Wikidata subsetting: approaches, tools, and evaluation

Seyed Amir Hosseini Beghaeiraveri [a,*,**], Jose Emilio Labra Gayo [*, b], Andra Waagmeester [*, c], Ammar Ammar [d], Carolina Gonzalez [e], Denise Slenter [d], Sabah Ul-Hasan [f], Egon Willighagen [d], Fiona McNeill [g] and Alasdair J G Gray [a]

[a] *School of Mathematical and Computer Science, Heriot-Watt University, Edinburgh, UK*
E-mail: sh200@hw.ac.uk
E-mail: A.J.G.Gray@hw.ac.uk
[b] *University of Oviedo, Oviedo, Spain*
E-mail: labra@uniovi.es
[c] *Micelio*
E-mail: andra@micel.io
[d] *Dept of Bioinformatics - BiGCaT, NUTRIM, Maastricht University*
E-mails: a.ammar@maastrichtuniversity.nl, denise.slenter@maastrichtuniversity.nl, egon.willighagen@maastrichtuniversity.nl
[e] *The Scripps Research Institute*
E-mail: agonzalez@scripps.edu
[f] *Hologic Inc*
E-mail: bysabahulhasan@gmail.com
[g] *School of Informatics, The University of Edinburgh UK*
E-mail: f.j.mcneill@ed.ac.uk

Abstract. Wikidata is a massive Knowledge Graph (KG) including more than 100 million data items and nearly 1.5 billion statements, covering a wide range of topics such as geography, history, scholarly articles, and life science data. The large volume of Wikidata is difficult to handle for research purposes; many researchers cannot afford the costs of hosting 100 GB of data. While Wikidata provides a public SPARQL endpoint, it can only be used for short-running queries. Often, researchers only require a limited range of data from Wikidata focusing on a particular topic for their use case. Subsetting is the process of defining and extracting the required data range from the KG; this process has received increasing attention in recent years. Specific tools and several approaches have been developed for subsetting, which have not been evaluated yet. In this paper we survey the available subsetting approaches, introducing their general strengths and weaknesses, and evaluate four practical tools specific for Wikidata subsetting – WDSub, KGTK, WDumper, and WDF – in terms of execution performance, extraction accuracy, and flexibility in defining the subsets. The results show that all four tools have a minimum of 99.96% accuracy in extracting defined items and 99.25% in extracting statements. The fastest tool in extraction is WDF, while the most flexible tool is WDSub. During the experiments, multiple subset use cases have been defined and the extracted subsets have been analyzed, obtaining valuable information about the variety and quality of Wikidata, which would otherwise not be possible through the public Wikidata SPARQL endpoint.

Keywords: Knowledge Graph, Wikidata, Subsetting, Big Data, Accuracy, Performance

[*] These authors contributed equally to this work and share first authorship
[**] Corresponding author. E-mail: sh200@hw.ac.uk.

1. Introduction

Wikidata [1] is a collaborative and open knowledge graph founded by the Wikimedia Foundation on 29 October 2012. The initial purpose of Wikidata is to provide reliable structured data to feed other Wikimedia projects such as Wikipedia. Wikidata contains 101,449,901 data items and more than 1.4 billion statements, as of 19 January 2023. Wikidata and its RDF and JSON dumps are licensed under Creative Commons Zero v1.0 [1], making it publicly available for all commercial or non-commercial use cases. It can be queried directly over a free SPARQL endpoint[2], a free query service GUI[3] and is interlinked with the other Linked Open Data on the web [1].

Wikidata is a key player in Linked Open Data and provides a massive amount of linked information about items in a wide range of topics. The topical coverage of Wikidata spans from scientific research and historical events to cultural heritage and everyday facts. With its ability to integrate data from multiple sources, Wikidata serves as a powerful tool for knowledge management and data integration. Its structured format and rich linking capabilities make it an ideal resource for machine learning and artificial intelligence applications. Although there is this massive data, most research and industrial use cases need a *subset* of items, statements, and metadata. This paper discusses the new research problem of Wikidata subsets, their definition, the significance of their extraction, and the methods to retrieve them.

1.1. The significance of Subsets

Having subsets of KGs have many benefits. The first benefit of subsetting is avoiding size issues. General purpose KGs such as Wikidata are valuable sources of facts about various topics. On the Linked Data Web, they serve as a common linking point between inter-, and sometimes intra-, domain KGs[4]. However, their increasing size makes them costly and slow to use locally. Additionally, the large volume of data in Wikidata increases the time required to run complex queries. This often restricts the types of queries that can be posed over the public endpoint since it has a strict 60-second limit on the execution time of queries. Any query that takes more time to execute than this will timeout[5].

Downloading and using a local version of Wikidata is one way of circumventing the timeout limit. However, it is not a cheap option due to the size of the data. Wikidata JSON dump of 14 December 2022 is 112GB in a compressed format. A suggested hardware required to have a personal copy of Wikidata includes 16 vCPUs, 128GB memory, and 800GB of raided SSD space[6]. A Google Cloud computation engine with these specifications would cost more than $527 per month[7]. Although the costs for infrastructure are relatively affordable, considering the value and potential use cases of having a local copy of Wikidata, there are many instances where only a small portion of Wikidata is relevant. In such cases, hosting a complete copy can be considered excessive and unnecessary. This makes it difficult to secure the necessary funds for such an infrastructure.

Additionally, out of this 112GB of data, one might need no more than 1GB on a specific topic. There are several use case scenarios where users do not need access to all topics in a massive general-purpose KG. A small and complete enough subset can be more suitable for many purposes. For example, a subset of all information about genes, proteins, drugs, and diseases can be used in pharmaceutical research [2]. Even in general-purpose use cases covering broad domains, small subsets can help. For example, in an open-domain Question Answering interface, the system may detect the domain category of a given question first, then refer to the smaller subsets in the detected domain to retrieve the facts, speeding up the query time. With a small subset, inference strategies can be applied to the data and completed in a reasonable time. Subsets can also be published along with papers, which provides better reproducibility of experiments [3]. Small subsets are also easier to archive and are more likely to be reused [4].

[1] https://creativecommons.org/publicdomain/zero/1.0/ - accessed 19 February 2023
[2] https://query.wikidata.org/bigdata/namespace/wdq/sparql?query={SPARQL} - accessed 19 January 2023
[3] https://query.wikidata.org/ - accessed 19 January 2023
[4] https://lod-cloud.net/ accessed 20 February 2022
[5] https://en.wikibooks.org/wiki/SPARQL/Wikidata_Query_Service/
[6] See this post: https://addshore.com/2019/10/your-own-wikidata-query-service-with-no-limits/
[7] Estimated by Google Cloud Pricing Calculator: https://cloud.google.com/products/calculator/#id=32eca290-7628-48af-9988-20508f4bc861 accessed 11 February 2023

Various topical archives can be created from Wikidata, which gives better access to the data, while multiple time snapshots can be built from this data. Subsets enable complex querying on cheap servers or personal computers — reducing the overall cost — and making the experiments reproducible.

Establishing comparison platforms is the other benefit of subsetting. Consider the aim to examine a feature unique to Wikidata (e.g., referencing). As there is no comparable KG, different subsets of Wikidata in multiple topics can be used as comparison parties. Also, random subsets of Wikidata can be regarded as a random samples of Wikidata items and statements. Subsetting also allows us to see whether there is uniform coverage of references across all of Wikidata and identify variations between different contributor communities.

Another advantage of subsets is populating new topic-oriented KGs. An example of Dan Brickley can be taken in this context: "Subsetting KGs is like cutting a plant and placing it in a new pot. So it can grow and become a new topic-specific KG ..."[8]. For example, in the case of extracting a life science subset of Wikidata, the extracted subset can be considered a life science knowledge graph, which can subsequently be enriched with additional triples, creating a new Life Science KG based on the Wikidata data model and enriching its contents with other contents.

1.2. What Is a Subset?

In its broadest sense, subsetting refers to extracting the relevant parts from a KG. Considering a KG (regardless of semantics) as a collection of nodes, edges and an associated ontology, a subset can be an arbitrary number of combinations of these three. Thus, in a broad definition, any query graph pattern can be considered a subset, but subsets can include more general cases. Including repetitive graph algorithms such as shortest paths and connectivity [5]. To the best of our knowledge, there is no precise formal definition for submitting accepted by the community [6].

The input of the subsetting process is generally a KG. Over the KG, *filters* are applied to separate the desired parts of the graph. The output of this process can be in the form of a graph (directed edge-labelled or property graph) in various formats, tables, or JSON. The most straightforward way to subset an RDF KG is to use SPARQL **CONSTRUCT** queries on the endpoints of a triplestore. This method is suitable for simple and small subsets but has limitations for large and complex subsets. SPARQL endpoints are usually slow and have run-time restrictions. Moreover, recursive data models are not supported in standard SPARQL implementations [7].

1.3. Objectives and Contribution

This research aims to collect all available Wikidata subsetting approaches and tools, test their capabilities, and analyse their advantages and disadvantages. The scope is individual, independent, local and arbitrary subsetting, i.e., use cases where users can subset Wikidata locally over any subsetting filters they desire without relying on publicly available servers or datasets. The main reason is that public servers usually apply limitations on the type and run-time of applications. The contributions of this paper are:

1. A survey of emerging practical knowledge graph subsetting tools (Section 3);
2. Performance analysis of practical Wikidata subsetting tools (Section 4);
3. Discussion of the flexibility of practical subsetting tools through tangible Life Science subsetting use cases (Section 5).

This paper first reviews the Wikidata RDF model and the terminology used in the paper in Section 2. In Section 3, a survey of the available methods for subsetting will be presented in detail. In Section 4, the apper investigates the performance (run-time and extraction statistics) and accuracy (what has been extracted and excluded) of the state-of-the-art subsetting tools. In Section 5, a discussion of the flexibility of the practical tools will be given by going through three Life Science subsetting use cases. Finally, the paper will be concluded in Section 6.

[8] BioHackathon Europe 2021, Project 21: Handling Knowledge Graphs Subsets (group discussions). Notes: https://seyedahbr.github.io/Blog/Biohackathon21.html - accessed 12 Feburary 2023

2. Wikidata RDF Model

The fundamental components of Wikidata are *items* which are concepts or entities from the real world, such as humans, chemicals, articles, etc. and *properties*, which are relationships between two items or between items and values. Items and properties have internal identifiers: item IDs start with a 'Q', and property IDs with a 'P' character, followed by an incremental number in their category. Relationships between entities create claims: a property that explains a fact about an item. Claims can be enriched by adding qualifiers to provide contextual information and/or references, to provide provenance and form *statements*. In other words, statements are those claims having some additional contextual metadata.

Wikidata is powered by the *Wikibase*[9] software collection which provides applications and libraries for creating, managing and sharing structured data, created by Wikimedia Foundation and is freely available as a Docker image[10]. Wikibase provides a syntax highlighting SPARQL query interface that supports federated queries, a Javascript-based GUI for populating data, and a Blazegraph triplestore [8] to store and manage RDF data. Wikibase also provides the EntitySchema extension that supports Shape Expressions, which as will be described later, has a role in subsetting. Wikibase has several other software components that are needed to create a knowledge base similar to Wikidata data model. Data can be exported in many formats like JSON, RDF/XML, OR N3, and it defines its data model which is used by Wikidata. In addition to Wikidata, there are other open KGs hosted in Wikibase instances, e.g., the Rhizome [9], FactGRID [10], and EU Knowledge Graph [11].

Wikidata uses reification based on intermediate nodes to store contextual metadata, known as *qualifiers*, and provenance metadata, known as *references*, for statements. As an example, Figure 1 shows the representation for the *speed limit (P3086)* statement in *Germany (Q183)*. The top of the image shows the representation of this statement in the Wikidata GUI. The bottom of the image shows the RDF graph of the information. The speed limit statement value can be reached directly by the **wdt:P3086**. To access qualifiers, references, and the rank of the statement, the intermediate 'Statement Node' must be used, represented with a **wds:** prefix. This intermediate node can be accessed by the **p:** combined with the same statement property identifier. From the statement node, qualifiers are accessible by the **pq:**, references by the **prov:wasDerivedFrom**, ranks by the **wikibase:rank**, the default value-unit with **psv:**, and the conversion to the default URI mapping using **psn:**. Note that in Wikidata, values can be simple literals (i.e. text or values), IRIs, or can be complex data types called a *full value*, storing more metadata about a literal value such as units, ranges, precision, and the calendar used [12]. Another important notion in Wikidata is the rank of statements. In Wikidata, statements can have normal, preferred, or deprecated ranks. Deprecated rank refers to a property value that is not considered correct (based on the statement's context, such as qualifiers or references). In Wikidata, "statements that have the best non-deprecated rank for given property" are called *Truthy statements* [12]. In other words, a deprecated statement can never be a truthy statement. Items, statements, contextual metadata, provenance metadata, and all other parts of this reification can be used to define a subset.

3. Subsetting State of the Art

Subsetting is a recent research problem in KGs. To the best of our knowledge, the early demand for creating a biomedical subset of Wikidata was in 2017 [13], the subsetting discussions in the Wikidata biomedical community were concretely started at the 12th international SWAT4HCLS conference in 2019 by Andra Waagmeester *et al.* [14] and then followed in Project 35[11] of ELIXIR BioHackathon-Europe 2020 [15], Project 21[12] of ELIXIR BioHackathon-Europe 2021, and Project 11[13] of ELIXIR BioHackathon-Europe 2022 [16].

[9] https://wikiba.se/ - accessed 12 December 2022
[10] https://hub.docker.com/r/wikibase/wikibase - accessed 15 December 2022
[11] https://github.com/elixir-europe/BioHackathon-projects-2020/tree/master/projects/35 - accessed 20 December 2022
[12] https://github.com/elixir-europe/BioHackathon-projects-2021/tree/main/projects/21 - accessed 20 December 2022
[13] https://github.com/elixir-europe/biohackathon-projects-2022/tree/main/11 - accessed 20 December 2022

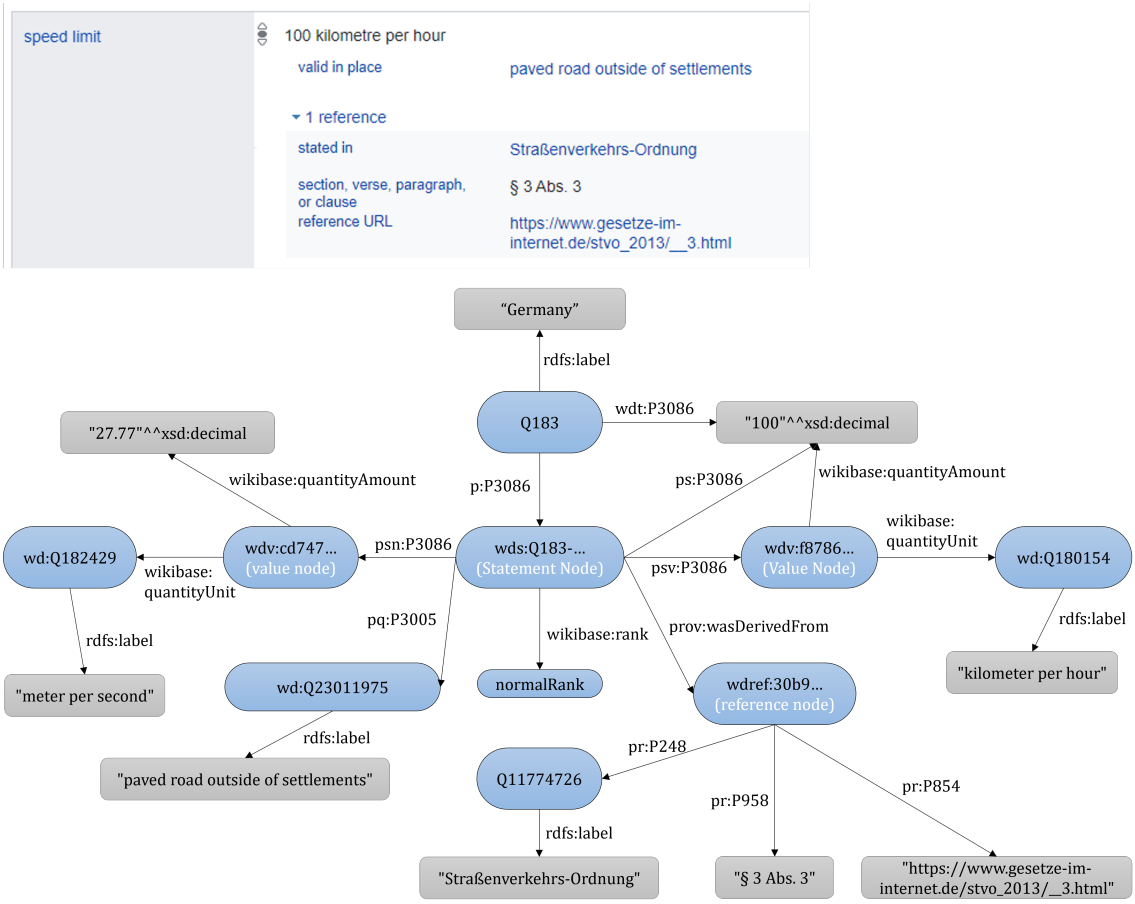

Fig. 1. Top: One of the *speed limit (P3086)* statements of *Germany (Q183)* on Wikidata GUI (retrieved on 14 December 2022). Bottom: various elements that can be extracted from Wikidata from *Germany (Q183)*.

3.1. General Purpose Subsetting Approaches

Matsumoto *et al.* [17] have introduced the Graph-to-Graph Mapping Language (G2GML) that aims to convert RDF graphs to property graphs. G2G Mapper[14] is a tool that receives a mapping configuration file written in G2GML and an RDF turtle file (or a SPARQL endpoint) as input and creates a property graph from the RDF data specified by the input mapping. Although the purpose of the G2GML language was to generate property graphs from RDF graphs to take advantage of the property graphs, it can be used as a subsetting tool; however, the output will be a property graph. For subsetting, an RDF output is preferable as it is standardized, and evaluating them is straightforward. Another limitation is that one needs to completely define the Wikidata ontological structure and data model in the form of property graphs, especially references. In that way, a mistake or forgotten property can affect the future evaluation of the subset.

Mimouni *et al.* [18, 19] use a concept called the Context Graph to generate a smaller dataset than the original massive KGs such as DBPedia and Wikidata, which enables them to test their knowledge base completion method on this dataset instead of the entire KG. The context graph construction algorithm starts with an initial set of *seed nodes*, and in each round, adjacent nodes of the seed set (that are not in a forbidden set) and their relations are added to the seed nodes. This operation continues for several rounds called the *radius*. The context graph production process seems to be suitable for generating random subsets; however, it is not an integrated method for generating

[14]GitHub: https://github.com/g2glab/g2g - accessed 20 December 2022

subsets around a topic. To produce subsets around a topic, it is necessary to identify the member entities of a particular topic. However, there is no such concept in the context graph. One has to extract all the nodes related to a topic from the beginning and put them in the initial seed set. On the other hand, extracting node neighbours to a $radius \geqslant 2$ may enter information that is not relevant to the topic. Another limitation is that this approach is not able to extract Wikidata contextual metadata, especially references.

Henselmann and Harth [20] developed an algorithm for creating on-demand subsets around a given topic from Wikidata, starting from a seed set of nodes and performing multiple SPARQL queries to obtain the desired triples. Their approach can be used to create subsets around topics. However, the authors do not provide use cases or evaluation of their algorithm, thus it is more a theoretical approach than a practical tool. The proposed algorithm and its SPARQL queries are also not compatible with references. Aghaei *et al.* [21] proposed an approach to create an on-demand sub-graph of a KG for Question Answering (QA), which is a common approach in heuristic-based QA over KGs [21]. In this approach, a set of entities is first fetched from the question. A neighbour graph query pattern is then used to create a knowledge sub-graph of those nodes' neighbours and their relationships from the KG. Similar to the context graph approach, the neighbour nodes are extracted up to a specific distance (hop). The limitation of this subsetting approach is that those are specific-purpose methods designed to answer natural language questions. These methods create the subset at the moment of answering the question, do not care about extracting the contextual metadata, and do not return the constructed subset as a portable output.

Shape Expressions (ShEx) [22] is a structural schema language allowing validation, traversal and transformation of RDF graphs? There are several ShEx validator implementations, e.g., shex.js [23] and PyShex [24], which receive a ShEx schema as the input and validate an RDF graph over it. These validators can keep track of the triples traversed during validation and return the matched triples out (called 'slurping'), which can be used to define data schemata which could result in extracting a subset. ShEx is a language for validating RDF data, and its evaluators are for checking the shape of the graph against a schema, not for extracting. Although the language has the most flexible way to define subsets, its evaluators' slurping capabilities are limited as they can not handle the massive size of Wikidata.

3.2. Practical Tools

WDumper[15] [25] is a third-party tool for creating custom and partial RDF dumps of Wikidata suggested at the Wikidata database download page [26]. The WDumper backend uses the Wikidata Toolkit (WDTK) Java library to apply filters on the Wikidata entities and statements, based on a specified configuration that is created by its Python frontend. This tool needs a complete JSON dump of Wikidata and creates an N-Triple file as output based on filters defined in the configuration file. This tool can be used as a topical subset creator; however, it cannot be said that WDumper can build a complete topical subset. This is due to the limitations of this tool, e.g., not supporting extracting the subclasses and the lack of making connections between separate filters. With a few changes and using a Python random generator script[16], WDumper can be extended to extract random subsets from Wikidata of any size [27]. Beghaeiraveri et al. [6] introduced the concept of Topical Subsetting over Wikidata using WDumper, extracting four topical Wikidata subsets. Beghaeiraveri et al. [28] used WDumper to extract six Wikidata topical subsets corresponding to six Wikidata Wikiprojects: Gene Wiki, Taxonomy, Astronomy, Music, Law, and Ships. Topical and random subsets of Wikidata are being used as the comparison platform for evaluating Wikidata references [29].

The flexibility of the ShEx language motivated researchers to develop a specific subsetting tool for Wikidata based on this language. WDSub [30] is a subsetting tool implemented in Scala that accepts ShEx schemata and extracts a subset corresponding to the defined schema from a local Wikidata JSON dump. The extractor part of the WDSub is similar to WDumper, i.e., the WDTK java library. In addition to traditional ShEx schemata in ShExC format, WDSub has its own subsetting language, WDShEx [31], which is a shape expression language based on ShEx and optimized for Wikidata RDF data model. WDSub can produce both RDF and Wikibase-like JSON outputs.

[15] Demo: https://wdumps.toolforge.org/ - accessed 20 December 2022
[16] https://github.com/seyedahbr/wdumper/blob/12f0ddf/extensions/create_random_spec.py - accessed 10 June 2023

Table 1
The summary of subsetting tools capabilities. WIP stands for Work In Progress.

Tool	Output Format	Input Format	Subset Definition Language	Average Infrastructure Requirements	Requires Full Dump	Live Subsetting	Supports Massive Data	Supports Qualifiers	Supports References	Graph Traversal	Further Output Transforms	Analytics
ShEx + Slurp	RDF	RDF (SPARQL)	ShEx	PC	-	+	-	+	+	+	-	-
WDumper	RDF (N-Triple)	Wikibase JSON	JSON spec file	PC	-	-	+	+	+	-	-	-
WDSub	Wikibase JSON/RDF	Wikibase JSON	ShEx/ WShEx	PC	+	-	+	+	+	-	-	+
SparkWDSub	Wikibase JSON	Wikibase JSON	ShEx	Spark Cluster	+	-	+	WIP	WIP	+	-	-
WDF	NDJSON	Wikibase JSON	Command line filters	PC	+	-	+	+	+	-	-	-
KGTK	TSV/RDF	Wikibase JSON/RDF	Kypher	PC	+	-	+	+	WIP	+	+	+
SPARQL Construct Queries	RDF	RDF (SPARQL)	SPARQL	PC	-	+	-	+	+	-	+	-

Knowledge Graph Toolkit (KGTK) [32, 33] is a collection of libraries and programs to manipulate KGs. KGTK is designed to make working with knowledge graphs easier, both for populating new KGs or developing applications on top of KGs. It is implemented in Python, including a command-line tool for multiple utilities such as importing and exporting Knowledge from various formats (e.g., RDF, CSV, JSON), merging and combining KGs data, creating KGs from unstructured sources, querying and analyzing KG data, etc. The fundamental operations in KGTK are importing and querying. KGTK imports massive KGs and converts the data to TSV files, and uses a Cypher-inspired language (called Kypher) to query from these TSV files. In the context of Wikidata, KGTK has been deployed in multiple quality and population-related studies (such as [34, 35]). However, its main limitation in Wikibase-driven datasets is not to support indexing of referencing metadata.

Wikibase Dump Filter (WDF) [36] is a Node.js tool to filter and process the JSON data dumps Wikibase, developed and maintained by the Wikimedia Foundation. Similar to WDumper, WDF is an item-based filtering tool, i.e., it applies different filters on items, claims, qualifiers and other Wikibase JSON dump components to create a new dump of desired items of Wikidata. It can also be used to filter revision dumps of Wikibase-driven datasets. WDF can transform the filtered data into CSV, as well as NDJSON[17].

Table 1 shows the summary of tool capabilities. The first column lists the name of the tool. The second column lists the output format the tool generates. The third column shows the required data input format for the tool. The fourth column lists the language/format used to define a subset. The fifth column reflects the average hardware and software infrastructure required for using the tool to extract a subset. The sixth column reflects whether or not a full Wikidata dump download is required for subsetting. The seventh column indicates whether or not the tool runs on live data. The eighth column reflects whether or not the tool is scalable for large subsets or extract subsets of Massive KG. The ninth column indicates whether or not the tool supports extracting qualifiers. The tenth column indicates whether or not the tool supports extracting references. The eleventh column indicates whether or not the tool provides support for graph traversal (i.e. exploring paths between nodes, including cycles to define a subset). The twelfth column indicates whether or not it is possible to use the tool to perform additional data transformations (e.g., RDF format conversion) without third-party tools once the subset is extracted. The thirteenth column reflects whether the tool provides analytics about the content of the extracted subset, e.g., the number of triples and items. Note that approaches such as the Context Graph or Aghaei et al. are not listed as those are one-purpose and cannot be reused for arbitrary subsetting.

The table shows that no tool provides all positive functionalities. Most of the tools can be run on PCs except SparkWDSub, which is designed for scalability purposes. WDumper is considered a tool that requires no access

[17] http://ndjson.org/ - NDJSON is a line-separated file in which every line is a valid JSON value. In WDF output, each line is a JSON blob of Wikidata JSON dump representing one item.

Table 2
Details of the 3 January 2022 Wikidata dump used as the data input for experiments.

Dump Date	Dump Format	Compressed Size	Total Items	Total Statements
3 Jan 2022	JSON.gz	102GB	95,900,304	1,353,626,249

to the local dump as it is available from an online demo which uses the latest Wikidata JSON dump. None of the practical tools is capable of live subsetting. Instead, they can deal with massive dumps, where ShEx slurping and SPARQL queries fail. Supporting graph traversal is also a challenging feature available in KGTK and SparkWDSub amongst the practical tools (however, SparkWDSub is in the early development stages).

The SPARQL `CONSTRUCT` queries can be considered the most available approach for subsetting Wikidata and other KGs, while regarding independent and local subsetting, they have limitations that exclude them from being a practical approach. The first limitation is defining a subset with `CONSTRUCT` queries is time-consuming, as the end-user needs to write the entire graph shape they want to extract. For example, if the end-user defines a subset of Genes, (in addition to the select filters) they should explicitly define what statements, labels, qualifiers, and references should be in the output. Once the scope of the subset gets complicated, specifying the connectivity of the output graph is even more challenging. Such detailed graph patterns can also be outdated very fast as the RDF data is schema-independent; therefore, users should constantly review and modify their queries. Another limitation is the query endpoint. Public endpoints usually apply concrete run-time and query-type limitations, which reduces the capabilities of `CONSTRUCT` queries (for example, users can not extract `rdf:type` triples or write heavy queries with more classes included), and raising a local endpoint (on a Blazegraph instance or other triplestores) returns us to the cost limitations again. Another limitation to use SPARQL `CONSTRUCT` queries to describe the subsets is the lack of support for recursion so it would not be able to handle the definition of subsets with cyclic data models. Overall, while SPARQL `CONSTRUCT` queries are a tangible approach for small subsetting use cases, we don't consider them a practical solution for subsetting.

4. Performance and Accuracy Evaluation

This section is dedicated to an evaluation experiment on the performance and accuracy of the four practical tools: WDSub, WDumper, WDF, and KGTK. Considering the size of Wikidata, the subsetting tools need to extract data in a feasible time. A fast extraction can reduce processing costs and pave the way for regular subset updates and live subset generation. Subsetting tools should also create accurate outputs. Accuracy in this context means the output of a subsetting tool should include all desired statements and exclude any other data. To assess the performance and the accuracy of the practical Wikidata subsetting tools, a unified test on each subsetting tool is performed and the extraction time and the content of the output is reported. The scripts, schemas, and SPARQL queries of this experiment can be found in the GitHub repository of the paper [37]. The extracted subsets, along with can be found on Zenodo [38].

4.1. Experimental Methodology

In addition to the size of Wikidata, there are other factors contributing to the speed of subset extraction: (i) the number and complexity of filters applied to the input, (ii) the type of the output data (RDF, JSON, etc.), and (iii) the internal operations of the tool. By keeping the input dump and the desired filters fixed, the internal operations run-time is calculated.

4.1.1. Input Dump

The Wikidata JSON dump of 3 January 2022 [39] is used as the input to the four subsetting tools. Table 2 shows the details of the input dump. The input dump was downloaded from the Wikidata Database Download page [40].

4.1.2. Subsetting Filters (Performance Test)

The experiment considers a life-science subset of Wikidata as the test use case with the following conditions.

- The subset includes all and only '*instances of(P31)*' *gene (Q7187)*, *protein (Q8054)*, *chemical compound (Q11173)*, and *disease (Q12136)*.
- The subset does not include the instances of subclasses. For example, if the tools extract the instances of *gene (Q7187)* class, instances of the *operon (Q139677)* class should not appear in the output.
- The subset includes all statements about the items but does not require to include qualifiers or references.

chemical compound (Q11173), *disease (Q12136)*, *gene (Q7187)*, and *protein (Q8054)* are the main Gene Wiki WikiProject classes. Each of these classes includes several subclasses in Wikidata that have many instances. For example, the *gene (Q7187)* class has 1,004,350 subclasses[18] of which *operon (Q139677)* is one. The condition of including no subclass examines the sensitivity in detecting the defined class only. Since KGTK cannot index and extract references, no filters are applied to the references to keep the evaluation equal.

4.1.3. Subsets Validation (Accuracy Test)

To measure the accuracy, after finishing the extraction and recording the execution time and the raw volume of the output, we perform the following set of queries on the input (Wikidata dump) and the output of each tool:

Condition 1: The total number of items (Q-IDs) that are instances of *chemical compound (Q11173)*, *disease (Q12136)*, *gene (Q7187)*, and *protein (Q8054)* classes.
Condition 2: The total number of statements of the items that are instances of *chemical compound (Q11173)*, *disease (Q12136)*, *gene (Q7187)*, and *protein (Q8054)* classes.
Condition 3: The total number of items (Q-IDs) that are instances of *operon (Q139677)* and *acid (Q11158)*.

Comparing the results of Condition 1 and Condition 2 in the input dump and the output of each tool is a measure of how well the tools extract what they are supposed to fetch. Condition 3 checks the existence of two subclass instances (Operon as a subclass of Gene, and Acid as a subclass of Chemical Compound), aiming to avoid including false positives. The Operon and Acid are arbitrary subclasses; however, operons have an extra semantic relation to genes (an operon is a functioning unit of DNA containing a cluster of genes) and proteins, while acids do not have such extra relation to chemical compounds. In that way, the two subclassing relations can be further compared and the misconfiguration of the tools can be found.

4.1.4. Output Format

In this experiment, the output type of WDumper and WDSub is RDF. WDumper creates GZip NTriple files. WDSub creates GZip Turtle files. WDF produces NDJSON files. The output type of KGTK is a TSV file. There are also differences in the size of different RDF formats. The type and the format of the outputs is reported; however, the difference should be kept in mind when comparing the results. The calculated time includes serialization to RDF and the time required to write to disk.

4.2. Experimental Setup

4.2.1. Host Machine

The experiments were performed on a multi-core server powered by 2 AMD EPYC 7302 CPUs (16 cores and 32 threads per CPU), 320GB of memory, and 2 hard disks: a 256GB SSD that runs the operating system (CentOS 7 kernel 3.10.0-1160.81.1.el7.x86_64 amd64) and a 6TB HDD that is used for the extraction steps.

4.2.2. Software Versions

Table 3 shows the versions of subsetting tools and software used for compiling. All versions were available on 12 November 2022. WDumper has no released version; therefore, the used commit ID is mentioned. All tools except WDF have Docker containers; however, all mentioned versions are cloned and compiled with no need to have root permissions. For KGTK, the repository-recommended binary package in Conda is installed, using pip. To the best of our knowledge, WDSub and KGTK are being upgraded regularly.

[18] https://w.wiki/69Bt - queried 24 December 2022

Table 3
Software versions and compiler/interpreter used.

Tool	Version	Compilers/Interpreters details	License
WDSub	version 0.0.28	sbt version 1.6.2, openjdk version “11"	MIT
WDumper	commit dc325fc	gradle 7.0.2, openjdk version “11"	MIT
WDF	version 5.0.7	npm 8.19.2	MIT
KGTK	version 1.4.3	conda 22.9.0	MIT

4.2.3. Experimental Run

A Python script[19] runs each tool three times separately from the moment of starting with the raw input dump to the moment it saves the output on disk. In this way, the time required for any indexing and pre-processing of the dump (if any), as well as the time of writing the output, is included in the extraction time, which is in line with the local and independent subsetting scope. Since the host machine is assumed to run other tasks at the same time, the extraction is repeated three times and the average and the standard deviation of the three runs are presented. While WDF and KGTK accept the filtering embedded in the command line, WDumper accepts a JSON specification file[20] and WDSub accepts a ShEx schema[21]. RDF outputs of WDumper and WDSub were imported in Blazegraph triplestore. In all cases, the recommended configurations and command line arguments which are mentioned in the online documentation of the tools are deployed. Note that amongst the four tools, KGTK supports multithreading. However, KGTK focuses on handling KGs on laptop computers [41]; therefore, its recommended settings use only six threads[22]. Then a set of SPARQL queries[23] has been performed to count the instances and statements in RDF outputs. For KGTK which produces TSV outputs, a Python script (using `pandas` package) has been used [24]. For counting the number of instances and statements in the WDF output and the input dump (which are JSON files), a parallelized Python script[25] with efficient time consumption has been used. Note that while each Wikidata JSON dump has an RDF pair dump, these two different serializations are not identical [42]. Therefore the JSON dump is queried directly using the Python parallel program.

4.3. Performance Test Results

Table 4 shows the output detail, results of extraction time, and the total number of distinct items and statements in the output of each tool. The output of WDSub and WDumper is significantly smaller due to compression. The KGTK output is not compressed; however, it is still as small as WDSub and WDumper. It is because other tools extract the entire metadata of the matched item, including labels, descriptions, qualifiers, etc., while KGTK extracts the statement triples only. In its TSV output, KGTK keeps the Q-IDs only and omits any prefixes, which results in light and fast-writing outputs. Note that KGTK can be set to extract other metadata; however, performing this requires additional conditions and filters, which are not necessary for the experimental scenario (see Section 4.1.2) and increases its extraction time. As well, WDumper, WDF, and WDSub can be set not to extract metadata; however, applying such filters enforces unnecessary overhead in their extraction time.

The extraction times show that WDF is the fastest tool. Part of that is because JavaScript is efficient in reading JSON files. The WDF filters are also basic, and parsing the conditions can be done straightforwardly. KGTK is the

[19] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/blob/a517842/performance-experiments/tool_runner.py - accessed 10 June 2023

[20] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/blob/a80a867/performance-experiments/gene_protein_disease_chemicals.json - accessed 10 June 2023

[21] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/blob/0d54e50/performance-experiments/gene_protein_disease_chemicals.shex - accessed 10 June 2023

[22] It is worth reporting that KGTK v1.5.3 over 32 threads and avoiding deprecated statements has been run and the tool was unsuccessful to return an output after three days of processing.

[23] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/tree/1eab3d9/performance-experiments/sparql - accessed 10 June 2023

[24] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/blob/dc2869c/performance-experiments/count_instances_tsv.py - accessed 10 June 2023

[25] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/blob/d63a3b1/performance-experiments/count_instances_json_iter.py - accessed 4 June 2023

Table 4
The results of running the four practical tools: size and type of the output, the average (Avg.) and standard deviation (STD) of the extraction time, the number of items and the number of statements.

Tool	Output Type	Output Size (GB)	Extraction Time (sec)		Items	Statements
			Avg.	STD		
WDSub	ttl.gz	2.7	43,060	126	3,434,509	38,372,871
WDumper	nt.gz	3.1	23,427	97	3,434,538	38,373,706
WDF	ndjson	36	13,876	52	3,434,538	38,373,706
KGTK	tsv	3.6	17,148	1,020	3,434,506	38,366,611

second fast tool which benefits from multithreading, providing a high variance of extraction time. KGTK extraction includes two stages: importing Wikidata and the query itself. In these experiments, 40% of the KGTK run-time was spent importing the Wikidata JSON dump and converting it into three TSV files corresponding to nodes, edges, and qualifiers. The rest 60% of the run-time was spent on the query. KGTK creates a graph cache in SQLite format from the edges TSV file once the first query is performed, which significantly speeds up subsequent queries to at most one hour. Thus, most of the query run-time is spent creating the graph cache for the first time. With such a feature, KGTK can be used to compute the graph cache once. Then the graph cache can be shared by Wikimedia or third-party associates for queries. However, in the context of this investigation, since the paper considers autonomous and arbitrary subsetting (and not publicly available servers), the graph cache processing in the run-time is included. Although WDF and WDumper traverse the JSON dump similarly line by line, and WDumper is a compiled tool, WDumper is slower. A part of this slowness is because WDumper serializes the matched JSON blobs to RDF. Also, WDumper can accept more complex filters that create a level of overhead in extraction (regardless of having a simple specification input). The same is true for WDSub. The RDF serializer in WDumper and WDSub is the same; however, the WDSub filtering system (based on ShEx) can parse quite complex filters at the SPARQL level, which creates a massive overhead. WDumper also has a better level of multithreading than WDSub.

Comparing the number of extracted items and statements shows that KGTK has the least number. The reason behind the higher ratio of missed items and statements in KGTK output is not clear, but it cane be hypothesized to be due to the greater complexity of indexing and query procedures in KGTK compared to other tools, a higher likelihood exists for skipping more blobs during intermediate steps due to their un-parsability. The number of extracted items and statements in WDF and WDumper is identical, although this identicality is coincidental as these numbers are the distinct add-up of four different classes. The disaggregated statistics, as discussed in Section 4.4, show that these tools extract a different number of instances in each class.

4.4. Accuracy Test Results

Table 5 shows the result of accuracy test queries on the input dump and each tool separately. In the Condition 1 column, the number of instances of each class can be seen. Compared to the input dump, all tools missed extracting some Q-IDs except WDF. The WDF filter matching process is the simplest among the available tools. It involves scanning the input dump line by line, with each line containing a JSON blob corresponding to a Wikidata item. The filters provided are then applied to the values within each JSON blob, and if a successful match is found, the entire blob is returned. Moreover, the number of extracted statements matches the input dump, highlighting the exceptional accuracy of WDF compared to other tools. The ratio of the missing items in other tools is less than 0.05%, and the ratio of missing statements is less than 0.75%. From 1,196,532 gene instances in the input dump, WDSub did not extract 44, and WDumper and KGTK did not extract 29 gene instances. Although the rate is acceptable, a 100% accuracy is expected for this task. Reviewing the gene instances items that are present in the input dump but are not

Table 5
Accuracy test results of the four tools.

	Condition 1		Condition 2	Condition 3	
	Class	**Items**	**Statements**	**Class**	**Items**
Input Dump	Gene	1,196,532	15,993,915	Operon	731
	Protein	987,636	11,365,759		
	Chemical Compound	1,244,881	10,942,239	Acid	22
	Disease	5,513	72,480		
WDSub	Gene	1,196,488	15,993,260	Operon	0
	Protein	987,614	11,365,230		
	Chemical Compound	1,244,859	10,941,100	Acid	7
	Disease	5,511	72,416		
WDumper	Gene	1,196,503	15,993,730	Operon	0
	Protein	987,636	11,365,759		
	Chemical Compound	1,244,874	10,941,562	Acid	7
	Disease	5,512	72,477		
WDF	Gene	1,196,532	15,993,915	Operon	0
	Protein	987,636	11,365,759		
	Chemical Compound	1,244,881	10,942,239	Acid	7
	Disease	5,513	72,480		
KGTK	Gene	1,196,503	15,988,146	Operon	0
	Protein	987,636	11,366,235		
	Chemical Compound	1,244,879	10,941,671	Acid	7
	Disease	5,512	71,933		

in the outputs of tools shows that the 29 missed items in KGTK[26] and WDumper[27] are identical. Plus additional 15 instances, those 29 items are missed in WDSub[28] too.

Analysis of the JSON blobs for some missed instances, such as *xmas-1 (Q29718370)*, *NGB (Q418553)*, *AH10.3 (Q29685684)*, and *EGAP798.1 (Q29678017)* revealed no issue concerning the *instance of (P31)* claims which serve as the basis for filters. However, we noticed some malformed characters, such as `<200d>` and `\\` in the Unicode label values[29]. Two other missed instances have a _ character in their Bangala label values[30]. The effect of bad characters has been already reported in the context of Wikidata RDF dumps [6, §4.4]. The fact that WDF did not miss any items or statements leads us to assume that other tools may struggle with parsing those certain values within different parts of the JSON blobs, such as names, descriptions, or date-time values, resulting in the skipping of the entire blob. All three tools involve intermediate operations that are potentially sensitive to datatype parsing. In the case of WDumper and WDSub, the RDF serialization step may cause unparsing, while KGTK utilizes a ShEx engine, which can introduce further sensitivity. Moreover, KGTK's use of the graph cache is considered a potentially sensitive stage, especially since missed items appear in the output of the importing step (the initial "nodefile.tsv") but are not extracted during the query step. Malformed characters of this nature can arise either from internal misfunctioning of the Wikibase software or through direct entry by contributors. Another observation is that 19 *instance of (P31)* claims for 15 Q-IDs were duplicated in the KGTK output[31], i.e., the entire `<item,P31,Q7187>` was

[26] List of missed gene instance Q-IDs: https://zenodo.org/record/8015611/files/dump_kgtk_unique_items.txt?download=1 - accessed 10 June 2023

[27] List of missed gene instance Q-IDs: https://zenodo.org/record/8015611/files/dump_wdumper_unique_items.txt?download=1 - accessed 10 June 2023

[28] List of missed gene instance Q-IDs: https://zenodo.org/record/8015611/files/dump_wdsub_unique_items.txt?download=1 - accessed 10 June 2023

[29] See Line 111 of file 'item-Q418553-found.json' in [38] and Line 40 of file 'item-Q29718370-found.json' in [38] - accessed 8 Jun 2023

[30] See Line 11 of file 'item-Q29685684-found.json' in [38] and Line 11 of file 'item-Q29718370-found.json' in [38] - accessed 8 Jun 2023

[31] List of duplicated gene instance Q-IDs: https://zenodo.org/record/8015611/files/kgtk_repetitive_items.txt?download=1 - accessed 10 June 2023

repeated. The reason for this phenomenon is unknown; however, checking one of the instances, *NGB (Q418553)*[32] shows that the item is an instance of *chemical compound (Q11173)* and *gene (Q7187)* at the same time. The presence of both gene and chemical compound classes in the extraction filters can lead to erroneous duplication of statements, potentially attributed to an internal malfunctioning of KGTK.

4.5. Discussion

Choosing amongst the available subsetting approaches depends on the task at hand. The methods introduced in Section 3.1 are single-purpose and usually cannot be reused to create any arbitrary subset. Amongst the practical tools (Section 3.2), the performance and accuracy evaluation showed that WDF has the fastest and most accurate performance; however, this tool is not flexible in defining subsets. This problem also exists in WDumper. In these two tools the inclusion and exclusion of items, statements, and contextual metadata can be defined, there is no possibility to make a connection between these conditions. For example, disease instances and chemical compound instances can be extracted together; however, if only the chemical compounds related to the extracted diseases are needed, this joined KG cannot be extracted with these tools.

KGTK and WDSub offer much higher flexibility due to their subset-defining structure derived from graph query languages. KGTK extracting data after a round of indexing relatively fast; however, in the context of Wikidata lacks indexing references, which is a major drawback. WDSub has the most flexible subset-defining structure in the Wikidata ecosystem and is reasonably accurate; however, response time is slow and still in its early stages of development (as of June 2023).

5. Flexibility Evaluation

The extent to which each tool supports common subsetting workflows is crucial. While Section 4 focuses on the performance and accuracy in a single subsetting scenario, it should be noted that tools offer varying degrees of support for various subsetting tasks, depending on their functionalities and features. The flexibility experiments showcase the range of potential applications and highlight the appropriateness of each tool for specific subsetting requirements. This section investigates more diverse subsetting tasks involving different parts of the Wikidata data model supported by each evaluated tool, thereby providing a more comprehensive understanding of their practical applicability. The first use case is the Gene Wiki project evolution from 2015 to 2022, the second is genes names and descriptions in four languages, and the third is instances of chemical compounds that are referenced with *reference URL (P854)*. All subsets were extracted using WDSub; however, the possibility of creating a similar subset using other practical tools is discussed. The scripts, schemas, and SPARQL queries of this experiment can be found in the GitHub repository of the paper [37].

5.1. Gene Wiki Evolution

The Gene Wiki Project [43] focuses on populating and maintaining Wikidata as a central hub of linked knowledge on genes, proteins, diseases, drugs, and related Life Science items. This project is one of the most active WikiProjects in terms of human and bot contribution [28]. The project is initiated based on a class-level diagram of the Wikidata knowledge graph for biomedical entities, which specifies 17 main classes [44]. The Wikidata WikiProject has extended the classes into 24 item classes.

The Gene Wiki evolution experiment aims to (i) capture a subsetting schema where the participating classes have connectivity to each other, and (ii) show the change in the amount of data instances from the early years of Wikidata. WDSub is deployed to extract the Gene Wiki subsets containing instances of the 20 classes pictured in [43] UML class diagram. The steps are:

– Creating a ShEx schema that represents the data model depicted in [43]. The ShExC format of the defined shapes is in Appendix A;

[32] See Lines 407-481 of file 'item-Q418553-found.json' in [38] - accessed 10 June 2023

Table 6
The number of instances for each Gene Wiki class from 2015 to 2022 and the number of instances on the live Wikidata Query Service (queried on 22 December 2022).

Class	2015	2016	2017	2018	2019	2020	2021	2022	Wikidata
active site	0	0	132	132	132	132	132	132	132
anatomical structure	4	62	470	483	614	732	738	812	746
binding site	0	0	76	76	76	77	77	77	76
biological pathway	0	0	425	2,754	2,929	3,279	3,429	3,486	3,554
biological process	11	12	31,263	31,222	42,058	43,417	42,061	41,857	42,449
cellular component	1	1	4,017	4,081	4,239	4,298	4,137	4,139	4,211
chemical compound	19,144	21,128	156,718	157,018	157,685	1,050,488	1,201,719	1,245,041	1,249,719
chromosome	0	0	149	152	432	9,167	9,224	9,223	9,224
disease	124	931	9,578	9,926	11,439	13,197	5,395	5,607	5,698
gene	17	20	679,372	677,836	811,574	1,196,193	1,196,334	1,211,506	Timed-Out
medication	46	2,127	2,459	2,472	2,699	3,210	3,336	3,424	3,450
molecular function	0	0	9,413	9,801	11,258	11,226	10,940	10,898	11,246
pharmaceutical product	0	0	1,067	1,067	2,725	2,754	2,759	2,774	2,784
protein domain	2	3	9,581	8,847	9,348	10,770	11,274	11,709	11,736
protein family	0	212	20,912	20,632	22,240	22,170	23,277	24,204	24,266
protein	118	166	450,785	487,781	579,979	980,520	985,755	988,099	Timed-Out
sequence variant	0	0	1,411	918	774	724	695	686	686
supersecondary structure	0	0	687	687	688	688	694	696	696
symptom	16	235	273	283	328	366	319	335	343
taxon	1,920,049	2,121,404	2,213,907	2,318,731	2,492,613	2,769,303	2,929,068	3,478,871	3,491,430

- Downloading the wikidata JSON dumps from 2015 to 2022 (exact dates are in Appendix B) which are available at Internet Archive[33];
- Deploying WDSub to create a subset from each dump.

A SPARQL query script is then run that counts the number of each item for each shape (class) and each link between shapes. Table 6 shows the number of instances for each class. The first attention-drawing point is the variation in the number of instances in different classes. The taxon, gene, protein and chemical compound classes have the highest number of items, such that more than 97% of the items in all the investigated dumps are instances of these four classes. Part of this heterogeneity is due to the nature of the abundance of classes. For example, the number of genes should be more than diseases, but it is not clear why in some classes the number of instances is so low, e.g., the number of anatomical structure instances seems less than expected. The number of instances in all classes except the biological process, cellular component, disease, molecular function, sequence variant, and symptom has increased continuously from 2015 to 2022. In addition to having the largest amount of data in all dumps, the data growth acceleration in the taxon, gene, protein, and chemical compound classes is also more than the other classes from 2015 and 2022. In all exceptional classes above, the peak point belongs to dump 2020. Then the number of instances decreases in 2021 and 2022, reaching the previous 2020 level in 2023, where the Wikidata SPARQL endpoint has been queried. The reason for this behaviour is not clear. It has been hypothesized that the number of instances was raised due to inaccurate bot activities in 2020, which was restored during human curations in the following two years and reached the same level again due to more accurate bots. Another observation is the low number of genes, proteins and chemical compound instances before 2017. The Gene Wiki WikiProject started and began populating data in 2015. These classes are the main focuses of the Gene Wiki community data population. It is found that the low number of instances in the 2015 and 2016 dumps is not due to the lack of A-Boxes, but due to the lack of *instance of (P31)* statements in the A-Boxes. Using *instance of (P31)* statements to specify the class of an item is a recent practice in Wikidata, thus, there was approximately the same number of the gene, protein, and

[33] https://www.wikidata.org/wiki/Wikidata:Database_download - accessed 14 February 2023

chemical compound instances in 2015 and 2016 on Wikidata identified by external identifiers such as Entrez Gene ID (P351), UniProt protein ID (P352), and InChI (P234), instead of *instance of (P31)* property.

The extracted subsets in this experiment can also be constructed by other practical tools of Section 3.2. The definition of these subsets in WDSub is based on writing a shape corresponding to each class containing the properties defined in the class diagram [43]. Such filters can be implemented by all other tools as well. In WDumper and WDF, one can simply write the corresponding filters based on the value of the parameters of the mentioned properties (the properties inside a Shape will be logical AND together). However, in some definitions, WDumper and WDF can not imitate the WDSub definition exactly. The reason for this is that in WDSub any number of relationships amongst shapes can be defined. For example, the `:active_site` class in Appendix A is related to the form `:protein_family` class via `wdt:P361` property. Now suppose the $\star$ operator in line 34 is replaced with a $+$. At extraction time, WDSub will not extract any active site instances that are not connected to at least one instance of a protein family. Unfortunately, such filtering and connections are not possible in WDumper and WDF. In these two tools, only one specific value can be defined for a property filter; it is not possible for the value to be of a specific class or related to other conditions (in WDumper, there is a possibility to define a condition saying a value should have existed, whatever that value is). KGTK can establish any relationship between conditions as its Kypher definition system is based on Cypher query language and has definition flexibility similar to ShEx. The extracted subsets can be found on Zenodo [45].

5.2. *Subsetting on Labels and Comments: Genes + Taxons*

Using the ShEx schema in Appendix C, a subset of Genes and Taxons instances from 2015 to 2022 is created, considering instances which have both labels and descriptions in English, Dutch, Farsi, and Spanish. Item instances that do not have a label or description in one of these four languages should not be extracted (aliases condition is considered with a $\star$ operator, which means that instances with zero aliases in the four languages can be in the subset).

Table 7 shows the number of instances separated by label, description, and alias languages along with the total number of extracted items. The difference between the number of labels, descriptions and aliases can also be seen. In general, English aliases are more than labels, which shows that on average each item has more than one English alias. By comparing between languages, it can be seen that the amount of labels, descriptions, and aliases in Farsi is lower than in other languages. This is more obvious in Genes compared to Taxons. In Spanish and Dutch, the number of labels and descriptions are close, which shows that wherever there is a label for this language, a description has also been added (note that labels and descriptions are usually added once for each language while aliases are more than one). While having fewer Farsi labels and aliases can be justified by the lack of proper translation, having fewer descriptions is due to the fewer Farsi-speaking participants (or their limited activity in Genes and Taxons). The low amount of data in Genes before 2017 which is explained in Section 5.1, can be seen here again. As the table shows, counting the number on Wikidata Query Service has been timed out in multiple taxon queries.

Subsetting on labels and comments can also be done by KGTK. KGTK and WDSub can define conditions even on the values of the label, e.g. define a shape (in KGTK a Kypher term) with a label condition the value specified to `"John Smith"` and extract all entities with the name John Smith from Wikidata. Filtering labels and comments is not possible with this flexibility in WDF and WDumper. In both WDF and WDumper, users can choose whether to skip labels and textual metadata (such as discriptions) along with the selected item. It is also possible to extract labels and comments in their specified languages (and not all languages). However, these options are considered post-filters, i.e., items are first selected based on property-based conditions, and then textual metadata can be ignored or kept on the selected items. Another limitation is that this option can be deployed either on all extracted items or non of them, e.g., it is not possible to extract a group of items with English labels and another group with Farsi labels. Initial selection based on language or value of a label/comment is not doable in WDF and WDumper. The extracted subsets can be found on Zenodo [46].

Table 7
The total and language seperated number of instances for Gene and Taxon class from 2015 to 2022 and the number of instances on the live Wikidata Query Service (queried on 14 February 2023).

Class	Casework	2015	2016	2017	2018	2019	2020	2021	2022	Wikidata
Gene	Total	17	20	679,372	677,836	811,574	1,196,193	1,196,334	1,211,506	1,215,324
	English Labels	16	18	679,365	677,827	811,567	1,196,185	1,196,326	1,211,497	1,215,314
	English Desc.	7	9	679,294	677,756	756,847	756,590	756,738	772,034	775,878
	English Aliases	2	15	1,954,528	1,843,927	1,810,033	1,945,779	1,947,441	1,975,129	1,980,192
	Spanish Labels	2	2	174,041	173,978	194,966	195,079	195,065	194,231	194,232
	Spanish Desc.	2	2	174,034	173,971	194,959	195,062	195,041	194,203	194,201
	Spanish Aliases	1	1	123	99	114	162	176	183	184
	Farsi Labels	1	0	130	132	528	814	872	917	1,033
	Farsi Desc.	0	0	37	37	38	58	60	65	67
	Farsi Aliases	0	0	22	21	21	21	21	24	21
	Dutch Labels	0	1	174,064	174,002	577,876	1,139,308	1,139,333	1,138,431	1,138,415
	Dutch Desc.	0	1	174,238	174,175	578,398	1,139,889	1,139,913	1,139,012	1,138,995
	Dutch Aliases	0	0	18	16	20	97	136	137	138
Taxon	Total	1,920,049	2,121,404	2,213,907	2,318,731	2,492,613	2,769,303	2,929,068	3,478,871	3,501,933
	English Labels	1,919,371	2,097,013	2,189,417	2,296,723	2,480,923	2,766,134	2,925,938	3,475,703	Timed-Out
	English Desc.	278,192	1,996,512	2,057,254	2,064,478	2,072,360	2,422,773	2,448,469	2,656,646	Timed-Out
	English Aliases	9,446	52,100	70,484	72,596	78,735	91,967	95,733	111,908	Timed-Out
	Spanish Labels	1,917,529	2,085,263	2,187,890	2,295,164	2,476,641	2,764,597	2,923,922	3,470,714	Timed-Out
	Spanish Desc.	18,846	24,991	770,220	1,610,043	1,622,695	1,625,360	1,626,266	1,628,861	Timed-Out
	Spanish Aliases	82,482	83,497	85,599	86,393	86,988	87,833	88,231	88,176	11,1641
	Farsi Labels	17,418	18,074	17,990	18,000	24,021	28,017	28,354	29,436	Timed-Out
	Farsi Desc.	169,462	167,849	166,932	166,880	166,773	167,075	166,900	167,226	Timed-Out
	Farsi Aliases	2,912	2,749	2,720	2,728	2,736	2,774	2,769	2,799	2,810
	Dutch Labels	926,956	2,089,454	2,191,695	2,297,575	2,478,838	2,766,321	2,926,153	3,474,191	Timed-Out
	Dutch Desc.	17,345	2,073,612	2,197,568	2,224,851	2,410,399	2,690,073	2,838,424	3,278,973	Timed-Out
	Dutch Aliases	29,744	31,425	32,467	33,471	34,289	35,977	36,739	37,767	38,151

5.3. Subsetting on References: Referenced Chemical Compounds

This section deploys references as filters and extract those chemical compound instances that their *instance of (P31)* fact has been referenced by a *reference URL (P854)*. Using WDSub, the scenario is to extract two different subsets according to the following schemas:

Schema 1 (referenced instances) This schema is designed to extract all *instances of (P31) chemical compounds (Q11173)* that have been referenced by at least one *reference URL (P854)*. Any chemical compound instances whose *instances of (P31)* fact have no reference using *reference URL (P854)* property should be excluded and not be in the subset. The schema can be seen in Appendix D.1.

Schema 2 (all instances) This schema extracts all *instances of (P31) chemical compounds (Q11173)*, no matter whether the *instances of (P31)* fact has been referenced or not. The schema can be seen in Appendix D.2.

The property *reference URL (P854)* provides primary external sources which are preferable provenance types according to Wikidata referencing policies. Choosing *instance of (P31)* statement is arbitrary. To observe the amount of such referenced statements, subsets from Wikidata dumps from 2015 to 2022 are extracted similar to Sections 5.1 and 5.2. To investigate whether extraction via Schema 1 includes only referenced instances, the following queries are performed on both Schema 1 and Schema 2 subsets:

Table 8
The number of referenced and not referenced chemical compound instances in Wikidata subsets from 2015 to 2022 and the Wikidata Query Service (queried on 14 February 2023).

chemical compound (Q11173)		2015	2016	2017	2018	2019	2020	2021	2022	Wikidata
Schema 1 (referenced instances)	**Query 1** (referenced instances)	1	0	26	27	25	18	16	16	Not Applicable
	Query 2 (all instances)	1	0	26	27	25	18	16	16	Not Applicable
Schema 2 (not referenced instances)	**Query 1** (referenced instances)	1	0	26	27	25	18	17	16	32
	Query 2 (all instances)	18,630	17,477	15,1970	15,1716	15,1506	1,036,696	1,187,186	1,109,165	1,251,822

Query 1 (referenced instances) Counts those chemical compound instances that their *instances of (P31)* statement has a *reference URL (P854)*[34].

Query 2 (all instances) Counts the number of chemical compound instances in general[35].

Table 8 shows the number of chemical compound instances obtained from performing the two queries on the 2015 to 2022 subsets. In the last column, it can be seen the number of referenced and not referenced chemical compound instances on Wikidata. As the results show, WDSub accurately excludes not referenced chemical compound instances in extraction. In all dumps, the number of referenced instances fetched by the referenced query (Query1) in the general subset (extracted using Schema 2) is equal to the total number of instances (fetched by the general query, Query2) in the referenced subset (extracted using Schema 1). The only inconsistency is in the column of dump 2021, where there are 17 referenced instances in the subset extracted by the general schema, while there are 16 instances in the subset extracted by the referenced schema. In other words, there is one referenced instance in the input dump which is not extracted by WDsub. However, this is not an unexpected missing item. The missed item is *nirmatrelvir (Q106405348)*, which has two separate *reference URL (P854)* values in its *instance of (P31)* statement. To extract shapes with more than one property, the ShEx schema requires the `EXTRA` qualifier to open the *reference URL (P854)* triple constraint. Thus, adding `EXTRA prov:wasDerivedFrom` to Line 11 of the Schema D.1 solves this inconsistency. Overall, the number of instances referenced by the *reference URL (P854)* property is low in all subsets and Wikidata. Subsetting based on references is not possible in KGTK, WDumper, and WDF. The extracted subsets can be found on Zenodo [47].

6. Conclusions

In this paper, the problem of subsetting in Wikidata was reviewed. As a continuously edited KG, Wikidata has a massive amount of data which cannot be queried from the SPARQL endpoint in all cases. Its weekly RDF and JSON dumps are maintained for a short period of time and hosting a Wikidata dump is costly. On the other hand, research and applications may need a specific scope of its data. Subsetting provides a platform to extract a dedicated part of the data from Wikidata, reducing the overall cost and facilitating the reproducibility of experiments.

The paper surveyed all available subsetting approaches over Wikidata and other KGs and explained their advantages and limitations. In the context of Wikidata, four subsetting approaches are distinguishable as practical subset-

[34] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/blob/0d54e50/flexibility-experiments/referenced-chemical-compounds/sparql/number_chemical_referenced.sparql - accessed 10 June 2023.

[35] https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/blob/0d54e50/flexibility-experiments/referenced-chemical-compounds/sparql/number_chemical_not_referenced.sparql - accessed 10 June 2023.

ting tools that can be deployed to extract a given defined subset: WDSub, WDumper, WDF, and KGTK. The performance, accuracy, and flexibility evaluations were then established over these four practical tools by defining several subsetting use cases. The results show that in terms of performance (i.e., the speed of extraction), WDF is the fastest tool and it can extract a subset in less than 4 hours. In terms of accuracy (i.e., extracting what is defined exactly, not more or less) the results show that WDF extracts all items and statements as exactly as they are present in the input dump. The ratio of missed items is less than 0.05% all tools missed less than %4 of items and that can be justified by the inconsistencies and syntax errors in the input dumps. In terms of flexibility (i.e., how much the tool allows the designer to define complex subsets on different parts of the Wikidata data model), three use cases have been defined and several subsets have been extracted from Wikidata dumps from 2015 to 2022. At first, a subsetting on different classes of the Gene Wiki WikiProject was performed and all tools supported such a subsetting. Then the subsets of genes and taxons were extracted based on having English, Spanish, Farsi, and Dutch labels and comments, which WDSub and KGTK supported such filtering. In the end, subsets of referenced chemical compounds were extracted and only WDSub was able to perform filters on references. The flexibility tests show that the most flexible tool for subsetting is WDSub, mainly because of its defining language which is ShEx and has the flexibility of SPARQL queries. During the subsetting, valuable information was gained about the amount of data in Wikidata from 2015 to 2022.

Subsetting faces many open questions. The first open question is subsetting other KGs, such as DBPedia, where the vocabulary is different and the dumps are not in JSON. There are also many massive collections of data supporting RDF, such as Uniprot and PubChem that can be the subject of subsetting. Another future work is to have flexibility and performance with one tool. WDSub is the most flexible tool but when you have flexibility, your filters take a longer time to be applied on the input dump items. SparkWDSub [48] is an under-development subsetting tool for Wikidata based on WDSub, which implements graph traversal for subset creation. To improve the speed, SparkWDSub uses the Apache Spark platform to distribute the computation. This tool is in the initial stages of development. Live subsets are the other future path. In this study (as well as in other related projects) several topical subsets have been extracted for which reusability is one of the main features. Over time with the new edits coming, the gap between these subsets and the corresponding data in Wikidata will increase. This gap can be reduced by repeating the subsetting process regularly, and by reducing the interval to an acceptable level (e.g., one day), end users can reach practically live subsets. A better solution is not to spend the extraction time for each repetition, instead, to generate the subset and apply the edits in real-time by establishing an active link between the Wikidata database and the subset. The main challenge in this task is hosting issues and the fact that Wikidata does not have a public API for establishing active links to the best of our knowledge. Subsetting also suffers from not having proper documentation. There is an essential need to aggregate and document all subsetting definition efforts as a training wiki, in which users can learn and define desired subsets effectively in a reasonable time.

Acknowledgement. This paper has progressed in several hackathons and tutorials of the ELIXIR BioHackathon-Europe series and SWAT4HCLS, and we would like to thank the organizers and participants. Suggestions and intellectual contributions of Dan Brickley, Lydia Pintscher, Eric Prud'hommeaux, Thad Guidry, and Filip Ilievski are greatly appreciated. This project has benefited from part of the following research grants: project PID2020-117912RB, ANGLIRU: Applying kNowledge Graphs to research data interoperabiLIty and ReUsability. The Alfred P. Sloan Foundation under grant number G-2021-17106 for the development of Scholia. The project R01GM089820 from the National Institutes of General Medical Sciences.

References

[1] D. Vrandečić and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Communications of the ACM* **57**(10) (2014), 78–85. doi:10.1145/2629489.

[2] A. Waagmeester, G. Stupp, S. Burgstaller-Muehlbacher et al., Wikidata as a knowledge graph for the life sciences, *eLife* **9** (2020), e52614, Publisher: eLife Sciences Publications, Ltd. doi:10.7554/eLife.52614.

[3] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg et al., The FAIR Guiding Principles for scientific data management and stewardship, *Scientific Data* **3**(1) (2016), 160018, Number: 1 Publisher: Nature Publishing Group. doi:10.1038/sdata.2016.18.

[4] L. Koesten, P. Vougiouklis, E. Simperl and P. Groth, Dataset Reuse: Toward Translating Principles to Practice, *Patterns* **1**(8) (2020), 100–136. doi:10.1016/j.patter.2020.100136. https://www.sciencedirect.com/science/article/pii/S2666389920301847.

[5] M.A. Rodriguez, The Gremlin graph traversal machine and language (invited talk), in: *Proceedings of the 15th Symposium on Database Programming Languages*, DBPL 2015, Association for Computing Machinery, New York, NY, USA, 2015, pp. 1–10. ISBN 978-1-4503-3902-5. doi:10.1145/2815072.2815073.

[6] S.A.H. Beghaeiraveri, A.J.G. Gray and F.J. McNeill, Experiences of Using WDumper to Create Topical Subsets from Wikidata, in: *CEUR Workshop Proceedings*, Vol. 2873, CEUR-WS, 2021, p. 13, ISSN: 1613-0073. https://researchportal.hw.ac.uk/files/45184682/paper13.pdf.

[7] J.E. Labra-Gayo, Creating Knowledge Graphs Subsets using Shape Expressions, *arXiv:2110.11709 [cs]* (2021), arXiv: 2110.11709. http://arxiv.org/abs/2110.11709.

[8] M. Cutcher, M. Personick and B. Thompson, The Bigdata® RDF Graph Database, in: *Linked Data Management*, Chapman and Hall/CRC, 2014, Num Pages: 46. ISBN 978-0-429-10245-5.

[9] Rhizome, Rhizome Artbase, 2021. https://artbase.rhizome.org/wiki/Main_Page.

[10] FactGrid, FactGrid, 2022. https://database.factgrid.de/wiki/Main_Page.

[11] D. Diefenbach, M.D. Wilde and S. Alipio, Wikibase as an Infrastructure for Knowledge Graphs: The EU Knowledge Graph, in: *The Semantic Web – ISWC 2021*, A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni and H. Alani, eds, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2021, pp. 631–647. ISBN 978-3-030-88361-4. doi:10.1007/978-3-030-88361-4_37.

[12] Wikimedia, Wikibase/Indexing/RDF Dump Format - MediaWiki, 2022. https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format.

[13] S. Lampa, E. Willighagen, P. Kohonen, A. King, D. Vrandečić, R. Grafström and O. Spjuth, RDFIO: extending Semantic MediaWiki for interoperable biomedical data management, *Journal of Biomedical Semantics* **8**(1) (2017), 35. doi:10.1186/s13326-017-0136-y.

[14] A. Waagmeester et al., Wikidata:WikiProject Schemas/Subsetting - Wikidata, 2019, https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas/Subsetting - accessed 31 December 2020. https://www.wikidata.org/wiki/Wikidata:WikiProject_Schemas/Subsetting.

[15] J.E. Labra-Gayo, A. González-Hevia, D. Fernández-Álvarez, A. Ammar, D. Brickley, A. Gray, E. Prud'hommeaux, D. Slenter, H. Solbrig, S.A.H. Beghaeiraveri, B. Fünkfstük, A. Waagmeester, E. Willighagen, L. Ovchinnikova, G. Benjaminsen, R. García-González, L.J. Garcia-Castro and D. Mietchen, Knowledge graphs and wikidata subsetting, Technical Report, 2021, Type: article. doi:10.37044/osf.io/wu9et.

[16] J.E. Labra-Gayo, A.C. González Cavazos, A. Waagmeester, N. Hofmann, S.A.H. Beghaeiraveri, E. Prud'hommeaux, S. Ul-Hasan, E. Willighagen and A. Ammar, Enhancement and Reusage of Biomedical Knowledge Graph Subset, Technical Report, 2022, Type: article. doi:10.37044/osf.io/n7qku.

[17] S. Matsumoto, R. Yamanaka and H. Chiba, Mapping RDF graphs to property graphs, *arXiv preprint arXiv:1812.01801* (2018).

[18] N. Mimouni, J.-C. Moissinac and A. Vu, Knowledge Base Completion With Analogical Inference on Context Graphs, in: *Semapro 2019*, 2019.

[19] N. Mimouni, J.-C. Moissinac and A. Tuan, Domain Specific Knowledge Graph Embedding for Analogical Link Discovery, *Advances in Intelligent Systems* (2020).

[20] D. Henselmann and A. Harth, Constructing demand-driven Wikidata Subsets., in: *Wikidata@ ISWC*, 2021.

[21] S. Aghaei, K. Angele and A. Fensel, Building Knowledge Subgraphs in Question Answering over Knowledge Graphs, in: *Web Engineering*, T. Di Noia, I.-Y. Ko, M. Schedl and C. Ardito, eds, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 237–251. ISBN 978-3-031-09917-5. doi:10.1007/978-3-031-09917-5_16.

[22] J.E. Labra-Gayo, E. Prud'Hommeaux, I. Boneva and D. Kontokostas, *Validating RDF data*, Vol. 7, Morgan & Claypool Publishers, 2017, pp. 1–328.

[23] E. Prud'hommeaux, shex.js, shexjs, 2022, original-date: 2015-08-06T07:18:07Z. https://github.com/shexjs/shex.js.

[24] H. Solbrig, Python implementation of ShEx 2.0, 2022, original-date: 2018-01-02T17:56:53Z. https://github.com/hsolbrig/PyShEx.

[25] B. Fünfstück, WDumper, 2019. https://github.com/bennofs/wdumper.

[26] Wikimedia, Wikidata:Database download, 2022. https://www.wikidata.org/wiki/Wikidata:Database_download.

[27] S.A.H. Beghaeiraveri, WDumper, 2021. https://github.com/seyedahbr/wdumper.

[28] S.A.H. Beghaeiraveri, A. Gray and F. McNeill, Reference Statistics in Wikidata Topical Subsets, in: *Proceedings of the 2nd Wikidata Workshop (Wikidata 2021)*, CEUR Workshop Proceedings, Vol. 2982, CEUR, Virtual Conference, October, 2021, ISSN: 1613-0073. https://researchportal.hw.ac.uk/files/53252708/Reference_Statistics_in_Wikidata_Topical_Subsets_corrected_version.pdf.

[29] S.A.H. Beghaeiraveri, Towards Automated Technologies in the Referencing Quality of Wikidata, in: *Companion Proceedings of The Web Conference 2022*, 2022. https://www2022.thewebconf.org/PaperFiles/8.pdf.

[30] J.E. Labra-Gayo, wdsub, Web Semantics Oviedo, University of Oviedo, 2022, original-date: 2021-07-05T09:27:56Z. https://github.com/weso/wdsub.

[31] J.E. Labra-Gayo, WShEx: A language to describe and validate Wikibase entities, in: *Proceedings of the 3rd Wikidata Workshop 2022 co-located with the 21st International Semantic Web Conference (ISWC2022)*, Vol. Vol-3262, 2022.

[32] F. Ilievski, D. Garijo, H. Chalupsky, N.T. Divvala, Y. Yao, C. Rogers, R. Li, J. Liu, A. Singh and D. Schwabe, KGTK: a toolkit for large knowledge graph manipulation and analysis, in: *International Semantic Web Conference*, Springer, 2020, pp. 278–293. https://arxiv.org/pdf/2006.00088.pdf.

[33] USC-ISI, KGTK: Knowledge Graph Toolkit, USC ISI I2, 2022, original-date: 2020-01-18T03:34:48Z. https://github.com/usc-isi-i2/kgtk.

[34] K. Shenoy, F. Ilievski, D. Garijo, D. Schwabe and P. Szekely, A Study of the Quality of Wikidata, *Journal of Web Semantics* **72** (2022), 100679, Publisher: Elsevier.

[35] F. Ilievski, P. Szekely and B. Zhang, Cskg: The commonsense knowledge graph, in: *European Semantic Web Conference*, Springer, 2021, pp. 680–696.

[36] maxlath, wikibase-dump-filter, 2022, original-date: 2016-04-27T22:18:04Z. https://github.com/maxlath/wikibase-dump-filter.

[37] kg-subsetting, kg-subsetting/paper-wikidata-subsetting-2023, kg-subsetting, 2023. https://github.com/kg-subsetting/paper-wikidata-subsetting-2023/releases/tag/v2.0.0.
[38] S.A.H. Beghaeiraveri, J.E. Labra-Gayo and A. Waagmeester, Wikidata Subsetting: Performance and Accuracy Experiment Datasets, Zenodo, 2023, https://doi.org/10.5281/zenodo.8015611. doi:10.5281/zenodo.8015611.
[39] Wikimedia, Wikidata json.gz Full Dump (3 Jan 2022), 2022. https://academictorrents.com/details/229cfeb2331ad43d4706efd435f6d78f40a3c438.
[40] Wikimedia, Wikidata:Database download, 2022. https://dumps.wikimedia.org/wikidatawiki/entities/.
[41] H. Chalupsky, P. Szekely, F. Ilievski, D. Garijo and K. Shenoy, Creating and Querying Personalized Versions of Wikidata on a Laptop (2021). http://arxiv.org/abs/2108.07119.
[42] L. Pintscher, Wikidata EntitySchemas Telegram Group, 2022, Message: https://t.me/c/1540810474/327. https://t.me/joinchat/ZeRz5wPDxpNkZGVk.
[43] Wikimedia, Wikidata:WikiProject Gene Wiki, 2020. https://www.wikidata.org/wiki/Wikidata:WikiProject_Gene_Wiki.
[44] S. Burgstaller-Muehlbacher, A. Waagmeester, E. Mitraka, J. Turner, T. Putman, J. Leong, C. Naik, P. Pavlidis, L. Schriml, B.M. Good and A.I. Su, Wikidata as a semantic framework for the Gene Wiki initiative, *Database (Oxford)* **2016** (2016). doi:10.1093/database/baw015.
[45] J.E. Labra-Gayo, S.A.H. Beghaeiraveri and A. Waagmeester, Generated Wikidata Subset for Gene Wiki Evolution, Zenodo, 2023, URLs: Dump 2015: https://zenodo.org/record/7869017, Dump 2016: https://zenodo.org/record/7883958, Dump 2017: https://zenodo.org/record/7872555, Dump 2018: https://zenodo.org/record/7872054, Dump 2019: https://zenodo.org/record/7871988, Dump 2020: https://zenodo.org/record/7871627, Dump 2021: https://zenodo.org/record/7870223, Dump 2022: https://zenodo.org/record/7869110.
[46] J.E. Labra-Gayo, S.A.H. Beghaeiraveri and A. Waagmeester, Generated Wikidata Subset for Genes + Taxons, Zenodo, 2023, URLS: Dump 2015: https://zenodo.org/record/7884057, Dump 2016: https://zenodo.org/record/7884081, Dump 2017: https://zenodo.org/record/7884116, Dump 2018: https://zenodo.org/record/7884297, Dump 2019: https://zenodo.org/record/7884316, Dump 2020: https://zenodo.org/record/7884424, Dump 2021: https://zenodo.org/record/7943929#.ZGSKw3bP2Uk, Dump 2022: https://zenodo.org/record/7944035#.ZGSTOXbP2Uk.
[47] S.A.H. Beghaeiraveri, J.E. Labra-Gayo and A. Waagmeester, Wikidata Subsetting: Reference-based Subsetting Experiment Datasets, Zenodo, 2023, https://doi.org/10.5281/zenodo.8015689. doi:10.5281/zenodo.8015689.
[48] J.E. Labra-Gayo, sparkwdsub, Web Semantics Oviedo, University of Oviedo, 2021, original-date: 2021-08-18T06:29:18Z. https://github.com/weso/sparkwdsub.

Appendix A. Gene Wiki ShEx

The ShExC shape expressions that is used to extract Gene Wiki subsets via WDSub is as follow:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX : <http://example.org/>

start= @:active_site OR
       @:anatomical_structure OR
       @:binding_site OR
       @:biological_pathway OR
       @:biological_process OR
       @:cellular_component OR
       @:chemical_compound OR
       @:chromosome OR
       @:disease OR
       @:gene OR
       @:mechanism_of_action OR
       @:medication OR
       @:molecular_function OR
       @:pharmaceutical_product OR
       @:pharmacologic_action OR
       @:protein_domain OR
```

```
        @:protein_family OR
        @:protein OR
        @:sequence_variant OR
        @:supersecondary_structure OR
        @:symptom OR
        @:taxon OR
        @:therapeutic_use

:active_site EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [ wd:Q423026 ]      ;
  wdt:P361 @:protein_family * ;
  wdt:P527 @:protein_family * ;
}

:anatomical_structure EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [ wd:Q4936952 ] ;
  wdt:P361 @:anatomical_structure * ;    # part of (P361)
  wdt:P527 @:anatomical_structure *      # has part(s) (P527)
}

:binding_site EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [ wd:Q616005 ] ;
  wdt:P361 @:protein_family * ;
  wdt:P527 @:protein_family * ;
}

:biological_pathway EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [ wd:Q4915012 ] ;
  wdt:P527 @:biological_pathway * ;
  wdt:P361 @:biological_pathway * ;
  wdt:P361 @:gene * ;
  wdt:P527 @:gene * ;
  wdt:P361 @:medication * ;
  wdt:P527 @:medication * ;
  wdt:P361 @:chemical_compound * ;
  wdt:P527 @:chemical_compound * ;
  wdt:P703  @:taxon * ;
  wdt:P1050 @:disease* ;
}

:biological_process EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [wd:Q2996394] ;
  wdt:P279 @:biological_process * ;      # subclass of (P279)
  wdt:P361 @:biological_process * ;      # part of (P361)
  wdt:P527 @:biological_process * ;      # has part(s) (P527)
  wdt:P128 @:biological_process * ;      # has part(s) (P527)
```

```
  wdt:P128 @:molecular_function * ;      # regulates (molecular biology) (P128)
  wdt:P361 @:medication * ;              # part of (P361)
  wdt:P527 @:medication * ;              # has part(s) (P527)
  wdt:P361 @:chemical_compound * ;       # part of (P361)
  wdt:P527 @:chemical_compound * ;       # has part(s) (P527)
  wdt:P279 @:biological_process *        # subclass of (P279)
}

:cellular_component EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [wd:Q5058355] ;
  wdt:P279 @:cellular_component * ;      # subclass of (P279)
  wdt:P361 @:cellular_component * ;      # part of (P361)
  wdt:P681 @:cellular_component * ;      # cell component (P681)
  wdt:P527 @:cellular_component * ;      # has part(s) (P527)
  }

:chemical_compound EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [ wd:Q11173 ] ;
  wdt:P3364 @:chemical_compound * ;
  wdt:P769 @:chemical_compound * ;
  wdt:P2868 @:pharmacologic_action * ;
  wdt:P769  @:pharmacologic_action * ;   # significant drug interaction (P769)
  wdt:P279  @:pharmacologic_action * ;   # subclass of (P279)
  wdt:P361  @:medication * ;             # part of (P361)
  wdt:P527  @:medication * ;             # has part(s) (P527)
  wdt:P2868 @:mechanism_of_action *;     # subject has role (P2868)
  wdt:P3489 @:disease * ;                # pregnancy category (P3489)
}

:chromosome EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [ wd:Q37748 ] ;
}

:disease EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31   [ wd:Q12136 ] ;
  wdt:P279  @:disease * ;
  wdt:P780  @:disease * ;                # symptoms and signs (P780)
  wdt:P828  @:taxon * ;                  # has cause (P828)
  wdt:P2293 @:gene * ;                   # genetic association (P2293)
  wdt:P927  @:anatomical_structure * ;   # anatomical location (P927)
  wdt:P2176 @:medication * ;             # drug or therapy used for treatment (P2176)
  wdt:P2176 @:chemical_compound * ;      # drug or therapy used for treatment (P2176)
  wdt:P2176 @:therapeutic_use * ;        # drug or therapy used for treatment (P2176)
  wdt:P2175 @:medication * ;             # medical condition treated (P2175)
  wdt:P2175 @:chemical_compound * ;      # medical condition treated (P2175)
  wdt:P2175 @:therapeutic_use * ;        # medical condition treated (P2175)
}
```

```
:gene EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31    [ wd:Q7187 ] ;
  wdt:P684   @:gene * ;                  # ortholog (P684)
  wdt:P2293  @:disease * ;               # genetic association (P2293)
  wdt:P703   @:taxon * ;                 # found in taxon (P703)
  wdt:P1057  @:chromosome * ;            # chromosome (P1057)
  wdt:P682   @:biological_process * ;    # biological process (P682)
  wdt:P688   @:protein * ;               # encodes (P688)
}

:mechanism_of_action EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [ wd:Q3271540 ] ;
}

:medication EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [ wd:Q12140 ] ;
  wdt:P2175 @:disease * ;                # medical condition treated (P2175)
  wdt:P3780 @:pharmaceutical_product * ;# active ingredient in (P3780)
  wdt:P769  @:pharmacologic_action * ;   # significant drug interaction (P769)
  wdt:P769  @:chemical_compound * ;      # significant drug interaction (P769)
  wdt:P769  @:therapeutic_use * ;        # significant drug interaction (P769)
  wdt:P2868 @:pharmacologic_action * ;   # subject has role (P2868)
  wdt:P2868 @:therapeutic_use * ;        # subject has role (P2868)
  wdt:P279  @:pharmacologic_action * ;   # subclass of (P279)
  wdt:P279  @:therapeutic_use * ;        # subclass of (P279)
  wdt:P2868 @:mechanism_of_action * ;    # subject has role (P2868)
  wdt:P2175 @:symptom *                  # medical condition treated (P2175)
}

:molecular_function EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31   [ wd:Q14860489 ] ;
  wdt:P361 @:molecular_function * ;
  wdt:P527 @:molecular_function * ;
  wdt:P279 @:molecular_function * ;
}

:pharmaceutical_product EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31   [ wd:Q28885102 ] ;
  wdt:P3781 @:therapeutic_use * ;        # has active ingredient (P3781)
  wdt:P3781 @:pharmacologic_action * ;   # has active ingredient (P3781)
  wdt:P3781 @:chemical_compound * ;      # has active ingredient (P3781)
  wdt:P3781 @:medication * ;             # has active ingredient (P3781)
  wdt:P3780 @:therapeutic_use * ;        # active ingredient in (P3780)
  wdt:P3780 @:pharmacologic_action * ;   # active ingredient in (P3780)
  wdt:P3780 @:chemical_compound * ;      # active ingredient in (P3780)
```

```
  wdt:P3780 @:medication * ;              # active ingredient in (P3780)
  wdt:P4044 @:disease *;                  # therapeutic area (P4044)
}

:pharmacologic_action EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [wd:Q50377224 ] ;
  wdt:P3780 @:pharmaceutical_product * ;# active ingredient in (P3780)
  wdt:P3781 @:pharmaceutical_product * ;# has active ingredient (P3781)
  wdt:P2175 @:disease * ;                 # medical condition treated (P2175)
  wdt:P2176 @:disease * ;                 # drug or therapy used for treatment (P2176)
}

:protein_domain EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [ wd:Q898273 ] ;
  wdt:P279 @:protein_domain * ;           # subclass of (P279)
  wdt:P128 @:protein_domain * ;           # regulates (molecular biology) (P128)
  wdt:P527 @:protein_domain * ;           # has part(s) (P527)
  wdt:P361 @:protein_domain * ;           # part of (P361)
}

:protein_family EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [ wd:Q417841 ] ;
  wdt:P527 @:protein * ;                  # has part(s) (P527)
  wdt:P279 @:protein_family* ;            # subclass of (P279)
  wdt:P527 @:protein * ;                  # part of (P361)
}

:protein EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31  [ wd:Q8054 ] ;
  wdt:P129 @:protein  * ;                 # physically interacts with (P129)
  wdt:P681 @:protein * ;                  # cell component (P681)
  wdt:P129 @:medication * ;               # physically interacts with (P129)
  wdt:P680 @:molecular_function * ;       # molecular function (P680)
  wdt:P681 @:cellular_component * ;       # cell component (P681)
  wdt:P681 @:anatomical_structure * ;     # cell component (P681)
  wdt:P682 @:biological_process * ;       # biological process (P682)
  wdt:P527 @:active_site * ;              # has part(s) (P527)
  wdt:P361 @:active_site * ;              # part of (P361)
  wdt:P527 @:protein_domain * ;           # has part(s) (P527)
  wdt:P361 @:protein_domain * ;           # part of (P361)
  wdt:P361 @:protein_family * ;           # part of (P361)
  wdt:P527 @:protein_family * ;           # has part(s) (P527)
  wdt:P527 @:active_site * ;
  wdt:P361 @:active_site * ;
  wdt:P361 @:binding_site * ;
  wdt:P527 @:binding_site * ;
  wdt:P129 @:chemical_compound  * ;       # physically interacts with (P129)
```

```
  wdt:P129 @:medication  * ;              # physically interacts with (P129)
  wdt:P702 @:gene * ;                     # encoded by (P702)
  wdt:P703 @:taxon * ;                    # found in taxon (P703)
}

:sequence_variant EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31    [ wd:Q15304597 ] ;
  wdt:P3433 @:gene * ;                    # sequence variant (Q15304597)
  wdt:P3355  @:chemical_compound * ;      # negative therapeutic predictor for (P3355)
  wdt:P3354 @:chemical_compound * ;       # positive therapeutic predictor for (P3354)
  wdt:P3354 @:medication * ;
  wdt:P3355 @:medication * ;
  wdt:P1057 @:chromosome * ;              # chromosome (P1057)
}

:supersecondary_structure EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [ wd:Q7644128 ] ;
  wdt:P361 @:protein * ;
  wdt:P361 @:protein_family * ;
  wdt:P361 @:protein_domain * ;
}

:symptom EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [ wd:Q169872 ] ;
  wdt:P2176 @:chemical_compound * ;   # drug or therapy used for treatment (P2176)
}

:taxon EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31 [ wd:Q16521 ] ;
}

:therapeutic_use EXTRA wdt:P31 {
  rdfs:label [ @en ] ;
  wdt:P31    [ wd:Q50379781 ] ;
}
```

Appendix B. Wikidata Dumps Dates

Appendix C. Genes + Taxons labeling and commenting ShEx

The ShExC shape expression that is used to extract Genes and Taxons subsets via WDSub based on labels, descriptions, and aliases in four languages is as follow:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>
```

Table 9
The exact dates, size, and download URL of Wikidata dumps used in the flexibility experiments.

Dump	Exact Date	Size	Download URL
2015	2015-06-01	4.5 Gb	https://archive.org/download/wikidata-json-20150601/wikidata-20150601-all.json.gz
2016	2016-06-13	7.19 Gb	https://archive.org/download/wikidata-json-20160613/wikidata-20160613-all.json.gz
2017	2017-08-21	15.7 Gb	https://archive.org/download/wikibase-wikidatawiki-20170821/wikidata-20170821-all.json.gz
2018	2018-01-15	26.48 Gb	https://archive.org/download/wikibase-wikidatawiki-20180319/wikidata-20180319-all.json.gz
2019	2019-01-21	48.14 Gb	https://archive.org/download/wikibase-wikidatawiki-20190121/wikidata-20190121-all.json.gz
2020	2020-11-02	83.94 Gb	https://archive.org/download/wikibase-wikidatawiki-20201102/wikidata-20201102-all.json.gz
2021	2021-05-31	93.93 Gb	https://archive.org/download/wikibase-wikidatawiki-20210531/wikidata-20210531-all.json.gz
2022	2022-06-30	107.66 Gb	https://archive.org/download/wikidata-20220630-all.json.gz/wikidata-20220630-all.json.gz

```
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX : <http://example.org/>
PREFIX schema: <http://schema.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

start= @:gene OR
       @:taxon

:gene EXTRA wdt:P31 {
  rdfs:label [ @en @es @fa @nl ] ;
  schema:description [ @en @es @fa @nl ] ;
  skos:altLabel [ @en @es @fa @nl ] * ;
  wdt:P31 [ wd:Q7187 ] ;
  wdt:P703 @:taxon * ;
}

:taxon EXTRA wdt:P31 {
  rdfs:label[ @en @es @fa @nl ] ;
  schema:description [ @en @es @fa @nl ] ;
  skos:altLabel [ @en @es @fa @nl ] * ;
  wdt:P31 [ wd:Q16521 ] ;
}
```

Appendix D. Referenced Chemicals ShExes

D.1. Schema 1 (referenced instances)

This schema extract those instances of chemical compounds that their *instances of (P31)* fact has been referenced by at least one *reference URL (P854)*:

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX pr: <http://www.wikidata.org/prop/reference/>


start = @<chemical_compound>
```

```
<chemical_compound> {
  wdt:P31 [ wd:Q11173 ] ;
  p:P31 {
    ps:P31 [ wd:Q11173 ] ; # is instance of (P31) chemical_compound (Q11173)
    prov:wasDerivedFrom @<reference>  # has a reference
  }
}
<reference>{
  pr:P854 .
}
```

D.2. Schema 2 (not referenced instances)

This schema extracts all instances of chemical compounds.

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX pr: <http://www.wikidata.org/prop/reference/>
PREFIX prov: <http://www.w3.org/ns/prov#>


start = @<chemical_compound>

<chemical_compound> {
  wdt:P31 [ wd:Q11173 ] + ; # is instance of (P31) chemical_compound (Q11173)
}
```