head entity or tail entity in a triple from the KB (Section 2). This method, however, has two problems: First, as we have already discussed, it does not work as well for the head entities (Section 2.1). Furthermore, traditional methods do not necessarily create negative statements that violate domain and range constraints. For example, a triple such as ⟨*Elvis*, *hasNationality*, *Priscilla*⟩ cannot be true since *hasNationality* requires a country as object. If such triples are not generated as negative examples, the model may produce them as predictions. Therefore, ReasonKGE sets out to generate negative examples by axioms – inspired by the NELL system [8], which also uses axioms for the generation of examples.

The framework of the proposed method is shown in Figure 3. The inputs of the framework are the KG and its ontology, whereas the outputs are negative samples, which can then be used for training the model in the next iteration. The first iteration simply generates the baseline model with a default sampling method. Here, traditional sampling methods are used to generate the negative facts, and the model is trained based on positive and negative facts to obtain the predictions. The predicted triples are checked for inconsistencies with respect to the underlying ontology with the help of a reasoner. The inconsistent triples are then generalized to other semantically similar triples which would also cause the same inconsistencies. Lastly, all the generated negative samples are fed back to the model for the next iteration of training. With each round of training, the model learns to identify inconsistencies and therefore make more consistent predictions.

The **consistency checking procedure** (step 4) is one of the main steps, that detects which predictions made by embedding model are inconsistent with the original KG ($\mathcal{G}$) and ontology $\mathcal{O}$. For computational purposes, this check is done only on the subset of relevant facts. The *relevant set* is defined as follows:

$$Relv(\alpha, \mathcal{G}) = \{\alpha\} \cup \{\beta \in \mathcal{G} | Ent(\beta) \cap Ent(\alpha) \neq \varnothing\} \tag{5}$$

Here, α is the predicted triple and β are triples in the KG. For example, consider α = ⟨*Samsung*, *locatedIn*, *Emmanuel Macron*⟩. For this prediction the relevant set could consist of the following triples:

$$\begin{aligned} Relv(\alpha, \mathcal{G}) = \{ & \langle Emmanuel\ Macon, livesIn, France\rangle, \\ & \langle Emmanuel\ Macron, spouse, Brigitte\ Macron\rangle, \\ & \langle Emmanuel\ Macron, type, person\rangle, \\ & \langle Samsung, type, company\rangle\} \end{aligned}$$

It is sufficient to perform consistency checking on $Relv(\alpha, \mathcal{G}) \cup \mathcal{O}$, instead of $\alpha \cup \mathcal{G} \cup \mathcal{O}$. In our example, a reasoner would flag this prediction as inconsistent, because the ontology tells us that *locatedIn* requires a location, and hence ⟨*Samsung*, *locatedIn*, *Emmanuel Macron*⟩ implies that the type of *Emmanuel Macron* is *location.* That contradicts the fact ⟨*Emmanuel Macron*, *type*, *person*⟩, together with the axiom that states that people and locations are disjoint. Thus, this triple can serve as a negative sample.

Further, in Step 5, the negative samples obtained via consistency checking are fed to a **generalization** module to obtain multiple similar inconsistent facts that have a similar structure within KG. This is beneficial in 2 ways: firstly, by generating several negative samples that cause the same inconsistency, the model would be able to learn the inconsistency pattern and thus, the prediction of similar incorrect triples in next training iterations would be avoided. Secondly, it enables us to obtain sufficient number of negative samples for a given triple during the training of the model. The generalization of inconsistent predictions is done in the following way: in an inconsistent predicted fact $\langle h, r, t\rangle$, t can be replaced with another entity k that has similar triples. For example, if $\alpha = \langle$*Samsung*, *locatedIn*, *Emmanuel Macron*⟩ is a predicted inconsistent triple, then we can take $\alpha = \langle Samsung, locatedIn, Joe\ Biden\rangle$ as another negative example if *Joe Biden* has the same neighbour triples as *Emmanuel Macron*, i.e. ⟨*Joe Biden*, *type*, *person*⟩.

The authors show experimentally that *ReasonKGE* achieved better results on the link prediction task as compared to traditional methods (TransE, ComplEx). Experiments conducted on the Yago3-10 dataset were particularly significant, as the model achieved more than 10% improvement for all the measures as compared to TransE. Additionally, *ReasonKGE* reduced the ratio of inconsistent predictions over the test set when compared to other models that employ static or random sampling techniques. A limitation of this method is the use of DL-Lite [3] ontologies, due to which, theoretically, not all possible similar negative samples will be obtained based on a given inconsistent prediction in the generalization step.

4.6 Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs

Similar to *ReasonKGE*, this paper proposes to improve KG embeddings by injecting available background knowledge in the form of ontological axioms [14]. The authors propose *TransOWL* and *TransROWL* models, as improved versions of the traditional embedding methods TransE and TransR respectively.

The injection of background knowledge during the training phase involves two main components - *reasoning* to add negative samples and *Background Knowledge (BK) injection* to add constraints on the scoring function.

- During **reasoning**, negative samples are generated by leveraging the ontological properties such as *domain*, *range*, *disjointWith*, *functionalProperty* with the help of the Apache Jena framework. For example, if a particular entity type (or concept in ontology terminology), let's say *location* is disjoint with another type e.g. *person*, then negative samples are generated by replacing the person entity in a triple with all location entities present in the

KG. Thus, for a triple $\langle Samsung, locatedIn, South\ Korea\rangle$, a list of negative samples can be generated by replacing $South\ Korea$ with *Joe Biden*, *Barack Obama*, *John Smith* and so on.

- During **BK injection**, ontological properties such as *equivalentClass*, *equivalentProperty*, *inverseOf* and *subClassOf* are applied for the definition of additional constraints on the scoring function such that resulting embedding vectors can reflect these properties. New triples corresponding to these properties are generated and added to the training set of the model. For example, for the *equivalentClass* property, if class A is equivalent to class B, then for a triple $\langle entity1, type, A\rangle$, it is possible to generate another triple $\langle entity1, type, B\rangle$ as well. Similarly this is performed for other properties as well and a considerable number of additional triples is generated before training the model.

The basic loss function for TransE is defined as

$$\sum_{\langle h,r,t\rangle\in\Delta,\langle h',r,t'\rangle\in\Delta'} [\gamma + f_r(t,h) - f_r(t',h')] \tag{6}$$

here $\gamma\ \geq 0$ is the hyperparameter *margin.* For TransOWL, this loss function is more complex due to the additional constraints from the axioms. For example, the addition of the the *inverseOf* axiom would add a term to the loss function as

$$\sum_{\langle t,s,h\rangle\in\Delta,\langle t',s,h'\rangle\in\Delta'} [\gamma + f_s(t,h) - f_q(t',h')] \tag{7}$$

where f is the scoring function, Δ refers to the set of additional triples generated by a reasoner and s is the inverse relation of r. Similarly, the constraints are added in the loss function for the other axioms as well. Experimental evaluation shows that the models generated through this procedure show improvement for link prediction as well as triple classification in KGs as compared to the original TransE and TransR models.

4.7 Knowledge Base Completion Using Embeddings and Rules

In [63], the authors propose to constrain knowledge graph embeddings by an altogether different type of axioms: cardinality axioms. This is done by an Integer Linear Programming problem: the objective function is computed using the scoring function of an embedding model under the constraints from the symbolic axioms.

Let the $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ and $\mathcal{R} = \{r_1, r_2, \ldots, r_m\}$ be the sets of entities and relations in a KG at hand. The linear problem is defined with mn^2 decision variables $\{x^k_{i,j}, 1 \leq i, j \leq n, 1 \leq k \leq m\}$ such that $x^k_{i,j}$ indicates whether the fact $\langle e_i, r_k, e_j\rangle$ is true or false. The weight of a triple is computed using the scoring function f of an embedding model. This results in an objective function of the form:

$$\max_{x^k_{i,j}} \sum_i \sum_j \sum_k f(\langle e_i, r_k, e_j\rangle) \cdot x^k_{i,j}$$

The constraints of this optimization problem are derived from four types of rules:

- **Type 1: noisy observation**. Observed triples are very likely to be true but KBs are prone to noise. In order to take into account the rare cases in which an observed fact is false, slack variables $\epsilon^k_{i,j}$ are introduced for each observed triple and the R1 constraint is added along with a penalization term in the objective function. This is a classical method in linear programming, which allows the easy identification of noisy triples.

- **Type 2: argument type expectation**. Some predicate-specific type constraints should be respected by the head and tail entities. This results in the R2 constraint in which $\mathcal{S}_k$ (resp. $\mathcal{O}_k$) contain the indexes of the entities that have the type of the head (resp. tail) of the relation r_k.
- **Type 3: at-most-one restraint**. Some relations can handle at most one head per tail (many-to-one) or one tail per head (one-to-many). For example, the relation *cityLocatedInCountry* is a one-to-many relation meaning that a city can be located in at most one country. Other relations are one-to-one. Those three types of relations result in three constraints R3.1, R3.2 and R3.3 in which $\mathcal{R}_{1-M}$, $\mathcal{R}_{M-1}$ and $\mathcal{R}_{1-1}$ are respectively the sets of one-to-many, many-to-one and one-to-one relations.
- **Type 4: simple implication**. A relation r_1 can imply another relation r_2, if $\langle x, r_1, y\rangle \Rightarrow \langle x, r_2, y\rangle$ for any entities x and y. It is denoted $r_1 \Rightarrow r_2$. This gives us the constraint R4.

With this, the final Integer Logic Program is:

$$\max_{x^k_{i,j}} \sum_i \sum_j \sum_k f(\langle e_i, r_k, e_j\rangle) \cdot x^k_{i,j}$$

$$(R1)\ x^k_{i,j} + \epsilon^k_{i,j} = 1, \forall (i,j,k) : \langle e_i, r_k, e_k\rangle \text{ is observed}$$

$$(R2)\ x^k_{i,j} = 0, \forall k, \forall i : e_i \notin S_k, \forall j : e_j \notin O_k$$

$$(R3.1)\ \sum_i x^k_{i,j} \leq 1, \forall k : r_k \in \mathcal{R}_{1-M}, \forall j$$

$$(R3.2)\ \sum_j x^k_{i,j} \leq 1, \forall k : r_k \in \mathcal{R}_{M-1}, \forall i$$

$$(R3.3)\ \sum_i x^k_{i,j} \leq 1, \sum_j x^k_{i,j} \leq 1, \forall k : r_k \in \mathcal{R}_{1-1}, \forall i, \forall j$$

$$(R4)\ x^{k_1}_{i,j} \leq x^{k_2}_{i,j}, \forall k_1, k_2 \text{ s.t. } r_{k_1} \Rightarrow r_{k_2}$$

$$\text{where } x^k_{i,j} \in \{0,1\}, \forall i,j,k;\ \epsilon^k_{i,j} \in \{0,1\}, \forall (i,j,k) : \langle e_i, r_k, e_k\rangle \text{ is observed}$$

In spite of their promising results, the authors highlight two main limitations to this approach. First, constraints do not take into account the possible many-to-many relations, possibly missing out some ontology information. Second, solving the integer linear programming problem is time consuming and the approach then lacks scalability. In this regard, the authors propose a divide-and-conquer strategy for future work.

4.8 Jointly embedding KGs and Rules

So far, we have constrained embeddings by axioms. There are, however, also approaches that use soft rules instead of axioms, and that learn embeddings jointly with confidence scores for these soft rules. The first of these [19] improves the training procedure of the TransE model [7] by a new training loss that integrates both observed triples and groundings of some logical rules. The method focuses on rules of only two shapes: $\forall x, y,\ \langle x, r_1, y\rangle \Rightarrow \langle x, r_2, y\rangle$ and $\forall x, y,\ \langle x, r_1, y\rangle \cap \langle y, r_2, z\rangle \Rightarrow \langle x, r_3, z\rangle$, where r_1, r_2 and r_3 are relations from the graph.

Following Rocktäschel et al. [45], the truth value of a grounded rule is computed from the truth values of the constituent triples and t-norm logic principles. For this, the truth value of a single triple is first defined as:

$$f(\langle x, r, y\rangle) = 1 - \frac{1}{3\sqrt{d}} ||\vec{x} + \vec{r} - \vec{y}||$$

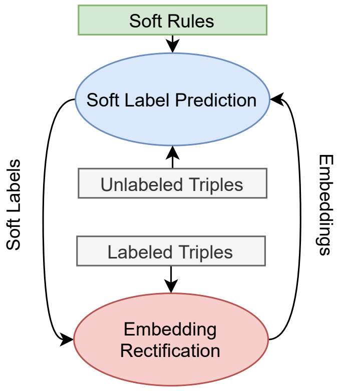


Figure 4 RUGE.

This is simply a normalization of the TransE [7] scoring function. To compute the truth value of more complicated logical formulae, this definition has to be broaden to negation, conjunction, and disjunction. The truth value of a negated triple $\neg p$ is simply $1 - f(p)$. The truth value of a conjunction is given by a t-norm, i.e., a function that is commutative, associative, and monotonous, and that has 1 as the identity element. The work of [19] uses simply the product as the t-norm, i.e., $f(p \wedge q) = f(p) \times f(q)$. With this, the truth value f of an implication is

$$\begin{aligned} f(p \Rightarrow q) &= f(\neg p \vee q) = f(\neg(p \wedge \neg q)) \\ &= 1 - f(p) \times (1 - f(q)) \\ &= 1 - f(p) + f(p) \times f(q) \end{aligned}$$

The only question left is how to generate the logical rules that are taken as input of this improved training procedure. A natural method could be to run a logical approach such as AMIE or RuDiK [17, 40]. The authors of [19] have a different approach that uses their method of scoring rule groundings to select the best ranking rules in a greedy manner.

4.9 Knowledge Graph Embedding with Iterative Guidance from Soft Rules (RUGE)

The Rule-Guided Embedding (RUGE) algorithm [20] is another method that learns embeddings jointly with confidence scores for logical rules. Its main steps are shown in Figure 4. The system starts out with soft rules (top of the figure), mined by the AMIE system [17]. These are instantiated to make predictions (see Section 2.1) – each with a confidence. The Embedding Step (bottom of the figure) takes as input the predictions of the rules as well as labeled triples from the KB. The embedding is trained on these two sources. This allows the prediction of new facts, which will in turn predict new facts by help of the rules. This process is iterated, thereby amplifying automatically the number of labeled examples.

The rule mining system gives each rule a confidence. However, this confidence concerns the rule as a whole, not an individual grounded variant of the rule, where all variables are instantiated. To compute the confidence of an individual grounded rule, the approach uses the scores $\phi(\cdot)$ of the embeddings of the facts that appear in the rule, as well as the score $s(\cdot)$ of the fact that the rule predicts, and proceeds according to the definitions of t-norm based fuzzy logics, in much the same way as [19].

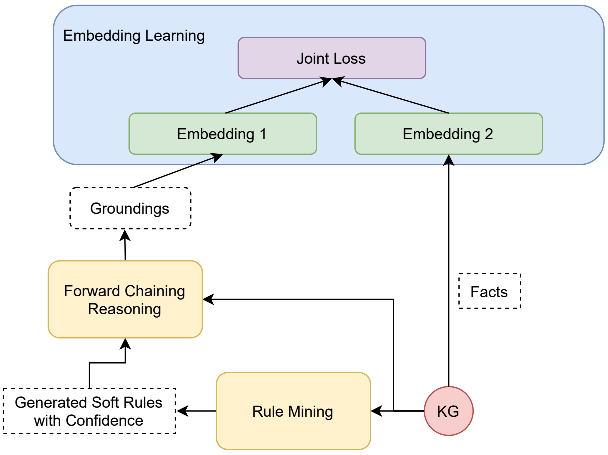


Figure 5 SoLE Architecture (stage 1 in yellow, stage 2 in blue).

The approach then aims to find a scoring function $s(\cdot)$ that is as close as possible to the current scoring function $\phi(\cdot)$, while at the same time making the confidences $\pi(\cdot)$ of all grounded rules as close as possible to 1 (the maximum). This is done by solving an optimization problem. This yields scores $s(\cdot)$ for facts that are predicted by the rules.

In the second step, the approach then corrects its embeddings $\phi(\cdot)$ so that they mirror (1) the truth value of facts that appear already in the KB and (2) the score $s(\cdot)$ for facts that do not appear in the KB, but were predicted by the rules. This updated embedding is then fed again into the rules, and the process is iterated. Experiments show that this method achieves significant improvements in link prediction task on Freebase and YAGO.

A variant of this approach is the SoLE system [71] ("Soft Logical rules enhanced Embeddings"), whose architecture is shown in Figure 5. Like RUGE, SoLE takes as input a KB and rules. It uses the rules to predict new facts in an iterative manner until no more facts can be predicted (a technique called *forward chaining*). The rules are then grounded, and a confidence score is computed for each grounded rule, not unlike this is done in RUGE as well. Different from RUGE, SoLE then minimizes a joint loss so as to find embeddings that can (1) predict the labels of triples contained the KB, while also (2) imitating the confidences of rules.

5 Rule Mining with embedding techniques

In the previous section, we have discussed several embedding methods that use logical techniques to improve their performance. The other direction is much less common: there are few methods that use embedding models in order to improve logical rule mining techniques. We will now present the most prominent ones.

5.1 ILP Rule Mining

A first small application of embeddings for rule mining is presented in an extension of RuDiK [40] by Ahmadi et al. [1]. The new system can also mine rules about class membership, such as "Politicians are not married to officeholders of a different party":

$$[\langle x, party, x_p\rangle \wedge \langle y, party, y_p\rangle \wedge x_p \neq y_p \\ \wedge \langle x, type, Politician\rangle \wedge \langle y, type, \mathit{Office\ Holder}\rangle] \Rightarrow \neg\langle x, spouse, y\rangle$$

The question is now what classes should be considered in such rules. Considering all classes may lead to rules that are too fine-grained. It would also be inefficient. Using only the top-level classes, in contrast, may miss out on useful rules that hold in a subclass.

RuDiK therefore clusters the instances of the KB. The method of choice here are entity embedding methods. The authors observe that the clusters obtained this way are more uniform in what concerns the structural similarity of entities (i.e., the outgoing relations that they share) than class membership. This is because two entities with different relations can belong to the same class, and entities with the same relations can belong to different classes. The embedding, in contrast, groups entities by their relations, which is more amenable to the rule mining.

It turns out that entities with popular classes, such as *Person*, can be spread across multiple clusters, but classes with finer granularity, such as *Politician* and *OfficeHolder* are grouped together. For each cluster, RuDiK then determines a class (e.g., the class that most entities in the cluster belong to). This class is then used for mining rules such as the one above.

5.2 Few-shot learning for label propagation

As stated in Section 2.1, a recurrent problem when working on KBs is the lack of negative statements. That makes it difficult to classify a prediction of any model. In an ideal situation, an operator would be available during training in order to manually tag generated facts as positive or negative. This is rarely the case because it is very costly but it could be very useful in the generation of false statements for example. This problem of manually tagging samples (here triples) is not specific to KB processing and it has given birth to a field of research called few-shot learning. This is the study of learning algorithms that work on a very small number of samples. It often applies in fields were the creation of supervision labels is costly, for example computer vision.

In [32] the authors propose a few-shot rule-based knowledge validation framework that uses an embedding model (HypER [5]) in order to propagate the decisions of a human operator to whom triples to tag are submitted. The goal of the method is to enrich the KB with positive and negative examples that allow a better evaluation of a set of rules. The proposed *propagation* method relies on a measure of similarity between facts. To compute the similarity, a vector representing each triple is computed by concatenating the embeddings of the entities. The propagation of manual labels is done locally to triples sharing the same relation, and so their embedding is omitted in the concatenation. For example, let's say that an operator labeled the fact $\langle \mathit{Barack\ Obama}, \mathit{marriedTo}, \mathit{Sasha\ Obama} \rangle$ as false, this label is going to be propagated to triples involving *Barack Obama* and *MarriedTo* or *MarriedTo* and *Sasha Obama* that are similar enough to the initial one. Eventually the set of manually labeled triples along with the automatically labeled ones improve the evaluation process of the rules. The authors apply their method to rules mined with AMIE [17] and RuDiK [40]. The proposed method uses HypER as embedding model but the authors insist on the fact that any model can be used for this task.

5.3 Approximate algorithms

AMIE [17] is an exhaustive rule mining system, i.e., it finds all rules above user-specified confidence and support thresholds. This makes AMIE quite heavy to run on large knowledge bases. AMIE+ [18] improved the runtime by approximating the computation of the confidence value of rules. This comes at a minor cost in the precision of the algorithm but allows reducing the computation time by several orders of magnitude.

Another way to speed up the rule mining is by *sampling*. The underlying intuition is that a rule of the form $\langle e, r_1, e_1\rangle \wedge \langle e_1, r_2, e_2\rangle \wedge \dots \wedge \langle e_n, r_{n+1}, e'\rangle \Rightarrow \langle e, r, e'\rangle$ can be seen as the co-occurrence in the knowledge graph (KG) of two paths from e to e': one of length 1 (passing through the relation r), and one of length $2n+1$ (through the entities e_1, e_2, $\dots$, e_n and the relations r_1, r_2, $\dots$, r_{n+1}). Exploring the possible rules then comes down to finding possible paths from one entity to another. This graph exploration is computationally expensive, and so the authors of [39] propose a two-step acceleration of the graph exploration and of the evaluation of the rules.

- First, the size of the graph is reduced by *sampling* the KG. Given a relation r that should appear in the head atom of the rule and a maximum length $l \geq 2$, the neighborhood of r is computed iteratively. We start from a set E_0, which includes any entity involved in a fact with r. We then compute E_i for $1 \leq i \leq l-2$ by including entities linked to some entity of E_{i-1} by any predicate. The neighborhood of r is then defined as $\mathcal{N}(r) = \cup_{i=0}^{l-2} E_i$ and it includes all entities relevant to find paths of maximum length l and then rules involving l atoms in the body and p in the head.
- Subsequently, instead of exhaustively exploring all the possible paths in the neighborhood of the relation r, the authors suggest to use a bilinear embedding model to learn matrix representations of relations (see Section 3.2). A relation path $r_1, r_2, \dots, r_l$ in the graph can then be represented as the product of the matrices of the relations $M_{r_1} \cdot M_{r_2} \cdot \dots \cdot M_{r_l}$. The similarity between the path (corresponding to the body of the potential rule) and r is computed using the matrix Frobenius norm $sim(r, [r_1, r_2, \dots, r_l]) = \exp(-||M_r - M_{r_1} \cdot M_{r_2} \cdot \dots \cdot M_{r_l}||_F)$.

The authors compare their approach to AMIE+, and show that the new approach mines more rules, and rules of better quality in terms of confidence. Furthermore, the process is much faster for rules that have the shape of paths.

6 Conclusion

Knowledge Bases (KBs) find many uses in AI applications, such as personal assistants, question answering systems, or text analysis. And yet, KBs are usually incomplete and miss facts. Two avenues of research have taken to predict missing facts: a symbolic one, based on rule mining, and a neural one, based on embeddings. Each of them has their respective strengths, and in this article we have presented an overview of both. We have also discussed recent studies on the criticism of the benchmark and protocols used during evaluation of embedding models. We have then presented approaches that successfully combine both symbolic and neural methods to perform fact prediction in KBs. While there are several approaches that use rules in order to improve embeddings, there are rather few approaches that use embeddings to improve rule mining. This may thus be an interesting direction for further research.

References

1 Naser Ahmadi, Viet-Phi Huynh, Vamsi Meduri, Stefano Ortona, and Paolo Papotti. Mining expressive rules in knowledge graphs. *Journal of Data and Information Quality (JDIQ)*, 12(2):1–27, 2020.

2 Farahnaz Akrami, Mohammed Samiul Saeef, Qingheng Zhang, Wei Hu, and Chengkai Li. Realistic re-evaluation of knowledge graph completion methods: An experimental study. In *ACM SIGMOD*, 2020.

3 Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The dl-lite family and relations. *Journal of artificial intelligence research*, 36:1–69, 2009.

4 Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, 2007.

5 Ivana Balažević, Carl Allen, and Timothy M Hospedales. Hypernetwork knowledge graph embeddings. In *ICANN*, 2019.

6 Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

7 Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, 2013.

8 Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

9 Yuanfei Dai, Shiping Wang, Neal N Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, 2020.

10 Tu Dinh Nguyen Dai Quoc Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *NAACL*, 2018.

11 Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

12 Boyang Ding, Quan Wang, Bin Wang, and Li Guo. Improving knowledge graph embedding using simple constraints. *arXiv preprint*, 2018. `arXiv:1805.02408`.

13 Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *ACM SIGKDD*, 2014.

14 Claudia d'Amato, Nicola Flavio Quatraro, and Nicola Fanizzi. Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In *ESWC*, 2021.

15 Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. Improved knowledge graph embedding using background taxonomic information. In *AAAI*, volume 33, 2019.

16 Jun Feng, Minlie Huang, Mingdong Wang, Mantong Zhou, Yu Hao, and Xiaoyan Zhu. Knowledge graph embedding by flexible translation. In *KR*, 2016.

17 Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.

18 Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. In *VLDBJ*, 2015.

19 Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *EMNLP*, 2016.

20 Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules. In *AAAI*, 2018.

21 Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. Learning to represent knowledge graphs with gaussian embedding. In *CIKM*, 2015.

22 Frank Lauren Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6:164–189, 1927.

23 Nitisha Jain, Jan-Christoph Kalo, Wolf-Tilo Balke, and Ralf Krestel. Do embeddings actually capture knowledge graph semantics? In *ESWC*, 2021.

24 Nitisha Jain, Trung-Kien Tran, Mohamed H Gad-Elrab, and Daria Stepanova. Improving knowledge graph embeddings with ontological reasoning. In *ISWC*, 2021.

25 Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, 2015.

26 Rudolf Kadlec, Ondřej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *RepL4NLP*, 2017.

27 Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *NeurIPS*, 2018.

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

29 Bhushan Kotnis and Vivi Nastase. Analysis of the impact of negative sampling on link prediction in knowledge graphs. *arXiv preprint*, 2017. `arXiv:1708.06816`.

30 Jonathan Lajus, Luis Galárraga, and Fabian M. Suchanek. Fast and Exact Rule Mining with AMIE 3. In *ESWC*, 2020.

31 Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2015.

32 Michael Loster, Davide Mottin, Paolo Papotti, Jan Ehmüller, Benjamin Feldmann, and Felix Naumann. Few-shot knowledge validation using rules. In *TheWebConf*, 2021.

33 Farzaneh Mahdisoltani, Joanna Asia Biega, and Fabian M. Suchanek. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*, 2015.

34 Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. An introduction to anyburl. In *KI*, 2019.

35 Pasquale Minervini, Luca Costabello, Emir Muñoz, Vít Nováček, and Pierre-Yves Vandenbussche. Regularizing knowledge graph embeddings via equivalence and inversion axioms. In *ECML PKDD*, 2017.

36 Sameh K Mohamed, Vít Novácek, Pierre-Yves Vandenbussche, and Emir Muñoz. Loss functions in knowledge graph embedding models. *DL4KG@ ESWC*, 2377:1–10, 2019.

37 Brian Murphy, Partha Talukdar, and Tom Mitchell. Learning effective and interpretable semantic models using non-negative sparse embedding. In *COLING*, 2012.

38 Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, 2011.

39 Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. An embedding-based approach to rule learning in knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1348–1359, 2021.

40 Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust discovery of positive and negative rules in knowledge bases. In *ICDE*, 2018.

41 Pouya Pezeshkpour, Yifan Tian, and Sameer Singh. Revisiting evaluation of knowledge base completion models. In *AKBC*, 2020.

42 Jay Pujara, Eriq Augustine, and Lise Getoor. Sparsity and noise: Where knowledge graph embeddings fall short. In *EMNLP*, 2017.

43 Simon Razniewski, Hiba Arnaout, Shrestha Ghosh, and Fabian M. Suchanek. Completeness, Recall, and Negation in Open-World Knowledge Bases. In *VLDB*, 2021.

44 Simon Razniewski, Fabian M. Suchanek, and Werner Nutt. But What Do We Actually Know? In *AKBC workshop*, 2016.

45 Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *NAACL*, 2015.

46 Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021.

47 Andrea Rossi and Antonio Matinata. Knowledge graph embeddings: Are relation-learning models learning relations? In *EDBT/ICDT*, 2020.

48 Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *ICLR*, 2019.

49 Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, 2019.

50 Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018.

51 Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *AAAI*, 2019.

52 Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*, 2013.

53 Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago - A Core of Semantic Knowledge . In *WWW*, 2007.

54 Fabian M. Suchanek, Jonathan Lajus, Armand Boschin, and Gerhard Weikum. Knowledge Representation and Rule Mining in Entity-Centric Knowledge Bases. In *RW*, 2019.

55 Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2018.

56 Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *ACL*, 2020.

57 Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *CVSC workshop*, 2015.

58 Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

59 Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *ICLR*, 2019.

60 Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

61 Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen, Dinh Phung, et al. A capsule network-based embedding model for knowledge graph completion and search personalization. In *NAACL*, 2019.

62 Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

63 Quan Wang, Bin Wang, and Li Guo. Knowledge base completion using embeddings and rules. In *ICOAI*, 2015.

64 Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, Samuel Broscheit, and Christian Meilicke. On evaluating embedding models for knowledge base completion. In *RepL4NLP*, 2019.

65 Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

66 Gerhard Weikum, Luna Dong, Simon Razniewski, and Fabian M. Suchanek. Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. In *Foundations and Trends in Databases*, 2021.

67 Alfred North Whitehead and Bertrand Russell. *Principia mathematica*. Cambridge University Press, 1913.

68 Han Xiao, Minlie Huang, Yu Hao, and Xiaoyan Zhu. Transg: A generative mixture model for knowledge graph embedding. *arXiv preprint*, 2015. `arXiv:1509.05488`.

69 Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint*, 2014. `arXiv:1412.6575`.

70 Rui Ye, Xin Li, Yujie Fang, Hongyu Zang, and Mingzhong Wang. A vectorized relational graph convolutional network for multi-relational network alignment. In *IJCAI*, 2019.

71 Jindou Zhang and Jing Li. Enhanced knowledge graph embedding by jointly learning soft rules and facts. *Algorithms*, 12(12):265, 2019.

72 Zhanqiu Zhang, Jianyu Cai, Yongdong Zhang, and Jie Wang. Learning hierarchy-aware knowledge graph embeddings for link prediction. In *AAAI*, 2020.

73 Xiang Zhao, Weixin Zeng, Jiuyang Tang, Wei Wang, and Fabian M. Suchanek. An Experimental Study of State-of-the-Art Entity Alignment Approaches . In *TKDE*, 2020.

Learning and Reasoning with Graph Data: Neural and Statistical-Relational Approaches

Manfred Jaeger
Aalborg University, Denmark

Abstract

Graph neural networks (GNNs) have emerged in recent years as a very powerful and popular modeling tool for graph and network data. Though much of the work on GNNs has focused on graphs with a single edge relation, they have also been adapted to multi-relational graphs, including knowledge graphs. In such multi-relational domains, the objectives and possible applications of GNNs become quite similar to what for many years has been investigated and developed in the field of statistical relational learning (SRL). This article first gives a brief overview of the main features of GNN and SRL approaches to learning and reasoning with graph data. It analyzes then in more detail their commonalities and differences with respect to semantics, representation, parameterization, interpretability, and flexibility. A particular focus will be on relational Bayesian networks (RBNs) as the SRL framework that is most closely related to GNNs. We show how common GNN architectures can be directly encoded as RBNs, thus enabling the direct integration of "low level" neural model components with the "high level" symbolic representation and flexible inference capabilities of SRL.

2012 ACM Subject Classification Computing methodologies → Logical and relational learning; Computing methodologies → Neural networks

Keywords and phrases Graph neural networks, Statistical relational learning

Digital Object Identifier 10.4230/OASIcs.AIB.2022.5

Category Invited Paper

1 Introduction

Learning and reasoning with graph and network data has developed as an area of increasing importance over recent years. Social networks, knowledge graphs, sensor and traffic networks are only some of the examples where graph structured data arises in important applications. Much of the attention currently focuses on *graph neural networks (GNNs)* as the technology for solving the challenges posed by this kind of data. While often very powerful in terms of scalability and predictive performance, graph neural networks suffer from the same drawbacks as other deep learning methods: lack of interpretability, limited support for the integration of prior domain knowledge, lack of robustness, and the inability to support more flexible reasoning than performing a specific task of prediction or synthetic graph generation. The field of *statistical relational learning (SRL)* has been concerned with learning and reasoning with graph and network data for over 20 years. Here the use of logic-based, symbolic representations and inference techniques, probabilistic graphical models, and relational database technology supports the construction of interpretable models via a combination of expert knowledge and machine learning, as well as a wide range of inference tasks, such as prediction and (most probable) explanations for varying and incomplete amounts of input data. On the other hand, SRL techniques lag behind GNNs in terms of scalability and predictive power in scenarios where the availability of extensive training data enables the training of the highly parameterized GNN models.

Combining the respective strengths of GNN and SRL technology is an emergent research area [49, 59, 69, 15]. Some works emphasize the complementarity of SRL and GNN approaches, and propose techniques that combine them in order to leverage the strengths of both [49, 59].

International Research School in Artificial Intelligence in Bergen (AIB 2022).
Editors: Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza; Article No. 5; pp. 5:1–5:42
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Overview of notation.

$G = (V, E)$	Graph with vertices (a.k.a. domain) V and edges E
n	Cardinality of V
N_i	Neighbors of node $i \in V$
$\mathcal{A}$	signature of node attributes
$\boldsymbol{A}$	$n \times \vert\mathcal{A}\vert$ matrix (table) of attribute values
$\mathcal{R}$	signature of relations of varying arities
$\mathcal{G}(V, \mathcal{R})$	Set of all graphs for signature $\mathcal{R}$ over domain V
$\boldsymbol{h}^k(i)$	Representation vector at level (layer) k for node $i \in V$
n_k	Dimension of vectors $\boldsymbol{h}^k(i)$
m	Number of hidden layers
$\{\!\!\{ \ldots \}\!\!\}$	Delimiters for multisets
P_V	Distribution over all graphs (for a given signature $\mathcal{R}$) with domain (vertex set) V
$F_{r(X_1,\ldots,X_k)}$	Probability formula defining the (conditional) probabilities for the k-ary relation r

In contrast, [69] directly uses relational logic as a high-level and flexible specification tool for neural architectures with a generic underlying training technique, thus essentially showing how a logic-based framework subsumes (graph) neural technology. In this article we, too, shall emphasize the overlap rather than the complementarity of SRL and GNN technology. We first give a brief overview of key aspects of modeling graph data using SRL and GNN approaches. Looking more closely at their respective semantics we then obtain a basis for establishing a direct correspondence between GNNs and some types of SRL models (Section 6), which gives rise to a direct encoding of GNNs as *relational Bayesian networks* (Section 7 and 9). We illustrate by examples the benefits of such an embedding of GNNs in a general SRL framework with regard to

- neuro-symbolic integration: the ability to combine symbolically specified (expert) knowledge with numerical optimization in neural architectures.
- flexible reasoning capabilities: the support provided by SRL frameworks for probabilistic reasoning beyond solving a fixed prediction task.

We consider in greater detail selected central themes that have played an important role in GNN and SRL research:

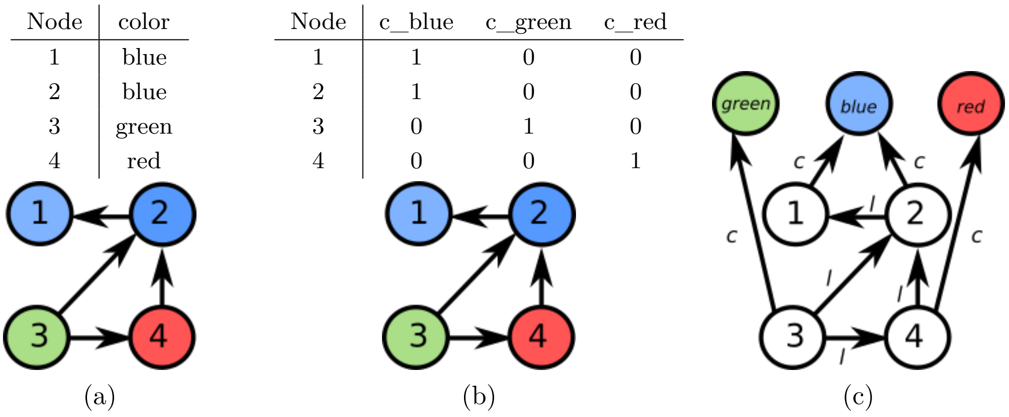
- Expressivity: what are the capabilities and limitations for the discriminative power of GNN/SRL models (Section 8)?
- Homophily: how do different approaches model and exploit the typical homophily properties in (social) networks?
- Aggregation: what operations can be used to aggregate information provided by related entities in a graph (Section 11)?

2 Graph Data

In its most basic form, graph data simply consists of a graph

$$G = (V, E) \tag{1}$$

defined by a set of vertices V, and a set of (directed or undirected) edges E. We shall limit ourselves to graphs with finite V, and use $n = |V|$ to denote its cardinality. We can then also assume that $1, \ldots, n$ are unique identifiers for the vertices, and we simply use



Node	color
1	blue
2	blue
3	green
4	red

Node	c_blue	c_green	c_red
1	1	0	0
2	1	0	0
3	0	1	0
4	0	0	1

(a) (b) (c)

Figure 1 Three representations of node attributes: (a) categorical attribute table, (b) one-hot attribute table, (c) multi-relational representation with binary attribute relation $c(olor)$ distinguished from original $l(ink)$ relation.

$i, j, \ldots \in \{1, \ldots, n\}$ to refer to elements of V. In an *attributed graph* the nodes are labeled with a set of attributes, which can be either Boolean, categorical or numeric. In a social network whose nodes represent people, for example, node attributes could be *gender* and *age*. We can write a graph whose nodes are labeled with k different attributes as

$$G = (V, E, \boldsymbol{A}), \tag{2}$$

where $\boldsymbol{A}$ is an $n \times k$ attribute matrix whose ith row contains the attribute vector for vertex i. The term “matrix” here is used in a loose sense: when some of the attributes are categorical, then corresponding columns in the matrix have symbolic values, and $\boldsymbol{A}$ would somewhat more appropriately be referred to as an attribute “table”. Using one-hot encodings of categorical attributes one can obtain a purely numerical attribute matrix $\boldsymbol{A} \in \mathbb{R}^{n \times k'}$. Figure 1 (a) and (b) illustrates these two alternative representations.

In many cases, graph models for real world networks require multiple edge relations. In a social network, for instance, there may be separate *friend* and *follower* connections between users. In a sensor or computer network, different types of connections (wireless, cable, ...) can be represented by different edge relations. This leads us to *multi-relational* graphs with several distinct edge relations. We write multi-relational graphs as

$$G = (V, \boldsymbol{E}, \boldsymbol{A}), \tag{3}$$

where now $\boldsymbol{E} = (E_1, \ldots, E_r)$ is a tuple of different edge relations. In the basic sense, edges are just either 'present' or 'absent', i.e., have a Boolean value. However, an edge labeling can also attach a categorical or numerical value to the edges, so that just like attributes, edge relations can be either Boolean, categorical, or numeric. Multi-relational graphs also provide an opportunity to represent node attributes as relations between the original graph vertices and attribute values that are materialized as additional nodes. This representation is illustrated in Figure 1 (c).

Going beyond binary edge relations, we can consider (directed) *hyperedges* $E \subseteq V^k$ for any $k \geq 3$. With few exceptions (e.g. [51]) hyperedges have not been considered in the GNN literature. From the predicate logic perspective of SRL, on the other hand, attributes, edges and hyperedges all are just relations with a certain *arity* $k \geq 1$. Under this uniform perspective of predicate logic, we can write an *attributed, multi-relational hypergraph* as

$$G = (V, \boldsymbol{R}) \tag{4}$$

where $\boldsymbol{R} = (R_1, \ldots, R_r)$ is a tuple of relations with arities $a_i \in \{0, 1, 2, 3, \ldots\}$ $(i = 1, \ldots, r)$, all of which can be of type Boolean, categorical, or numeric. We use relations of arity 0 as representations of global graph properties such as *connected*, or a label like *toxic* for a graph representing a chemical molecule.

Knowledge Graphs

Knowledge graphs are often presented as a set KG of *triplets* of the form $(\mathit{source}, \mathit{relation}, \mathit{target})$. Each such triplet describes a specific relationship between a source and a target entity. There are several ways to cast such a collection of triplets as a graph in one of the forms considered above. A first option is to consider each triplet as a hyperedge of a single relation of arity 3. For illustrative purposes, we may call this single relation the *fact* relation, and expand the triplet notation to explicitly write $\mathit{fact}(\mathit{source}, \mathit{relation}, \mathit{target})$. In this view, relations are also treated as nodes (entities). While this perspective is somewhat encouraged by the triplet notation for knowledge graphs, it is usually not the one that underlies graphical representations of knowledge graphs, where typically a triplet is illustrated by two nodes for the source and target entities, connected by an edge labeled with *relation*. The underlying graph model then is that of a multi-relational graph with binary relations, and the perhaps more appropriate notation for a triplet is the classical form of writing $\mathit{relation}(\mathit{source}, \mathit{target})$. Moreover, expressing everything as triplets also requires a representation of node attributes in the form illustrated in Figure 1 (c), i.e. in the form of relations $\mathit{attribute}(\mathit{entity}, \mathit{attribute\_value})$. A specific feature of knowledge graphs compared to other multi-relational graphs is the number of distinct relations, which can easily lie in the thousands. Multi-relational graphs representing more specif domains (such as bibliographic or social networks), in contrast, mostly contain only a relatively small number of relations. When considering the application of machine learning solutions for graph data to knowledge graphs, one must therefore consider not only the ability of a solution to deal with multi-relational graphs, but also their scalability in terms of the number of relations.

3 Graph Neural Networks

We here give a high-level summary of some key features of graph neural networks. For more complete and details surveys also covering types of GNNs not considered here (e.g. graph auto-encoders), the reader is referred to [74, 79].

Graph neural networks compute *representations* of the nodes as real valued vectors. Also often referred to as *embeddings* or *feature vectors*, these representations can then be the basis for supervised tasks like node classification, graph classification and link prediction, or unsupervised tasks such as graph clustering. Often the representations are learned in an “end-to-end” fashion to support a particular task, but they can also be constructed in a stand-alone process in order to support a variety of downstream applications (e.g. the unsupervised version of GraphSAGE [24]).

In the following, we first assume that $G = (V, E, \boldsymbol{A})$ is an attributed graph as in equation (2). In a very general, abstract form, the computation of the node representations proceeds in multiple steps, starting with initial representations

$$\boldsymbol{h}^0(i) \in \mathbb{R}^{n_0} \qquad (i \in V). \tag{5}$$

The initial representations can be collected in an $n \times n_0$ matrix, denoted $\boldsymbol{H}^0$. Using $\boldsymbol{H}^0[i, \bullet]$ to denote the ith row of $\boldsymbol{H}^0$, we then have $\boldsymbol{h}^0(i) = \boldsymbol{H}^0[i, \bullet]$. This gives us a bit of notational redundancy, which can be quite convenient, however. We will consider the choice of initial representations in more detail below. For now it may be helpful to think of the attribute vector $\boldsymbol{h}^0(i) = \boldsymbol{A}[i, \bullet]$ as the initial representation.

Then, given a representation $\boldsymbol{H}^k$ computed in the kth step, the $(k+1)$th representation is obtained as a function

$$\boldsymbol{H}^{k+1} = F^k(\boldsymbol{H}^k, G) \tag{6}$$

of the previous representation and the graph structure G. In most GNN architectures, the computation of the representation $\boldsymbol{h}^{k+1}(i)$ for node i only depends on the rows of $\boldsymbol{H}^k$ corresponding to i itself, and its neighbors. Then (6) can be expressed as follows:

$$\boldsymbol{h}^{k+1}(i) = F^k(\boldsymbol{h}^k(i), \{\!|\boldsymbol{h}^k(j)|j \in N_i|\!\}, G), \tag{7}$$

where N_i denotes the set of graph neighbors of i, and the delimiters $\{\!|\ldots|\!\}$ indicate that this is a multiset. Based on this dependence of the representation update of a node i on its neighbors, the basic GNN computations have been described using *information diffusion* or *message passing* metaphors [67, 21]. A fundamental distinction arises according to whether the updates (7) are performed in a *recurrent* or *fixed* architecture. In the former, the update function F^k is the same for all k, and updates are performed until a fixed point $\boldsymbol{H}^{k+1} = \boldsymbol{H}^k$ is reached (e.g.[67]). In the latter, updates are performed for a fixed number of steps $k = 0, \ldots, m$, and each step may be defined by a different update function F^k. In this article we focus on fixed architectures, which also constitute the majority of current GNN frameworks. In any case, a final node representation $\boldsymbol{H}^m$ is used to compute an output function. In node classification applications, an output is computed for each node i, based on that node's representation:

$$\boldsymbol{o}(i) = O(\boldsymbol{h}^m(i)). \tag{8}$$

When the task is graph classification, then an output is computed based on the representations of all nodes in the graph. This particular type of function is usually referred to as a *readout*:

$$\boldsymbol{r} = R(\{\!|\boldsymbol{h}(i)|i \in V|\!\}). \tag{9}$$

Figure 2 gives a schematic illustration of the computation graph for a fixed architecture with $m = 2$ steps, both for a node classification and a graph classification scenario.

Link prediction tasks can be addressed by GNNs in several ways. The most direct one is to use a GNN architecture as shown in Figure 2, but without the final output or readout layers. Instead, link prediction is directly performed based on the representations of the nodes, for example simply using the dot product $\boldsymbol{h}^m(i) \cdot \boldsymbol{h}^m(j)$ as a predictive score for the existence of a link between i and j [68]. A very different approach is to turn the link prediction problem into a graph classification problem for the "enclosing subgraph" of the candidate edge [78].

Equation (7) is only an abstract description of the computation of $\boldsymbol{H}^{k+1}$. All concrete functions implementing this description need to deal with the multiset argument $\{\!|\boldsymbol{h}^k(j)|j \in N_i|\!\}$ which differs in cardinality for different nodes i. In order to turn this into a fixed dimensional input an *aggregation* function such as *sum* or *mean* is usually applied. Making this aggregation step explicit, we can re-write (7) as

$$\boldsymbol{h}^{k+1}(i) = F_1^k(\boldsymbol{h}^k(i), agg\{\!|F_2^k(\boldsymbol{h}^k(j), G)|j \in N_i|\!\}, G), \tag{10}$$

which also allows that the representations $\boldsymbol{h}^k(j)$ are transformed by a function F_2^k before aggregation.This transformation may also depend on the graph structure G. This, for example, enables a scaling of $\boldsymbol{h}^k(j)$ before aggregation by a factor $1/\sqrt{d_j}$ with d_j the degree

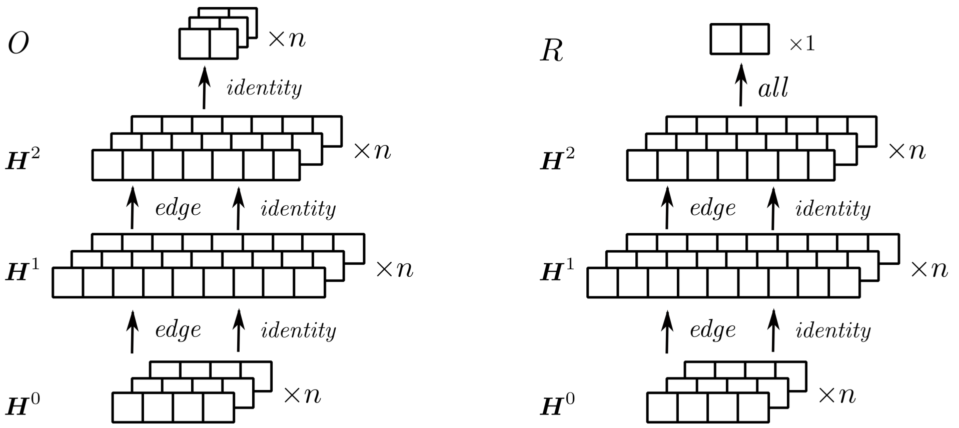


Figure 2 Computation graphs for basic fixed architecture showing the representation vectors $\boldsymbol{h}^k$ at each step, and indicating the relations *edge*, *identity* and/or *all* between nodes that defines the dependence of a representation at the next step on representations/output at the previous step. Left: architecture for node classification with one output per node; right: architecture for graph classification, with one global *readout* output function.

of j in G, which is required to cover the original graph convolutional network [41] within the formulation of (10). In most cases, the dependence of F_1^k, F_2^k on G will only be through the degrees of the nodes i, j, and often there is no such dependence at all.

The abstract functions F_1^k, F_2^k are defined in reality by neural network layers. These are usually quite simple in nature, and typically just consist of a linear function (layer) followed by a non-linear activation function.

Multi-Relational GNNs

The largest part of the GNN literature focuses on the case of attributed graphs with a single edge relation (2). Statistical relational learning, on the other hand, is concerned with rich structures as described by (4). In many applications, however, multi-relational graphs (3) are sufficient, and GNNs can quite easily be adapted to also deal with these [56, 68, 48]. The functional form (10) then becomes

$$\begin{aligned}\boldsymbol{h}^{k+1}(i) = F_1^k(\boldsymbol{h}^k(i), agg\{\!|F_{2,1}^k(\boldsymbol{h}^k(j), G)|j \in N_i^{E_1}|\!\}, \ldots, \\ agg\{\!|F_{2,r}^k(\boldsymbol{h}^k(j), G)|j \in N_i^{E_r}|\!\}, G), \quad (11)\end{aligned}$$

where now $N_i^{E_h}$ denotes the set of i's neighbors according to relation E_h.

We note here that we have limited our exposition to the most fundamental forms of GNN "message passing" architectures. This basic form has seen modifications and generalizations in many directions, including the addition of *attention mechanisms*[71], or *skip connections* that make computations at one layer dependent not only on the output of the previous layer. Most of the following considerations on the relationship between GNN and SRL models carry over to such more general forms of GNNs.

4 Statistical Relational Learning

Statistical relational learning (SRL) [20, 60] is a fairly diverse field encompassing different approaches to combine elements of relational logic representations, probabilistic graphical models, logic programming, and relational databases, for probabilistic reasoning and learning about entities and their relationships. SRL frameworks use the language of relational logic to represent basic facts in the form of *atomic expressions*, or *atoms* for short:

$$\mathit{friend}(\mathit{lars}, \mathit{giovanni}),\ \ \mathit{color\_green}(\mathit{emerald}),\ \ \mathit{connected}(X, \mathit{router\_48264}), \ldots$$

Generally, an atom is a relation symbol followed by a list of arguments with a length corresponding to the relation's *arity*. Arguments can be *constants* representing specific entities, or *variables*. A usual convention is that variables start with upper-case letters, whereas constants start with lower-case letters. Thus, in our examples above, X is a variable, and all other arguments are constants. An atom that only contains constants as arguments is called *ground*. It represents a specific fact that can either be true or false. In a pure predicate logic interpretation, atoms can only evaluate to Boolean values. Thus, the color of an entity would need to be represented in the form $\mathit{color\_green}(X)$ or $\mathit{color}(X, \mathit{green})$, as in Figure 1 (b) or (c). However, in SRL frameworks this can often be loosened to also allow categorical atoms such as $\mathit{color}(X) \in \{\mathit{red}, \mathit{green}, \mathit{blue}\}$, or numerical atoms such as $\mathit{length}(X) \in \mathbb{R}$. It is thus apparent that the fundamental building blocks of SRL frameworks describe multi-relational hypergraphs as in (4). However, depending on the SRL context and background, these kinds of structures also go by very different names, such as *Herbrand interpretations* [12], or *possible worlds* [62]. In order to maintain a close connection with the preceding sections, we shall here continue to speak about graphs (which always are understood to be multi-relational hypergraphs).

SRL frameworks define probability distributions over graphs. More specifically, consider a fixed set $\mathcal{R}$ of relations (of different arities). We also call $\mathcal{R}$ a *signature* of relation symbols. Let V be a finite set of vertices (more commonly referred to as a *domain* in SRL contexts), and let $\mathcal{G}(V, \mathcal{R})$ be the set of all graphs with vertex set V for the relations $\mathcal{R}$. An *SRL model* then defines a mapping that assigns to every finite set V a probability distribution over $\mathcal{G}(V, \mathcal{R})$ [34].

▶ **Example 1.** Classic random graph models such as the basic Erdős-Rényi model [17], the stochastic blockmodel [27], or the preferential attachment model [3], are SRL models in our sense with a signature $\mathcal{R}$ consisting of a single binary $\mathit{edge}(X, Y)$ relation (however, this does not mean that every SRL framework as detailed below is necessarily able to capture all of these random graph models). All these models define for every cardinality $|V| = n$ a probability distribution over all graphs with n nodes.

We call an *SRL framework* any specific system of representation and inference tools for SRL models. More specifically, an SRL framework provides:

- *Syntax and semantics*: a formal language for the specification of SRL models, and a semantic specification of the probability distribution that is defined by the model.
- *Inference*: general algorithms for computing for a given vertex set V, and two subsets $A, B \subseteq \mathcal{G}(V, \mathcal{R})$, the conditional probability $P_V(A|B)$ under the distribution P_V defined by the model for the domain V. Different frameworks will differ in how the subsets A, B can be defined, but all will usually allow to specify sets by ground atoms: $P_V(\mathit{republican(mary)}|\mathit{friends(mary,carl)}$ will then stand for $P_V(A|B)$, with A the set of all graphs where the (Boolean) attribute *republican* is true for entity *mary*, and B the

set of all graphs where the relation *friends* holds between the entities *mary* and *carl* (assuming that V contains entities *mary* and *carl*, and that the signature $\mathcal{R}$ contains the relations *republican* and *friends*).

- *Learning*: methods for statistical learning of an SRL model from data. One here distinguishes *parameter learning* and *structure learning*. The latter refers to learning the high-level, symbolic part of the model specification, and the former to fitting numeric parameters of the model.

Within the very diverse landscape of SRL frameworks, one can identify a number of major paradigms, which we briefly survey in the following (a slightly more detailed exposition along similar lines can be found in [32]). For the following we assume that all relations are Boolean or categorical, which is the original and main focus of SRL frameworks.

Bayesian Network Constructors

Here the actual distribution P_V for a given V is eventually represented by a Bayesian network whose nodes are all the ground atoms that can be formed from relations in $\mathcal{R}$ with entities from V. The SRL model provides a general blueprint for how such a Bayesian model representation is constructed for any domain V. Languages for defining such blueprints fall into two main categories: *rule based* and *graphical templates*. The basic building blocks of rule based approaches are logical implications between atoms, such as

$$\mathit{infected}(X) \leftarrow \mathit{contact}(X,Y) \tag{12}$$

which are further annotated with quantitative probabilistic information. The qualitative parts of the rules (as shown in (12)) then define the graphical dependency structure in the Bayesian network for P_V, whereas the probabilistic annotations (not shown) define the quantitative conditional probability specifications. As (12) illustrates, the rules will usually only contain variable symbols, not constants referring to specific entities, and thereby are applicable to arbitrary domains V. This brand of SRL frameworks has its roots in what originally was called *knowledge-based model construction* [7, 53], and is further represented by *Bayesian logic programs* [37] and *relational Bayesian networks* [29].

As illustrated by (12), the directed dependencies expressed by the rules often coincide with causal dependencies. However, it is not generally necessary that the rules have a causal background, or even that they comply with an existing causal direction (it would be perfectly valid, if rather counter-intuitive, to construct a model including a rule $\mathit{infected}(X) \leftarrow \mathit{fever}(X)$ that inverts the causal direction).

Template-based frameworks follow essentially the same modeling paradigm, but using graphical representations of Bayesian network fragments as the basic representational building block [46, 45]. Another paradigm for abstract graphical representations that can be compiled into Bayesian networks is based on entity-relationship diagrams as used in relational databases [18, 25].

Markov Network Constructors

This type of SRL frameworks is mostly represented by *Markov logic networks* [62], which are closely related to *exponential random graph models* that have a long history in statistics and discrete mathematics. Markov logic networks also use logic-based representations to specify blueprints for the construction of a probabilistic graphical model for a specific distribution P_V. However, the target model now is an undirected Markov network, rather than a directed Bayesian network. Instead of directed implications, the logical building blocks are disjunctions (a.k.a. clauses) of atoms (possibly negated):

$$\neg \mathit{friends}(X,Y) \vee \neg \mathit{republican}(X) \vee \mathit{republican}(Y), \tag{13}$$

which also are annotated with numerical weights. Such a clause represents a Boolean *feature* of entity pairs (X,Y), and the associated weight specifies whether graphs in which this feature holds for many concrete entity pairs are more or less probable. Instead of directed dependencies as in Bayesian networks, these features can now specify undirected, symmetric (non-causal) dependencies. The clause in (13), for example, can be rewritten as $\mathit{friends}(X,Y) \rightarrow (\mathit{republican}(X) \rightarrow \mathit{republican}(Y))$, and thus expresses a *homophily* features: friends are likely to have the same political leanings.

Probabilistic Logic Programming

Another major line of SRL developments is rooted in logic programming, and the machine learning tradition of *inductive logic programming.* A logic program such as

$$\begin{array}{l} \mathit{edge}(a,b) \\ \mathit{edge}(b,c) \\ \mathit{path}(X,Y) \leftarrow \mathit{edge}(X,Y) \\ \mathit{path}(X,Y) \leftarrow \mathit{edge}(X,Z), \mathit{path}(Z,Y) \end{array} \tag{14}$$

defines a unique *least Herbrand model*, which in our terminology is just a multi-relational graph. For the program (14) this is the graph over $V = \{a,b,c\}$ in which the relation *edge* contains the tuples $(a,b),(b,c)$, and the relation *path* contains the tuples $(a,b),(b,c),(a,c)$. In probabilistic logic programming the clauses are annotated with probabilities. Randomly selecting clauses according to their probabilities then induces a probability distribution over logic programs, and hence a probability distribution P_V over Herbrand models over V. As described here, (14) would only define a probability distribution P_V over the fixed domain $V = \{a,b,c\}$, and thus lack the generality we required of an SRL framework. However, concrete constants are usually only included in listings of simple ground facts, whereas general modeling rules are formulated at the generic level using only variables. This makes the framework still modular, and by substituting other sets of ground facts, the generic model can be applied to arbitrary domains V. Early examples of this probabilistic logic programming approach are [66, 58]. A more recent system at a very mature level of development is the ProbLog framework [39].

Inference

All the frameworks outlined above support to compute conditional probabilities of the form

$$P_V(q|e_1,\ldots,e_m), \tag{15}$$

where $q, e_1, \ldots, e_m$ all are ground atoms. There is no fundamental conceptual difference between graph classification, node classification, or link prediction, which are only distinguished by the arity of the query atom q.

▶ **Example 2.** Let V be a domain of n individuals. For some individuals we have made observations on the *infected* attribute, as well as on *contact* relations. For an individual *mary* we want to predict whether she is infected. This would be accomplished by computing the node classification query

$$P_V(\mathit{infected}(\mathit{mary})|\mathit{contact}(\mathit{mary},\mathit{john}), \mathit{contact}(\mathit{john},\mathit{anne}), \mathit{infected}(\mathit{john}), \neg\mathit{infected}(\mathit{anne}))$$

In this example the input information for the query atom *infected*(*mary*) only consists of the observations of four ground atoms related to entities in V. However, it may very well be the case that much more about the graph is known. For example, the *contact* relation might be fully observed, in which case the query would be conditioned on the whole *contact* graph.

Suppose, conversely, that we have comprehensive observations of the *infected* attribute, and want to infer the *contact* relations. This would lead to link prediction queries such as

$$P_V(\mathit{contact}(\mathit{john}, \mathit{anne}) | \mathit{infected}(\mathit{mary}), \neg\mathit{infected}(\mathit{john}), \neg\mathit{infected}(\mathit{anne})).$$

Note that the model used for answering this query is the same generative model P_V as in the node classification query before. Finally, suppose we have a predicate *spreading* that represents for the whole population V whether the infection is currently spreading in V. A probabilistic model for *spreading* might be based on the number of pairs of individuals in V that are in contact, and one of which is infected. Then a query like

$$P_V(\mathit{spreading}() | \mathit{infected}(\mathit{mary}), \neg\mathit{infected}(\mathit{john}), \mathit{contact}(\mathit{john}, \mathit{anne}), \mathit{contact}(\mathit{mary}, \mathit{john}))$$

would predict the *spreading* status of the whole domain (this query in a realistic scenario would be conditioned on much more comprehensive input observations than shown here). To emphasize the view of the query predicate as a relation of arity 0, we write it here in the format *spreading*().

SRL frameworks in the Bayesian or Markov network constructor classes can use existing inference algorithms for these types of probabilistic graphical models for the computation of arbitrary queries (15). However, constructing the graphical model for the full distribution P_V in order to compute the query probability (15), which often only refers to a small subset of the entities in V, may be very wasteful. Significant effort, therefore, has been spent on the question whether inference could be performed more directly on the basis of the high-level SRL model specification, instead of its compilation into a standard graphical model [30, 57, 13]. However, decisive breakthroughs in this area of so-called *lifted inference* remain elusive.

Inference for probabilistic logic programming frameworks is somewhat different in nature. Here the probability of the query atom q will depend on which of the possible randomly sampled programs support the derivation of the query atom q. The calculation of the query probability therefore is essentially based on the construction of all possible proofs of q from the rules and facts in the probabilistic logic program, and then a calculation of the probability that a given proof is actually supported by a randomly sampled subset of the facts and rules.

Learning

Learning of SRL models can be separated into learning the *structure* of the high-level symbolic component of the model, and learning numerical *parameters*. For the parameter learning task often quite effective methods exist. In many cases, also learning from incomplete data is supported by statistical learning techniques like expectation-maximization. Structure learning, on the other hand, amounts to a search in a complex combinatorial space of symbolic representations. Full structure learning without any prior constraints provided by (expert) domain knowledge is certainly not a solved problem at this point. We will come back to this issue in Section 12.

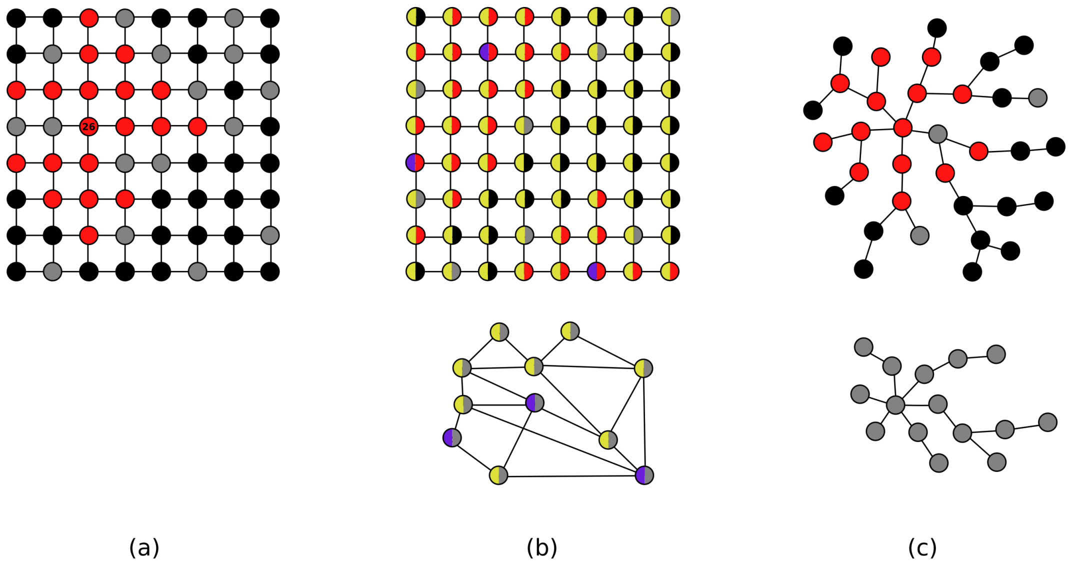


Figure 3 Transductive and inductive scenarios. (a): transductive, learnable from node identifiers; (b): inductive, learnable from node attributes; (c): inductive, learnable from graph structure. Graphs for training and (transductive) prediction in the top row; new test graphs underneath.

5 Transductive and Inductive Inference

Before we investigate at greater depth the relationship between GNN and SRL models we briefly discuss the difference between *transductive* and *inductive* inference settings. Roughly speaking, the former describes scenarios where at the time of model learning the nodes for which predictions are going to be made are already known. The latter describes scenarios where a predictive model is learned from training data, and that predictive model later is applied to formerly unknown nodes, or completely new graphs. Figure 3 illustrates this difference. In all graphs the target of prediction is a class label with values 'red' and 'black'. Figure 3 (a) shows a partially labeled graph where nodes do not have attributes other than the class label. Unlabeled nodes are shown in gray. For the particular unlabeled nodes in this graph one could learn to predict that a node is red if it is at most 3 hops away from node '26'. This model makes its predictions based on the relationship to a specific node in the training graph, and therefore does not generalize to other graphs. The learning scenario and the constructed model hence are transductive. The top of Figure 3 (b) shows a partially labeled graph where nodes also have an attribute *color* with values 'yellow' and 'blue'. This attribute here is assumed to be observed for all nodes. One could here learn a model that predicts a node to be 'red' if it is at most 2 hops away from a blue node. This model can be used to both classify the unlabeled nodes in the training graph shown in the top of the figure, as well as the nodes in a completely new and unlabeled graph shown underneath. The ability for inductive generalization is not always dependent on the existence of node attributes. Figure 3 (c) (top) shows a partially labeled graph without any node attributes. In this case one could learn to predict a node to be red if it is at most 2 hops away from a node that has a degree ≥ 5. This model would again be able to classify nodes in completely new and unlabeled graphs, like the one shown at the bottom or Figure 3 (c).

SRL models as we have described them in Section 4 are inductive in nature: they define for arbitrary new node sets V a probability distribution (and hence a prediction model) over $\mathcal{G}(V, \mathcal{R})$. This implies that SRL models cannot be defined in terms of particular node identifiers such as “node 26” in Figure 3 (a), which is reflected in syntax rules according to which elements like (12) or (13) can not contain constants denoting specific domain entities (and recall that in a slightly modified form, this constraint also applies to probabilistic logic programming models (14)). This focus on inductive models is more a historic convention than a necessary feature of SRL models: for example, a version of a rule like (12)

$$\mathit{blue}(X) \leftarrow \mathit{edge}(X, Y_1) \wedge \mathit{edge}(Y_1, Y_2) \wedge \mathit{edge}(Y_2, \mathit{node}_{26})$$

that contains references to a specific node would be able to express our transductive model for Figure 3 (a).

Whether a GNN model is inductive or transductive essentially depends on the initial representations $\boldsymbol{h}^0(i)$ used as inputs to the model. If these initial representations are node attribute vectors (as tacitly assumed in Section 3) then the resulting model will be inductive and able to handle scenarios like the one in Figure 3 (b). However, GNNs can also operate on initial node representations that are one-hot-encodings of node identifiers, which then leads to transductive models suitable to handle the situation in Figure 3 (a). In scenarios as depicted in Figure 3 (c), neither node identifiers nor node attributes would be available as initial representations. In this case one an think of $\boldsymbol{h}^0(i)$ as consisting of a constant not depending on i. The representation $\boldsymbol{h}^1(i)$ obtained after one round of aggregation would then already be able to encode the degree of the node i, which with two additional layers then enables our predictive model for Figure 3 (c).

In the following detailed comparison of SRL and GNN models we focus on inductive versions for both types of modeling frameworks, noting that the analogies we obtain in the inductive setting will carry over to transductive scenarios when for SRL frameworks we permit the use of node identifiers.

6 Semantics: a comparison

For SRL frameworks we have identified a common, well-defined semantics. Denoting the set of probability distributions over $\mathcal{G}(V, \mathcal{R})$ as $\Delta\mathcal{G}(V, \mathcal{R})$, we can write this semantics as a mapping

$$V \mapsto \Delta\mathcal{G}(V, \mathcal{R}). \tag{16}$$

▶ **Example 3.** Consider a relational signature $\mathcal{R} = \mathit{follower}, \mathit{influencer}$, where *follower* is a Boolean binary relation, and *influencer* is a Boolean node attribute. Let V be any fixed, finite domain. A distribution $P_V \in \Delta\mathcal{G}(V, \mathcal{R})$ can be defined by first defining the distribution over the *follower* relation, and then the conditional distribution of the *influencer* attribute given the *follower* relation, i.e., factorizing the joint distribution over both relations as

$$P_V(\mathit{follower}, \mathit{influencer}) = P_V(\mathit{follower}) \cdot P_V(\mathit{influencer}|\mathit{follower}). \tag{17}$$

The simplest way to define $P_V(\mathit{follower})$ is via an Erdős-Rényi random graph model, according to which each edge $\mathit{follower}(i, j)$ has a constant probability $p \in [0, 1]$ of being true, regardless of the cardinality of V:

$$P_V(\mathit{follower}(i, j)) = p. \tag{18}$$

Observe here the subtle difference: $P_V(\mathit{follower}(i, j))$ in (18) denotes the probability for the single ground atom (i.e., Boolean random variable) $\mathit{follower}(i, j)$, whereas $P_V(\mathit{follower})$ in (17) denotes the distribution over the whole *follower* relation, i.e., the joint distribution over all $\mathit{follower}(i, j)$ atoms with $i, j \in V$.

Assuming that *follower*(i,j) means that i is a follower of j, then the probability of *influencer*(i) given the *follower* relation could be defined as a function of i's in-degree (number of incoming edges) in this relation, which we denote as $d_i^{in,follower}$. For example, one could define

$$P_V(\mathit{influencer}(i)|\mathit{follower}) = 1 - q^{d_i^{in,follower}}, \tag{19}$$

for some $q \in [0,1]$. Again note that the left side of (19) denotes the conditional probability of the ground atom *influencer*(i) given the complete specification of the *follower* relation over the whole domain. For the specification on the right side of (19) we extract from the *follower* relation the in-degrees of nodes as relevant features. (19) is a standard *noisy-or* model for independent causal influences. The combined model composed of (18) and (19) would be supported by almost every SRL framework in the Bayesian network constructor family. Also a GNN implementing (18) as a link prediction model would be easy to construct. (19) is based on an underlying probabilistic interpretation, which leads to aggregation by multiplication, rather than summation. This would typically not be supported by standard GNN architectures. However, a slightly different function such as

$$P_V(\mathit{influencer}(i)|\mathit{follower}) = \sigma(d_i^{in,follower}), \tag{20}$$

where σ denotes the sigmoid function, can be easily implemented as a node classification GNN.

The preceding example has illustrated how the definition of a generative SRL model can be accomplished by the specification of several conditional probability distributions, each of which resembles a node classification or link prediction model, similar to what can be implemented in the form of a GNN. In the following we analyze this relationship more closely.

GNNs tend to be defined in terms of their computational architecture, rather than a semantic specification of the types of functions they compute. For our purpose, however, it is important to clarify the mathematical structure of GNN functions. The nature of the output computed by a GNN depends on their use for a node classification, link prediction, or graph classification application. In all cases, however, a node-level representation is the crucial (intermediate) output of a GNN (i.e. the representations $\boldsymbol{H}^2$ in Figure 2). More critical than the question of what outputs a GNN computes is the question on what class of inputs it can operate. For now we shall limit our considerations to the case of GNNs operating on simple attributed graphs $G = (V, E, \boldsymbol{A})$. Let $\mathcal{A}$ denote a fixed set of attributes. For a given set V we then denote by $\mathcal{G}(V, \mathcal{E}, \mathcal{A})$ the set of all graphs with node set V, and attributes $\mathcal{A}$.

Figure 2 may suggest that the inputs to a GNN are feature vectors $\boldsymbol{H}^0[i, \bullet]$ for all nodes i. The GNN would then accept as inputs all graphs $(V, E, \boldsymbol{A})$ with fixed V, E, but arbitrary node attribute values $\boldsymbol{A}$. This is also consistent with the functional representations (7),(8), respectively (7),(9), if one interprets the occurrence of G here as fixed parameters of the function, not as inputs that can be varied freely. To obtain a clearer picture, we have to leave the high level of abstraction adopted in Section 3, and consider the specific functions usually implementing the function pattern (7) (or the already slightly more specific version (10)). The most basic form of this function is

$$\boldsymbol{h}^{k+1}(i) = f\left(\boldsymbol{W}^k\boldsymbol{h}^k(i) + \boldsymbol{U}^k \sum_{j\in N_i} \boldsymbol{h}^k(j) + \boldsymbol{b}^k\right) \tag{21}$$

where $\boldsymbol{W}^k, \boldsymbol{U}^k$ are $(n_{k+1} \times n_k)$-dimensional matrices, $\boldsymbol{b}^k$ is a n_{k+1}-dimensional (bias) vector (where n_k denotes the dimension of the node representations at step k), and f is an element-wise activation function (cf. e.g. [23, Equation (5.7)], [4, Equation (2)]).

In the multi-relational version (11) this becomes

$$\boldsymbol{h}^{k+1}(i) = f\left(\boldsymbol{W}^k \boldsymbol{h}^k(i) + \sum_{h=1}^{m} \boldsymbol{U}_h^k \sum_{j \in N_i^{E_h}} \boldsymbol{h}^k(j) + \boldsymbol{b}^k\right), \tag{22}$$

which captures the essence of e.g. [56, Equation 4] or [68, Equation 2] (but omitting normalization operations in the aggregations).

The learnable model parameters here are the $\boldsymbol{W}^k, \boldsymbol{U}^k, \boldsymbol{b}^k$. If these matrices/vectors have dimensions that do not depend on $n = |V|$, a trained model can be applied to graphs $(V, E, \boldsymbol{A})$ with arbitrary V and E. A case where these matrices may actually be specific for a particular $|V|$ is when the initial representation $\boldsymbol{h}^0$ is a one-hot-encoding of node identifiers. This, however, only makes sense in a transductive learning setting. In inductive scenarios, the model must be able to generalize to graphs of sizes other than the size of the training graph, and the G arguments in (7),(10) must be seen as free inputs of the GNN model. Assuming now such an inductive setting, and focusing on the node embedding functionality of GNNs, one can describe the semantics of GNNs that can take (multi-relational) graphs with arbitrary V as inputs as a mapping

$$\bigcup_V \mathcal{G}(V, \mathcal{R}) \rightarrow \bigcup_{n \geq 1} \mathbb{R}^{n \times n_m}, \tag{23}$$

such that an attributed graph $(V, E, \boldsymbol{A})$ or multi-relational graph $(V, \boldsymbol{R})$ is mapped to an output in $\mathbb{R}^{|V| \times n_m}$ (n_m being the dimension of the final node embedding vectors). If one rather considers the end-to-end semantics of a GNN as a node classifier, link predictor, or graph classifier, then the semantics becomes

$$\bigcup_V \mathcal{G}(V, \mathcal{R}) \rightarrow \bigcup_{n \geq 1} \mathbb{R}^{n^c \times k}, \tag{24}$$

where $c = 1$ for node classification, $c = 2$ for link prediction, $c = 0$ for graph classification, and k is the number of possible (node/edge/graph) classes.

Comparing SRL semantics (16) and inductive GNN semantics (23) or (24), we already observe that both are functions defined on (finite) multi-relational graphs for a given signature, i.e. the space $\bigcup_V \mathcal{G}(V, \mathcal{R})$. However, an SRL model computes for an input graph $(V, \boldsymbol{R})$ a probability $P_V(\boldsymbol{R})$, whereas on the right-hand side of (24) we find a matrix of feature vectors for nodes, edges, or the whole graph. To bring these two types of function values together, we take a closer look at how probability distributions on $\mathcal{G}(V, \mathcal{R})$ can be defined following the factorization strategy already illustrated in Example 3.

Given a fixed V, a multi-relational relational graph $G = (V, \boldsymbol{R})$ is determined by the definitions of the relations $\boldsymbol{R} = R_1, \ldots, R_r$ over V. Let P_V denote the probability distribution over $\Delta\mathcal{G}(V, \mathcal{R})$ defined by an SRL model. Then P_V can be factored according to the chain rule [44, Sec. 2.1.2.2] as

$$P_V(\boldsymbol{R}) = P_V(R_1) \cdot P_V(R_2|R_1) \cdot \ldots \cdot P_V(R_h|R_1, \ldots, R_{h-1}) \cdot \ldots \cdot P_V(R_r|R_1, \ldots, R_{r-1}). \tag{25}$$

Equation (25) is a probabilistic law that holds for any distribution $P_V \in \Delta\mathcal{G}(V, \mathcal{R})$, no matter how P_V is defined or learned. In the following we write $R_{1:k}$ for the tuple of relations $R_1, \ldots, R_k$, and $\mathcal{R}_{1:k}$ for the corresponding signature of relation symbols. For the class of SRL frameworks that we have described as 'Bayesian network constructors', the chain rule also serves as the core of the representation strategy: in these SRL frameworks, the specification of the distribution P_V is decomposed into specifications of conditional probability distributions. This decomposition need not be performed at the level of whole relations as in (25). Often one rather factors the distribution at the level of the ground atoms: a definition of relation R_h over V is equivalent to a truth assignments to all ground atoms $R_h(\boldsymbol{i}) \mapsto \{\mathit{true}, \mathit{false}\}$ ($\boldsymbol{i} \in V^{arity(R_h)}$; for the case of R_h being a Boolean relation). Furthermore, also the definition of a relation-level factor $P_V(R_h|R_{1:h-1})$ can most easily be done at the level of ground atoms in the form

$$P_V(R_h|R_{1:h-1}) := \prod_{\boldsymbol{i} \in V^{arity(R_h)}} P_V(R_h(\boldsymbol{i})|R_{1:h-1}). \tag{26}$$

Note that (26) now is based on the assumption that ground atoms in the relation R_h are conditionally independent given the relations $R_{1:h-1}$. In the form (26) the specification of $P_V(R_h|R_{1:h-1})$ becomes a mapping of the form

$$\mathcal{G}(V, \mathcal{R}_{1:h-1}) \to [0,1]^{n^{arity(R_h)}}. \tag{27}$$

This is still assuming that R_h is Boolean, and thus $P_V(R_h(\boldsymbol{i})|R_{1:h-1})$ is given by a single probability value for $R_h(\boldsymbol{i})$ being *true*. With a little additional notation, this generalizes to arbitrary categorical R_h. The factorization strategy employed by an SRL model for P_V, and the specification of $P_V(R_h|R_{1:h-1})$ will be uniform across different V. We can therefore also say that an SRL framework in the Bayesian network constructor class defines for each relation R_h a mapping

$$\bigcup_V \mathcal{G}(V, \mathcal{R}_{1:h-1}) \to \bigcup_{n \geq 1} [0,1]^{n^{arity(R_h)}}, \tag{28}$$

which now is almost identical to (24). It becomes completely identical, if we assume that the GNN uses softmax normalization on its output to also generate a probability distribution over node or link labels.

To summarize: the specification of a generative distribution P_V can be accomplished via (25) as a series of probabilistic node classification and link prediction operations (and possibly predictions of relations of arity higher than 2, if such are present in $\mathcal{R}$). If one further assumes that predictions for the atoms of each relation are independent of each other, then an SRL model that uses the chain rule as its underlying representation paradigm essentially consists of r functions, each of which is equivalent to a GNN function.

In this section we have focused on a comparison of the SRL generative models with (discriminative) GNN models that are designed for specific prediction tasks. GNN models have also been proposed for graph generation [42, 76, 47, 11], and one may wonder whether these are not a more natural point of reference for comparison with SRL frameworks. This is not the case, however: most of the proposed generative GNN models are in the tradition of classic random graph models, and their primary purpose is to learn GNN models from which random graphs can be effectively sampled, such that the distribution of key graph statistics (e.g., degree distribution, clustering coefficients) in the randomly sampled graphs matches the distribution in the training data. This is different from the typical SRL scenario, where the purpose is not to generate full graphs according the distribution P_V, but to answer queries of the form (15). Thus, the objectives for which generative SRL models are built are much more aligned with the objectives of predictive GNN models, than with the objectives of generative GNN models. An exception to this observation is [42] where the generative model is applied to link prediction. This, however, is in a transductive setting where the generative model is only fitted to a single graph for which then missing links are predicted, thus again being very different in nature from SRL.

7 RBNs

We now review the SRL framework of *relational Bayesian networks (RBNs)* [29, 34], which directly follows the representation paradigm outlined by (25) and (26). The RBN language consists of a syntax for logic-functional expressions called *probability formulas* for the specification of the conditional probabilities $P_V(R_h(\boldsymbol{i})|R_1,\ldots,R_{h-1})$ appearing in (26). We here give a description of the RBN language that uses a slightly more verbose syntax than the one introduced in the original papers. The difference is entirely cosmetic, however, and consists of little more than an alternative choice of terminal symbols in the grammar. The language of probability formulas consists of four different syntactic constructs. For each construct we define the syntax, and the semantics that defines how for a tuple $\boldsymbol{i} \in V^{arity(R_h)}$ in a graph $G = (V, R_1, \ldots, R_{h-1})$ the formula evaluates to the probability $P_V(R_h(\boldsymbol{i})|R_1,\ldots,R_{h-1})$. In the following, F denotes a probability formula, and $eval(F, \boldsymbol{i}, G)$ the probability value it defines for $\boldsymbol{i}$ in G.

Constants

For a real number $q \in [0, 1]$

$$F \equiv q \tag{29}$$

is a probability formula. $eval(F, \boldsymbol{i}, G) = q$ for all $\boldsymbol{i}, G$.

Atoms

For a relation R_j with $j \in 1, \ldots, h-1$, and variable symbols $Y_1, \ldots, Y_{arity(R_j)}$

$$F \equiv R_j(Y_1, \ldots, Y_{arity(R_j)}) \tag{30}$$

is a probability formula with the semantics

$$eval(F, \boldsymbol{i}, G) = \begin{cases} 1 & \text{if } R_j(\boldsymbol{i}) \text{ is true in } G \\ 0 & \text{if } R_j(\boldsymbol{i}) \text{ is false in } G \end{cases} \tag{31}$$

This innocuous definition is quite significant, as it transforms logic-symbolic information represented by a relation R_j into numeric data.

WIF-THEN-ELSE

Assume that F_1, F_2, F_3 are probability formulas. Then

$$F \equiv \texttt{WIF}\ \ F_1\ \ \texttt{THEN}\ \ F_2\ \ \texttt{ELSE}\ \ F_3 \tag{32}$$

is a probability formula. 'WIF' here stands for “weighted if”. The semantics is a weighted mixture of probabilities:

$$\begin{aligned} &eval(\texttt{WIF}\ \ F_1\ \ \texttt{THEN}\ \ F_2\ \ \texttt{ELSE}\ \ F_3, \boldsymbol{i}, G) = \\ &\qquad eval(F_1, \boldsymbol{i}, G)\,eval(F_2, \boldsymbol{i}, G) + (1 - eval(F_1, \boldsymbol{i}, G))\,eval(F_3, \boldsymbol{i}, G). \end{aligned} \tag{33}$$

Before we move on to the fourth and most important construct for probability formulas, we illustrate the use of the ones introduced so far.

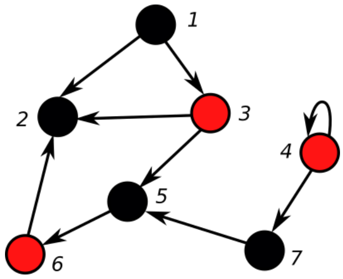


Figure 4 A small graph for two relations.

▶ **Example 4.** Figure 4 shows a small graph G for two relations consisting of a node attribute $R_1 = \mathit{red}$, and a binary relation $R_2 = \mathit{edge}$. Here the color black is just the negation of the Boolean attribute *red*. As always in SRL frameworks, relations are defined on ordered tuples, that means edges are directed.

Let now *positive* be a new Boolean node attribute. Associating with *positive* the constant probability formula $F_{\mathit{positive}(X)} \equiv 0.3$ would define a probability distribution according to which each node i has a constant probability of 0.3 of being *positive*. If *new_edge* is a new binary relation, then, similarly, $F_{\mathit{new\_edge}(X,Y)} \equiv 0.3$ would define a distribution over the *new_edge* relation according to which $\mathit{new\_edge}(X, Y)$ is true with probability 0.3 for each pair $i, j \in \{1, \ldots, 7\}$.

For a slightly more interesting example, let

$$F_{\mathit{positive}(X)} \equiv \texttt{WIF } \mathit{red}(X) \texttt{ THEN } 0.3 \texttt{ ELSE } 0.9.$$

Then $\mathit{eval}(F_{\mathit{positive}(X)}, i, G) = 0.3$ for the red nodes $i = 3, 4, 6$ of G, and $\mathit{eval}(F_{\mathit{positive}(X)}, i, G) = 0.9$ for the non-red nodes. For *new_edge* we could also define

$$F_{\mathit{new\_edge}(X,Y)} \equiv \texttt{WIF } \mathit{edge}(Y, X) \texttt{ THEN } 0.5 \texttt{ ELSE } 0.$$

This specification would add for every edge (j, i) in the existing *edge* relation with probability 0.5 the reverse edge (i, j) to the *new_edge* relation. We can add the condition that in the existing edge the source node must be red in order for the reverse edge to be generated:

$$\begin{aligned} F_{\mathit{new\_edge}(X,Y)} \equiv \; & \texttt{WIF } \mathit{edge}(Y, X) \texttt{ THEN WIF } \mathit{red}(Y) \\ & \qquad\qquad\qquad\qquad \texttt{THEN } 0.5 \\ & \qquad\qquad\qquad\qquad \texttt{ELSE } 0 \\ & \qquad\qquad\quad \texttt{ELSE } 0 \end{aligned}$$

We can allow some “syntactic sugar” to make expressions like this more compact and readable, and write

$$F_{\mathit{new\_edge}(X,Y)} \equiv \texttt{WIF } \mathit{edge}(Y, X) \wedge \mathit{red}(Y) \texttt{ THEN } 0.5 \texttt{ ELSE } 0,$$

with the understanding that this is not a proper extension of the representation language, but only a shorthand for expressions that are constructed according to the existing syntax rules.

The three constructs introduced so far allow to condition the probability of $R_h(\boldsymbol{i})$ on Boolean combinations of properties of $\boldsymbol{i}$ according to the relations $R_1, \ldots, R_{h-1}$. The fourth and central syntactic construct enables us to condition probabilities for $\boldsymbol{i}$ on properties of other entities $\boldsymbol{j}$. This construct requires the distinction between input relations $\mathcal{R}_{in}$ and probabilistic relations $\mathcal{R}_{prob}$ described at the end of Section 6.

Combination Functions

Assume that $F_1,\ldots,F_t$ are probability formulas.

$$\begin{array}{ll} F \equiv & \texttt{COMBINE}\ F_1,\ldots,F_t \\ & \texttt{WITH}\ <\textit{combination function}> \\ & \texttt{FORALL}\ <\textit{variables}> \\ & \texttt{WHERE}\ <\textit{Boolean}\ \mathcal{R}_{in}\ \textit{condition}> \end{array} \tag{34}$$

is a probability formula. This syntax rule is dependent on a few supplementary specifications: $<$ *variables* $>$ is simply a list of variable names. A $<$ *Boolean* $\mathcal{R}_{in}$ *condition* $>$ is a Boolean expression built from atomic expressions that can be either atoms $R(\boldsymbol{Y})$ with $R \in \mathcal{R}_{in}$, or equalities $Y = Z$ between variables (again, no identifiers for specific entities i are allowed). A $<$ *combination function* $>$, according to the original definition of [29], is any function that maps multisets of probability values $\{\!|p_1,\ldots,p_K|\!\}$ to a probability value. The most important such combination functions are

$$\textit{noisy-or}\{\!|p_1,\ldots,p_K|\!\} = 1 - \prod_{i=1}^{K}(1-p_i) \tag{35}$$

$$\textit{mean}\{\!|p_1,\ldots,p_K|\!\} = \frac{1}{K}\sum_{i=1}^{K} p_i \tag{36}$$

One can relax the condition that both input and output values always have to be probabilities, and also allow e.g. summation:

$$\textit{sum}\{\!|p_1,\ldots,p_K|\!\} = \sum_{i=1}^{K} p_i. \tag{37}$$

However, when such constructs are used which can generate numbers outside of $[0,1]$, then for the eventual specification of the conditional probability $P_V(R_h(\boldsymbol{i})|R_1,\ldots,R_{h-1})$ these numbers have to be brought back into the $[0,1]$ interval. The most useful tool for this is the combination function which by a slight abuse of terminology we call the logistic-regression function:

$$\textit{logistic-regression}\{\!|p_1,\ldots,p_K|\!\} = \frac{1}{1+\textit{exp}(-\sum_{i=1}^{K} p_i)} \tag{38}$$

Note that we have overloaded the term ’combination function’ to both denote the probability formula (34), and the concrete numerical combination functions at its core. A full formal specification of the semantics of a combination function construct requires some care regarding the variables that appear in different components of the formula, and how substitutions of domain entities for these variables are performed. However, the basic principle can be described quite easily: suppose that $<$ *variables* $>\equiv Y_1,\ldots,Y_k$. Then, for a given graph $(V, R_1,\ldots,R_{h-1},\boldsymbol{R}_{in})$, and a tuple $\boldsymbol{i}$, the $<$ *Boolean* $\mathcal{R}_{in}$ *condition* $>$ defines the set of all $\boldsymbol{j} \in V^k$ that make the condition true when one substitutes j_l for Y_l ($l = 1,\ldots,k$), and the elements of $\boldsymbol{i}$ for other designated variables appearing in the Boolean condition. Let $J(\boldsymbol{i})$ be the set of these satisfying k-tuples of domain elements. For each $\boldsymbol{j} \in J(\boldsymbol{i})$, and each F_m ($m = 1,\ldots,t$) the value $\textit{eval}(F_m,\boldsymbol{i},\boldsymbol{j},G)$ is already defined. Then the value of the formula (34) is

$$\textit{eval}(F,\boldsymbol{i},G) = \textit{comb}\{\!|\textit{eval}(F_m,\boldsymbol{i},\boldsymbol{j},G)|m = 1,\ldots,t;\boldsymbol{j} \in J(\boldsymbol{i})|\!\} \tag{39}$$

where *comb* is the combination function declared in the `WITH` clause of (34).

▶ **Example 5.** Consider again the graph of Figure 4. Assume now that *edge* $\in \mathcal{R}_{in}$ is an input relation, whereas *red* $\in \mathcal{R}_{prob}$ is probabilistic. Then we can define the conditional probability for the *positive* node attribute by

$$\begin{aligned} F_{positive(X)} \equiv\ & \texttt{COMBINE}\ 0.7 \cdot red(Y) \\ & \texttt{WITH}\ noisy\text{-}or \\ & \texttt{FORALL}\ Y \\ & \texttt{WHERE}\ edge(Y,X). \end{aligned} \tag{40}$$

The product $0.7 \cdot red(Y)$ in the COMBINE clause here again is a syntactic shorthand for what in principle is another *product* combination function. This formula expresses a standard causal model according to which each *red* source node of an edge causes the target node to be *positive* with probability 0.7. For each $i \in \{1,\ldots,7\}$ here $J(i)$ is just the set of nodes j with $edge(j,i)$. We obtain

$$eval(F_{positive(X)}, i, G) = \begin{cases} 0 & \text{for } i = 1,3,6 \\ 1-(1-0.7) = 0.7 & \text{for } i = 4,5,7 \\ 1-(1-0.7)^2 = 0.91 & \text{for } i = 2 \end{cases}$$

If we want to prevent self-loops of red nodes from being possible causes for that node to be *positive* (as happens for node 4 in the example), we can strengthen the WHERE clause to $edge(Y,X) \wedge Y \neq X$.

The formula is an example of a typical model for information or causal influence propagation along edge relations, and is closely related to the message passing principle in GNNs. However, combination functions are not limited to this pattern of information diffusion. If we modify the WHERE clause to be just the logical constant *true*, then $J(i) = \{1,\ldots,7\}$ and $eval(F_{positive(X)}, i, G) = 1-(1-0.7)^3$ for all i. Thus, the probability for *positive*(i) now depends uniformly for all i on the global graph feature of the total number of *red* nodes. Exactly the same formula could then also be employed to define the probability for a global graph label *class*(). Going in the opposite direction, we could use still the same formula to define the probabilities for the binary *new_edge* relation (defining an Erdős-Rényi random graph model where the edge probability is a function of the number of red nodes in the graph). Generally, a probability formula with k "free variables" ($k = 1$ in (40), and $k = 0$ when the WHERE clause is changed to *true*) can be used to define conditional probabilities for relations of arities $\geq k$.

8 Expressivity

In Section 6 we have identified at an abstract level the similarities between what GNNs and SRL frameworks represent and compute. The high-level semantic analogies do not mean that the concrete functions that are supported by GNN or SRL models have much in common. However, already Example 5 has indicated that there are some commonalities between the message passing operations in GNNs, and probabilistic combination operations in SRL, especially RBNs. In this and the next section we will establish strong correspondences between the concrete modeling capacities of GNNs and RBNs.

The question of expressivity has been investigated for SRL frameworks [31], and also has received considerable attention in recent years for GNNs [75, 51, 4, 64, 19]. We start by looking a bit deeper into the expressivity of GNNs.

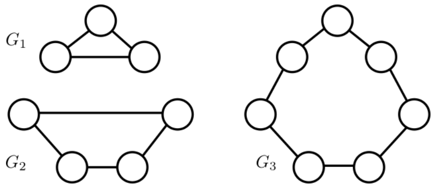


Figure 5 Indistinguishable nodes and graphs.

8.1 GNN expressivity

Broadly speaking, here expressivity relates to a GNNs capability to differentiate between different inputs. The focus can be on differentiating between different input graphs, or between nodes in graphs. In both cases, the ability to differentiate between inputs is a pre-condition for being able to support a rich class of predictive functions.

The node-level version of expressivity can be cast as the following question: for what graphs $G = (V, E), G' = (V', E')$, and nodes $i \in V, i' \in V'$, can a particular GNN architecture (or a certain class of architectures) learn representations $\boldsymbol{h}^m(i), \boldsymbol{h}^m(i')$, such that $\boldsymbol{h}^m(i) \neq \boldsymbol{h}^m(i')$? At graph level, the question becomes for which pairs of graphs G, G', the value of the readout function (9) can be different. Since (9) depends on node representations as input, the discrimination capabilities at node and graph level are tightly linked.

▶ **Example 6.** Figure 5 (adapted from [1]) shows three graphs G_1, G_2, G_3. The nodes here do not have any attributes or identifiers, so the initial representations $\boldsymbol{h}^0(i)$ would be the same constant for all nodes i in all three graphs. In a computation of $\boldsymbol{h}^1$ by any form of message-passing update (7), each node i will also obtain the same representation $\boldsymbol{h}^1(i)$, because all nodes sum identical $\boldsymbol{h}^0$ representations for exactly two neighbors j. By induction, representations $\boldsymbol{h}^k(i), \boldsymbol{h}^k(j)$ for $i \neq j$ can never become different at any step k. At the graph level, however, the three graphs could be distinguished by a final readout aggregator (9), because that would receive as input multisets of different cardinalities for the three graphs. Finally, if one considers the graph $G_4 = G_1 \cup G_2$, then this graph would no longer be distinguishable from G_3, because now (9) receives as input for the both graphs multisets of the same cardinality of identical node representations.

Many different approaches have been proposed to make GNNs more expressive than what can be achieved by the basic form of message passing (21) that we assumed in Example 6. One possible strategy is to consider node identifiers: it is clear that when unique node identifiers are used as initial representations, then already the initial representation distinguishes all nodes, and our expressivity question at the node level becomes moot (though graph-level discrimination is not immediately solved by node identifiers). However, as discussed in Section 6, the use of node identifiers would severely limit inductive generalization capabilities of models that depend on them (cf. Figure 3). A few papers have studied the use of randomly generated initial attributes as a means to combine some benefits of identifiers with (still somewhat limited) generalization abilities [65, 1]. A full review of these approaches is beyond the scope of this article. However, the following example (inspired by [65]) illustrates the main traits of these approaches.

▶ **Example 7.** Consider again the graphs in Figure 5. We shall see that by assigning random initial node attributes, we can construct a GNN, which otherwise follows the simple architecture (21), that can identify nodes that lie on a cycle of length 3, and hence can distinguish the nodes in G_1 from the nodes in G_2 and G_3. Due to the probabilistic nature of the construction, this will only be guaranteed with a certain probability $1 - \delta$ that can be brought arbitrarily close to 1.

Let N be an integer that one should think of as being significantly larger than the cardinalities n of our input graphs. For a graph $G = (V, E)$, and node $i \in V$ we generate a random initial N-dimensional $\boldsymbol{h}^0(i)$ in the form of a random one-hot vector (i.e., $\boldsymbol{h}^0(i)$ has a 1 in one randomly chosen position, and 0s everywhere else). Let $idx(i) \in 1, \ldots, N$ denote the index at which $\boldsymbol{h}^0(i)$ is 1. For a given $\delta > 0$ we can choose an N, such that for graphs with $|V| \leq n$ with probability at least $1 - \delta$ the $idx(i)$ are different for all nodes $i \in V$. The $\boldsymbol{h}^0(i)$ then can be seen as random node identifiers. For $k = 1, 2, 3$ let the dimension of $\boldsymbol{h}^k$ be $2N$. We use the first N components of these representations to just copy the initial random identifiers $\boldsymbol{h}^0$. The last N components are used to represent which nodes are reachable by a path of length k. This can be accomplished by functions of the form (21) as follows: for $k = 1$ let $\boldsymbol{W}^0$ be the $2N \times N$ matrix that consists of an $N \times N$ identity matrix in the upper half, and is zero in the lower half. Similarly, $\boldsymbol{U}^0$ is the $2N \times N$ matrix that has the identity matrix in the lower half. With $\boldsymbol{b}^0 = \boldsymbol{0}$ and f the identity function, then $\boldsymbol{h}^1(i)$ will contain a copy of i's random initial one-hot vector in the first N components, and $\boldsymbol{h}^1(i)[N + idx(j)] = 1$ iff $(i, j) \in E$, i.e., j is reachable from i by a path of length 1. The construction for $k = 2, 3$ is almost the same, with minor modifications: the matrices $\boldsymbol{W}^{k-1}, \boldsymbol{U}^{k-1}$ are now $2N \times 2N$ matrices that have $N \times N$ identity matrices in the upper left and lower right quadrant, respectively. The argument vector of $f()$ in (21) can now have integer values > 1 in some of the components $N + idx(j)$ if there exist multiple paths from i to j. This can be brought back to a pure 0/1-valued indicator vector for the existence of paths by using for f the truncated Relu function $f(x) = min(1, Relu(x))$.

Node i now lies on a cycle of length 3 iff $\boldsymbol{h}^3(i)$ has a 1 both in components $idx(i)$ and $N + idx(i)$ (i.e. i is reachable from itself by a path of length 3). Defining $\boldsymbol{h}^4(i) = Relu(\boldsymbol{h}^3(i)[1 : N] - \boldsymbol{h}^3(i)[N + 1 : 2N] - \boldsymbol{1}) \in \mathbb{R}^N$ (which still fits the functional form (21)) then gives an N-dimensional representation of i that has a single 1 in component $idx(i)$ if i lies on a cycle of length 3, and is 0 everywhere otherwise. A final summation $\boldsymbol{h}^5(i) = \sum_{h=1}^{N} \boldsymbol{h}^4(i)[h]$ then gives a scalar that classifies i as lying on a length 3 cycle or not.

Example 7 is quite representative of the general results of [65, 1] in that:

- the random initial features are exploited by otherwise standard GNN architectures;
- high-dimensional representations $\boldsymbol{h}^k$ are required;
- for a given level $1 - \delta$ of confidence in the correctness of the outcomes, the set of possible inputs has to be constrained, and thus the inductive generalization capabilities are limited.

Several other approaches have been proposed for increasing the expressivity of GNNs:

- the use of *higher order* GNNs in which representations are not associated with single nodes, but with tuples or sets of nodes [51, 72].
- more sophisticated functions than simple summation as in (21) for aggregating representations of neighbor nodes. These functions ideally are injective, i.e., map distinct multiset inputs to distinct outputs, and thereby preserve discriminative information provided by graph neighbors to the highest possible extent [75]. We return to this in Section 11.2.

8.2 The ACR architecture and first-order logic

A relatively simple approach to increase the expressivity of the basic message passing architecture (7) was proposed in [4], based on the observation that with (7) node representations $\boldsymbol{h}^k(i)$ are limited to information that is visible within a k-hop neighborhood of i. This can easily be remedied by already allowing in the computation of node representations global readout aggregations (9). The abstract representation update function then becomes

$$\boldsymbol{h}^{k+1}(i) = F^k(\boldsymbol{h}^k(i), \{\!\{\boldsymbol{h}^k(j)|j \in N_i\}\!\}, \{\!\{\boldsymbol{h}^k(j)|j \in V\}\!\}, G), \tag{41}$$

which can be instantiated to a concrete form analogous to (21):

$$\boldsymbol{h}^{k+1}(i) = f\left(\boldsymbol{W}^k\boldsymbol{h}^k(i) + \boldsymbol{U}^k\sum_{j\in N_i}\boldsymbol{h}^k(j) + \boldsymbol{R}^k\sum_{j\in v}\boldsymbol{h}^k(j) + \boldsymbol{b}^k\right). \tag{42}$$

For the resulting *aggregate-combine-readout (ACR)* GNN architecture, [4] then derive an expressivity analysis using first-order predicate logic. We shall here not give a full review of first-order logic (FOL) (standard references are [63, Chapter 8], [16]), but only illustrate the main issues by examples.

A first-order formula $\phi(X)$ with one *free variable* X can define properties of nodes in a graph. For example, the formula

$$\phi(X) \equiv \exists Y_1, Y_2, Y_3 : (E(X,Y_1)\wedge E(X,Y_2)\wedge E(X,Y_3)\wedge\neg Y_1 = Y_2\wedge\neg Y_1 = Y_3\wedge\neg Y_2 = Y_3) \tag{43}$$

says that X is connected to three nodes Y_1, Y_2, Y_3 that are all different, i.e., X has a degree of at least 3. Assuming that the nodes in the graph have color attributes *red*,*green* and *blue* (also allowing that several of these attributes are true at the same time for a single node), then

$$\phi(X) \equiv \mathit{blue}(X) \wedge \exists Y : \mathit{red}(Y) \tag{44}$$

says that X is blue, and there exists at least one node Y that is red. The *two-variable fragment* of FOL, denoted FOL_2, consists of all formulas that contain at most 2 distinct variables. Thus, (44) belongs to FOL_2, while (43) does not. An extension of the syntax of FOL is by *counting quantifiers* $\exists^{\geq k}$ that directly state that there exist at least k different entities with a certain property. Using counting quantifiers, one can rephrase (43) as

$$\phi(X) \equiv \exists^{\geq 3} Y : E(X,Y). \tag{45}$$

This formula is equivalent to (43), but now it only makes use of two distinct variables: (45) is an element of the two-variable fragment with counting quantifiers, denoted FOLC_2.

First-order logic and each of its fragments or extensions has a certain ability to discriminate nodes in a graph. Specifically, consider the set of graphs $\mathcal{G}(\cdot,\mathcal{E},\mathcal{A}) := \cup_V \mathcal{G}(V,\mathcal{E},\mathcal{A})$ where the signature $\mathcal{A}$ only contains Boolean attributes. A *(Boolean) node property* for this set of graphs is a mapping ρ that takes a graph $G = (V, E, \boldsymbol{A}) \in \mathcal{G}(\cdot,\mathcal{E},\mathcal{A})$ and a node $i \in V$ as input, and returns 0 or 1. A node property is captured by a logic formula $\phi(X)$ if $\rho(G, i) = 1$ iff $\phi(X)$ evaluates to true for $X = i$. A central result of [4] then is

▶ **Theorem 8** ([4, Theorem 5.1]). *If a node property ρ is captured by a formula in* FOLC_2*, then ρ can be computed by an ACR-GNN of the form (42) with f the truncated Relu activation function, and the node attribute vectors $\boldsymbol{A}[i, \bullet]$ as initial representations.*

This result is remarkably similar to an expressivity result for RBNs given in [29]. Adapted to our current context, that result can be stated as

▶ **Theorem 9** ([29, Theorem 1]). *If a node property ρ is captured by a formula in FOL, then there exists a probability formula $F_{\rho(X)}$ that only uses the noisy-or combination function, such that* $\mathrm{eval}(F_{\rho(X)}, i, G) = \rho(G, i)$.

The proof of Theorem 8 is non-trivial, and depends on an alternative characterization of FOLC$_2$ as a special modal logic. The proof of Theorem 9, on the other hand, is straightforward, as the construction of a probability formula corresponding to a given FOL formula ϕ can simply follow the structure of ϕ, using wif-then-else constructs to capture Boolean operations, and noisy-or combination functions to capture existential quantification.

Theorem 9 is somewhat stronger than Theorem 8, as FOL is more expressive than FOLC$_2$. Moreover, the original theorem of [29] is more general than what is stated in Theorem 9, as beyond node properties it also covers properties of whole graphs, and of k-tuples ($k \geq 2$) of nodes. The ability to express features of k-tuples of nodes via probability formulas with k free variables is key for the high flexibility and expressivity of RBNs and many other SRL frameworks (using, of course, somewhat different representation techniques than probability formulas). This also ensures that probability formulas are still more powerful than higher order GNNs mentioned above.

Theoretical expressivity analyses are mostly based on classes of properties that can be expressed in a formal framework, such as logic characterizations that we have focused on here, or classes of *Weisfeiler-Lehman (WL)* graph isomorphism tests, which have played a central role in the expressivity analysis of GNNs [64]. However, in reality a GNN or SRL model will rather need to represent complex noisy relationships, not clear-cut logical properties. In the next section we will show that RBNs can also represent all functions that do not represent logic properties, and which can be represented by standard GNN architectures.

9 RBN encodings of GNNs

In this section we show how an ACR-GNN composed of layers of the form (42) can be encoded as a probability formula as introduced in Section 7. Let $\mathcal{N}$ be an ACR-GNN defined by matrices/vectors $\boldsymbol{W}^k, \boldsymbol{U}^k, \boldsymbol{R}^k, \boldsymbol{b}^k$ $(k = 1, \ldots, m)$, as well as a final output (8) or readout (9) layer. Assume, for now, that the function f in (42) is the sigmoid activation function. Also assume that all node attributes $\mathcal{A}$ are Boolean, represented by one-hot encodings in $\boldsymbol{h}^0$. We show that for each k, and each $l = 1, \ldots, n_k$, there exists a probability formula $F_{\boldsymbol{h}^k[l]}(X)$, such that for all attributed graphs $G = (V, E, \boldsymbol{A})$, and all $i \in V$:

$$\boldsymbol{h}^k(i)[l] = eval(F_{\boldsymbol{h}^k[l]}, i, G). \tag{46}$$

First consider $k = 0$ and $1 \leq l \leq n_0$. Then there exists an attribute $A \in \mathcal{A}$, and a truth value $\tau \in \{true, false\}$, such that $\boldsymbol{h}^0(i)[l]$ is the 0,1-valued indicator for whether node i has value τ for A. For our Boolean attributes A this somewhat redundant encoding could obviously be reduced to a single 0,1-valued input, using 0 for $\tau = false$, and 1 for $\tau = true$. A "mechanical" application of one-hot encodings will give us this redundant two component encoding, however. We then define

$$F_{\boldsymbol{h}^0[l]}(X) \equiv \begin{cases} A(X) & \text{if } \tau = true \\ \neg A(X) & \text{if } \tau = false \end{cases} \tag{47}$$

where $\neg A(X)$ is a shorthand for `WIF` $A(X)$ `THEN` 0 `ELSE` 1. Now assume that formulas $F_{\boldsymbol{h}^k[l]}$ have been constructed for some $k \geq 0$. We then can first define formulas that compute the two sums in (42). For the first sum over the neighbor representations, we can use

$$\begin{aligned} F_{\sum_{E(\cdot,X)} \boldsymbol{h}^k[l]}(X) \equiv\ & \texttt{COMBINE } F_{\boldsymbol{h}^k[l]}(Y) \\ & \texttt{WITH } sum \\ & \texttt{FORALL } Y \\ & \texttt{WHERE } E(Y, X). \end{aligned} \tag{48}$$

A similar formula $F_{\sum_V \boldsymbol{h}^k[n_k]}()$ is used to represent the second sum ranging over all nodes $j \in V$. In that formula the `WHERE` clause simply is the Boolean *true* constant, and the formula then does not depend on the node X. Let us abbreviate the first sum in (42) by $\overline{\boldsymbol{h}^k}^i$ (this one depends on i), and the second sum by $\overline{\boldsymbol{h}^k}$ (no dependence on i). Then

$$\boldsymbol{h}^{k+1}(i)[l] = f(\boldsymbol{W}^k[l,\bullet] \cdot \boldsymbol{h}^k(i) + \boldsymbol{U}^k[l,\bullet] \cdot \overline{\boldsymbol{h}^k}^i + \boldsymbol{R}^k[l,\bullet] \cdot \overline{\boldsymbol{h}^k} + \boldsymbol{b}^k[l]). \tag{49}$$

Expanding the dot products between n_k-dimensional vectors contained in this expression, we can write this as the probability formula

$$\begin{array}{lll} F_{\boldsymbol{h}^{k+1}[l]}(X) \equiv & \texttt{COMBINE} & \boldsymbol{W}^k[l,1] \cdot F_{\boldsymbol{h}^k[1]}(X), \\ & & \vdots \\ & & \boldsymbol{W}^k[l,n_k] \cdot F_{\boldsymbol{h}^k[n_k]}(X), \\ & & \boldsymbol{U}^k[l,1] \cdot F_{\sum_{E(\cdot,X)} \boldsymbol{h}^k[1]}(X), \\ & & \vdots \\ & & \boldsymbol{U}^k[l,n_k] \cdot F_{\sum_{E(\cdot,X)} \boldsymbol{h}^k[n_k]}(X), \\ & & \boldsymbol{R}^k[l,1] \cdot F_{\sum_V \boldsymbol{h}^k[1]}(), \\ & & \vdots \\ & & \boldsymbol{R}^k[l,n_k] \cdot F_{\sum_V \boldsymbol{h}^k[n_k]}(), \\ & & \boldsymbol{b}^k[l] \\ & \texttt{WITH} & \textit{logistic regression} \\ & \texttt{FORALL} & \\ & \texttt{WHERE} & \textit{true} \end{array} \tag{50}$$

The products appearing here are products of scalar quantities defined by probability formulas, and strictly speaking another shorthand for formulas of the form `WIF` F_1 `THEN` F_2 `ELSE` 0. Formula (50) is a degenerate combination function in the sense that it does not aggregate over any entities, as visible from the empty `FORALL` clause (the following `WHERE` clause then is somewhat redundant). Aggregation here only is over the fixed number of $t = 3n_k + 1$ sub-formulas. Since the *logistic-regression* combination function sums its arguments, and then applies the sigmoid function, (50) computes exactly (48), when f there is the sigmoid.

In a similar manner, also probability formulas representing the components of an output (8) or readout (9) layer can be constructed. To accommodate other activation functions, such as Relu or truncated Relu, corresponding combination functions have to be used.

In terms of representation size, the encoding of (42) by probability formulas clearly leads to a significant blow-up, as the matrix-vector multiplications are decomposed down to the level of operations on scalars. It is important to realize, however, that mathematically the encoding is faithful: the evaluation of the probability formulas leads to exactly the same basic multiplication, addition, and sigmoid application operations, as in a "forward propagation" evaluation of the neural network layers. More importantly, also gradient-descent based learning of the parameters $\boldsymbol{W}^k, \boldsymbol{U}^k, \boldsymbol{R}^k, \boldsymbol{b}^k$ using a standard algorithm like LBFGS [54] or ADAM [40] leads to exactly the same algorithmic steps, assuming that equivalent loss functions for the final output of the probability formula, respectively the ACR-GNN, are used (cf. Section 12.2).

We have here shown how GNNs can be represented as RBNs (focusing on ACR-GNNs, but similar constructions can be done for other GNN architectures). The central message-passing paradigm of GNNs can also be captured by other SRL frameworks than RBNs, especially frameworks in the Bayesian network constructor class, which all contain conditioning on relational neighbors as a central modeling tool. However, RBNs are specifically well-suited for a direct encoding of GNN architectures, because of the following features:

- The translation of symbolic to numeric data performed by the semantics (31) of atomic probability formulas directly bridges the gap between symbolic SRL and numeric GNN approaches.
- The recursive syntax definition of probability formulas directly corresponds to the "deep" structure of GNN architectures.

Obviously, just encoding a GNN as an RBN serves little purpose if one then solves identical tasks using the RBN representation, as one would solve with a GNN. Indeed, this would be a rather bad idea, because even though the RBN representation is, in principle, mathematically and algorithmically equivalent to the GNN model, it is in practice computationally much less efficient. A main reason for this is that GNNs only permit aggregation operations over a node's neighbors (or, for a readout, over all nodes in the graph), and this corresponds to simple matrix-vector multiplications involving the adjacency matrix. RBNs, on the other hand, support aggregations over all kinds of sets of tuples of nodes that can be defined with a Boolean condition in the `WHERE` clause of a combination function. The retrieval of the relevant tuples is implemented by what amounts to a general database query function. Even the simple queries '`FORALL` Y `WHERE` $E(Y,X)$' we encounter in the RBN encoding of a GNN are then a bit more involved to compute than simply retrieving row X of E's adjacency matrix. Encoding a GNN model in an RBN can be beneficial, however, if one then leverages capabilities of SRL models that are not provided by a GNN:

- Solving inference tasks other than the single prediction task for which a GNN is trained.
- Combining low-level "neural" model components with higher level symbolic representations, e.g. expressing expert domain knowledge.

We will illustrate the first point in Example 11 below. First we consider an example for ACR-GNNs in their original form, however.

▶ **Example 10.** Barceló et al. [4] considered logically defined Boolean class labels for nodes in attributed graph over a signature of color attributes $\mathcal{A} = \{\mathit{blue}, \mathit{green}, \mathit{yellow}, \mathit{red}, \mathit{purple}\}$. The simplest label definition considered in [4] is expressed by the FOLC$_2$ formula

$$\alpha_1(X) \equiv \exists^{[8,10]} Y(\mathit{blue}(Y) \wedge \neg \mathit{edge}(X,Y)), \tag{51}$$

where $\exists^{[8,10]}$ is shorthand for $\exists^{\geq 8} \ldots \wedge \neg \exists^{\geq 11} \ldots$. Using the property α_1 defined by this formula, a more complex property α_2 is defined by

$$\alpha_2(X) \equiv \exists^{[10,20]} Y(\alpha_1(Y) \wedge \neg \mathit{edge}(X,Y)). \tag{52}$$

According to Theorem 8, the node properties (51), (52) can be captured by ACR-GNNs. For (51) it is sufficient to use an ACR-GNN with a single ACR layer (42) of dimension $n_1 = 2$: the parameters $\boldsymbol{W}^1, \boldsymbol{U}^1, \boldsymbol{R}^1, \boldsymbol{b}^1$ can be set such that $\boldsymbol{h}^1(i)[0]$ becomes a 0,1-valued indicator for whether node i has at least 8 blue non-neighbors, and $\boldsymbol{h}^1(i)[1]$ indicates whether this number is no more than 10. Based on this representation, an output layer (8) can then provide an exact classification of (51). Similarly, (52) can be represented by a two-layer ACR-GNN with $n_1 = n_2 = 2$.

Table 2 ACR-GNN accuracies for α_2.

	$n_i =$				
m	2	4	16	64	128
1	0.756|0.407	0.698|0.697	0.830|0.713	0.831|0.716	0.833|0.714
2	0.696|0.502	0.683|0.649	0.833|0.704	0.891|0.886	0.869|0.795

Using the ACR-GNN implementation provided by the authors of [4] [1] we can re-create and extend some of their experiments. Table 2 shows the accuracies for the property α_2 that are obtained by ACR-GNN models trained on randomly generated graphs of sizes $40 \leq n \leq 50$. To test the generalization capabilities of the learned models, they are tested on graphs in the same size range, and also on a test set of slightly larger graphs with sizes ranging in $51 \leq n \leq 60$. In Table 2 the accuracies for these two different test sets are shown in the format $<$ *small graph accuracy* $>$ | $<$ *large graph accuracy* $>$. In our experiment we vary the number of layers $m = 1$ or $m = 2$, and their dimensions n_i (for $m = 2$, always $n_1 = n_2$). The base frequency of nodes with the α_2 property is 0.643 and 0.396 for the small and large graphs test sets, respectively. One can see that even though the architecture with $m = 2$ and $n_i = 2$ is sufficient to capture α_2 in principle, the stochastic gradient optimization here does not succeed to construct a model with an accuracy that is notably better than a baseline predictor. To obtain higher accuracies, a significant over-parameterization is required ($n_i = 64$ corresponds to the experimental setting of [4]). We will return to the benefits of over-parameterization in neural network learning in Section 12.3. For the simpler property α_1, a similar experiment leads to perfect accuracies of 1.0 in all settings, and for both test sets.

▶ **Example 11.** We now consider RBN models for the simpler target α_1 (51). We consider three different models:

- RBN-manual: a manually designed RBN that directly encodes the logical formula α_1 following the construction behind Theorem 9.
- RBN-gnn-learned: an RBN encoding of an ACR-GNN with one hidden layer of dimension 4, following the construction described in Section 9. As described in Example 10, this is (more than) enough to represent a precise model for α_1. Parameters learned from training data using stochastic gradient descent.
- RBN-gnn-manual: the same RBN structure as the previous, but parameters set manually to encode α_1 (most of the manually set parameters are zeros, since only two of the four dimensions are actually needed).

For all three models we can consider a *discriminative* model where the attribute *blue* is assumed to be a fixed input (as it would be for a GNN model), and a *generative* version where we also take *blue* to be probabilistic. This is easily effected by adding to the RBN a very simple formula

$$F_{blue(X)} \equiv 0.26$$

that specifies the marginal probability for a node to be blue. The value 0.26 is the empirical probability of *blue* in the training data, which in our two manual models is set manually, and in the learned model is learned jointly with the other model parameters. Note that the *edge* relation still is assumed to be given as input, so we still do not have a fully generative model for graphs, but only a conditional model for the attributes given the graph structure.

[1] `https://github.com/juanpablos/GNN-logic`

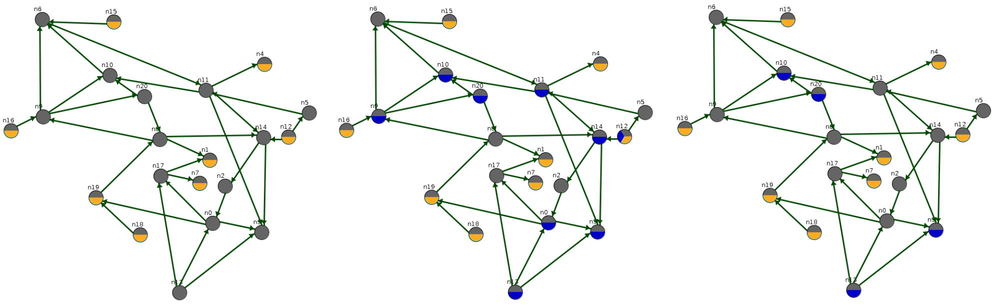


Figure 6 Most probable explanations in Example 11.

Evaluated on two test sets with different graph sizes, as described in Example 10, all three discriminative models achieve an accuracy of 1.0 for both test sets (recall that the same was observed for ACR-GNNs on the simple α_1 target). Using the generative models, we now can also consider queries beyond the prediction of α_1. One interesting type of query is to turn the role of input attribute *blue* and target α_1 around, and assume that given observed α_1 labels, we want to predict the (now) unobserved attribute *blue*. One particularly interesting version of this question is to ask for the *most probable explanation* (MPE) of the observed α_1 values, i.e. to find the assignment of *blue* attributes that makes the observed α_1's most probable.

Figure 6 on the left shows a small graph with 21 nodes where nodes with the observed α_1 label are marked by an orange color segment. Edges here are shown as directed, but this direction is ignored by the models. From any of our generative models we can now calculate an MPE assignment vector $\boldsymbol{b}$ for the *blue* attribute that maximizes the probability $P_V(\boldsymbol{\alpha}_1|\boldsymbol{b})$ (this is a non-trivial optimization problem, which is tractable for this small example). Figure 6 in the middle shows the MPE assignment of *blue* that is obtained from either of the two manual RBN models. In this explanation, there is a total of 9 blue nodes, and nodes have the α_1 label iff they are connected to at most one of these blue nodes, i.e., (51) holds. Performing MPE inference for the RBN-gnn-learned model yields the assignment of the *blue* attribute shown in the right graph of Figure 6. This explanation is inconsistent with the logical definition of the α_1 label, since here there are only 4 blue nodes in the domain, which implies that α_1 should be false for all nodes, contrary to the given observation. This indicates that the learned model does not generalize to this MPE inference task.

The reason for this failure of the learned model could be twofold: 1) the model, even though perfectly accurate on test sets that are very similar or only slightly larger than the training examples, does not generalize to our MPE query graph, which is smaller than the training examples, and may also exhibit different connectivity patterns of the *edge* relation; 2) the training objective is not appropriate for solving MPE tasks. Turning to point 2) first: as we will show in Section 12.2 below, the objective function we use for training the RBN is equivalent to the classification loss function used for training an ACR-GNN classifier. However, both are equivalent to maximizing the log-likelihood, which can be seen as a "universal" objective for learning a generative probabilistic model for all probabilistic inference tasks. This means that even if right from the beginning we had intended our model to be used for MPE inference as considered here, our training objective would not have been different. That indeed point 1) seems to be the issue is revealed when we now consider our original classification task for the graph of Figure 6: we use this graph with the *blue* attribute set as in the middle graph of Figure 6. For this input graph the learned model only achieves an accuracy of 0.47 for classifying the α_1 label for the 21 nodes. Thus, the learned model does not capture the logical nature (51) of the target well enough in order to generalize to input graphs that are rather different in size (and possibly structure) from the training examples.

Lastly, we consider an RBN model that combines domain knowledge with learning: suppose it is known that the α_1 label depends on the number of blue non-neighbors, but the exact bounds $[8, 10]$ in (51) are unknown. Our qualitative knowledge then is captured by the logical form of the probability formulas in RBN-manual. The missing quantitative information corresponds to unknown values of the numeric constants in these formulas. Learning these parameters gives us a model RBN-manual-learned with manually defined structure and learned parameters. This model then turns out to perform perfectly well both on our classification and MPE tasks.

10 Dealing with Homophily

A key property in network data is the phenomenon of *homophily*: connected entities tend to be similar. This phenomenon is particularly well documented for social networks, where, e.g., a *friendship* relation between two people is indicative of similar political leanings, social status, and other properties (cf. (13)). Similar in other types of networks: in bibliographic networks, papers citing each other are likely to be in the same subject area. In sensor or traffic networks, entities that are connected by a spatial proximity relation tend to have similar properties. When a given class label exhibits homophily, it will be important to exploit this for classification. Taking homophily into account has two different aspects:

- Collective classification: the prediction of class labels for entities should be done jointly for all (unlabeled) entities, such that the joint labeling exhibits homophily.
- Autoregression: the prediction of a class label depends on observed labels for some entities.

These two aspects are non-exclusive, but distinct. For illustration, consider the two small graphs in Figure 7. In the graph on the left an attribute with possible values 'yellow' and 'blue' is observed for all nodes, whereas the class label with values 'red' and 'black' is unobserved for all nodes. In the graph on the right, the class label is observed for three nodes. Assuming that the class label exhibits homophily (maybe this is learned from some other labeled training graphs), one would want to assign to all the nodes belonging to one of the two cliques in the graph the same label. This would be a case of collective classification, where the label assigned to one node constrains what label we assign to other nodes. In the graph on the right, homophily would indicate that the nodes in the clique at the top should be labeled 'red', and the nodes in the bottom clique be labeled 'black'. This would then require an autoregressive component in the classification. Homophily mostly plays a role in transductive problem settings, as illustrated by the graph on the right.

Among SRL frameworks, the Markov network constructor and probabilistic logic programming types are able to model homophily most easily. For Markov logic networks an example is already given by (13): the undirected nature of these logical feature specifications and the Markov network semantics fit very well mutual, symmetric dependencies between attributes due to homophily. Probabilistic logic programs can represent autocorrelation via clauses like

$$\mathit{republican}(X) \leftarrow \mathit{friends}(X, Y) \wedge \mathit{republican}(Y).$$

The least fixed-point semantics of the logic program then allows to propagate the *republican* label from some observed republicans to other unlabeled entities (the probabilistic component of the program would make this propagation probabilistic). For SRL frameworks of the

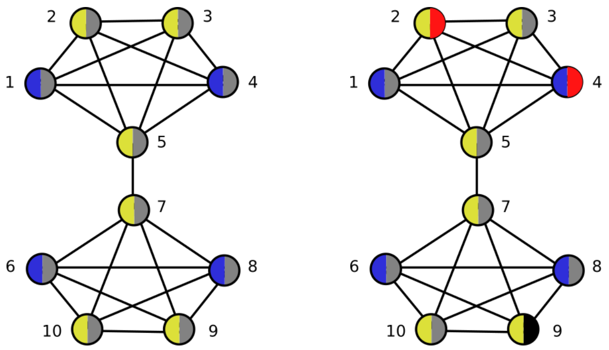


Figure 7 Homophily challenge.

Bayesian network constructor type the required acyclicity of probabilistic dependencies is a certain hurdle for a direct modeling of homophily. We will show below how this hurdle can be overcome.

GNNs encounter inherent challenges for dealing with homophily. A GNN for predicting a node class label is relying on node features other than the label for making this prediction. If nodes are indistinguishable based on the available features, they can only be assigned the same label (cf. Section 8). For our example in Figure 7 on the right this means that a predictive model for the class label that only is a function of the observed *color* attribute and *edge* relation must give identical labels to the nodes in the sets $\{1, 4, 6, 8\}$, $\{5, 7\}$ and $\{2, 3, 9, 10\}$, respectively, because nodes in these sets are pairwise indistinguishable based on these two relations. This limitation of GNNs has motivated the combination of GNNs with Markov logic networks in [59].

A standard strategy in probabilistic modeling for representing dependencies that cannot be explained by observed features is the introduction of *latent variables*. For RBNs, modeling with latent numeric relations has been introduced in [36]. Originally mostly motivated by applications in community detection, the same technique also applies to classification under homophily.

▶ **Example 12.** A numeric k-ary relation r simply is an assignment of a real number to ground atoms $r(i_1, \ldots, i_k)$ $(i_j \in V)$. Numeric relations can represent actual observable numeric data, such as numeric node attributes or edge weights. We use numeric relations as latent features that are not observed, and that are not part of the generative model. For the simple scenario as depicted in Figure 7 we may assume that both the *edge* relation and the class label depend on an unobserved node feature that determines both the propensity of nodes to connect by an edge, and the likely value of their class label. Representing this node feature by a latent numeric attribute *latent*(X), we can model the *edge* and *class* relation by the two formulas

$$\begin{aligned} F_{class(X)} &\equiv \texttt{COMBINE}\ latent(X)\ \ \texttt{WITH}\ logistic\ regression \\ F_{edge(X,Y)} &\equiv \texttt{COMBINE}\ latent(X) \cdot latent(Y)\ \ \texttt{WITH}\ logistic\ regression \end{aligned} \tag{53}$$

Assuming that all $i \in V$ are assigned a value $latent(i) \in \mathbb{R}$, the probability for $class(i)$ then simply is $1/(1 + e^{-latent(i)})$, and the probability for $edge(i, j)$ is $1/(1 + e^{-(latent(i) \cdot latent(j))})$ (we omit the *color* attribute here, since it is not instrumental for the prediction of the class label). The extremely simplistic model (53) could be refined by allowing more than one latent attribute (i.e., a latent feature vector, rather than a latent scalar value), and refining the functions that map latent feature values to probabilities.

Table 3 Latent feature values and predicted probabilities from model (53).

i	1	2	3	4	5	6	7	8	9	10
$latent(i)$	1.49	1.52	1.49	1.52	1.15	−1.49	−1.15	−1.49	−1.52	−1.49
$P(class(i) = red)$	0.81	[1.0]	0.81	[1.0]	0.75	0.18	0.24	0.18	[0.0]	0.18

Given observed *edge* and *class* data, one can learn values for the *latent* attribute that best explain the data based on the maximum likelihood principle (cf. Section 12.2 below). Importantly, learning latent numerical relations in this manner is treated just as part of the general parameter learning problem, and does not require any special purpose algorithms. Learning the latent values and predicting the class labels for the 10 nodes gives the result shown in Table 3. We see that nodes 1-5 in the upper clique are clearly classified as red, and those in the lower clique as black.

Somewhat analogous to the latent variable modeling approach we have taken here within the RBN setting are the *auto-encoders* in neural networks, and especially graph auto-encoders for graph data [42]. Graph auto-encoders also construct latent feature representations of nodes that are calibrated to explain observed edges with a function corresponding to our $F_{edge(X,Y)}$ in (53) (this type of model actually has a longer history in statistical graph analysis [26]). Originally proposed in the context of link prediction tasks, graph autoencoders could also be a basis for node classification under homophily.

11 Aggregation

11.1 Invariance and Sum Aggregation

The core element both in GNN and SRL models of relational data is the aggregation of features of neighboring nodes. Since there is no defined order on the neighbors of a node, such an aggregation should not be based on any assumed ordering. In our definitions, such as (7) for GNNs, and the definition for combination functions in Section 7 for RBNs, this has been taken into account by specifying that the aggregation must be a function of a representation of the neighbors as a multiset. Aggregation by summation as in (21) is consistent with such a multiset view of the inputs, because the sum does not depend on the order of summation. In practice, however, graphs are usually represented by adjacency matrices or adjacency lists that induce an (arbitrary) order on the nodes.

In the case of GNNs, matrix and vector representations of graphs and neighbor lists are often assumed in the basic definitions of the models (e.g. [41, 79, 74], whereas e.g. [24] uses set notation in the model specification). The requirement that the implicit ordering of nodes induced by these representations does not affect the results is then often just implicitly taken into account by only considering order independent operations like summation, or the maximum or minimum operators. A more explicit and systematic consideration of admissible aggregation operations on vector representations of multisets has been initiated in [77]. We here briefly review these initial results. Reformulating the definitions of [77] slightly, we formalize our requirement as follows.

▶ **Definition 13.** *Let $\mathcal{X}$ be a set (to be thought of as a set of possible feature values). A function*

$$f : \bigcup_{n\in\mathbb{N}} \mathcal{X}^n \to \mathbb{R}$$

is called an $\mathcal{X}$-tuple aggregator*. The function* f *is* permutation invariant*, if for all* n *and all permutations* π *of* $\{1,\dots,n\}$*, and all* $x_1,\dots,x_n \in \mathcal{X}$

$$f(x_1,\dots,x_n) = f(x_{\pi(1)},\dots,x_{\pi(n)}) \tag{54}$$

holds.

We note that regardless of the nature of $\mathcal{X}$, the value of the aggregator here is required to be a real number. This is somewhat contrary to the idea that an aggregator of values from $\mathcal{X}$ should return another value in $\mathcal{X}$. However, often $\mathcal{X}$ will itself just be $\mathbb{R}$, or a subset of $\mathbb{R}$, in which case the return value of f lies in the same space as the values it aggregates. The following is then proposed as Theorem 2 in [77], and here given in a generalized form as described in Appendix A.3 in [73].

▶ **Theorem 14** ([77, 73])**.** *Let* $\mathcal{X}$ *be countable. Then an* $\mathcal{X}$-tuple aggregator f *is permutation invariant iff there exist functions* $\phi : \mathcal{X} \to \mathbb{R}$ *and* $\rho : \mathbb{R} \to \mathbb{R}$*, such that*

$$f(x_1,\dots,x_n) = \rho(\sum_i \phi(x_i)). \tag{55}$$

The restriction of this theorem to countable $\mathcal{X}$ has caused some concern already in [77], and amplified in [73]. However, countability of $\mathcal{X}$ per se is not a major problem: in reality, initial (node) features will be categorical attributes (or one-hot encodings thereof), or finite precision numerical attributes. Thus, a countable $\mathcal{X}$, in principle, is sufficient to represent such initial features. Importantly, then, if $\mathcal{X}$ is countable, then so is $\bigcup_{n\in\mathbb{N}} \mathcal{X}^n$, and hence the range of f in $\mathbb{R}$. This means that countability of the feature space $\mathcal{X}_k$ at the kth GNN layer will be guaranteed, if the input feature space is countable.

The implications of Theorem 14 have sometimes been overstated: by appealing to universal function approximation properties of neural networks [28] it is suggested that it is sufficient to use summation for aggregation, in combination with additional perceptron layers that are trained to implement ρ and ϕ (e.g.[23, Section 5.2.2], [75]). However, as already pointed out in [73], the function approximation properties of neural networks are not sufficient in this case. To illustrate this point, we here reproduce the proof of Theorem 14 as given in [73].

Proof of Theorem 14. Let $x_1, x_2, \dots$ be an enumeration of $\mathcal{X}$. Let $p_1, p_2, \dots$ be an enumeration of all prime numbers. Define $\phi(x_i) := -\log p_i$. Then, by the unique prime factorization property of the integers, for any tuple $\boldsymbol{x} = (x_{i_1},\dots,x_{i_n}) \in \mathcal{X}^n$ the sum $\sum_{j=1}^n \phi(x_i) = \log \frac{1}{\prod_{j=1}^n p_{i_j}}$ is an $\mathbb{R}$-valued code $\Phi(\boldsymbol{x})$ for $\boldsymbol{x}$, such that the encoding Φ is injective on $\bigcup_{n\in\mathbb{N}} \mathcal{X}^n$, and hence invertible. One can now simply define $\rho(r) := f(\Phi^{-1}(r))$ to obtain a representation of f in the form (55). ◀

The functions ϕ, ρ as constructed in this proof are outside the scope of the universal representation theorems for neural networks, which only apply to functions that are defined on a compact subset of the reals, and are continuous. To even be able to consider continuity in the sense of the representation theorems, the function ϕ would need to be defined on $\mathbb{R}$ (or $\mathbb{R}^m$, for some $m > 1$), not on $\mathcal{X}$. This, however, can be overcome by assuming (without much loss of generality) that $\mathcal{X} \subset \mathbb{R}$, and that ϕ actually is defined on all of $\mathbb{R}$. However, then an extension to $\mathbb{R}$ of ϕ as constructed in the proof can either not be continuous (if $\mathcal{X}$ has an accumulation point in $\mathbb{R}$), or not be limited to a compact set (if $\mathcal{X}$ is unbounded in $\mathbb{R}$). Due to these limitations of Theorem 14 there is still a need to consider aggregators outside the class defined by (55). Possibilities include the use of a fixed selection of standard aggregators [10], or the use of a parametric family of aggregators whose trainable parameters can be optimized to learn customized aggregation functions for each learning task [55].

11.2 Injectivity and Expressivity

The key element of the proof of Theorem 14 is the construction of an injective function $\Phi(\boldsymbol{x})$ on the space of multisets $\boldsymbol{x}$. The problem of constructing injective functions on multisets of feature values has been a key element in the study of the expressiveness of GNNs, also outside the context of investigating the universality properties of sum-aggregation (e.g.[75, 64]). An injective aggregation function would allow to preserve the full information of a node's neighbors' features $\{\!\!\{\boldsymbol{h}^k(j) | j \in N_i\}\!\!\}$ in the updated node representation $\boldsymbol{h}^{k+1}(i)$, and thereby enable the construction of maximally discriminative node (or graph) classifiers. However, classic results in mathematics impose strict limits to the endeavor of implementing injective aggregation functions using continuous functions as provided by neural network layers. For our purpose, we can formulate this as follows:

▶ **Theorem 15** ([8, 70]). *There does not exist a continuous injective function from $\mathbb{R}^n$ to $\mathbb{R}^m$ if $n > m$.*

In our context, n in this theorem would be the cardinality of a multiset of real numbers, and usually $m = 1$ as the target dimension for the aggregator. We now have to be a little bit careful, since our desired permutation invariance (54) actually says that we want our aggregator not to be injective on $\mathbb{R}^n$, but only to return distinct values for different multisets. We can use ordered vectors as unique representatives for multisets: defining

$$\mathbb{R}^{n,\leq} := \{(r_1, \ldots, r_n) \in \mathbb{R}^n | r_i \leq r_{i+1}, i = 1, \ldots, n-1\}, \tag{56}$$

we obtain a one-to-one correspondence of multisets of cardinality n and $\mathbb{R}^{n,\leq}$. We can now re-state Theorem 15 as

▶ **Theorem 16** ([8, 70]). *There does not exist a continuous injective function from $\mathbb{R}^{n,\leq}$ to $\mathbb{R}^m$ if $n > m$.*

This theorem imposes limits on the possibility of constructing general and expressive aggregation function for the feature space $\mathcal{X} = \mathbb{R}$ already in the case of a fixed cardinality of multisets. However, the theorem does not exclude the possibility of continuous functions on $\mathbb{R}^{n,\leq}$ that are injective on a countable subset $\mathcal{X}^n \subset \mathbb{R}^n$, which, as argued above, may be all we need. In fact, as the following example shows, this can be done.

▶ **Example 17.** Let $\mathcal{X} = \mathbb{N}$. We construct a function $f : \cup_{n\in\mathbb{N}}\mathbb{R}^n \to \mathbb{R}$, such that:

(i) the restrictions of f to arguments of fixed dimensions n are continuous;
(ii) f is permutation invariant;
(iii) f is injective for multisets of values from $\mathcal{X}$.

We first map vectors in $\mathbb{R}$ to their ordered representatives in $\mathbb{R}^{n,\leq}$:

$$f_{ord} : \cup_{n\in\mathbb{N}}\mathbb{R}^n \to \cup_{n\in\mathbb{N}}\mathbb{R}^{n,\leq}.$$

In procedural terms, $f_{ord}(\boldsymbol{r})$ is the application of a sorting algorithm to $\boldsymbol{r} \in \mathbb{R}^n$. Seen as mapping from $\mathbb{R}^n$ to $\mathbb{R}^{n,\leq}$ this is a function that satisfies properties (i)–(iii). In fact, f_{ord} is not only continuous, but also differentiable. For vectors in $\boldsymbol{r} \in \mathbb{R}^{n,\leq}$ we now define

$$f_{prime} : \boldsymbol{r} \mapsto \prod_{j=1}^{n} p_j^{r_j},$$

where, as in the proof of Theorem 14, p_j is the j'th prime number. As in that proof, the unique prime factorization of integers implies that the restriction of f_{prime} to $\cup_{n\in\mathbb{N}}\mathcal{X}^{n,\leq}$ is injective. Since f_{prime} also is continuous, we then obtain that $f = f_{prime} \circ f_{ord}$ satisfies (i)–(iii).

Table 4 Learning in GNN and SRL: a summary of correspondences.

		GNN	SRL
Structure	Space	NN architectures	(Logical) model structure
	Manual specification by	NN engineers	SRL experts, domain experts
	Learned by	Optimization/search in combinatorial spaces	
Parameters	Space	High-dimensional	Low-dimensional
	Manual specification	Never	Possible
	Objective	Loss function (cross-entropy, MSE, . . .)	Likelihood (plain, conditional, pseudo-, . . .)

In theoretical terms, the function f we have constructed has all the properties one could desire. In addition to (i)-(iii) it also has the important property that for $\boldsymbol{r} \in \mathbb{N}^n$ the value $f(\boldsymbol{r})$ is again an element of $\mathbb{N}$, so that the same f can be used over multiple iterations of aggregation. The requirement that the initial input features are in $\mathbb{N}$ is not a serious (theoretical) limitation, since any countable $\mathcal{X}$ could be mapped into an integer encoding. Note that a mapping of an initial input feature space $\mathcal{X}$ to an integer encoding is not subject to the continuity concerns that we otherwise have, since it can be implemented as a data preprocessing step, and need not be computed by internal (continuous) neural network functions.

In practical terms, however, f is unmanageable, due to the very large numbers produced by f_{prime}, which would soon cause numeric overflow in an implementation. Furthermore, f_{prime} again does not fulfill the requirements of the universal approximation results for neural networks. It appears to be an open question whether a function f with (i)-(iii) can be constructed that is numerically manageable, such that it can be approximated by standard neural architectures.

Theoretical questions about permutation invariance, canonical forms (based on summation as the core aggregation step), and expressivity of aggregation functions that have arisen in the field of GNNs have not been considered previously in analogous lines of investigation in SRL, even though aggregating (or combining) information from related entities also is a core element of SRL modeling. There are several reasons for this: first, permutation invariance only becomes an issue when one represents graphs by adjacency matrices. The logic-based background of SRL, and the associated “possible worlds” view of multi-relational graphs, favors a representation of graphs as the set of ground atoms that are true. When, in this manner, all fundamental definitions about syntax and semantics of SRL models are based on sets (or multisets) rather than matrices and vectors, permutation invariance never becomes an issue. The characterization of general, canonical forms of aggregation, however, would still be of interest at least for those SRL frameworks that include explicit aggregation or combination operators: these are most, if not all, of the frameworks that fall into the Bayesian network constructor category, as exemplified by the combination function construct in RBNs. Markov network constructors and probabilistic logic programming approaches, on the other hand, perform aggregation more implicitly through a single, fixed multiplicative mechanism (based on products, rather than sums as in (55).

12 Parameter and Structure Learning

In this section we discuss the role of structure and parameters in SRL and GNN learning. A summary of some of the correspondences we find is given in Table 4.

12.1 Structure

An SRL model consists of a “structure” that is given by dependency relations expressed using a logic-based or graphical representation, and numeric parameters that are needed to quantitatively define a probability distribution. The learning task for SRL models then consists of the two parts of learning the structure, and learning the parameters. In Markov logic networks, for example, the structure consists of all the logical clauses (13), and the parameters of the numeric weights attached to these clauses. In the ProbLog probabilistic logic programming language, the structure consists of clauses of the form (14), together with probabilities assigned to certain ground facts. In RBNs, the structure consists of the functional form of probability formulas, and parameters are the constants (29) of the formula. Since the structure represents interpretable, meaningful dependencies between different relations and attributes, it may also be elicited (at least in part) by domain experts.

▶ **Example 18.** Consider again the scenario of Example 3, and suppose one wants to create an SRL model for predicting whether a person should be labeled as an influencer. A social network expert would probably be able to say that whether or not a person is an influencer depends (maybe among other factors) on the number of his/her followers. In a probabilistic logic programming framework, this would lead us to include the clause

$$\mathit{influencer}(X) \leftarrow \mathit{follower}(Y, X) \tag{57}$$

in our model. The clauses in Markov logic networks do not represent directed implications like (57) but undirected logical properties or features that are deemed relevant for the probability of a possible world. Our knowledge about a connection between the *influencer* attribute and the *follower* relation can be incorporated into the model by constructing several features that express combinations of these two:

$$\begin{array}{l} \mathit{influencer}(X) \vee \mathit{follower}(Y, X) \\ \mathit{influencer}(X) \vee \neg\mathit{follower}(Y, X) \\ \neg\mathit{influencer}(X) \vee \mathit{follower}(Y, X) \\ \neg\mathit{influencer}(X) \vee \neg\mathit{follower}(Y, X) \end{array} \tag{58}$$

The fact that a greater number of followers increases the probability for being an influencer would here be encoded not already through the structure of the model, but by the relative magnitudes of the numeric weights associated with these four different features. Relational Bayesian networks again encode directed dependencies. Here our knowledge would imply that the probability formula for the *influencer* attribute should include the construct

$$\begin{array}{lll} F_{\mathit{influencer}(X)} \equiv & \ldots & \\ & & \texttt{COMBINE} < \mathit{probability\ formula} > \\ & & \texttt{WITH} < \mathit{combination\ function} > \\ & & \texttt{FORALL}\ Y \\ & & \texttt{WHERE}\ \mathit{follower}(Y, X) \\ & \ldots & \end{array} \tag{59}$$

This is only a partial specification of the probability formula for *influencer*, which leaves open the details of how the dependency on the number of followers should be aggregated in the combination function construct, and what other dependencies of the *influencer* attribute need to be encoded in its probability formula.

There is an apparent decrease in the ease-of-use from the logic programming via the Markov network to the RBN framework for encoding expert knowledge in the model structure. The first two frameworks allow modular specifications where different pieces of knowledge can be represented by separate logic-based representations. In the case of RBNs, all relevant knowledge for the attribute *influencer* needs to be collected in the single probability formula $F_{influencer(X)}$. If additional knowledge was provided that the *influencer* attribute also depends on a known *has_youtube_channel* attribute, then in a probabilistic logic or Markov logic framework this could be incorporated by adding new clauses, leaving the existing (57)(58) untouched. In RBNs, on the other hand, the additional knowledge regarding *has_youtube_channel* needs to be integrated with the previous knowledge inside the formula (59).

However, modular specifications, though intuitive on the surface, pose their own challenges. In pure logic, the total knowledge expressed by a set of formulas is simply the conjunction of the knowledge expressed by each single formula. In a logic-based, modular specification of a generative probabilistic model, the semantic contribution of each model component to the overall probability distribution defined by the model can not be defined by a "local semantics" of the component. The impact of each component on the probabilities defined always depends on the full model that it is part of. In short, syntactic modularity here does not translate into semantic modularity.

It should be apparent, now, that even though SRL frameworks support the integration of domain knowledge into the model construction process, this can not happen without a thorough understanding of the semantics and algorithmics of the SRL framework being used. Thus, expert-driven model development here requires both a domain and an SRL expert.

This example has highlighted the ability to (partially) construct the structure of an SRL model manually based on domain knowledge. It is, of course, a central objective to also use machine learning for determining the structure of a model, which leads to search and optimization problems in very large combinatorial spaces of possible model structures. For probabilistic logic programming frameworks, often search techniques from the field of inductive logic programming are adapted (e.g.[5]). Structure learning for Markov logic networks has received a particularly large amount of attention. Here inductive logic programming techniques have also been exploited [43, 50]. Other approaches (e.g. [38]) exploit more novel machine learning techniques, but still include an element of heuristic (beam) search over possible clauses. For RBNs in the structure learning problem has only been addressed in the context of a somewhat simplified framework [33].

A (graph) neural network model also consists of a structure (here often called the architecture) and its parameters. However, the balance between the tasks of structure design, or structure learning, on the one hand, and parameter learning on the other hand, is quite different. Whereas in SRL structure learning is perhaps the greatest and most fundamental challenge, one would typically view the learning problem of a GNN almost exclusively as a parameter optimization task. The network architecture may either be taken to be a given "standard solution", or obtained from a manageable set of candidate architectures via tuning such hyperparameters as the number and dimensions of network layers. The increasing complexity and variability of available neural components, however, also has given rise to the field of *neural architecture search* [14], which begins to share some characteristics with SRL structure learning.

12.2 Parameter learning

An SRL model usually contains only a moderate number of numerical parameters (the bound < 100 parameters probably covers a large fraction of SRL models presented in the literature). When these parameters have a clear, interpretable, statistical meaning then even parameters may be amenable to specification by domain experts. For example, in the probability formula

$$F_{\mathit{influencer}(X)} \equiv \texttt{WIF}\ \mathit{has\_youtube\_channel}(X)\ \texttt{THEN}\ \ 0.35\ \texttt{ELSE}\ \ 0.01$$

the parameters 0.35 and 0.01 correspond to the statistical frequencies of influencers among people who do, respectively do not, own a Youtube channel. In the absence of adequate training data, such parameters could be assessed (approximately) by human experts. In most cases, however, parameters of an SRL model should be learned from data.

Suppose, then, that an SRL structure has been fixed, and that this structure is parameterized by k real-valued parameters. Then a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^k$ defines an SRL model, i.e., for each domain V we have the probability distribution $P_V^{\boldsymbol{\theta}}$ on $\mathcal{G}(V, \mathcal{R})$ (the signature $\mathcal{R}$ always being fixed). Training data consists of a number of observed graphs $(V_1, \boldsymbol{R}_1), \ldots, (V_N, \boldsymbol{R}_N)$. The domains of the training graphs may be different, may be all the same $V_1 = \ldots, V_N$, or the data may only consist of a single graph ($N = 1$). In all cases, we can score $\boldsymbol{\theta}$ by the log-likelihood:

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P_{V_i}^{\boldsymbol{\theta}}(\boldsymbol{R}_i). \tag{60}$$

In the case of Markov logic networks, the exact probabilities $P_{V_i}^{\boldsymbol{\theta}}(\boldsymbol{R}_i)$ are intractable to compute, and an approximate *pseudo-likelihood* is used instead. We note that our objective (60) does not contain a regularization term. This is because the problem of overfitting can arise (and must be dealt with) already at the stage of structure learning/design.

A major strength of probabilistic generative models lies in their ability to learn from incomplete data. Suppose our data consists of partially observed graphs $(V_1, \tilde{\boldsymbol{R}}_1), \ldots, (V_N, \tilde{\boldsymbol{R}}_N)$ where the domains V_i are fully observed, but for the relations we have only partial observations $\tilde{\boldsymbol{R}}_i$, meaning that for $R \in \mathcal{R}$ and entities $h, j \in V_i$ the atom $R(h, j)$ may have values *true*, *false* and *unknown*. Being a generative model, our current parameters then define for each possible completion $\boldsymbol{R}_i$ of $\tilde{\boldsymbol{R}}_i$ the probability

$$P_{V_i}^{\boldsymbol{\theta}}(\boldsymbol{R}_i | \tilde{\boldsymbol{R}}_i). \tag{61}$$

Taking the probability distribution over complete observations thus defined as an imputed complete dataset, one can then apply optimization techniques for complete datasets to optimize the parameter $\boldsymbol{\theta}$. Iterating the steps of imputing the *expected* complete data, and *maximizing* the likelihood function for the imputed data gives the famous *Expectation-Maximization (EM)* algorithm for learning from incomplete data. The EM algorithm is a general paradigm of almost universal applicability in statistical learning. However, in order to be feasible in practice for a particular model class (SRL or other), efficient techniques have to be developed for that particular model class to implement the computation of expected completions, and the subsequent optimization of the parameters. The expectation step often is computationally quite expensive, which means that learning from incomplete data usually is significantly more time consuming than learning from complete data.

GNN parameters are learned by minimizing a *loss function.* When the task for which a GNN is trained is classification, then usually the *cross-entropy loss* is used. Assume, for example, that the task is classification of a Boolean node label $C(X)$, and that we use a GNN architecture along the lines shown in Figure 2 on the left, where the output layer applies a *softmax* function to guarantee that the outputs represent a probability distribution over the possible class labels. Training data will consist of labeled nodes, which in general could be given by training examples

$$(V_1, C(j_1), \boldsymbol{R}_1), \ldots, (V_N, C(j_N), \boldsymbol{R}_N)$$

where j_i is a node in V_i, and $C(j_i) \in \{\mathit{true}, \mathit{false}\}$ is the observed label. Again, a fixed signature $\mathcal{R}$ for attributes and relations (other than the class label C) is given, and $\boldsymbol{R}_i$ consists of complete observations of $\mathcal{R}$ for V_i. Often, all examples will come from a single graph, i.e. $V_1 = \ldots = V_N$ and $\boldsymbol{R}_1 = \ldots, \boldsymbol{R}_N$. Let $\boldsymbol{\theta}$ be a setting for the weights in the network. Then the network produces outputs $\boldsymbol{o}^{\boldsymbol{\theta}}(j_i)$ that are 2-dimensional non-negative vectors for which $\boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[0] + \boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[1] = 1$. Assuming that the first output component is associated with the label *true*, and the second with the label *false*, we then obtain the cross-entropy loss

$$-\left(\sum_{i:C(j_i)=\mathit{true}} \log \boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[0] + \sum_{i:C(j_i)=\mathit{false}} \log \boldsymbol{o}^{\boldsymbol{\theta}}(j_i)[1] \right). \tag{62}$$

Under the probabilistic interpretation of the outputs, and the assumption that all training examples are independent, this is the negative log-likelihood. In particular, considering the case $(V_i, \boldsymbol{R}_i) = (V_{i'}, \boldsymbol{R}_{i'})$, (62) incorporates the conditional independence assumption (26) for the distribution $P_V(C|\mathcal{R})$. Thus, the loss minimization objective of GNN training here is exactly the same as the likelihood maximization objective in learning an SRL model for the conditional distribution $P_V(C|\mathcal{R})$, if the SRL model makes assumption (26). The latter is the case, for example, when $P_V(C|\mathcal{R})$ is specified by an RBN consisting of a single probability formula $F_{C(X)}$. Parameter (weight) vectors $\boldsymbol{\theta}$ of GNNs are usually much larger than those of SRL models, and overfitting can also occur as a result of pure parameter learning. Therefore, the cross-entropy loss (62) will often be combined with a regularization term for $\boldsymbol{\theta}$.

12.3 From sparse to over-parameterizations

As noted in the preceding sections, SRL and GNN models are typically distinguished by huge differences in the size of their parameterizations. SRL models combine structure that encodes relevant features and dependencies with a sparse parameterization that quantifies these dependencies. GNNs follow the deep learning philosophy that feature discovery is automated as a part of the parameter learning problem [22, Chapter 1]. The use of high-dimensional parameter spaces in GNN architectures serves two distinct purposes:

- *Model capacity:* providing a rich hypothesis space that can capture complex relevant features.
- *Facilitating optimization:* gradient descent is more effective in higher-dimensional spaces.

We speak of over-parameterization when a model architecture contains more tunable parameters than are actually required to perfectly capture a target concept. As we have observed in Example 10, neural network training can be more effective in overparameterized than 'minimally sufficient' model architectures. This somewhat counter-intuitive observation has been made and studied by many authors, e.g. [52, 9]. A partial explanation is provided by consideration of the limit case where the dimensions of hidden layers (and number of weight parameters) goes to infinity [6, 2]. In this limiting case, the last hidden layer will contain sufficiently rich features (regardless of the weight settings at lower layers) such that learning can be reduced to a convex optimization problem for the weights at the output layer [6].

13 Conclusion and Outlook

We have studied similarities and differences in graph and network analysis using the tools of statistical relational learning and graph neural networks. We have emphasized the commonalities of these two paradigms, especially with regard to SRL frameworks of the Bayesian network constructor type. In particular for the relational Bayesian network framework we demonstrated the capability to directly encode GNNs without modifications or additions to the original RBN framework. This directly enables forms of neuro-symbolic integration by RBN models that combine neural encoding components with higher-level symbolic representations. As we have seen in Section 9, this can be exploited to tackle a larger variety of inference tasks, and to combine learning with expert-driven model specifications. However, in order to obtain maximal benefits from such combinations, several challenges have still to be met:

- Interpretability: symbolic representations are typically more interpretable for a human user than a neural network model. However, an RBN component that directly encodes a GNN module is not more interpretable than the original GNN. A challenge therefore is whether a learned RBN containing over-parameterized GNN components can be reduced to a smaller and more interpretable model, e.g. by some form of model distillation. In contrast to other approaches towards interpretability via surrogate model learning (e.g. [61]), the original and the simplified model here would live both in the same hypothesis space of RBNs.
- Trading structure learning for parameter learning: the success of GNN technology indicates that the overall strategy of reducing the learning problem as much as possible to a parameter learning problem has advantages over the SRL strategy that places primary importance on the structure of a model. This leads to the question of whether structure learning can be reduced to parameter learning. A very small-scale and simplistic instance of this was already presented in [35], where it was suggested to learn the appropriate combination function in a model by learning a mixture model over several candidate combination functions, and then selecting the one with the dominant coefficient in the learned mixture.
- Closing the efficiency gap: as discussed in Section 9, an RBN encoding of a GNN is mathematically equivalent, and, in principle, has the same parameter learning complexity as the original GNN. In practice, however, there is a significant gap, because the general purpose RBN algorithms do not automatically leverage the limited functional structures encountered in GNN encodings.

References

1 Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of IJCAI 2021*, 2021.

2 Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

3 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

4 Pablo Barceló, Egor Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020.

5 Elena Bellodi and Fabrizio Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(2):169–212, 2015.

6 Yoshua Bengio, Nicolas Le Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. *Advances in neural information processing systems*, 18:123, 2006.

7 J. S. Breese, R. P. Goldman, and M. P. Wellman. Introduction to the special section on knowledge-based construction of probabilistic decision models. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(11), 1994.

8 Luitzen EJ Brouwer. Beweis der Invarianz des n-dimensionalen Gebiets. *Mathematische Annalen*, 71(3):305–313, 1911.

9 Alon Brutzkus and Amir Globerson. Why do larger models generalize better? a theoretical perspective via the xor problem. In *International Conference on Machine Learning*, pages 822–830. PMLR, 2019.

10 Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33, 2020.

11 Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. Scalable deep generative modeling for sparse graphs. In *International Conference on Machine Learning*, pages 2302–2312. PMLR, 2020.

12 L. De Raedt. *Logical and Relational Learning.* Springer, 2008.

13 R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1319–1325, 2005.

14 Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

15 Varun Embar, Sriram Srinivasan, and Lise Getoor. A comparison of statistical relational learning and graph neural networks for aggregate graph queries. *Machine Learning*, pages 1–20, 2021.

16 Herbert B Enderton. *A mathematical introduction to logic.* Elsevier, 2001.

17 Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

18 N. Friedman, Lise Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

19 Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pages 3419–3430. PMLR, 2020.

20 L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning.* MIT Press, 2007.

21 Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

22 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

23 William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

24 William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1024–1034, 2017. URL: `http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs`.

25 D. Heckerman, C. Meek, and D. Koller. Probabilistic entity-relationship models, PRMs, and plate models. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning.* MIT Press, 2007.

26 Peter D Hoff, Adrian E Raftery, and Mark S Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

27 Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

28 Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

29 Manfred Jaeger. Relational Bayesian networks. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13)*, pages 266–273, Providence, USA, 1997. Morgan Kaufmann.

30 Manfred Jaeger. On the complexity of inference about probabilistic relational models. *Artificial Intelligence*, 117:297–308, 2000.

31 Manfred Jaeger. Model-theoretic expressivity analysis. In L. De Raedt, K. Frasconi, P.and Kersting, and S.H. Muggleton, editors, *Probabilistic Inductive Logic Programming*, volume 4911 of *LNCS*, pages 325–339. Springer, 2008.

32 Manfred Jaeger. Probabilistic logic and relational models. In Reda Alhajj and Jon Rokne, editors, *Encyclopedia of Social Network Analysis and Mining*, pages 1–15. Springer New York, New York, NY, 2017. doi:10.1007/978-1-4614-7163-9_157-1.

33 Manfred Jaeger, Marco Lippi, Andrea Passerini, and Paolo Frasconi. Type extension trees for feature construction and learning in relational domains. *Artificial Intelligence*, 204:30–55, 2013. doi:10.1016/j.artint.2013.08.002.

34 Manfred Jaeger. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32:179–220, 2001.

35 Manfred Jaeger. Parameter learning for relational Bayesian networks. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

36 Jiuchuan Jiang and Manfred Jaeger. Numeric input relations for relational learning with applications to community structure analysis. *CoRR*, abs/1506.05055, 2015. arXiv:1506.05055.

37 K. Kersting and L. De Raedt. Towards combining inductive logic programming and Bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP-2001)*, Springer Lecture Notes in AI 2157, 2001.

38 Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. Learning Markov logic networks via functional gradient boosting. In *2011 IEEE 11th international conference on data mining*, pages 320–329. IEEE, 2011.

39 Angelika Kimmig, Bart Demoen, L De Raedt, V. Santos Costa, and Ricardo Rocha. On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming*, 11(2-3):235–262, 2011. doi:10.1017/S1471068410000566.

40 Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, 2014. arXiv:1412.6980.

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

42 Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint*, 2016. arXiv:1611.07308.

43 Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448, 2005.

44 Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

45 Kathryn Blackmond Laskey. MEBN: A language for first-order Bayesian knowledge bases. *Artificial Intelligence*, 172(2-3):140–178, 2008. doi:10.1016/j.artint.2007.09.006.

46 Kathryn Blackmond Laskey and Suzanne M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. In *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 334–341, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

47 Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint*, 2018. `arXiv:1803.03324`.

48 Yao Ma, Suhang Wang, Chara C Aggarwal, Dawei Yin, and Jiliang Tang. Multi-dimensional graph convolutional networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 657–665. SIAM, 2019.

49 Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31:3749–3759, 2018.

50 Lilyana Mihalkova and Raymond J Mooney. Bottom-up learning of Markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*, pages 625–632, 2007.

51 Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019.

52 Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2018.

53 L. Ngo and P. Haddawy. Probabilistic logic programming and Bayesian networks. In *Algorithms, Concurrency and Knowledge (Proceedings ACSC95)*, Springer Lecture Notes in Computer Science 1023, pages 286–300, 1995.

54 Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

55 Giovanni Pellegrini, Alessandro Tibo, Paolo Frasconi, Andrea Passerini, and Manfred Jaeger. Learning aggregation functions. In *Proceedings of the Thirty International Joint Conference on Artificial Intelligence (IJCAI-21)*. International Joint Conferences on Artificial Intelligence, 2021.

56 Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

57 D. Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

58 David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

59 Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph Markov neural networks. In *International conference on machine learning*, pages 5241–5250. PMLR, 2019.

60 Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189, 2016.

61 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

62 M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

63 S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition edition, 2010.

64 Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint*, 2020. `arXiv:2003.04078`.

65 Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.

66 T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*, pages 715–729, 1995.

67 Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

68 Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

69 Gustav Šourek, Filip Železnỳ, and Ondřej Kuželka. Beyond graph neural networks with lifted relational neural networks. *Machine Learning*, pages 1–44, 2021.

70 J. van Mill. Domain invariance. Encyclopedia of Mathematics. URL: `http://encyclopediaofmath.org/index.php?title=Domain_invariance&oldid=16623`.

71 Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

72 Clément Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In *NeurIPS*, 2020.

73 Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494. PMLR, 2019.

74 Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2021.

75 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

76 Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.

77 Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: `https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

78 Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

79 Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Automating Moral Reasoning

Marija Slavkovik ✉ 🏠
University of Bergen, Norway

Abstract

Artificial Intelligence ethics is concerned with ensuring a nonnegative ethical impact of researching, developing, deploying and using AI systems. One way to accomplish that is to enable those AI systems to make moral decisions in ethically sensitive situations, i.e., automate moral reasoning. Machine ethics is an interdisciplinary research area that is concerned with the problem of automating moral reasoning. This tutorial presents the problem of making moral decisions and gives a general overview of how a computational agent can be constructed to make moral decisions. The tutorial is aimed for students in artificial intelligence who are interested in acquiring a starting understanding of the basic concepts and a gateway to the literature in machine ethics.

2012 ACM Subject Classification Computing methodologies → Philosophical/theoretical foundations of artificial intelligence

Keywords and phrases Machine ethics, artificial morality, artificial moral agents

Digital Object Identifier 10.4230/OASIcs.AIB.2022.6

Category Invited Paper

1 Introduction

Artificial intelligence (AI) is concerned with the problem of using computation to automate tasks that require intelligence [13]. Artificial intelligence, since 1956 when it was named and established [35], has been increasingly contributed towards automating tasks that require manipulation of information, production line tasks (robotics) and most recently pattern identification and learning [15].

In a society, we all affect each other with our activities and decisions. Ethics (or moral philosophy) is concerned with understanding and recommending right and wrong behaviours and decisions [25]. The right decisions being characterised by taking into consideration not only ones own interest, but also the interest of others [29]. The more computationally automated tasks are used to complement or replace people's tasks, the more concerns we have to ensure that the resulting actions and choices are not only correct and rational, but also do not have a negative ethical impact on society. As Rosalind Picard puts it "The greater the freedom of a machine, the more it will need moral standards" [42]. AI Ethics is a new, interdisciplinary, sub-field of AI that aims to address precisely this issue.

One way to ensure that AI has a non-negative ethical impact on society is to ensure that we do have an insight into, and measures to control the impact AI has [22]. Various different research approaches are being developed towards this end, in computer science, but also in philosophy, organisational science, law etc. Algorithmic accountability studies how to ensure that society and stakeholders can establish the right relationship with the people who research, develop, deploy and use AI algorithms [53]. Transparency is concerned with ensuring that the adequate type of information about how an AI system works is made available to a given stakeholder [20, 54]. Fairness is concerned with ensuring that like individuals and groups are treated alike by decision-making algorithms [17]. Explainable AI is concerned with the problem of finding ways to extract information from AI algorithms that justifies the choices that algorithm takes and use that information to adequately explain that information to a given stakeholder [34, 28].

International Research School in Artificial Intelligence in Bergen (AIB 2022).
Editors: Camille Bourgaux, Ana Ozaki, and Rafael Peñaloza; Article No. 6; pp. 6:1–6:13
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Another way to ensure AI has a non-negative ethical impact is to consider that moral reasoning is itself a cognitive task that we can consider automating. Machine ethics, or artificial morality, is a sub-field in AI that is researching this approach. In general, machine ethics is "is concerned with the behaviour of machines towards human users and other machines" [5]. The problem of automating moral reasoning can be considered as a problem of moral philosophy, whereas one is interested in questions such as: should machines be enabled with ethical reasoning [24, 8], which norms should machines follow [33], can machines ever be moral agents [16], etc. As a problem of computer science, machine ethics focuses on the question of how to automate moral reasoning [6, 51].

Here we are concerned with the question of how to automate moral reasoning. Although this problem, and machine ethics in general, have been raised since 2006 [5], it is an extremely difficult problem that requires a lot of improvement in the state of the art in AI and moral philosophy. We discuss the basic approaches in machine ethics, the advantages and challenges of each. These lecture notes are structured as follows.

We start with Section 2 in which we discuss what is decision making and how decision-making is distinguished from moral decision-making. Decisions are made by an agent. In Section 3 we discuss what computational agents are, what does it mean for a computational agent to be autonomous and what kind of moral agents can computational agents be. One way to automate moral reasoning is to follow a specific moral theory. In Section 4 we give a very quick overview of what is a moral theory and some of the more known moral theories from moral philosophy. In Section 5 we discuss two general approaches to building artificial moral agents, we discuss open research problems and challenges. In Section 6, we end with a discussion on how to find out more about machine ethics, beyond the scope of this tutorial.

2 Moral decision making

Decision-making is a cognitive process that is studied from many disciplines including cognitive science, neuroscience, psychology, and economy. Decision theory is a field which studies the choices that an agent does (descriptive decision theory) and should (prescriptive decision theory) make when faced with a formally specified decision problem [41]. Artificial Intelligence typically follows the model of decision-making from economy since the goal typically is to automate rational decision-making [45]. We summarise that model.

As [31] put forward, decision-making is taken to comprise of at least four steps:

1. identify the problem for which a decision needs to be made,
2. evaluate the objectives and preferences that apply,
3. analyze the decision problem and its constraints, and develop or identify the possible options from which to choose,
4. choose from the identified options following some reasoning.

To put this into perspective, consider an example. In step one, I as an agent recognise that I am hungry and that this is a problem. Next, I evaluate my objectives and preferences. So my objective here are to stop being hungry which means get something to eat. My preference would be to eat a hot meal but not cook and at the same time, to eat something soon. Thus my problem becomes finding a place to buy food from. I now (step 3) have to identify the constraints: how much I want to spend, how far I am willing to go and which establishments are open today. These constraints help me make a list of possible restaurants to choose from. In the last step I choose from among the alternatives using some type of reasoning, like for example the closest and cheapest Indian food restaurant.

Decisions are taken by an agent. In computer science artificial intelligence[1], the requirement of what constitutes an agent is very low. Anything that has the ability to perceive its environment through sensors and can act upon that environment through actuators is considered to be an agent [45]. For a computational agent the environment is essentially a data construct consisting of lists of information with "sensors" being various dynamic inputs that change, add or remove data available to the agent. The source of that input can be an actual device that measures aspects of the environment and converts it into data or it can be a human inputting the data directly. An actuator is a device that produces motion and changes a physical environment in that way. For a computational agent, the "actuators" are its ability to change its own environment, namely its ability to alter data that is available to not only themselves but also some other person or agent.

The difference between a computational agent and an embodied agent (such as a robot) is perhaps best illustrated by considering as an example a program that plays chess. While the chess playing program does not "see" a chess board, or physically move chess pieces, it still plays chess which is considered an activity that requires intelligence. The environment it works with is a digital representation of the chess board and the changes on it. What is important in playing a move is not that the chess piece is physically moved on the board, but to make the choice of which chess piece to move in order to win.

The problem identification, the evaluation of objectives, preferences and constraints that apply is part of situational awareness and for most computational agents this is supplied as data. Situational awareness is the perception of one's environment, the events in that environment with respect to time or space, internalisation and utilization of that perception as information, and the projection of the future state of the environment and its elements. The available options and their characteristics are often also made available with the agent expected to do the evaluation of the options towards finding the optimal choice with respect to the given objectives, preferences and constraints. While people are capable of reiterating steps 3 and 4 of the decision-making process by identifying missing information and procuring it, those same activities, as well as situational awareness in general, are a hard problem in AI [45].

What is the difference between decision-making and moral decision-making? The difference is in whose objectives, preferences and constraints we choose to apply. As we can see from the four step model, decision-making is a process that only considers the objectives, preferences and constraints of the agent that makes the decision. Moral decision making requires us to consider the objectives, preferences and constraints of others. For example, when making a list of options, the environmental impact of different food options can be considered. Although I strongly prefer meat to tofu, I can choose to rank the vegan options over the meat options because sourcing that food does not cause suffering to animals.

It is, of course, in general not clear how that information on the objectives, preferences and constraints of others is to be sourced, to which extent it should be considered etc. In moral philosophy, numerous different approaches to answering these questions for human decision-makers have been discussed and we will give a brief overview of some of the dominant ones in Section 4. In the next section, we will consider artificial moral agency: to which extent *can* a computational agent make moral decisions given that they do not have full agency to do so due to the lack of situational awareness and other constraints on information processing capabilities.

[1] Artificial Intelligence can be studying in other fields than computer science such as philosophy

Before we move on, although it is not directly relevant here, for completeness, it is important to mention the relationship between economy and moral philosophy. Rational decision-making is studied in economy, whereas what are good and bad decisions is studied in moral philosophy. One dominant perspective on moral decision making in economy is that of Sen [46]. In economy, a decision problem is represented with a set of available alternatives, and the agent's preference order over those alternatives. The decision-making process is then choosing the alternative that maximizes the expected utility for the agent. Sen [46, 47] attempts to model morality by assuming morality to be an ordering over the agent's preference of alternatives, namely an ordering over orderings. So a moral decision is then to choose which preference order over the available alternatives to follow.

Dietrich and List [21] also present a decision-theoretic consideration of moral decision making. They show how to represent a moral theory in terms of two parameters: "(i) a specification of which properties of the objects of moral choice matter in any given context, and (ii) a specification of how these properties matter." We discuss what moral theories are in Section 4.

3 Artificial Moral Agents

Autonomy is the ability of an agent to govern itself, which includes its ability to identify problems and make decisions to resolve them. Both [36] and [51], some of the pioneers of machine ethics, observe that the extent to which an artificial agent would be able to do moral decision-making depends on the extent of autonomy of the agent. We typically do not talk about the autonomy of artificial agents, but the level of autonomy of a system, which we then refer as an autonomous system. Before introducing the different types of artificial moral agents that [36] and [51] have proposed, we introduce the levels of autonomy of an autonomous system.

An autonomous system is a system, software or device alike, that is capable of some degree of operation without human control. Systems that do not have autonomy are divided into: controlled, supervised and automatic. Controlled systems are systems that have no autonomy and require continuous human control to operate. An example of a controlled system is the standard electrical iron for clothes: for the iron to be used, a person must hold plug it in, hold it and move it. Supervised systems are systems that are capable of some short periods of unsupervised activity, but require a human operator to both start and end that period of activity. An example of a supervised system is the standard washing machine. A person needs to load it and choose a program, and it operates without oversight executing the washing program, and the human operator needs to unloaded after the program is completed. Automatic systems are able to operate without any human supervision, but they can execute only a very limited range of activities in a fully controlled environment: all their choices are pre-programmed. An example of an automatic system is an elevator. It responds to a human input in a specified way, but it operates (moves up, down, stops, opens and closes doors) without the oversight of human operator.

Autonomous systems are systems that are able to operate without human oversight for long periods of time, they are able to process signals from the environment, use them to reason about their choices and actions, and be able to perform actions that have an effect on the environment. In autonomous systems we discern, according to Parasuraman et al. [40], ten levels of autonomy of decision and action selection (ranked here from high to low), listed as follows.

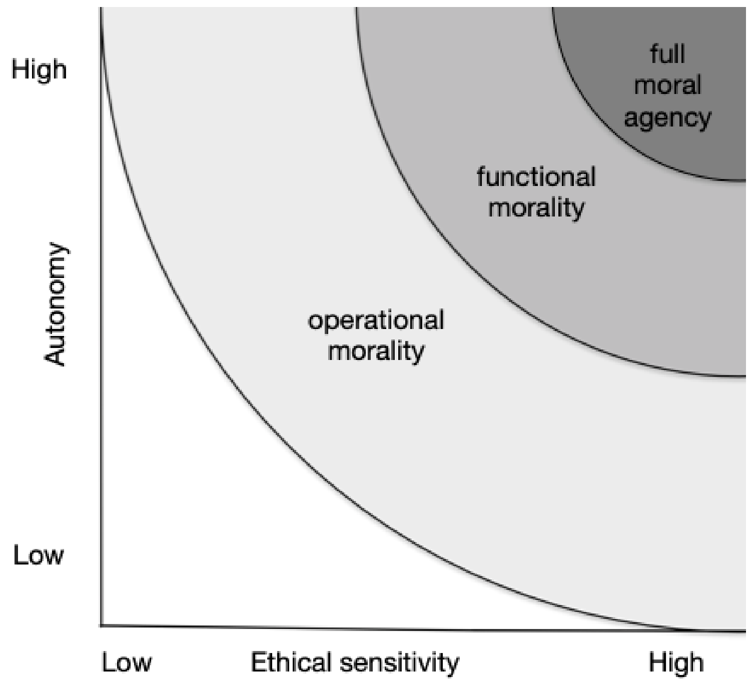


Figure 1 Degrees of artificial morality [51, Chapter2].

1. The system decides everything, acts autonomously and can ignore human input and control
2. Informs the human about its choices only if it, the system chooses to do so
3. Informs the human about its choices only if asked
4. Executes its decisions autonomously/automatically and then necessarily informs the human about the decision-making process
5. Allows the human a limited time to veto a decision made autonomously before execution, or
6. Executes the decision only if the human approves it, or
7. Suggest a decision (an alternative) to the human
8. Narrows the selection of options to choose from to a human, or
9. The system offers a complete set of decision/action alternatives, or
10. The system does not make any decision-making, the human must make all the decisions and actions.

Wallach and Allen [51] observe that the ability of computational agents to make moral decisions is restricted by their autonomy and by their *ethical sensitivity*[2]. They offer a graph of different types of artificial moral agency, which we reproduce in Figure 1.

[51] define operational morality as the morality of the artificial agents for which "the moral significance of their actions lies entirely in the humans involved in their design and use". This means that the artificial agent itself does not make moral decisions, but (in matters of morality) follows the instructions of a human operator. Functional morality is defined

[2] [51] do not explicitly define ethical sensitivity, but it is understood that it refers to the ability of the artificial agent to take into account the objectives, preferences and constraints of other when making decisions.

as the property of artificial agents who have the ability to make moral decisions without direct instructions from humans. It is understood that this ability is contextual, namely only applies under given circumstances. Full moral agency is the ability to have situational awareness and fully autonomously make moral decisions.

[36] also offers a four tier distinction between artificial moral agents, but does not base it explicitly on levels of autonomy or "ethical sensitivity" of the agent. Instead, he considers choice making artificial agents whose operation affects others in society in a positive or a negative way. For these agents he considers whether the agent uses moral choice relevant information at all, and if it does whether it sources it itself or it is in some way provided. [36] discerns: ethical impact agents, implicit ethical agents, explicit ethical agents and fully ethical agents.

Ethical impact agents are artificial agents which by virtue of existing bring about positive or negative impact on the lives of people. An ethical impact agent is an autonomous system or an AI system that is a disruptive technology, namely it changes society, and makes life better or worse for people in it by changing how certain tasks or operations are executed. An ethical impact agent does not itself make moral decisions at all or considers the objectives, preferences or constraints of others in its decision-making.

Implicit ethical agents implicitly do moral decision making. Namely, they are either fully constrained from choosing unethical options or the options they are considering are made available already evaluated with respect to how they affect the objectives, preferences and constraints of others. The evaluation of what is right and what is wrong thus is performed entirely by the human designers or operators of the artificial agent.

Explicit ethical agents do explicit moral decision making. This means that they are able to evaluate if an option is more or less ethical than another, possibly by also sourcing their own information for this evaluation. The degree to which they can perform moral decision making, would of course depend on the limited abilities of the artificial agent and might also be contextual. The evaluation of what is right and what is wrong is still by a large extent determined by the information supplied by the human designers or operators, but the artificial agent also contributes.

Fully ethical agents for Moor [36], just like for Wallach and Allen [51], are agents who have situational awareness and fully autonomously make moral decisions.

The definitions of Moor [36] are not meant to be operational. It is in general very difficult to evaluate the impact of an artificial agent, and also to draw a line to indicate which agents have and which do not have such impact. A mobile phone can be considered an ethical impact agent: it helps to solve crimes (a net positive impact) and it eases surveillance of people (a net negative impact[3]). To be able to ascertain whether an agent is an implicit or explicit moral agent one necessarily needs to have access to the agent's programming. [23] propose to refine the taxonomy of [36] towards making it more operational.

[23] propose that implicit can be considered those agents who engage in moral decision making without using their own autonomy. For ethically sensitive contexts, implicit ethical agents defer to the human operator, either directly or by accessing information made available. Explicit ethical agents do use the autonomy they have to make moral decision. Both explicit and implicit ethical agents thus have sufficient situational awareness to recognise that moral decision making is needed. However, implicit agents, do not use their autonomy, even if it is high. Their moral choices are either constraint or otherwise governed by a human operator. In contrast, explicit moral agents do use the autonomy they have to make moral choices.

[3] Personal opinion

One idea on how to develop implicit or explicit ethical agents is having their moral choices be governed by theories developed by moral philosophy. The alternative is that the artificial agents are informed directly by a person or societies views on what is right and wrong, see for example [39, 11, 44].

4 Moral philosophy

A detailed overview of moral philosophy is outside of the scope of this lecture. Perhaps choosing which moral philosophy works to present and which to ignore in a short list, itself is a moral choice. Instead we give a definition of what a moral theory is and what the main types of moral theories have been considered in moral philosophy. Each of the theories have strengths and weaknesses. Furthermore, every theory is developed to be applied by a human agent.

Moral philosophy is considered to include three main areas of study: meta-ethics, normative ethics and applied ethics [25]. Meta-ethics is concerned with the concepts of right and wrong themselves and how the validity of these concepts can be established. Normative ethics is concerned with developing means to identify what are the right and wrong decisions, actions, states of the world etc. Applied ethics is concerned with issuing recommendations of what is the right thing to do for a specific person in a specific situation. An example of applied ethics are the biomedical ethics rules that govern the conduct of, among others, medical doctors [12].

In machine ethics, we are primarily interested in normative moral philosophy and the moral theories developed within it. A moral theory is an explanation of what makes an action right, or what makes an entity good [50]. It is a reasoning system that can be used to establish the righteousness of an action or the goodness of an entity etc. The moral theories that are concerned with the discerning between good and bad entities are called theories of value, whereas those concerned with discerning between good and bad choices or actions are called theories of obligation.

Many moral theories have been proposed. Vaughn [50] argues that for a theory of reasoning about right and wrong to be usable, it needs to satisfy the following basic criteria. It needs to be consistent with considered judgments and our moral experience. Considered judgements are morally relevant preferences or decisions society has already made by carefully considering the complexity of a given problem and the intended and unintended consequences of alternative options. Our own moral experience vaguely describes what most people have been raised to intuitively consider right or wrong in most situations, such as for example stealing or betraying a confidence. Further, a moral theory should be useful in moral problem solving and it should be coherent. Usefulness means that anyone can apply it to make moral decisions, whereas coherence means that it should identify the same moral choices when presented with the same problem features.

When a decision is made, three aspects of the decision can be considered to be most relevant for identifying if that decision is good or bad. These are: the agent that makes the decision (their intentions, objectives and incentives included), the decision itself and available alternatives, and lastly the consequences of that decision.

Moral theories that put most relevance on the properties of the agent that makes the decisions are called *virtue theories*. The theories which place most relevance on the decision and alternatives are called *deontological theories*. *Consequentialist theories* deem that what ultimately decides whether an action is good or bad is the consequences of that action.

Virtue theories prescribe not how to make a decision but what intentions, objectives and preferences, i.e. virtues, the agent should have in order to choose right. The moral conduct of the agent emerges from their moral virtues. A virtue is a stable disposition to act and feel according to some ideal model of excellence. A notable virtue theory is Aristotelian ethics. Aristotle claimed that intellectual virtues can be taught but that moral virtues can be learned only through practice. He argued that for an agent to be virtuous they have to aim to achieve the golden mean which is finding a balance between two behavioural extremes. While at first glance it may seem that virtue theories are not particularly suited for developing artificial ethical agents, this is not necessarily the case. For example, [48] argues how virtue ethics can be taken from theory to implementation.

Deontological theories prescribe that the righteousness of a decision should be based on whether the chosen option is itself right or wrong under a series of rules, rather than on who is executing it [2]. Deontological theories typically prescribe obligations, permissions, prohibitions that the agent should follow when choosing between alternatives. Deontological theories also prescribe ethical values or ethical principles that one should follow in order to identify the right choices. In a very abstract way, they can be seen as providing heuristics for what are the objectives, preferences and constraints of others that the agent should taken into consideration during moral decision-making.

A notable deontological theory is Kantian ethics. Kant argued that reason alone leads us to the right and the good. According to him, moral law is a set of imperatives one should follow: hypothetical or categorical. A hypothetical imperative tells us what we should do if we have certain duties (obligations), while a a categorical imperative tells us what we should do regardless of our wants and needs. For example, [43] and [14] argue how Kantian ethics can be used to develop artificial moral agents. It should also be noticed that deontic logic has been developed to formalise reasoning about obligations and norms [26].

Consequentialist theories are perhaps the first thing that does come to mind when one considered moral theories. These theories prescribe that a decision is moral if it is motivated by assessing the consequences of the available options, namely what kind of states of affairs they bring about [2].

The most notable consequentialist group of theories is utilitarian ethics. Utilitarianism is the theory asserting that the morally right action is the one that produces the most favourable balance of good over evil, everyone considered [50]. Act-utilitarianism is the theory that the morally right actions are those that directly produce the greatest overall good, everyone considered. Given that it is not always practical to assess the consequences of each available alternative, sometimes rules can be developed to identify the actions that typically have desirable consequences. Rule-utilitarianism is the theory that the morally right action is the one covered by a rule that if generally followed would produce the most favourable balance of good over evil, everyone considered [50]. Since utilitarianism essentially prescribes quantifying the goodness of an action and reduces moral decision-making to maximising the utility of an option, the idea that this moral theory can be implemented in an artificial agent is not strange and has been considered early on, see for example Gips [27] and Anderson and Andrson[4].

5 Top down and bottom up automation

How should one go about developing an artificial moral agent? The first choice is to asses to which degree can we predict the moral decision-making problems that the artificial agent is expected to handle and what is the impact of the choices that the agent will make on the environment. This assessment will inform us towards whether we need to build an implicit or an explicit ethical agent. Beyond this choice, Wallach, Allen and Smit [52] argue that artificial moral agents can be built either by following a top-down or a bottom-up approach, not excluding hybrids of these two approaches as well. Top-down and bottom-up approaches are typical heuristics in problem solving deployed in engineering. Both top-down and bottom-up approaches can be executed for both implicit and explicit agents.

Following the top-down approach, a problem is iteratively broken down into smaller problems until we reach a problem small enough that we know how to solve. The solutions of those smaller problems combined constitute the solution of our original problem. Procedural programming operates following the top-down approach, where an algorithm breaks down the problem into a set of basic instructions that the computer can execute.

Following the bottom-up approach, we start by describing the features or criteria of the solution of the problem. Different actions are then pieced together, possibly in a trial-and-error fashion, until a solution is reached that satisfies the prescribed criteria. Declarative programming operates following a bottom-up approach where we describe the sought information or alternative by giving constraints and preferences that should apply to it.

Using the top-down to build an explicit ethical agent means answering the question: how can we build the artificial agent to follow a given moral theory? To build an implicit agent top-down means to answer the question: how can we follow a given moral theory to build an artificial agent? Both explicit and implicit agent construction requires facing the challenge of choosing a moral theory. Virtually all of the theories from moral philosophy can be difficult even for people to follow. For each of them there exist examples of situations, called moral dilemmas, in which the theory does not provide a satisfactory method to choose what to do. The approach then is not to attempt to fully implement a moral theory, or expect that the artificial agent will succeed in implementing it where humans have failed.

Building explicit top-down agents requires difficult AI problems to be solved, such as situational awareness, prediction of consequences of ones actions. So it is the state of the art in AI that is also a limitation to the abilities of artificial moral agents of this kind. Building implicit top-down agents is somewhat more attainable, but requires that the agent is given a pre-determine set of rules, built by a human operator following a moral theory, that they should apply when making moral decisions. This in turn, is only possible when the human operator can to a large degree predict the problems the artificial agents will face and the choices that would be available.

The advantage of the top-down approach is that the resulting agent, whether implicit or explicit, will follow a “tried and tested” theory. This makes it possible to test if the agent is making the correct choices according to some theory. Top-down approaches are also *verifiable* [19].

When building artificial moral agents, it is not sufficient to enable them to make moral decisions. We need to also be able to prove that we have done a good job. We need to be able to test whether the end behaviour we obtain is indeed ethical and correct with respect to some specification of correctness. For top-down agents testing and verification is made easier by knowing what their moral choices should be compared to. The top-down reasoning of the agent also allows itself to be formally verified [19, 18].

The motivation behind the bottom-up approach in building artificial agents draws from the observation that people typically make moral decisions without following a specific moral theory. We have a sense of right and wrong which we have developed over years of interactions with people, by observing how our decisions affect others and how we are affected by the decisions of others. The bottom-up approach aims to enable an artificial agent to learn little by little to discern right from wrong, emulating what people do.

To build a bottom up explicit ethical agent, one needs to figure out how to build an agent that learns to behave morally. In contrast, to build a bottom up implicit agent, one needs to figure out how to build an agent that given examples of right and wrong learns to identify moral choices correctly.

The advantages of the bottom-up approach are, clearly that one avoids the problem of choosing and implementing a specific moral theory. The approach is robust in the sense that it does not run the risk of running into a situation for which a moral decision cannot be made because the moral theory is under-specified. This robustness can also be a limitation of the bottom-up approach. As agents learn the moral behaviour they may end up learning something which we as humans do not recognise as moral. Examples are not hard to imagine, but we have already been able to witness some of them. Consider for example Tay[4]. Tay was a chatter-bot developed by Microsoft that was supposed to learn how to interact with people on Twitter, but was taken down after "learning" to post offensive tweets.

A limitation for explicit ethical bottom-up agent is also the dependence of its success on solving hard AI problems, just like the explicit ethical top-down agents. A specific limitation of the implicit ethical bottom-up agents is that they require reliable examples of right and wrong. This type of data is not readily available and it needs to be purposefully created. One challenge to creating this data set is that it would be an expensive undertaking. Another challenge is moral: who gets to supply the examples? How do we determine what are the representative examples of teachers of right and wrong for machines?

This shortage of examples has caused that deep learning, despite being the driver of much of the commercial success of recent AI [15], is virtually not used at all in machine ethics. Jiang et al [30] can perhaps be considered an exception. Jiang et al. use deep learning to build a question answering system that evaluates the morality of certain actions, however this system is not meant to be used by machines but by humans. Examples of bottom-up artificial agents either rely on symbolic learning, e.g., Anderson and Anderson [7] or on reinforcement learning [38, 9, 1]. The challenge of taking the reinforcement learning approach then becomes how to specify the objective function for the bottom-up artificial agent. This opens up again the problems of who gets to supply this information.

Lastly, compared to the top-down approach, the bottom-up approach does not lend itself as easy for testing and verification. What should the decisions of the artificial agent be compared to? Two moral agents might make two different choices in a moral decision making situation because they interpret the situation differently, and it is hard to claim that one choice is moral and the other is not.

The Ethical Turing test has been proposed [3] as a possible way to asses whether the artificial agent makes moral choices. This test would work as the original Turing test. An artificial moral agent would be given examples of decision problems and its answers will be compared to that of a moral philosopher or other expert in ethical decision making [7, 32]. Arguments have also been put forwards towards why the Ethical Turing test is not an adequate approach to testing if moral behavior in an artificial agent has been attained [10].

6 Beyond this tutorial

In this tutorial so far we had not discussed specific examples of artificial moral agents. This tutorial is not intended to be a systematic review of implemented machine ethics systems. Two such reviews exist and the reader can consult [49] and [37]. A very practical reason for

[4] `https://en.wikipedia.org/wiki/Tay_(bot)`

avoiding discussing implementations of artificial agents here is that these implementations vary vastly in the approaches they use and considerable background knowledge in various reasoning and learning methods would be necessary to understand the implementations. Throughout the text, however, numerous references are given to these specific systems and the interested reader can follow them and explore them for learning more.

It has to be mentioned that a considerable challenge to learn and conduct research in machine ethics is that research articles in machine ethics appear in a variety of AI venues, but also in volumes in engineering, decision theory, organisation theory and of course, philosophy.

References

1 David Abel, James MacGlashan, and Michael L Littman. Reinforcement learning as a framework for ethical decision making. In *Workshops at the thirtieth AAAI conference on artificial intelligence*, 2016.

2 Larry Alexander and Michael Moore. Deontological Ethics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2021 edition, 2021.

3 Colin Allen, Gary Varner, and Jason Zinser. Prolegomena to any future artificial moral agent. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(3):251–261, 2000. doi:10.1080/09528130050111428.

4 Michael Anderson and Susan Leigh Anderson. Machine ethics: Creating an ethical intelligent agent. *AI magazine*, 28(4), 2007.

5 Michael Anderson and Susan Leigh Anderson. The status of machine ethics: A report from the aaai symposium. *Minds Mach.*, 17(1):1–10, March 2007. doi:10.1007/s11023-007-9053-7.

6 Michael Anderson and Susan Leigh Anderson, editors. *Machine Ethics*. Cambridge University Press, 2011.

7 Michael Anderson and Susan Leigh Anderson. Geneth: A general ethical dilemma analyzer. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 253–261. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8308.

8 Susan Leigh Anderson. Asimov's "three laws of robotics" and machine metaethics. *AI Soc.*, 22(4):477–493, 2008. doi:10.1007/s00146-007-0094-5.

9 Stuart Armstrong. Motivated value selection for artificial agents. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

10 Thomas Arnold and Matthias Scheutz. Against the moral turing test: Accountable design and the moral reasoning of autonomous systems. *Ethics and Inf. Technol.*, 18(2):103–115, June 2016. doi:10.1007/s10676-016-9389-x.

11 Seth D. Baum. Social choice ethics in artificial intelligence. *AI Soc.*, 35(1):165–176, 2020. doi:10.1007/s00146-017-0760-1.

12 Tom L. Beauchamp and James F. Childress. *Principles of Biomedical Ethics*. Oxford University Press, USA, 2001.

13 Richard E. Bellman. *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, 1978.

14 Oliver Bendel, Kevin Schwegler, and Bradley Richards. Towards kant machines. In *2017 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 27-29, 2017*. AAAI Press, 2017. URL: http://aaai.org/ocs/index.php/SSS/SSS17/paper/view/15278.

15 Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. Deep learning for AI. *Communications of the ACM*, 64(7):58–65, June 2021. doi:10.1145/3448250.

16 Bartosz Brożek and Bartosz Janik. Can artificial intelligences be moral agents? *New Ideas in Psychology*, 54:101–106, 2019. doi:10.1016/j.newideapsych.2018.12.002.

17 Alexandra Chouldechova and Aaron Roth. A snapshot of the frontiers of fairness in machine learning. *Commun. ACM*, 63(5):82–89, 2020. doi:10.1145/3376898.

18 Louise A. Dennis, Martin Mose Bentzen, Felix Lindner, and Michael Fisher. Verifiable machine ethics in changing contexts. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 11470–11478. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17366.

19 Louise A. Dennis, Michael Fisher, Marija Slavkovik, and Matt Webster. Formal verification of ethical choices in autonomous systems. *Robotics Auton. Syst.*, 77:1–14, 2016. doi:10.1016/j.robot.2015.11.012.

20 Nicholas Diakopoulos. Transparency. In Markus D. Dubber, Frank Pasquale, and Sunit Das, editors, *The Oxford Handbook of Ethics of AI*. Oxford University Press, July 2020. doi:10.1093/oxfordhb/9780190067397.013.11.

21 Franz Dietrich and Christian List. What matters and how it matters: a choice-theoretic representation of moral theories. *Philosophical Review*, 126(4):421–479, 2017.

22 Virginia Dignum. *Responsible Artificial Intelligence - How to Develop and Use AI in a Responsible Way*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2019. doi:10.1007/978-3-030-30371-6.

23 Sjur Dyrkolbotn, Truls Pedersen, and Marija Slavkovik. On the distinction between implicit and explicit ethical agency. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '18, pages 74–80, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3278721.3278769.

24 Amitai Etzioni and Oren Etzioni. Incorporating ethics into artificial intelligence. *The Journal of Ethics*, 21:403–418, 2017.

25 James Fieser. Ethics. In Michael Boylan, editor, *Internet Encyclopedia of Philosophy*. ISSN 2161-0002, 2021.

26 Dov Gabbay, John Horty, and Xavier Parent. *Handbook of Deontic Logic and Normative System*. College Publications, UK, 2013.

27 James Gips. Toward the ethical robot. In Kenneth M. Ford, C. Glymour, and Patrick Hayes, editors, *Android Epistemology*, pages 243–252. MIT Press, USA, 1994.

28 David Gunning and David Aha. Darpa's explainable artificial intelligence (XAI) program. *AI Magazine*, 40(2):44–58, June 2019. doi:10.1609/aimag.v40i2.2850.

29 R. M. Hare. *Community and Communication*, pages 109–115. Macmillan Education UK, London, 1972. doi:10.1007/978-1-349-00955-8_9.

30 Liwei Jiang, Jena D. Hwang, Chandra Bhagavatula, Ronan Le Bras, Maxwell Forbes, Jon Borchardt, Jenny Liang, Oren Etzioni, Maarten Sap, and Yejin Choi. Delphi: Towards machine ethics and norms. *CoRR*, abs/2110.07574, 2021. arXiv:2110.07574.

31 Gregory E Kersten and Stan Szpakowicz. Decision making and decision aiding: defining the process, it representations, and support. *Group Decision and Negotiation*, 3(2):237–261, 1994.

32 Hyeongjoo Kim and Sunyong Byun. Designing and Applying a Moral Turing Test. *Advances in Science, Technology and Engineering Systems Journal*, 6(2):93–98, 2021. doi:10.25046/aj060212.

33 Bertram F. Malle, Paul Bello, and Matthias Scheutz. Requirements for an artificial agent with norm competence. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '19, pages 21–27, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3306618.3314252.

34 Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019. doi:10.1016/j.artint.2018.07.007.

35 James Moor. The dartmouth college artificial intelligence conference: The next fifty years. *AI Magazine*, 27(4):87, December 2006. doi:10.1609/aimag.v27i4.1911.

36 James H. Moor. The nature, importance, and difficulty of machine ethics. *IEEE Intelligent Systems*, 21(4):18–21, July 2006. doi:10.1109/MIS.2006.80.

37 Vivek Nallur. Landscape of machine implemented ethics. *Sci. Eng. Ethics*, 26(5):2381–2399, 2020. doi:10.1007/s11948-020-00236-y.

38 Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R. Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. Teaching ai agents ethical values using reinforcement learning and policy orchestration. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6377–6381. International Joint Conferences on Artificial Intelligence Organization, July 2019. `doi:10.24963/ijcai.2019/891`.

39 Ritesh Noothigattu, Snehalkumar (Neil) S. Gaikwad, Edmond Awad, Sohan Dsouza, Iyad Rahwan, Pradeep Ravikumar, and Ariel D. Procaccia. A voting-based system for ethical decision making. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1587–1594. AAAI Press, 2018. URL: `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17052`.

40 Raja Parasuraman, Tom .B. Sheridan, and Christopher D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 30(3):286–297, 2000. `doi:10.1109/3468.844354`.

41 Martin Peterson. *An Introduction to Decision Theory*. Cambridge Introductions to Philosophy. Cambridge University Press, 2 edition, 2017. `doi:10.1017/9781316585061`.

42 Rosalind W. Picard. *Affective Computing*. MIT Press, 1997.

43 Thomas M. Powers. Prospects for a kantian machine. In Michael Anderson and Susan LeighEditors Anderson, editors, *Machine Ethics*, pages 464–475. Cambridge University Press, 2011. `doi:10.1017/CBO9780511978036.031`.

44 Iyad Rahwan. Society-in-the-loop: programming the algorithmic social contract. *Ethics and Information Technology*, 20(1):5–14, March 2018. `doi:10.1007/s10676-017-9430-8`.

45 Steward Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 4 edition, 2020.

46 Amartya Sen. Choice, orderings and morality. In Stephan Körner, editor, *Practical Reason*, pages 54–67. Camalot Press, Oxford, 1974.

47 Amartya K. Sen. Rational fools: A critique of the behavioral foundations of economic theory. *Philosophy & Public Affairs*, 6(4):317–344, 1977. URL: `http://www.jstor.org/stable/2264946`.

48 Jakob Stenseke. Artificial virtuous agents: from theory to machine implementation. *AI & SOCIETY*, 2021. `doi:10.1007/s00146-021-01325-7`.

49 Suzanne Tolmeijer, Markus Kneer, Cristina Sarasua, Markus Christen, and Abraham Bernstein. Implementations in machine ethics: A survey. *CoRR*, abs/2001.07573, 2020. `arXiv:2001.07573`.

50 Lewis Vaughn. *Beginning Ethics: An Introduction to Moral Philosophy*. W. W. Norton & Company, New York City, 2014.

51 Wendell Wallach and Colin Allen. *Moral Machines: Teaching Robots Right from Wrong*. Oxford University Press, Inc., USA, 2008.

52 Wendell Wallach, Colin Allen, and Iva Smit. Machine morality: bottom-up and top-down approaches for modelling human moral faculties. *AI & SOCIETY*, 22(4):565–582, 2008. `doi:10.1007/s00146-007-0099-0`.

53 Maranke Wieringa. What to account for when accounting for algorithms: A systematic literature review on algorithmic accountability. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, FAT* '20, pages 1–18, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3351095.3372833`.

54 Alan F. T. Winfield, Serena Booth, Louise A. Dennis, Takashi Egawa, Helen Hastie, Naomi Jacobs, Roderick I. Muttram, Joanna I. Olszewska, Fahimeh Rajabiyazdi, Andreas Theodorou, Mark A. Underwood, Robert H. Wortham, and Eleanor Watson. Ieee p7001: A proposed standard on transparency. *Frontiers in Robotics and AI*, 8:225, 2021. `doi:10.3389/frobt.2021.665729`.