YAGO 4.5: A Large and Clean Knowledge Base with a Rich Taxonomy

Fabian M. Suchanek
Telecom Paris
Institut Polytechnique de Paris
Palaiseau, France
fabian.suchanek@telecom-paris.fr

Mehwish Alam
Telecom Paris
Institut Polytechnique de Paris
Palaiseau, France
mehwish.alam@telecom-paris.fr

Thomas Bonald
Telecom Paris
Institut Polytechnique de Paris
Palaiseau, France
thomas.bonald@telecom-paris.fr

Lihu Chen
INRIA Saclay, France
Palaiseau, France
lihu.chen@inria.fr

Pierre-Henri Paris
Telecom Paris
Institut Polytechnique de Paris
Palaiseau, France
pierre-henri.paris@telecom-paris.fr

Jules Soria
Telecom Paris
Institut Polytechnique de Paris
Palaiseau, France
jules.soria@alumni.ip-paris.fr

ABSTRACT

Knowledge Bases (KBs) find applications in many knowledge-intensive tasks and, most notably, in information retrieval. Wikidata is one of the largest public general-purpose KBs. Yet, its collaborative nature has led to a convoluted schema and taxonomy. The YAGO 4 KB cleaned up the taxonomy by incorporating the ontology of Schema.org, resulting in a cleaner structure amenable to automated reasoning. However, it also cut away large parts of the Wikidata taxonomy, which is essential for information retrieval. In this paper, we extend YAGO 4 with a large part of the Wikidata taxonomy – while respecting logical constraints and the distinction between classes and instances. This yields YAGO 4.5, a new, logically consistent version of YAGO that adds a rich layer of informative classes. An intrinsic and an extrinsic evaluation show the value of the new resource.

CCS CONCEPTS

• **Information systems → World Wide Web**.

KEYWORDS

Knowledge Bases; Knowledge Graphs; Taxonomies; YAGO

ACM Reference Format:
Fabian M. Suchanek, Mehwish Alam, Thomas Bonald, Lihu Chen, Pierre-Henri Paris, and Jules Soria. 2024. YAGO 4.5: A Large and Clean Knowledge Base with a Rich Taxonomy. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24), July 14–18, 2024, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3626772.3657876

1 INTRODUCTION

A Knowledge Base (KB), also called Knowledge Graph, is a directed labeled multi-graph, where the nodes are entities (such as the United States of America, Eleanor Roosevelt, or the UN Declaration of Human Rights), and the edges are relations between these entities (such as which person is a citizen of which country, or which person contributed to which artifact) [73]. Similar entities are grouped into classes (such as the class of countries, the class of people, or the class of artifacts), and these classes form a taxonomy, where more general classes (such as living beings) subsume more special classes (such as humans). KBs find applications in question answering, natural language processing [76], and knowledge injection into language models [37, 54]. They are used in particular also in information retrieval, e.g., to enhance the understanding of queries and documents [64], to expand queries [17, 58], to summarize documents [5], to facilitate semantic search [15, 20], or for entity retrieval [14, 30, 62]. Major industry players such as Google, Apple, Microsoft, and Meta all build and use KBs [52]. There are also numerous public KBs, including both domain-specific ones and general-purpose ones.

Wikidata. One of the largest general-purpose KBs nowadays is Wikidata [75]. It provides a wealth of facts about nearly every domain of common human discourse, with more than 100 million entities and around 1.4 billion facts about them. Each entity has an abstract identifier (such as *Q83396*), which makes the identifiers language-independent and persistent in time. Tens of thousands of people contribute to the project. At the same time, being a collaborative KB, Wikidata suffers from a lack of agreement on the schema level: there are several classes that are difficult to distinguish for the uninitiated user (e.g., *geographical location* (Q2221906), *location* (Q115095765), *geographic region* (Q82794), *physical location* (Q17334923), and *geographical area* (Q3622002)); there are more than ten thousand relations; constraints are defined but not enforced (*Grotesco* (Q10509019) is a subclass of Q49094906, which is not a class); classes and instances are mixed (*scientist* (Q901), e.g., is both a subclass of *person* (Q215627) and an instance of *profession* (Q28640)); there are more than 2.7M classes of which only 3% are instantiated (1M subclasses of *chemical entity* (Q43460564) have no instance); and the taxonomy contains cycles (there are 47 pairs of classes that are subclasses of each other, e.g., *method* (Q1799072) and *technique* (Q2695280), and 15 cycles of length 3 or more, e.g., *axiom* (Q17736), *first principle* (Q536351), *principle* (Q211364)). Finally, the abstract identifiers for Wikidata properties and entities make downstream applications more difficult.

YAGO 4. The YAGO KB has been in existence since 2008 [9, 27, 39, 72]. Its fourth version [74] was designed to address the shortcomings of Wikidata: It combines the data about instances from Wikidata with the taxonomy and properties from Schema.org – an ontology developed by a W3C Community Group[1]. Filtering and constraint enforcement made YAGO 4 a KB that allows for automated reasoning. However, this merger came at the expense of abandoning nearly the entire class taxonomy of Wikidata. That is a disadvantage because classes can express facts that are very hard to model correctly by RDF properties [57] – like saying that something is a "train ferry route", a "financial regulatory agency", or a "de facto consulate". As a consequence, one of the major criticisms that users advanced was that the class hierarchy of YAGO 4 was too sparse.

Contributions. In this paper, we show how this shortcoming of YAGO 4 can be resolved, while still maintaining the logical consistency and semantic coherence of YAGO. We carefully incorporate selected parts of the Wikidata taxonomy into the taxonomy of Schema.org. This leads to considerable challenges. Numerous organically grown branches of the Wikidata taxonomy have to be disentangled. Furthermore, many classes in Wikidata are both instances and classes and pose a challenge for modeling. The transformation also poses engineering challenges: Wikidata comprises more than 120 GB, even compressed, which must be parsed and processed. We will describe how we surmounted these challenges, and what open problems still remain. The resulting resource, which we call YAGO 4.5, contains 132M facts and is logically consistent.

Our paper is structured as follows: Section 2 recalls related work, Section 3 discusses design decisions, Section 4 discusses implementation issues, and Section 5 presents the resulting KB, before Section 6 concludes.

2 RELATED WORK

General-Purpose Knowledge Bases. The Semantic Web comprises hundreds of KBs[2]. Many of these are tailored for specific domains or applications, such as the Gene Ontology [3] for biological processes, functions, and cellular components. However, in this paper, we are concerned with general-purpose KBs that do not focus on a specific domain. In the following, we describe several prominent general-purpose KBs. **ConceptNet** [71] is a semantic network that primarily deals with common sense knowledge. This makes it an orthogonal project to YAGO concerned with facts about instances concerning nearly all human knowledge. **BabelNet** [50] is a large multilingual encyclopedic dictionary derived from WordNet, Wikipedia, Wikidata, and several other sources. BabelNet focuses on relations between concepts and words and has neither a taxonomy nor a schema. **DBpedia** [4] is a large-scale, multilingual KB derived from Wikipedia (and, more recently, Wikidata). Unlike YAGO, it lacks information about the temporal validity of the facts. Moreover, the automated generation from the Wikipedia infoboxes and the priority for recall over precision has led DBpedia to be not fully consistent [1, 21, 67]. The manually curated part of DBpedia contains just 4M instances[3]. **Freebase** [10] was an extensive KB consisting of metadata compiled from various sources. The project was discontinued in 2015, and its content was transferred [59] to Wikidata. **Wikidata** [75], a project of the Wikimedia Foundation, is a collaboratively edited KB that supports other Wikimedia projects like Wikipedia and Wikimedia Commons. It is by far the largest open KB. However, due to its collaborative nature, its taxonomy has grown convoluted and complicated, with inconsistencies in hierarchies and data models as well as rule violations – making it hard to use even by contributors [11, 60, 68].

In this landscape, **YAGO** positions itself as a large general KB for facts about instances, with a taxonomy, manually defined properties, and logical constraints. Its key property is that it is a centrally controlled data source, which allows it to establish certain guarantees for the quality of its data [9, 27, 39, 72, 74]. The latest version, YAGO 4 [74], was designed to be clean enough to perform automated reasoning on it. However, its taxonomy is very parsimonious, which is the challenge that we address in the present paper.

Upper Ontologies. Top ontologies, also known as upper or foundational ontologies, provide a domain-independent framework for organizing knowledge across various fields. We discuss some of the most prominent projects below and refer the reader to Mascardi et al. [42] for a comprehensive comparison. **Cyc** [34] is one of the oldest and most comprehensive upper ontologies, developed by Cycorp, aiming to represent general human knowledge and common sense reasoning. **SUMO** [46] is an open-source upper ontology with a formal structure for organizing and integrating domain-specific ontologies. It consists of a core set of general concepts and relations and domain-specific extensions that cover various fields, such as biology, finance, and geography. **DOLCE** [43] is another top ontology which focuses on capturing the ontological categories underlying natural language and human cognition. **BFO** [2] is an upper ontology that was created based on the ontologies related to the domain of geospatial information. **WordNet** [48] contains lexical and semantic relationships between sets of synonymous words (synsets). WordNet does not define properties, and the project was discontinued in 2012.

Schema.org [25] differs from the above in that it is a collaborative project initiated by major companies like Google, Microsoft, and Yahoo to provide a shared vocabulary for annotating Web content with structured data. While not a top ontology in the traditional sense, Schema.org plays a crucial role in the Semantic Web ecosystem by promoting standardized vocabularies for describing entities and their properties, thus facilitating data interoperability and integration. It is broadly adopted (with more than 12 million websites in 2016) and benefits from strong industry support, making it a highly reliable and sustainable choice for building a KB [47].

Taxonomy Induction and Expansion. Many recent studies automate taxonomy induction and expansion, including Online Catalog Taxonomy EnrichmenT (OCTET) [41], TaxoCom [33], TaxoOrder[70], TaxoExpan [65], HiExpan [66], TaxoEnrich [31],

[1]https://www.w3.org/community/schemaorg/
[2]http://cas.lod-cloud.net/
[3]https://www.dbpedia.org/resources/ontology/

and taxonomy induction from a set of terms [40]. Our own previous YAGO versions automatically mapped WordNet synsets to Wikipedia categories [72]. In contrast to these automated approaches, YAGO 4 and the new YAGO 4.5 use a manual mapping. This is because there are only a few dozen classes to be mapped.

Wikidata Efforts. There is a community effort to map Wikidata properties and classes to Schema.org[4]. These mappings use *exact match (P2888)* (64 mappings), *equivalent class (P1709)* (332 mappings), *equivalent property (P1628)* (84 mappings), *external subproperty (P2236)* (22 mappings), and *external superproperty (P2235)* (6 mappings). However, the effort was discontinued in 2017. These mappings inspired the mappings of YAGO 4, which in turn were the basis for the mappings that we use in this paper.

Beyond that, the Wikidata community has conducted a survey on ontology issues that its contributors face[5]. It lists in particular a "Messy upper-level ontology", a "Mix-up of meta levels", "Exchanged sub-/superclasses", "Redundant classification", "Cycles" (in the taxonomy), and "Unclassified items". However, as a community effort, Wikidata has to count on the collaboration of its contributors to solve these issues. This is a lengthy and incremental process that depends on individual commitment and consensus. The discussion dryly notes, for example, that the "messy upper-level ontology" is "not fully solvable without some dictator to decide and enforce it". This dictator, of course, cannot and should not exist in a community-driven effort such as Wikidata. In YAGO, in contrast, decisions can be taken and enforced effectively, as the team of contributors is much smaller. Indeed, YAGO has always had the strategy of ingesting instance data from large resources in a bottom-up fashion, but enforcing a top-level taxonomy, relations, and constraints in a top-down fashion. This is also the strategy that we use in this paper to extract a clean subset of the Wikidata taxonomy for YAGO.

3 DESIGNING YAGO

3.1 Design Rationale

Our goal is to have a clean upper taxonomy for YAGO, which is precise and non-redundant to allow for automated reasoning. Our choice falls on Schema.org, for reasons we have elaborated in Section 2: the taxonomy is concise, maintained by a W3C consortium, and finds applications well beyond its original purpose of annotating Web pages. It has the right level of detail for our purposes and does not digress into philosophical concepts. It defines not just classes, but also relations.

One could argue that the upper-level taxonomy is sufficient and that one should not aim to add more fine-grained classes – least of all the Wikidata taxonomy: it contains overly specific classes with few instances, some of its classes are not useful for large KB applications (such as *multi-organism process* (Q22269697) for elections), and sibling classes could be further grouped into common superclasses (e.g., Wikidata is missing a class that regroups human-made places, making do instead with *human-made geographic feature* (Q811430), *human-made geographic feature* (Q811463), and *human-made geographic object* (Q35145743)). All of this, one could argue, makes it an ill-suited candidate for a taxonomy.

[4]https://github.com/okfn-brasil/schemaOrg-Wikidata-Map
[5]https://www.wikidata.org/wiki/Wikidata_talk:Ontology_issues_prioritization#Overview_of_potential_solutions

However, these issues fade when a clean upper-level taxonomy is put on top: less useful lower-level classes (such as Q22269697) disappear because they are not subclasses of the clean upper-level classes. The missing grouping, likewise, can be achieved by the upper-level taxonomy – for example for places. Finally, even if a Wikidata class contains few instances, it can still carry meaningful information. For example, it is informative for the human user to know that an entity is a "General aviation monoplane with 1 tractor-piston-propeller engine" (Q33110974). It was precisely that lack of such classes that users deplored for YAGO 4. Such classes do not carry logically formalized meanings. (It would be cumbersome to formally express that something is a "General aviation monoplane with 1 tractor-piston-propeller engine" by RDF statements [56, 57].) Rather, the purpose of these lower-level classes is to convey informal information to the human user. The Wikidata classes clearly serve this purpose.

3.2 Design Principles

Our goal is to integrate the upper taxonomy from Schema.org with the lower taxonomy from Wikidata. The following design principles drive our integration:

1. Prefer properties over class membership. Some information can be expressed either by a property (*hasNationality UnitedStates*) or by a class membership (*type American*). When designing the schema, we give preference to properties and choose classes only if the class appears in the domain or range of a property. In our example, *American* does not appear as a range or domain of any property (its superclass *Human* does). Hence, the class *American* should not exist, and the nationality should be expressed by a property. The reason for avoiding classes when possible is that OWL DL (the reasoning formalism we target) does not allow expressing properties about classes (i.e., we cannot attach properties to the class *American*, while we can for the instance *UnitedStates*). This choice is also consistent with the way Wikidata and YAGO model properties.

2. Choose the property with fewer objects. When we have the choice between a property and its inverse property (e.g., *hasCitizenship* and *hasCitizen*), we choose the one that has, on average, fewer objects per subject (i.e., *hasCitizenship*). The reason for this choice is that it allows seeing the properties as attributes "about" the subject. For example, the Wikipedia page of Eleanor Roosevelt lists her US-American nationality, because the nationality is perceived as a property "of a person". At the same time, the Wikipedia page about the United States does not list all people of American nationality, because a citizen is not perceived as a property "of a country". Our criterion formalizes this intuition. It is indeed *de facto* used by all major KBs [23].

3. The upper taxonomy exists to define formal properties that will be populated. All classes of the upper taxonomy shall define formal properties (e.g., an *Airline* has an *iataCode*, which justifies its existence as an upper-level class). Both the domain and the range of these properties have to be upper-level classes. The reason for this design choice is that it makes the upper taxonomy a self-contained schema. Schema reasoning can then be restricted to this upper-level taxonomy without the need to search for property definitions in the lower classes.

4. The lower taxonomy exists to convey human-intelligible information about its instances in a non-redundant form. While the upper-level taxonomy contains the schema, the lower-level taxonomy targets mainly human users. Our design principle thus tells us to remove classes that add no information over upper taxonomy classes, to eliminate links in the taxonomy that are redundant due to transitivity, to merge classes that are hard to distinguish for the uninitiated user, and to remove classes that are not populated.

3.3 Upper Taxonomy

We now discuss how we construct the upper-level taxonomy of YAGO 4.5.

Upper-level classes. As for YAGO 4 [74], we start with the taxonomy of Schema.org. It defines one top-level class *schema:Thing* with 11 subclasses. We exclude the class *Action*[6], which models mainly Web user actions. We exclude *BioChemEntity* and *MedicalEntity* because these are domain-specific concepts. This leaves us with 8 top-level classes (*CreativeWork, Event, Organization, Taxon, Person, Place, Product, Intangible*), which we accept as subclasses of *Thing*. All top-level classes are declared disjoint (except places/organizations, and products/creative works).

Fictional entities. To deal with fictional entities, we add a class *yago:FictionalEntity* as a subclass of *schema:Thing*. This class defines properties such as *yago:createdBy* and *yago:appearsIn*, thus justifying its existence as a class under Design Principle 1. Fictional entities are not disjoint with any other class, as anything can also exist in fiction. A fictional entity is an instance of both *yago:FictionalEntity* and the class it belongs to in fiction. For example, a fictional human is an instance of both *yago:FictionalEntity* and *schema:Person*. This has the disadvantage that fictional humans will be counted as humans in count queries. However, it has the advantage that one can easily reason on fictional humans, as they will share all the properties that we declared for *schema:Person*. Wikidata goes a different way, by recreating the entire class hierarchy with its properties also for fictional beings, mapping each class of fictional entities to its real-world class counterpart. This choice can for sure be defended, but for YAGO, it would have severely convoluted the schema: it would have required duplication of all class and property specifications. The current modelization already has an advantage over the modelization in previous versions of YAGO (which simply merged real and fictional entities), as well as over other top-level ontologies such as DOLCE, BFO, and SUMO (which do not model fictional entities at all).

Intangibles. For our new YAGO, we added the following classes that are not in Schema.org, but that are necessary to define the ranges of properties (under Design Principle 3 above): *yago:Award*, *yago:Gender* (which differs from *schema:GenderType* in that it allows more than two values), and *yago:BeliefSystem* (for religious adherence). All are subclasses of *schema:Intangible*. The other subclasses of *Intangible* that Schema.org defines are mostly Web-specific (e.g., *ActionAccessSpecification*). Since these classes would not have instances, let alone populated properties from Wikidata, we removed them under Design Principle 3.

[6]For ease of reading, we omit prefixes where these can be inferred.

Schema.org has a subclass *Occupation* of *Intangible*, and models occupations by the property *hasOccupation*. However, if we model occupations by a property, (1) we lose the class hierarchy of occupations (*Physicist subClassOf Scientist* etc.), and (2) we lose the ability to add properties to specific professions (such as the *doctoral advisor* for scientists). Hence, by Design Principle 1 above, we model professions rather as subclasses of *Person*.

Places. When it comes to places, the taxonomy of Schema.org is heavily oriented towards the annotation of Web pages, with subclasses such as *Accommodation*, *Residence*, *LocalBusiness*, etc. These classes do not define properties that we could populate from Wikidata, and therefore, we remove them under Design Principle 3. We then manually created a taxonomy of subclasses of *schema:Place*, which distinguishes *schema:Landform* (areas with a boundary given by nature), *schema:AdministrativeArea* (boundary given by human administration) and the newly created *yago:HumanMadePlace* (boundaries given by human physical construction) and *yago:AstronomicalObject* (with boundaries in space). For the former, we add a subclass *yago:Way*, which regroups all ways of transit (roads, canals, railway lines, etc.).

General considerations. Under Design Principle 3, we keep only those classes from Schema.org that add new properties (plus their super-classes, all the way up to *schema:Thing*). This results in 41 upper classes. As in YAGO 4, all of the above is expressed as SHACL constraints[7] on the classes of Schema.org. Thus, there is no special syntax, code, or formalism for these declarations, and they are all part of the YAGO KB as normal facts.

3.4 Lower Taxonomy

Mapping to Wikidata. The lower levels of the YAGO 4.5 taxonomy come from Wikidata. As in YAGO 4, each class in the YAGO upper taxonomy is manually mapped to one or more classes in Wikidata. This happens, likewise, in a fully declarative way with a simple RDF statement that links the Schema.org-class by a special predicate to the Wikidata class(es). The mapping can happen at any level of the upper taxonomy: general classes such as *Organization* are mapped to Wikidata, and more special classes such as *Corporation* are mapped as well. A mapping can give rise to the following constellations:

One-to-one mapping. One upper class is mapped to one Wikidata class. All of the subclasses of the Wikidata class are glued under the upper class, but its super-classes are not imported.

One-to-many mapping. One upper class is mapped to several Wikidata classes. Again, all subclasses of these are glued under the upper class. This has the effect of merging the Wikidata classes. We do this, e.g., for classes that are equivalent for our purposes (such as *geographical region* (Q82794) and *geographical area* (Q3622002)).

One-to-none mapping. One upper class is mapped to no Wikidata class – only its subclasses are mapped to Wikidata. This has the effect that there cannot be direct instances of the upper class. Nor can there be subclasses other than the ones we declared. This is a new mechanism that did not exist in YAGO 4. We use it for classes with a convoluted taxonomy in Wikidata.

[7]https://www.w3.org/TR/shacl/

We use a one-to-none-mapping for the following classes:

schema:Thing. In YAGO 4, this class was mapped to the top-level class *entity* (Q35120) in Wikidata, which resulted in more than 1 million direct instances of schema:Thing in Yago. This defies Design Principle 3, because *schema:Thing* defines only very few properties. Hence, we now accept only entities that fall into one of the manually approved subclasses of *Thing*. This results in a clean top-level taxonomy, discarding meta-classes (e.g., *class or metaclass of Wikidata ontology* (Q21522864)), overlapping classes (e.g., *geographic entity* (Q27096213) and *location* (Q115095765)) and too specific classes (there are 6 top classes in Wikidata with less than 40 instances each, e.g., *converter* (Q35825432)).

schema:Place. The taxonomy of Wikidata for places is highly convoluted, with classes that are difficult to distinguish such as *terrain* (Q14524493), *geographical location* (Q2221906), *geographical region* (Q82794), *geographical area* (Q3622002), and *location* (Q115095765). Hence, we do not map *schema:Place*, and accept only instances of its manually designed subclasses. This discards 2,861 classes (from 29,826, i.e., less than 1%) and 137k instances (from 19M, i.e., less than 1%).

schema:Intangible. Intangible classes in Wikidata, likewise, are highly convoluted, with classes such as *class* (Q5127848), *process* (Q3249551 and Q67518233) or *role* (Q4897819). Hence, here, too, we would have difficulties establishing properties to comply with Design Principle 3. Therefore, we apply a one-to-none mapping and accept only instances and subclasses of the manually declared subclasses of *Intangible*.

Importing the subclasses. Once the mapping has been defined manually, the subclasses of Wikidata can be imported automatically. To this end, we consider the subclass graph of Wikidata, which we construct as follows: every entity of Wikidata that has a *subClassOf* relationship becomes a node in this graph. It is linked to its superclasses by the *subClassOf* relationship. We then iterate through all upper classes of YAGO, and whenever we hit a class that is mapped to Wikidata, we glue the entire sub-DAG of that Wikidata class under the YAGO top-level class. This approach differs from the approach in YAGO 4, where we mapped only classes with a Wikipedia article. However, this limited the taxonomy to just a handful of classes per instance, which proved to be too few.

Several caveats are to be respected in this merging process to respect Design Principle 4: as in YAGO 4, we ensure that we do not add any link that would create a loop in the taxonomy (57 loops were removed). We do not add transitive links (40k such links were removed). We do not add a link if that would make a class a transitive subclass of two top-level classes that we declared disjoint (9k links were removed). We also remove, from the sub-DAG that we import, all Wikidata classes that have their own mapping to YAGO upper classes.

Excluded classes. We exclude housekeeping classes of Wikidata that we blacklist manually: classes of Wikimedia pages, disambiguation pages, lists, and the like. For our new YAGO, we also exclude linguistic objects (such as characters (Q3241972), phrases (Q187931), numbers (Q11563), etc.), many of which are technically infinite and would otherwise make up 700k entities in YAGO. We also remove abstract objects such as actions and occurrents, as these have rather philosophical subclasses that are of limited use for our purposes (e.g., *Multi-organism process* (Q22269697), etc.). The same fate is bestowed on scholarly articles. The addition of all obtainable scholarly articles to Wikidata was controversial[8], and in YAGO, they would make up 39M of entities, almost half of all entities. Hence, we decided to remove them. Finally, under Design Principle 4, we also remove all classes that do not have instances (1.3M).

3.5 Instances

Identifiers. As in YAGO 4, every instance is automatically equipped with a readable name. We use the title of the corresponding Wikipedia page as an entity identifier (as in *yago:Eleanor_Roosevelt*). If there is none, or if the same Wikipedia page is used by more than one entity, we use the English label of the entity and concatenate it with the Wikidata Q-id to avoid ambiguity (as in *yago:Brazilian_jiu_jitsu_competition_Q105086361*). If there is none, we use a label that contains legal Turtle characters, concatenated with the Wikidata id (which was not done in YAGO 4). If there is no such label, we use the Wikidata id (with a YAGO-prefix for uniformity). The Turtle standard[9] allows percentage codes in local names, but many parsers (e.g., the one of Hermit[10]) cannot deal with them. Hence, we replace all characters that are not letters or numbers by their hexadecimal Unicode, so that two identifiers that differ only in their inadmissible characters are still distinct.

Instances vs. Classes. Wikidata contains several items that are both instances and classes. For example, *English* (Q1860) is an instance of *Natural Language* (Q33742), as well as a subclass of *Anglic* (Q1346342). That makes sense because there can be several subclasses of *English*, such as e.g., *American English* (Q7976). As per our discussion in Section 3.4, *English* thus becomes a class. At the same time, we would also like to say that Eleanor Roosevelt spoke English, i.e., we would like to make a statement about a class. The dominant reasoning language, OWL 2, allows such statements by a mechanism called *punning*[11]. However, OWL 2 punning works essentially by allowing the same identifier (*English*) to denote two distinct elements (a class and an instance). This means that neither standard description logic reasoners nor the query language SPARQL can deal with expressions where a variable is bound to an identifier that is simultaneously a class and an instance [35]. Thus, it would be impossible to say that Eleanor Roosevelt spoke a subclass of Anglic. (This problem did not appear in YAGO 4, where intermediate classes were eliminated aggressively, and survived only as instances.) We solve this problem as follows: whenever we encounter a Wikidata fact that would link an instance to a class, we create a generic instance of the class and use it as an object. In our example, we create a fact saying that Eleanor Roosevelt spoke *yago:English_language_generic_instance*. The intuition is that Roosevelt spoke something that is an instance of English. This mechanism is in line with Approach 2 in [53], and kicks in for awards, belief systems, academic titles, and languages.

This technique works well for the objects of statements, which generally have an existential interpretation (Eleanor Roosevelt did

[8]https://www.mail-archive.com/wikidata@lists.wikimedia.org/msg06716.html
[9]https://www.w3.org/TR/turtle/
[10]http://www.hermit-reasoner.com/
[11]https://www.w3.org/TR/owl2-new-features/#F12:_Punning

not speak all dialects of English, but there is one that she spoke). It works less well for subjects of statements. For example, commercial products are classes in Wikidata (and rightly so, as different people can own different instances of the same product). However, if we take commercial products as classes, we cannot attach their manufacturer, date of inception, awards, etc. to them. It would be semantically wrong to attach these to a generic instance of the product, as they apply to the line of products itself. It would also be semantically wrong to create an axiom that says that all instances of the class have that property (not every single iPhone has won an award). Hence, we make every item that is a product (as identified by the *manufacturer* (P176) relation in Wikidata) an instance.

3.6 Properties and Constraints

Properties and constraints work largely as in YAGO 4: for *schema:Thing* and each sub-class from Schema.org, we manually define the properties that Schema.org provides. By Design Principle 3, we keep only those properties of Schema.org that are (1) of general interest (removing specialized properties such as *hasDriveThroughService*) and (2) existent in Wikidata (because otherwise they would not be populated). For the new YAGO, we added a few properties, most notably for the new top-level classes *yago:Award* and *yago:FictionalEntity*, and also for countries (Schema.org does not contain the relation *hasCapital*). Each of the 100 most frequent relations in Wikidata is either mapped to YAGO or excluded on purpose (e.g., because of redundancy). Our design document[12] lists the most frequent Wikipedia properties, together with their mappings to YAGO 4.5.

An instance can have only those properties that are declared for its class or superclasses. We manually add SHACL constraints [32] for maximum cardinality (31 constraints in total), patterns of literals (e.g., for ISBNs; 9 in total), and domain and range constraints (for each relation). Each relation is mapped manually and declaratively to a relation in Wikidata, and populated from there. A fact is accepted only if both its subject and its object conform to the domain and range constraints (this removes roughly 6% of facts in our dataset). We take only the facts that Wikidata labels as "truthy" (which exclude disputed statements). We extract time stamps for facts from Wikidata, and attach them to the YAGO facts in the RDF-star model [26].

The entire taxonomy of upper classes, the definition of properties, the accompanying SHACL constraints, and the mapping to Wikidata classes take the form of a single Turtle file. It forms an integral part of YAGO and can be downloaded along with the rest of the KB. All of this works exactly as in YAGO 4, and we refer the interested reader to the corresponding publication [74].

Unlike YAGO 4, the new YAGO simplifies numerical quantities: while these were previously values with a range and a unit, they are now simple literals. Our design document[12] lists, explains and justifies the design decisions for each class and relation in detail.

4 IMPLEMENTING YAGO

Parsing, analyzing, and transforming a KB of the size of Wikidata (766 GB as of April 2023) is no easy feat. We describe here the challenges we encountered when creating YAGO 4.5, and how we surmounted them, in the hope that this will be useful for other users of Wikidata and YAGO.

Infrastructure. The code of YAGO 4 was written in Rust. While this ensured high performance and compile-time flagging of code problems, it also complicated the maintenance of the project , and we have, therefore, rewritten the code from scratch in Python. The original YAGO 4 loaded the data into a RocksDB key-value store. This had the advantage that many costly operations (such as constraint checks) could be run directly on the data store. At the same time, loading the entire data into the data store and indexing it could easily take a day. The new system, therefore, stores all data (intermediate and final) in files on the hard drive, which has the advantage that intermediate results can be inspected and re-used.

Data formats. Wikidata exists in the "full" version and the "truthy" version[13]. While the truthy version is much smaller, we need the full version to extract time stamps for facts (a feature YAGO has had since Version 2 [27]). One has the choice between the NT format (easier to parse) and the Turtle file (smaller by a factor of 2), and our choice falls on the latter. The file can be downloaded as BZ2 and GZIP, and we strongly recommend the GZIP version. First, the unpacking is much faster (in the order of hours instead of the order of days in our case). Second, BZ2 allows no way of seeing the compressed file size without unpacking it. Finally, GZIP files can be processed sequentially by Python without unpacking them. For parsing Wikidata, we experimented with RDFlib. However, the library failed for certain characters in URIs, which caused an unrecoverable abortion in the middle of the parsing. Furthermore, the generation of URIs (expanding the prefixes) causes a large overhead. Therefore, we wrote our own Turtle parser and graph database, which, for our limited application, turned out to be rather simple (500 lines of code). These design choices mean that our code does not use any external libraries.

While our initial input (Wikidata and schema.org) is in Turtle, we chose TSV as our intermediate file format, because it allows for much faster parsing. In addition, we can attach more information to each fact (time stamp, source, etc.) in the form of supplementary columns.

Data processing. Our system proceeds in 6 sequential steps. Each step reads the output files of the previous step, and produces new output files, as in [9]. Each of these steps can be run on its own, and each of the steps has its own set of test input files with gold-standard output files, which we can use to check if the step works as expected.

Two of our steps need to process the entire Wikidata file, which is done in parallel. We experimented with Python multithreading, only to find that, due to the Global Interpreter Lock, it does not fully utilize multiple processor cores for CPU-bound tasks. The correct construction is multiprocessing, which uses one processor per process. Since processes cannot efficiently share data, each process has to load a copy of the data that it needs. Each process writes out its results to its own temporary file, which we then merge with the others. Each process i of the n processes starts at position $(i-1)/n \times N$ of the Wikidata file (where N is the size of the file). From that position on, the process scrolls forward to the next item

[12] https://yago-knowledge.org/data/yago4.5/design-document.pdf

[13] https://www.wikidata.org/wiki/Wikidata:Database_download

declaration, where it starts its work. It proceeds until it hits the item declaration that follows position $i/n \times N$. Since the file is UTF-8 encoded, the initial position may hit the middle of a character that is encoded by more than one byte. This is not a problem because the UTF-8 standard can distinguish the middle-bytes from the initial bytes in an encoded stream.

Steps. We generate YAGO on a Unix machine with 90 CPUs and 800GB of RAM. We proceed in 6 steps:

1. **Create schema:** The manual definition of the schema is loaded, and the relevant parts of the Schema.org-taxonomy are extracted, as described in Section 3.3. This process operates only on manually defined files and hence terminates on our machine in less than a second.
2. **Create taxonomy:** Wikidata is parsed for classes, and a loop-free taxonomy is constructed, as described in Section 3.4. This step is parallelized and takes 4 hours.
3. **Create facts:** Wikidata is parsed for facts, each predicate is mapped to a YAGO predicate as described in Section 3.6, the subject of the fact is type-checked, and the objects are type-checked if they are literals. The objects that are not literals cannot yet be type-checked because we do not yet have a complete list of all instances at this stage. This step, likewise, is parallelized and takes 4 hours.
4. **Type-check facts:** The previous step has given us a list of facts, which also contains the class that each instance belongs to. We load this list into memory and run through all facts to type-check the object of each fact. This step runs on YAGO data, and not on Wikidata, and thus does not need parallelization. It takes 1:30h to run.
5. **Create ids:** Among those facts that survived the type check, we map each entity to its legible YAGO name, as described in Section 3.6. This takes one hour.
6. **Create statistics:** Debugging and testing are an integral part of the development. The last step counts the number of instances per class and of facts per predicate. It creates a visualization of the taxonomy and a random selection of entities for manual check. This process also takes one hour.

Thus, the overall process takes about 12 hours.

5 RESULT

5.1 Resource

Size. Table 1 shows the statistics of YAGO 4.5[14] and puts them in perspective with Wikidata and YAGO 4. Both versions of YAGO have vastly less predicates than Wikidata. As explained previously, this is deliberate: according to Design Principle 3 (Section 3.2), YAGO accepts only those predicates that are defined in its schema. That said, YAGO 4.5 does cover most of the 100 most frequent predicates of Wikidata (see Section 3.6). When we now turn to compare the two versions of YAGO, we see that YAGO 4.5 has fewer properties than YAGO 4. This is due to the removal of inverse properties, which we removed under Design Principle 2 (e.g., we removed *hasParent* because we already have *hasChild*; 6 such cases); properties related to scholarly articles (*citation* etc., 4 in total, which we removed according to the discussion in Section 3.4); biochemical properties (11 in total, removed because of a lack of expertise in our team); and properties of numerical literals (*value* etc., 6 in total, removed because literals became simple values in YAGO 4.5). The loss is thus deliberate. A detailed comparison of the properties in YAGO 4, YAGO 4.5, and Wikidata is in our design document[11]. The same goes for the facts of YAGO 4: 238M facts describe scholarly articles (mainly citations, pagination, and publication dates), 13M facts describe literals and 2.5M facts are redundant because of an inverse property. If these are discarded, the new YAGO contains slightly more facts. The dump size is still smaller because we use the Turtle file format instead of NT. Concerning the classes, the picture is as rosy as intended: YAGO 4.5 has vastly more classes.

	Wikidata	YAGO 4	YAGO 4.5
Entities	103M	67M (37M)	49M
of which generic	0	0	62k
Classes	2.8M	10k	133k
Predicates	11k	140 (124)	108
Facts	500M	343M (89M)	132M
Type facts	106M	70M (33M)	53M
Label facts	795M	303M	479M
Meta facts	12M	2.5M	7M
Dump size	766GB	280GB	142GB

Table 1: Size of YAGO 4.5. Facts exclude *type*, *label*, *comment*, *alternateName*, *sameAs*, and *mainEntityOfPage* facts. In brackets: without redundant properties, properties describing literals, and properties describing scholarly articles.

Data Format. The final file format of YAGO is Turtle. We separate the subject, predicate, object, and dot by a tabulator, so that our files are *de facto* also TSV files. YAGO is split into the following files:

- **Schema**: upper taxonomy, property definitions, and SHACL constraints.
- **Taxonomy**: the entire taxonomy of YAGO (all *subClassOf* facts).
- **Facts**: all facts about entities that have an English Wikipedia page.
- **Beyond Wikipedia**: all facts about other entities.
- **Meta**: all temporal annotations (in the RDF-star file format [26]).

Downstream applications can load only the files they need, and exclude, e.g., meta facts or facts about entities that do not have an English Wikipedia page.

5.2 Evaluation

Intrinsic evaluation. To evaluate the quality of the new KB, we draw inspiration from the criteria for ontology evaluation that a recent survey identified [77]: **consistency** refers to the absence of logical contradictions. For this purpose, we verified the logical consistency of YAGO using the OWL API[15] and Pellet [69], an OWL DL reasoner for Java, in 4 hours. Additionally, we validated the

[14]Version yago-4.5.0.2

[15]https://owlcs.github.io/owlapi/

Criterion	Operationalization	Wikidata	YAGO 4	YAGO 4.5
Consistency	Absence of contradictions	no	**yes**	**yes**
Complexity	Top-level classes	41	2714	**9**
	Number of paths to root	44	**1.1**	2.3
Modularity	Disjointness axioms	0	18	**24**
Conciseness	Taxonomic loops	62	0	**0**
	Redundant taxonomic links	377k	1216	**0**
	Redundant relations	118	6	**0**
	Classes without instances	2.6M	73	**0**
Understandability	Human-readable names	0%	89%	**91%**
Coverage	Classes per instance	**8.4**	3.6	7.8
	Facts per instance	4.8	**5.1**	2.7

Table 2: Quality measures, inspired by [77]

SHACL shapes with Jena SHACL[16] in 1h30. **Complexity** refers to the extent to which the ontology is complicated. We propose to measure it (1) by the number of top-level classes (i.e., the number of direct subclasses of *Thing*; the fewer the better), and (2) by the average number of paths from an instance to the root in the taxonomic tree (the fewer the better). **Modularity** is the degree to which the ontology is composed of discrete subsets. In our case, these subsets are the disjoint classes, and we report the number of (non-redundant and actively enforced) class disjointness statements as an indicator. **Conciseness** requires the absence of redundancies, and we count the number of taxonomic loops, taxonomic links that are redundant because of transitivity, and relations whose inverse also exists. **Understandability** is the degree to which the ontology can be comprehended. This is difficult to operationalize, but we can at least report the percentage of identifiers that have human-readable names. **Coverage** refers to the degree to which the ontology covers the domain knowledge. We report the number of classes and facts per instance (excluding labels etc.).

Table 2 shows these measures for YAGO 4, YAGO 4.5, and Wikidata. The latter has vastly more facts per instance than YAGO. This is to be expected, as YAGO is a clean subset of Wikidata. YAGO 4, too, has more facts per instance than YAGO 4.5. However, this is mainly due to the 174M facts about citations for 40M scholarly articles. YAGO 4 also has a lower number of paths to the root, which is due to its sparse taxonomy. On all other measures, YAGO 4.5 scores much better than both YAGO 4 and Wikidata: it has a clean upper-level taxonomy of just nine top-level classes – instead of the dozens of Wikidata or the thousands that YAGO 4 attached to the taxonomic root for lack of good intermediate classes. YAGO 4.5 is also free of all types of redundancy that plagued Wikidata and YAGO 4. On average, there are just 2.6 paths to the root, as opposed to 44 in Wikidata (not accounting for cycles). At the same time, YAGO 4.5 nearly replicates the taxonomic richness of Wikidata, with 7.8 classes per entity – twice as many as in YAGO 4.

Extrinsic evaluation. To show the value of the new YAGO 4.5 over YAGO 4, we applied it to the task of entity disambiguation (also called entity linking). Given a KB and a text that contains an entity mention, the task is to link it to the corresponding entity in the KB.

[16]https://jena.apache.org/documentation/shacl/

	[1, 5)	**[5, 10)**	**[10, 20)**	**[20, ∞)**	**Macro**
samples	13971	3452	1374	203	19000
YAGO 4	77%	59%	53%	20%	52%
YAGO 4.5	**80%**	**64%**	**58%**	**31%**	**58%**

Table 3: Disambiguation accuracy by candidate mention.

This task is important for analyzing large document sets such as the Panama Papers[17] or newspaper archives [19], and it is useful also in information retrieval [44, 80]. Entity disambiguation is particularly difficult with ambiguous entity names. For example, in "I love the city of light, Paris", the word "Paris" could refer to the capital of France, but also to several dozen cities of that name in the US, not to mention people of that name such as Paris Hilton or the Greek hero Paris. We use the BLINK [79] dataset, which is based on a 2019 English Wikipedia dump. We collect 19 thousand samples from this dataset in such a manner that (1) every mention appears only once (so that there is less bias towards popular entities), and (2) all entities exist in both YAGO 4 and YAGO 4.5. For each entity mention, we collect the possible candidate entities by finding all entities that share a label with the mention. We use the end-to-end entity linking system ExtEnD [6], pre-trained on Longformer [8]. It takes as input a single piece of text, which consists of the input text concatenated with the separator [SEP] and a sequence of entity candidates. Each entity candidate is given in textual form. For our experiments, we use the main label of the entity, followed by the classes of which the entity is an instance. In our example, we produce "I love the city of light, *Paris* [SEP] Paris [capital, city, place]; Paris Hilton [singer, actress, human]; ...". Then, the system indicates the start and end token of the span containing the predicted entity candidate.

Table 3 shows the accuracy of the disambiguation, grouped by the number of candidate entities per mention. The disambiguation works much better on YAGO 4.5 – especially on mentions that have many candidates. This shows the usefulness of the new YAGO taxonomy. All code and data of this study are publicly available[18] for reproducibility.

[17]https://medium.com/@ambiverse

[18]https://github.com/tigerchen52/eval_yago_el/

5.3 Applications

Benchmarking. Previous versions of YAGO have been widely used as benchmark datasets in entity type prediction [29] and link prediction [24]. YAGO43KET [49] is a benchmark dataset for entity type prediction, which has been created from the subset of YAGO 3.0. It contains 43k entities with their type information. YAGO3-10 is a benchmark for link prediction that consists of all entities of YAGO 3.0 having at least 10 predicates [18]; it consists of 120k entities (mainly people) and 37 predicates. YAGO11K [16] is also a subset of Yago 3.0, and it is used to evaluate algorithms for temporal link prediction [36, 78] and temporal relation prediction [13].

YAGO in Information Retrieval. YAGO is helpful for Information Retrieval (IR) systems in order to enhance their understanding of queries and documents beyond the scope of word tokens and plain texts [64]. YAGO can help IR mainly in two applications: Document Retrieval and Entity Retrieval. *Document retrieval* is the task of finding relevant textual resources for a given user query. Several works leverage entity information from YAGO to expand queries, incorporating rich features into the retrieval process [17, 58]. Another approach uses YAGO ontology to construct document summarization tools, aiding in the efficient retrieval of key information from extensive document collections [5]. Facts in YAGO can also be used to represent both queries and documents, facilitating semantic search in information retrieval [15, 20]. *Entity Retrieval* aims to retrieve and rank entities in a document collection (or a knowledge base). Several benchmarks for entity retrieval are derived from YAGO [12, 28, 51]. Existing studies rely on information in YAGO to retrieve various entities, including geographic entities [62], Point of Interest (POI) entities [14], and temporal scopes of entities [30].

Other Applications. The YAGO KB has found quite a number of other applications in the past [63]. Together, the YAGO publications have been cited more than 10,000 times in total so far, according to Google Scholar. As every user is free to download the KB and to use it (or not), we do not have an overview of the projects that use YAGO. We just know that the KB (in various versions) has been downloaded about 6000 times during the past year (heuristically excluding bots). YAGO 4.5 can be plugged in as-is into any system that uses YAGO 4 (as is currently happening with Qlever [7]). We expect our YAGO 4.5 to be even more useful than YAGO 4, as it provides fine-grained information about instances by the new classes. This will be useful to any application that draws on classes: entity retrieval, semantic annotation of documents, graphical exploration of instances, similarity computations of entities, of documents, or of queries and documents, and, as shown, entity disambiguation.

Availability. YAGO 4.5 is available on the YAGO Web site[19]. It contains download links for the data, the Design Document that details the mapping of schema.org to Wikidata, the documentation of the data format, the list of publications, and the names of the contributors. The Web page also offers an interactive browser for the KB (loaded with a subset of the entities). A SPARQL endpoint is provided[20]. The URIs of YAGO 4.5 are dereferenceable. YAGO 4.5 is available with Creative Commons Attribution-ShareAlike License (as imposed by the license of schema.org). The source code of YAGO 4.5 is available via GitHub[21] under a Creative Commons Attribution license.

[19]https://yago-knowledge.org
[20]https://yago-knowledge.org/sparql

6 CONCLUSION

In this paper, we have presented a method to merge the Wikidata taxonomy with the taxonomy of Schema.org, giving rise to a new version of the YAGO knowledge base, YAGO 4.5. Our work adds a rich layer of informative classes to YAGO, while at the same time keeping YAGO logically consistent. Going beyond YAGO, we have introduced, discussed, and justified several design principles that can be of use for other KB projects as well, most notably concerning the choice between properties vs classes, of properties vs inverse properties, and the modeling of fictional entities. We have also described implementation experiences that can help other researchers who work on Wikidata.

Several open challenges remain: first, we did not include the classes of *BioChemEntities* and *MedicalEntities*. These can be added in the future as a domain-specific extension. Now that the general framework is in place, more upper classes and properties can be defined. An interesting avenue of research in this direction would be the (automated) translation of textual class descriptions (such as *de facto embassy* (Q5244910)) into logical properties and constraints [38]. This would require a better understanding of commonsense statements [57]. Another challenge is the maintenance: while the code can be rerun on newer versions of Wikidata, the manual mapping of classes might have to be adapted if the upper taxonomy of Wikidata changes – as in previous versions of YAGO. Constraints, too, are currently defined manually. They could be mined automatically instead [22, 45, 55, 61]. The automated maintenance of structured data and its schema is an interesting challenge for the research community as a whole.

ACKNOWLEDGMENTS

This work was partially funded by the French National Agency for Research under the Grant "NoRDF" ANR-20-CHIA-0012-01 (https://nordf.telecom-paris.fr/en/).

REFERENCES

[1] Abián, D., Guerra, F., Martínez-Romanos, J., Trillo-Lado, R.: Wikidata and dbpedia: a comparative study. In: IKC (2018)
[2] Arp, R., Smith, B., Spear, A.D.: Building ontologies with basic formal ontology. Mit Press (2015)
[3] Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., et al.: Gene ontology: tool for the unification of biology. Nature genetics (2000)
[4] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: Dbpedia: A nucleus for a web of open data. In: ISWC (2007)
[5] Baralis, E., Cagliero, L., Jabeen, S., Fiori, A., Shah, S.: Multi-document summarization based on the yago ontology. Expert Systems with Applications **40**(17) (2013)
[6] Barba, E., Procopio, L., Navigli, R.: ExtEnD: Extractive entity disambiguation. In: ACL (2022)
[7] Bast, H., Buchhold, B.: Qlever: A query engine for efficient sparql+ text search. In: CIKM (2017)
[8] Beltagy, I., Peters, M.E., Cohan, A.: Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150 (2020)
[9] Biega, J.A., Kuzey, E., Suchanek, F.M.: Inside YAGO2s: A Transparent Information Extraction Architecture . In: WWW demo track (2013)
[10] Bollacker, K.D., Evans, C., Paritosh, P.K., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD (2008)

[21]https://github.com/yago-naga/yago-4.5

[11] Brasileiro, F., Almeida, J.P.A., Carvalho, V.A., Guizzardi, G.: Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata. In: WWW (2016)
[12] Chen, L., Varoquaux, G., Suchanek, F.M.: Learning high-quality and general-purpose phrase representations. arXiv preprint arXiv:2401.10407 (2024)
[13] Chen, Z., Xu, C., Su, F., Huang, Z., Dou, Y.: Incorporating structured sentences with time-enhanced BERT for fully-inductive temporal relation prediction. In: SIGIR (2023)
[14] Cinar, E.R., Altingovde, I.S.: Exploiting cluster-skipping inverted index for semantic place retrieval. In: Chen, H., Duh, W.E., Huang, H., Kato, M.P., Mothe, J., Poblete, B. (eds.) SIGIR (2023)
[15] Corcoglioniti, F., Dragoni, M., Rospocher, M., Aprosio, A.P.: Knowledge extraction for information retrieval. In: ESWC (2016)
[16] Dasgupta, S.S., Ray, S.N., Talukdar, P.: Hyte: Hyperplane-based temporally aware knowledge graph embedding. In: EMNLP (2018)
[17] Demidova, E., Zhou, X., Nejdl, W.: Efficient query construction for large scale data. In: Jones, G.J.F., Sheridan, P., Kelly, D., de Rijke, M., Sakai, T. (eds.) SIGIR (2013)
[18] Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2d knowledge graph embeddings. In: AAAI (2018)
[19] Economist, T.: Who the economist has written about over the past 175 years (2022)
[20] Ercan, G., Elbassuoni, S., Hose, K.: Retrieving textual evidence for knowledge graph facts. In: ESWC (2019)
[21] Färber, M., Bartscherer, F., Menne, C., Rettinger, A.: Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. Semantic Web (2018)
[22] Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases . In: WWW (2013)
[23] Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast Rule Mining in Ontological Knowledge Bases with AMIE+ . In: VLDBJ (2015)
[24] Gesese, G.A., Biswas, R., Alam, M., Sack, H.: A survey on knowledge graph embeddings with literals: Which model links better literal-ly? Semantic Web **12**(4) (2021)
[25] Guha, R.V., Brickley, D., Macbeth, S.: Schema.org: evolution of structured data on the web. Commun. ACM (2016)
[26] Hartig, O.: Rdf* and sparql*: An alternative approach to annotate statements in RDF. In: ISWC poster track (2017)
[27] Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia . In: Artificial Intelligence (2013)
[28] Hoffart, J., Yosef, M.A., Bordino, I., Fürstenau, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., Weikum, G.: Robust disambiguation of named entities in text. In: EMNLP (2011)
[29] Hu, Z., Gutiérrez-Basulto, V., Xiang, Z., Li, R., Pan, J.Z.: Multi-view contrastive learning for entity typing over knowledge graphs. In: EMNLP (2023)
[30] Jatowt, A., Kawai, D., Tanaka, K.: Timestamping entities using contextual information. In: SIGIR (2017)
[31] Jiang, M., Song, X., Zhang, J., Han, J.: Taxoenrich: Self-supervised taxonomy completion via structure-semantic representations. In: WWW (2022)
[32] Knublauch, H., Kontokostas, D.: Shapes constraint language (shacl) (2017), https://www.w3.org/TR/shacl/, w3C Candidate Recommendation 11(8)
[33] Lee, D., Shen, J., Kang, S., Yoon, S., Han, J., Yu, H.: Taxocom: Topic taxonomy completion with hierarchical discovery of novel topic clusters. In: WWW (2022)
[34] Lenat, D.B., Guha, R.V.: Representation and inference in the cyc project (1990)
[35] Lenzerini, M., Lepore, L., Poggi, A.: Metamodeling and metaquerying in owl2ql. Artificial Intelligence **292** (2021)
[36] Li, Z., Jin, X., Li, W., Guan, S., Guo, J., Shen, H., Wang, Y., Cheng, X.: Temporal knowledge graph reasoning based on evolutional representation learning. In: SIGIR (2021)
[37] Liu, Q., Yogatama, D., Blunsom, P.: Relational memory-augmented language models. TACL (2022)
[38] Lu, X., Liu, J., Gu, Z., Tong, H., Xie, C., Huang, J., Xiao, Y., Wang, W.: Parsing natural language into propositional and first-order logic with dual reinforcement learning. In: Coling (2022)
[39] Mahdisoltani, F., Biega, J.A., Suchanek, F.M.: YAGO3: A Knowledge Base from Multilingual Wikipedias . In: CIDR (2015)
[40] Mao, Y., Ren, X., Shen, J., Gu, X., Han, J.: End-to-end reinforcement learning for automatic taxonomy induction. In: ACL (2018)
[41] Mao, Y., Zhao, T., Kan, A., Zhang, C., Dong, X.L., Faloutsos, C., Han, J.: Octet: Online catalog taxonomy enrichment with self-supervision. In: SIGKDD (2020)
[42] Mascardi, V., Cordì, V., Rosso, P.: A Comparison of Upper Ontologies. In: AI/TABOO workshop (2007)
[43] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L.: The wonderweb library of foundational ontologies. WonderWeb deliverable D (2002)
[44] Meij, E., Balog, K., Odijk, D.: Entity linking and retrieval. In: SIGIR (2013)
[45] Meilicke, C., Chekol, M.W., Ruffinelli, D., Stuckenschmidt, H.: An introduction to anyburl. In: KI (2019)
[46] de Melo, G., Suchanek, F.M., Pease, A.: An Application: Extending SUMO with Wikipedia . In: Adam Pease: Ontology: A Practical Guide. (2011)
[47] Mika, P.: On schema. org and why it matters for the web. IEEE Internet Computing (2015)
[48] Miller, G.A.: WordNet: An electronic lexical database. MIT press (1998)
[49] Moon, C., Harenberg, S., Slankas, J., Samatova, N.: Learning contextual embeddings for knowledge graph completion. In: PACIS (2017)
[50] Navigli, R., Ponzetto, S.P.: Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artif. Intell. (2012)
[51] Noullet, K., Mix, R., Färber, M.: Kore 50dywc: An evaluation data set for entity linking based on dbpedia, yago, wikidata, and crunchbase. In: LREC (2020)
[52] Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J.: Industry-scale knowledge graphs: Lessons and challenges: Five diverse technology companies show how it's done. Queue (2019)
[53] Noy, N., Uschold, M., Welty, C.: Representing classes as property values on the semantic web (2005), https://www.w3.org/TR/2005/NOTE-swbp-classes-as-values-20050405/
[54] OpenAI: Chatgpt plugins. https://openai.com/blog/chatgpt-plugins (2023)
[55] Ortona, S., Meduri, V.V., Papotti, P.: Robust discovery of positive and negative rules in knowledge bases. In: ICDE (2018)
[56] Paris, P.H., Aoud, S.E., Suchanek, F.M.: The Vagueness of Vagueness in Noun Phrases. In: AKBC (2021)
[57] Paris, P.H., Suchanek, F.M.: Non-named entities - the silent majority. In: ESWC short paper track (2021)
[58] Pasca, M., Alfonseca, E.: Web-derived resources for web information retrieval: from conceptual hierarchies to attribute hierarchies. In: SIGIR (2009)
[59] Pellissier Tanon, T., Vrandečić, D., Schaffert, S., Steiner, T., Pintscher, L.: From freebase to wikidata: The great migration. In: WWW (2016)
[60] Piscopo, A., Simperl, E.: Who Models the World?: Collaborative Ontology Creation and User Roles in Wikidata. ACM Hum. Comput. Interact. (2018)
[61] Rabbani, K., Lissandrini, M., Hose, K.: Extraction of validating shapes from very large knowledge graphs. VLDB J. **16**(5) (2023)
[62] Rae, A., Murdock, V., Popescu, A., Bouchard, H.: Mining the web for points of interest. In: SIGIR (2012)
[63] Rebele, T., Suchanek, F.M., Hoffart, J., Biega, J.A., Kuzey, E., Weikum, G.: YAGO: a multilingual knowledge base from Wikipedia, Wordnet, and Geonames . In: ISWC (2016)
[64] Reinanda, R., Meij, E., de Rijke, M., et al.: Knowledge graphs: An information retrieval perspective. Foundations and Trends in Information Retrieval **14**(4) (2020)
[65] Shen, J., Shen, Z., Xiong, C., Wang, C., Wang, K., Han, J.: Taxoexpan: Self-supervised taxonomy expansion with position-enhanced graph neural network. In: WWW (2020)
[66] Shen, J., Wu, Z., Lei, D., Zhang, C., Ren, X., Vanni, M.T., Sadler, B.M., Han, J.: Hiexpan: Task-guided taxonomy construction by hierarchical tree expansion. In: SIGKDD (2018)
[67] Sheng, Z., Wang, X., Shi, H., Feng, Z.: Checking and handling inconsistency of dbpedia. In: WISM (2012)
[68] Shenoy, K., Ilievski, F., Garijo, D., Schwabe, D., Szekely, P.A.: A study of the quality of Wikidata. J. Web Semant. (2022)
[69] Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Journal of Web Semantics **5**(2) (2007)
[70] Song, X., Shen, J., Zhang, J., Han, J.: Who should go first? a self-supervised concept sorting model for improving taxonomy expansion. arXiv preprint arXiv:2104.03682 (2021)
[71] Speer, R., Chin, J., Havasi, C.: Conceptnet 5.5: An open multilingual graph of general knowledge. In: AAAI (2017)
[72] Suchanek, F.M., Kasneci, G., Weikum, G.: Yago - A Core of Semantic Knowledge . In: WWW (2007)
[73] Suchanek, F.M., Lajus, J., Boschin, A., Weikum, G.: Knowledge Representation and Rule Mining in Entity-Centric Knowledge Bases . In: RW invited paper (2019)
[74] Tanon, T.P., Weikum, G., Suchanek, F.M.: YAGO 4: A Reason-able Knowledge Base . In: ESWC (2020)
[75] Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM (2014)
[76] Weikum, G., Dong, L., Razniewski, S., Suchanek, F.M.: Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. Foundations and Trends in Databases (2021)
[77] Wilson, R., Indika, W., Ginige, A.: Analysis of ontology quality dimensions, criteria and metrics. In: ICCSA (2021)
[78] Wu, J., Xu, Y., Zhang, Y., Ma, C., Coates, M., Cheung, J.C.K.: TIE: A framework for embedding-based incremental temporal knowledge graph completion. In: SIGIR (2021)
[79] Wu, L., Petroni, F., Josifoski, M., Riedel, S., Zettlemoyer, L.: Scalable zero-shot entity linking with dense entity retrieval. In: EMNLP (2020)
[80] Zwicklbauer, S., Seifert, C., Granitzer, M.: Robust and collective entity disambiguation through semantic embeddings. In: SIGIR (2016)