Distributed and Parallel Databases
https://doi.org/10.1007/s10619-022-07414-w

Bio-SODA UX: enabling natural language question answering over knowledge graphs with user disambiguation

Ana Claudia Sima[1] · Tarcisio Mendes de Farias[1,2,3] · Maria Anisimova[1,4] · Christophe Dessimoz[1,2,5,6] · Marc Robinson-Rechavi[1,3] · Erich Zbinden[1,4] · Kurt Stockinger[4]

Accepted: 21 June 2022

Abstract

The problem of natural language processing over structured data has become a growing research field, both within the relational database and the Semantic Web community, with significant efforts involved in question answering over knowledge graphs (KGQA). However, many of these approaches are either specifically targeted at *open-domain* question answering using DBpedia, or require *large training datasets* to translate a natural language question to SPARQL in order to query the knowledge graph. Hence, these approaches often cannot be applied directly to complex *scientific datasets* where no prior training data is available. In this paper, we focus on the challenges of natural language processing over knowledge graphs of scientific datasets. In particular, we introduce Bio-SODA, a natural language processing engine that does not require training data in the form of question-answer pairs for generating SPARQL queries. Bio-SODA uses a generic graph-based approach for translating user questions to a ranked list of SPARQL candidate queries. Furthermore, Bio-SODA uses a novel ranking algorithm that includes node centrality as a measure of relevance for selecting the best SPARQL candidate query. Our experiments with real-world datasets across several scientific domains, including the official *bioinformatics* Question Answering over Linked Data (QALD) challenge, as well as the CORDIS dataset of European projects, show that Bio-SODA outperforms publicly available KGQA systems by an F1-score of least 20% and by an even higher factor on more complex bioinformatics datasets. Finally, we introduce Bio-SODA UX, a graphical user interface designed to assist users in the exploration of large knowledge graphs and in dynamically disambiguating natural language questions that target the data available in these graphs.

Keywords Question answering · Knowledge graphs · Ranking

✉ Ana Claudia Sima
ana-claudia.sima@sib.swiss

Extended author information available on the last page of the article

Published online: 16 July 2022

1 Introduction

The problem of natural language processing over structured data has gained significant traction, both in the Semantic Web community—with a focus on answering natural language questions over RDF graph databases [1–3]—and in the relational database community, where the goal is to answer questions by finding their semantically equivalent translations to SQL [4–7]. Significant research efforts have been invested in particular in *open-domain* question answering over knowledge graphs. These efforts often use the DBpedia and/or Wikidata knowledge bases, that are composed of structured content from various Wikimedia projects such as Wikipedia. A growing ecosystem of tools is therefore becoming available for solving subtasks of the KGQA problem, such as entity linking [8–11] or query generation [12]. However, most of these tools are specifically targeted at question answering over DBpedia [13], not having been applied to other contexts, such as for scientific datasets.

On the one hand, encouraged by the recent success of machine learning methods, several new benchmarks for training and evaluating KGQA systems have been published [14, 15]. On the other hand, most of the existing datasets are synthetic (i.e., not based on real query logs) and generally limited to DBpedia or Wikidata, which may not be representative of knowledge graphs for scientific datasets.

For example, one of the major question answering datasets over DBpedia, LC-Quad [14], as well as its updated version, LC-Quad 2.0 [15], include only *simple multi-fact questions* that connect at most two facts. In other words, these queries cover at most two or three triple patterns, with a query graph spanning a maximum of two hops, whereas real-world questions tend to be much more complex. In particular, a study of SPARQL query logs [16] across multiple knowledge graphs, including DBpedia, has shown that a significant fraction of real-world queries have 10 triple patterns or more. It therefore remains unclear whether existing training sets can serve as representative for real-world natural language processing engines over knowledge graphs in general. All in all, data access and retrieval remain challenging for domain experts who are not familiar with structured query languages, nor with the data models of each scientific dataset that they use.

To illustrate the general problem of natural language processing over knowledge graphs, consider the simple data model in Fig. 1. Here we see that a drug could be a *possible disease target* for asthma (left branch), as well as potentially having *side effects* such as triggering asthma symptoms (right branch). Now consider the following natural language question: *"Which drugs are used for asthma?"*. Note that our knowledge graph has no concept or property called *used for*. Hence, this question cannot be easily translated without relying on external knowledge (*e.g.* training data), given that *used for* cannot be directly mapped to either of the two properties (*possibleDiseaseTarget* or *sideEffect*) shown in the figure. However, node centrality metrics, such as the PageRank score of nodes in the knowledge graph, can help capture "common sense" knowledge, *e.g.*, that *asthma* is more commonly a *Disease*, rather than a *Side Effect*.

As a step towards bridging the current gap in natural language processing for knowledge graphs of scientific datasets, we introduce Bio-SODA, a system

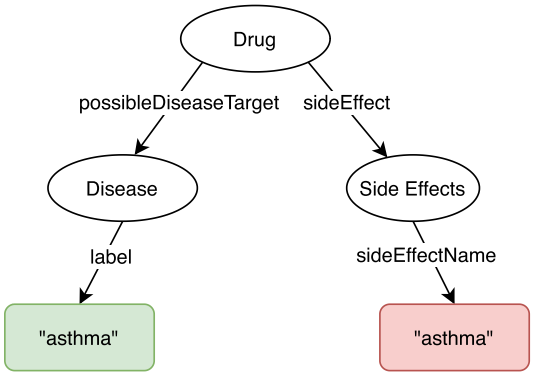

Fig. 1 Illustrative data model, simplified from the QALD4 benchmark datasets [17]. Consider the following question: *"Which drugs are used for asthma?"*. In the QALD4 dataset, "asthma" appears as both a disease instance (shown in green), as well as a side effect (shown in red). The second interpretation describes drugs that can *trigger* asthma symptoms. Therefore, it is the opposite of the user's intended question. However, the predicate *used for* in the question cannot be easily linked to either of the properties indicated through arrows in the image. Due to ambiguity, the question is difficult to translate correctly in the absence of external knowledge, without relying on training data (inferring that *used for* implies drug targeting disease) (Color figure online)

designed to answer *natural language questions* across knowledge graphs where *no prior training data* is available. Bio-SODA relies on a generic graph-based approach in order to translate natural language questions into SPARQL queries. Furthermore, Bio-SODA is designed to *compensate for incompleteness in the data*—either due to missing schema information or, to some extent, due to missing labels. Although these situations should not occur when following ontology engineering best practices for representing data in RDF, our experience in working with real-world datasets shows that these problems are frequent in practice.

We make our prototype implementation available open-source[1]. We also make available a live demo of Bio-SODA online[2], where each of the datasets considered in this paper can be queried. The prototype enables both keyword search, as well as full question answering in English. We chose bioinformatics as our *primary* target domain, motivated by the rapid growth of publicly available RDF data in this scientific domain. Specifically, around 8% of the Linked Open Data Cloud originates from the Life Sciences [18]. For the purpose of evaluating our system, we use several real-world datasets stemming from different domains. For example, we use the last bioinformatics question answering challenge released as part of the official Question Answering on Large Databases (QALD) series, namely the QALD4 biomedical task [17]. Importantly, to-date there is no sufficiently large training dataset of questions and corresponding SPARQL queries to enable the use of machine learning approaches for end-to-end Question Answering in the biomedical field.

[1] Code at https://github.com/anazhaw/Bio-SODA
[2] See demo at http://biosoda.expasy.org/

Finally, to demonstrate the generalizability of Bio-SODA to other domains, we also apply our system to an entirely different context, outside bioinformatics, namely on the CORDIS dataset describing European Union (EU) funded research projects[3]. This dataset is also used in the EU-project INODE (Intelligent Open Data Exploration) [19].

This paper, which is an extended version of [20], makes the following **contributions**:

- We introduce Bio-SODA—a novel natural language processing engine over knowledge graphs that *does not require prior training data* (question-answer pairs) for translating natural language questions into SPARQL.
- We define a novel ranking algorithm for selecting the best automatically generated SPARQL statements in response to a given natural language question. The ranking algorithm combines *syntactic and semantic similarity*, as well as *node centrality* in the knowledge graph. Many existing question answering systems either rely on simple metrics for ranking, such as the length of the answer query graph [6], or require extensive training data in order to learn a ranking function [21]. To the best of our knowledge, our approach is the first to take into account all three factors (syntactic and semantic similarity, as well as node centrality) for ranking queries.
- Our experiments on various real-world datasets show that Bio-SODA outperforms state-of-the-art KGQA systems by 20% on the F1-score using the official QALD4 biomedical benchmark and by an even higher factor on the more complex bioinformatics dataset.
- Finally, in addition to the work presented in the conference version of this paper (see [20]), here we introduce Bio-SODA UX, a prototype graphical user interface enabling users to interact with knowledge graph data and disambiguate natural language questions over the data dynamically; we demonstrate through selected use cases how the interface can assist users in exploring the available data and in finding the information of interest from the underlying knowledge graph.

The paper is structured as follows: Sect. 2 places our contribution in the context of the related work. In Sect. 3 we introduce some of the challenges of natural language processing over RDF-based knowledge graphs. In Sect. 4 we explain the high level architecture of Bio-SODA through a concrete example from the biomedical domain. In Sect. 5 we present the detailed system architecture of Bio-SODA followed by a description of the Bio-SODA UX user interface in Sect. 6. Next, we describe the datasets used for evaluation, their specific challenges and the results obtained, in Sect. 7. In Sect. 8 we discuss lessons learned from building a natural language processing system for real-world domain datasets. We outline directions for future work in Sect. 9.

[3] https://cordis.europa.eu/projects

2 Related work

The problem of natural language processing and question answering over structured data has been well-studied in recent years, with a growing number of published systems, particularly in open-domain question answering. Recent surveys on natural language interfaces to databases include [22, 23]. However, in this paper we focus on natural language interfaces to *RDF graph databases or RDF-based knowledge graphs*. Natural language interfaces to relational databases are outside the scope of this paper.

In parallel, the biomedical field has seen a growth of dedicated systems for question answering. Examples include GFMed [24] and Pomelo [25] – the two highest ranked systems in the QALD4 biomedical challenge – as well as more recent systems [26]. However, these are generally considered expert systems, with lower generalizability to other domains, given that they extensively rely on manually handcrafted rules and domain expertise.

Our work aims to bridge the gap between the two parallel efforts by solving the *common case* in a domain-independent manner. For this, Bio-SODA relies on a generic graph-based approach in order to generate a ranked list of candidate SPARQL queries from a given question. We enable the addition of custom rules only for *special cases* when needed.

Many recent KGQA systems [1, 3] have been evaluated using the LC-Quad benchmark of 5000 questions over DBpedia [14]. Although this benchmark is an important step forward, particularly for enabling machine learning approaches, it does not include complex multi-hop questions, which makes it unclear how the results would generalize to this case. For example, the current publicly available implementation of the SPARQL query generation system SQG [12], would not work for complex question answering on a new knowledge graph without significant changes to the code base, as it targets question answering over DBpedia and more specifically in the format required by the LC-Quad benchmark.

More recent KGQA systems, such as [3, 27], support multiple knowledge graphs, but are limited to queries with a complexity of at most three triple patterns. Similarly, existing end-to-end QA systems, based on machine learning approaches, such as [28], can only handle simple questions. These approaches have the added drawback that they only generate a single answer, as opposed to multiple candidates. Furthermore, end-to-end approaches suffer from the lack of explainability, which makes it challenging for users to validate the correctness of the result. Explainability in this context has therefore become an active area of research, with solutions proposed including translating back structured queries into natural language sentences [29–31] or summarizing the entities in the results [32].

Disambiguation is one of the major tasks of question answering systems. One possible solution for this is to limit the interface to a controlled natural language and involve the user in constructing questions from the available building blocks. Sparklis [33] is a query building system that enables answering controlled natural language questions over knowledge graphs out-of-the-box. However, this process

is manual and therefore time-consuming, which makes it less convenient than a true natural language interface.

One of the systems closest to ours is the KG-agnostic WDAqua-core1 [1]. The system supports multiple knowledge bases in several languages. However, the system is only available as a demo. Although the authors mention that node relevance can in principle be taken into account for ranking, it is not clear whether the approach was used in the evaluation or whether the ranking function was learned based on training data. An updated version of this question answering system, QAnswer, is presented in [34], however this system is also limited to support at most 3 triple-pattern queries.

3 Challenges of natural language processing over knowledge graphs

In this section we summarise some of the challenges of natural language processing over knowledge graphs, focusing on scientific knowledge graphs, which shape the architecture of the Bio-SODA system (described in Sects. 4 and 5).

- **Lack of training data.**
  For many scientific knowledge graphs there is no sufficiently long and diverse log of questions and their corresponding queries in order to derive a representative training set for a machine learning-based solution. So far, existing training corpora have proven costly to construct [14], with the added drawback that any semi-automatically generated dataset risks compiling a set of question-answer pairs that are non-representative for the information needs of real users of the KGQA system, *e.g.* domain experts.
- **Rule-based approaches perform well, but are costly to build and maintain.**
  So far, state-of-the-art solutions for question answering over generic RDF-based knowledge graphs have been mostly rule-based systems, relying on manually handcrafted rules. For example, GFMed [24] and Pomelo [25], the top 2 ranked systems in the QALD4 biomedical challenge, have achieved very good results in the challenge, but at the cost of very little generality. In essence, these systems suffer from significant overfitting: to be applicable to a new domain, their rule sets would need extensive or even complete rewriting. Moreover, even for a new dataset within the same domain, for which the schema differs, new rules need to be added in order to accommodate the differences.

  In some cases it is beneficial to incorporate a minimal set of rules in KGQA systems, particularly for deriving complex concepts. However, this should be a last resort and not the main translation mechanism, given that a large rule set is hard to maintain and scale.
- **Schema-less, incomplete data.**
  One of the strengths of relational databases is to have a database schema which enables strict data modelling and guarantees certain data integrity and data quality aspects. However, since RDF does not strictly enforce a (database) schema, real-world datasets using RDF knowledge graphs often exhibit poor structure [35, 36]. Typical examples are properties with missing or generic domains and

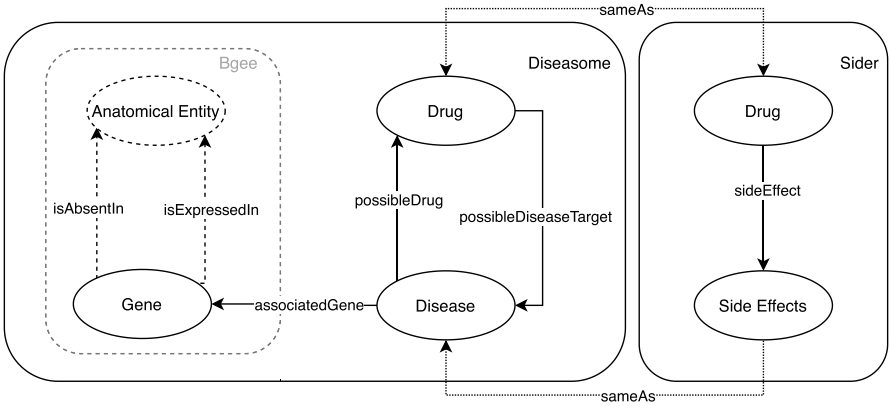

Fig. 2 Simplified data model based on the Bgee database and QALD4 [17] datasets. The data model is a multigraph, including disjoint properties – such as *isAbsentIn* and *isExpressedIn*, as well as inverse properties, such as *possibleDiseaseTarget* and *possibleDrug*. To make matters more complicated, a *Side Effect* and a *Disease* can be described by the same terms, with instances of the two classes being related via the *sameAs* property. As a result, even simple questions such as *"which drugs might lead to strokes?"* are hard to automatically translate correctly in the absence of external knowledge (*i.e.* "lead to" = "side effect")

ranges. In other words, a question answering system over RDF knowledge graphs typically does not have complete schema information. Hence, an important step when working with such incomplete knowledge graphs is to enrich the existing (incomplete) schema, for example, by inferring property ranges and domains based on instance-level data.

- **Disambiguation.**
  In many cases, different users have different expectations (query intents) when asking *the same* question. An example would be the question *What are all the Big Data projects?*, asked over the European Projects dataset. Possible interpretations of this request are either to retrieve all projects in a Big Data call, or all projects by institutions that have the term *Big Data* in their name or all projects whose title or abstract include the terms *Big Data* etc. The system should ultimately let users decide which interpretation was intended when asking the question, also informing them of the range of possible options, according to the available underlying data.

4 Bio-SODA: a high-level perspective

In this section we use a motivating example to illustrate the natural language processing pipeline of Bio-SODA.

Consider the data model illustrated in Fig. 2, which combines four different scientific databases. The database *Bgee* on the left contains information about genes and in which parts of the body (anatomical entity) a gene is expressed or absent. The database *Diseasome* in the middle contains information about diseases, as well as

drugs targeting each disease. In addition, the drugs are part of the pharmaceutical database *DrugBank* (not explicitly shown in the figure). Finally, the database *Sider* contains information about drugs and their side effects. Correspondences between equivalent drugs in Sider and DrugBank are made through the *sameAs* property.

Further assume that a domain expert is interested in answering the question: "*What are the drugs for diseases associated with the BRCA*[4] genes?".

The natural language processing pipeline of Bio-SODA for answering this question is illustrated in Fig. 3. In particular, the main steps involved in translating the natural language question to SPARQL are as follows: first, Bio-SODA matches question tokens, such as "drugs" and "diseases", against the data stored in the database, using an inverted index. This step is called *Lookup Candidate Match*. In this example, all tokens are of length one, *i.e.* composed of a single word. The inverted index enables retrieving not only the URI of each matching candidate, but also its *PageRank* score. An example is shown in parentheses for the first two tokens in the Figure. In addition, the inverted index retrieves the *class and property names* of the match (omitted in the figure for simplicity). For example, the lookup for "*BRCA*" retrieves instances of the class *Diseasome:Genes*, where the *rdfs:label* property matches the user token ("*BRCA1*", "*BRCA2*"). A few simplified Inverted Index entries are provided in Table 1.

In the *Ranking* step, candidates are grouped together according to class/property[5] and ranked according to string similarity and PageRank score.

In the *Query Graph Construction* step, all the ranked candidates are used to construct a query graph which represents one possible answer or interpretation of the natural language question. For simplicity, Fig. 3 only shows the query graph obtained for the top ranked candidate matches. However, Bio-SODA generates multiple alternative interpretations, for example, also including the interpretation considering *Sider:Drugs* instead of the *DrugBank:Drugs*. This can be tested in the demo page of Bio-SODA for QALD4.

Next, Bio-SODA generates the corresponding SPARQL query for each query graph. Finally, the results are returned by executing the query on the target knowledge graph (see bottom of Fig. 3).

5 Bio-SODA: system architecture

The main building blocks of the Bio-SODA system architecture, shown in Fig. 4, are the following:

- *Preprocessing Phase*: This phase includes building indexes for efficient lookup as well as automatically generating a schema graph, which will serve as the basis

[4] Note that, based on the biomedical literature, mutations in the two *BRCA* genes, BRCA1 and BRCA2 (stemming from *BReast CAncer*) are known to be associated with multiple types of cancer.

[5] a `FILTER` for the token *BRCA* is created on the *Diseasome:Genes* class

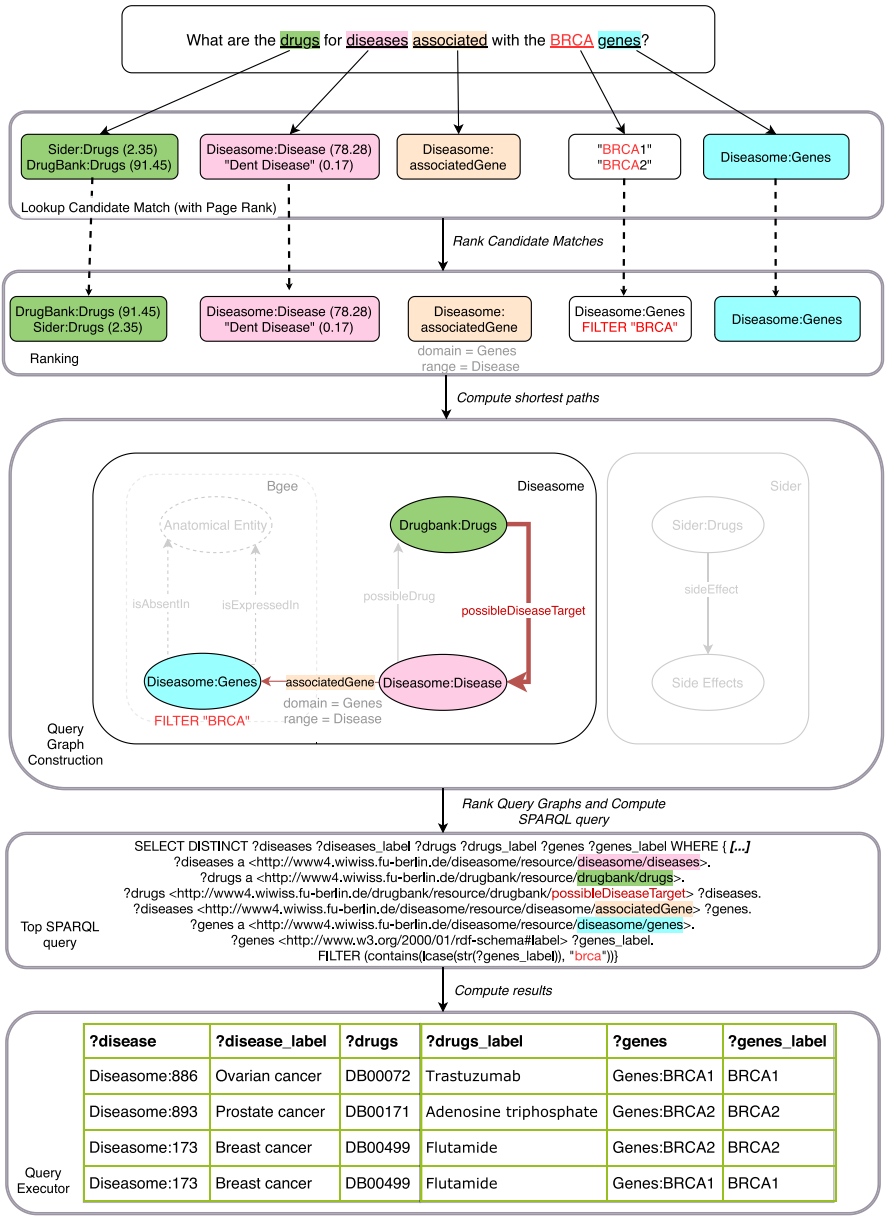

Fig. 3 Simplified answer pipeline for the query "*What are the drugs for diseases associated with the BRCA genes?*". For the sake of simplicity, PageRank scores are solely displayed when more than one match is found

for constructing candidate SPARQL queries in response to user questions. This phase is only executed once, when initialising the system.

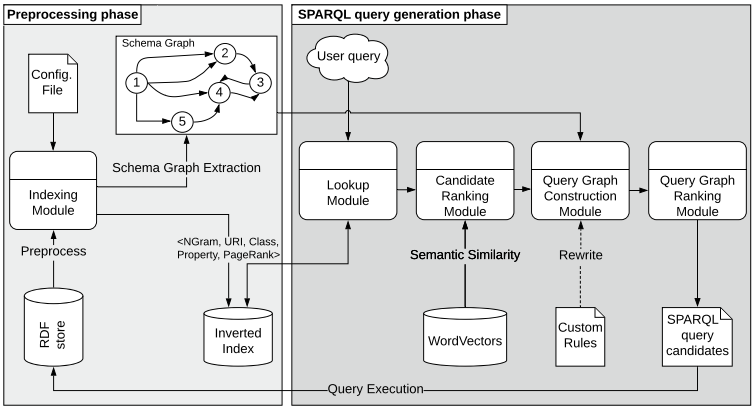

Fig. 4 Bio-SODA System Architecture

- *SPARQL Query Generation Phase*: This phase represents the natural language query translation process and includes (1) looking up query tokens in the database, (2) ranking the candidate tokens, (3) constructing the candidate query graphs, (4) ranking the query graphs in order of relevance to the user question; and finally (5) constructing a valid SPARQL query and presenting the results.

Additionally, the Bio-SODA UX interface[6], discussed in Sect. 6, introduces a further, iterative phase:

- *User Dialog and Disambiguation*:
  More and more RDF datasets are available in diverse scientific fields, yet for practitioners, they are often difficult to explore. Indeed, the increasing size and complexity of the data necessitate not only faster indexes but also smarter user interfaces providing dynamic querying and filtering possibilities. Our experience in prototyping Bio-SODA showed that, in order to enable data exploration, the system must also guide the user in the process of exploring the data models, assist in disambiguating questions, and finally dynamically choose the most relevant answers for specific use cases. To this purpose we designed the Bio-SODA UX interface, which can be operated online[7] to explore knowledge graphs and assist users without technical knowledge of the underlying data models or query languages in disambiguating questions targeting the data stored in the knowledge graphs.

We will now discuss these phases in more detail.

[6] Code also at https://github.com/anazhaw/Bio-SODA, see *biosodaUX* folder
[7] Demo available at https://biosoda.expasy.org/biosodaUX/

5.1 Preprocessing phase

The core component of this phase is the *Indexing Module*, which extracts the *Inverted Index* as well as the *Schema Graph* of the RDF data sources:

- *Inverted Index*: This index stores the vocabulary of the system. More precisely, all the properties that should be searchable from the RDF data store are indexed, according to a configuration file that specifies the list of properties of interest (by default, all string literals will be indexed). A further configuration option is whether URI fragments should also be parsed and indexed. In this case, these fragments are split by a predefined punctuation list, and through a camel case regex (*e.g.*, "possibleDiseaseTarget" will be indexed as the corresponding keywords "possible disease target").

  The inverted index is stored in a relational database for fast searches and it is used to match tokens (sequences of keywords in a user query) against the RDF data. More precisely, the index stores: keywords (N-grams of literals indexed), the indexed instance URI, the class of this instance, the property from which the keywords were indexed (*e.g.* label), as well as the PageRank score of the instance (see Table 1). PageRank scores are computed using the approach presented in [32].

  We note that the size of the inverted index depends on a few characteristics of the knowledge graph, including the verbosity of literals (*i.e.*, attributes that are strings), as well as the total number of attributes that should be indexed by Bio-SODA (an explicit list of these attributes can be provided in the system). For example, very verbose fields should not be indexed in their entirety, but perhaps in a summary form. This variability reflects also in the size of the inverted index compared to the size of the original dataset. Table 2 provides an overview across the 3 datasets considered in this study. For example, the QALD4 and Bioinformatics datasets also contain numerical data, which is not indexed, leading to a smaller index size than in the case of CORDIS. Generally, in terms of time, building the inverted index can take a few hours for large datasets, but this highly depends on the performance and availability of the SPARQL endpoint through which the dataset is accessible, given that the inverted index is built by querying this endpoint. We note that we have not focused on optimizing the inverted index construction and instead leave this as future work.
- *Schema Graph Extractor*: This module is used in order to enrich the (incomplete) schema of the knowledge graph(s) using instance-level data from the RDF store. The Schema Graph is essentially the *accurate schema of the integrated RDF data* which Bio-SODA automatically extracts from data instances[8]. Moreover, the Schema Graph serves as the basis for constructing candidate query graphs from selected entry points (*i.e.*, matches for tokens in a user question).

[8] Note that multiple RDF sources can be combined, as long as they are semantically aligned - *i.e.* they have at least one common concept, such as *Gene*.

Table 1 Inverted Index Sample

Lookup Key	URI	Class	Property	PageRank
Stroke	side_effects:C0038454	sider:side_effects	sider:side-EffectName	0.34
Drug	drugbank:drugs	owl:Class	rdfs:label	91
Drug	sider:drugs	owl:Class	rdfs:label	2.3
Possible disease target	diseasome:possible-DiseaseTarget	rdf:Property	uri_match	80

The lookup key is used for fast searches based on keywords from a user question. The remaining information is used in attaching candidate matches to the Schema Graph (see description in Sect. 5) in order to construct the corresponding query graphs. A lookup key can consist of multiple keywords. The same lookup key can appear multiple times

Computing a Schema Graph allows the system to compensate for incomplete schema information, for example, in cases where domains and ranges for properties are either missing or ill-defined. A second benefit of the Schema Graph is that it enables integrating multiple data models from different knowledge graphs. More precisely, since the search algorithm works at the level of the Schema Graph, it is agnostic to the actual physical representation of the data, meaning it can be easily extensible to support the case of multiple, complementary, knowledge graphs in the future. The minimal requirement for achieving this is that these KGs overlap, *i.e.*, they have classes in common, such that they can be joined in an integrated Schema Graph.

Extracting the schema graph is achieved via SPARQL queries that compute, for example, domains and ranges of all properties, based on the classes of the instances which they connect. As a simplified example, a triple asserting "Migraine → possibleDrug → Ibuprofen" will result in *Disease* → *possibleDrug* → *Drug* being added to the Schema Graph.

Currently, as a minimum requirement we assume that each instance in the RDF data has a well-defined class, *i.e.* an explicit *rdf:type*. If this is not the case, additional preprocessing with external tools (for example, using RDF schema discovery techniques [36]), would be required in order to properly define types for all RDF instances.

Table 2 Descriptions of the size of the 3 public datasets used in our evaluation and their corresponding inverted index

Dataset	Sources	Dataset Size	Index Size
QALD4-biomedical	Drugbank, Diseasome, Sider	200 MB	150 MB
Bioinformatics	Bgee, OMA	30 GB	8.5 GB
CORDIS	EU projects	1 GB	1 GB

We note here that indexing is a preprocessing step that is only required once, when the system is initialized. Afterwards, updates to the RDF store can be incorporated periodically through incremental updates (appends) to the inverted index, while the Schema Graph only needs to be recomputed in case of schema changes.

5.2 SPARQL query generation phase

Given a natural language question, the goal of the Bio-SODA system is to translate it into a set of ranked candidate SPARQL queries, such that the top ranked query is the closest to the user's query intent. In the following, we detail the role of each component involved in this translation process, namely the *Lookup Module*, the *Candidate Ranking Module*, the *Query Graph Construction Module*, the *Query Graph Ranking Module* and the *Query Executor Module*.

- *Lookup Module*:
  The lookup module has the role of retrieving the best candidate matches for tokens identified in a user query. A token is defined by the longest sequence of keywords that matches an entry in the Inverted Index (implemented in a relational database for fast searches). For example, in the question "*What are the possible disease targets of Ibuprofen?*" the two tokens extracted will be "*possible disease target*" (corresponding to an RDF property name) and "*Ibuprofen*" (corresponding to one or more Drug instances).
- *Candidate Ranking Module*:
  The lookup module can return a large number of candidate matches per token. In order to find best candidate matches, the ranking module *groups together equivalent matches and ranks them* in order of relevance to the initial query. For example, instances of the class *Drug* with matching *rdfs:label* are grouped together. In our running example illustrated in Fig. 3, the genes *BRCA1* and *BRCA2* are a match for the keyword *BRCA*.

  Furthermore, both *string similarity* and *node importance* are taken into account when ranking. Including the PageRank score as a measure of importance in the knowledge graph reduces the influence of the quality of labels assigned (labels which can be imprecise, see discussion in Sect. 3).

  The intuition behind this is that domain knowledge graphs usually cluster around a few important concepts, which will be reflected in the PageRank scores of the corresponding nodes. For example, UniProt[9] [37], a protein knowledge base containing more than 60 billion triples, currently includes only 177 classes. Out of these, only few classes, such as *Protein* and *Annotation*, have a central role, and will usually be the target of domain expert questions.

  Likewise, in the case of the CORDIS EU projects dataset (see Sect. 7 for details), two different classes of Projects are available, *EC-Project* and *ERC-Project*. However, there is significantly more information in the dataset for the first

[9] https://sparql.uniprot.org/

class. In the lack of query logs or handcrafted rules for mapping query tokens to the correct candidates, the PageRank score can serve as a good proxy for ranking candidates according to node centrality, similarly to the initial approach used by web search engines [38].

As an added benefit, scoring with PageRank also ensures that metadata matches are prioritized. For example, *Drug* as a class name will rank higher than an instance match.

Finally, to ensure that candidate matches not only have good string similarity, but are also *semantically similar*, word embeddings are also used in the candidate ranking. The similarity comparison ensures that spurious matches, such as *gene* compared to *oogenesis*, are discarded based on a pre-defined similarity threshold in the system configuration.

Any word embeddings can in principle be used with Bio-SODA. For the two main bioinformatics use cases considered in this paper, we use Word Vectors extracted from PubMed, as described in [39]. The candidate ranking module presents to the user top N matches per query token, where N is configurable in the system. We note that it is important to limit the number of matches per token for performance reasons. This is because the total number of candidate queries generated for a question with T query tokens (*i.e.* concepts searched by the user) will be in the worst case N^T (there are up to N matches for each of the T tokens).

- *Query Graph Construction Module*:

  The goal of this module is to use the matches from the previous step to *generate a list of candidate query graphs*. We extend the approach presented in [40] to translate matches to query graph patterns. More precisely, we apply the iterative algorithm shown in Algorithm 1: for each set of candidate matches (one match per query token), we augment the Schema Graph by attaching the candidate matches to their corresponding class. Next, we find the minimal subgraph that covers all matches. For this purpose, we solve the approximate Steiner tree problem by computing the minimal spanning tree that covers one match per token.

  Note that there might be multiple such subgraphs, given that two classes can be connected via multiple properties. However, unless the user can be involved in disambiguating, it is important to generate all the variants, given that two equal-length subgraphs might actually have opposite semantics. Recall the example shown in Fig. 2, where the properties *e.g*, *isAbsentIn versus isExpressedIn* both connect the same two classes, but represent disjoint result sets.

  Finally, in some cases handcrafted rules for inferring new concepts or relationships are required, due to the complexity of the corresponding query graphs. In such cases translating user questions into SPARQL cannot be done via simple entity linking methods. Therefore, if needed, our approach also supports adding rules to derive implicit information from the original knowledge graph as part of the question answering pipeline. These rules are implemented as sub-queries similar to the `SELECT` SPARQL query form. In this case, the rule head is the SPARQL query projection, and the rule body is the WHERE clause content.

- *Query Graph Ranking Module*:
  The query graph ranking module plays an important role in presenting the user with a meaningful, ordered list of results. In contrast to existing work, we do not return the *overall minimal subgraph* as the top result, but rather the *graph that maximizes the sum of the match scores* of the candidates covered. To understand why this is the case, consider the following question: *"What are the drugs for asthma?"*. This question translates to a 2-hop query graph, joining *Drug* and *Disease* via the *possibleDiseaseTarget* path (see Fig. 2). However, one likely scenario is that the description of a *Drug* instance includes the keyword *asthma*. In this case, the minimal query graph would be 1-hop only, retrieving only *Drug* instances that explicitly contain the keyword in the description, probably a small subset of all instances which have the corresponding *Disease* as a possible target. In this case, the minimal result would have good precision, but very low recall.
- *Query Executor Module*:
  Finally, the query executor translates the ranked query graphs into SPARQL queries, assigning meaningful variable names, also adding human-readable fields to the result set wherever possible. Importantly, we do not only return the best result, but rather a ranked list of possible interpretations (top N, where N is configurable in the system). This gives the user the opportunity to inspect the results in order to choose only the interpretation (*i.e.* SPARQL query) that matches the question intent.

Algorithm 1: Iterative graph-based approach for constructing query graphs from candidate matches

Data:
$M^{n \times t}$: the matrix of ranked candidate matches, where
n = the number of candidate matches per token,
t = the number of tokens in the user question.
M_i = a set of candidates covering one match per token (i.e. the i^{th} row vector of the $M^{n \times t}$ matrix).
G: Schema Graph of the RDF data
Result: S: the ranked set of candidate query graphs

```
foreach M_i ∈ M do
    QG_i = φ (empty graph)
    foreach candidate match T_j ∈ M_i do
        if T_j = a RDF property then
            Get domain D and range R of T_j from G;
            Add D and R as vertices to QG_i;
            Add edge T_j between D and R in QG_i;
            if multiple domains / ranges for T_j then Create a new
            copy of QG_i per alternative;
        else
            Compute in schema graph G:
            shortest paths between class of T_j and classes of other
            matches T_z in M_i;
            Add shortest paths to QG_i
            if multiple alternatives exist then
            Create a new copy of QG_i per alternative;
        end
    end
    Add spanning tree extracted from QG_i to result set S (Steiner tree
    approximation)
end
S_sorted = sort S by sum of match score of composing vertices. On a
tie, sort by the weight (i.e. the number of edges) of spanning tree.
return S_sorted
```

6 Bio-SODA UX: user dialog interface

6.1 Design

Since natural language questions can be very ambiguous and two users might mean different things when asking the same question, it is important that the system helps the user in exploring the data and in finding the correct answer to a question. Hence, we have built Bio-SODA UX, an interface that guides the user in the data discovery process by disambiguating natural language expressions that have multiple meanings.

To help the user understand the meaning of different results, the concepts are visually represented in a graph of the data model, supported by a data-driven color scheme and additional detailed information.

Bio-SODA UX provides more information about the underlying data and the meaning of the results on hover/click. Once the user received the first answer, a data table is populated with a result set. The immediate feedback of the system helps the user to identify potential refinements of the question or opportunities for disambiguation (i.e. choosing the right concepts). After each interaction with the user, the data table is reloaded to match the current state. While the default view shows only a few key features of the resulting data (e.g. Genes), the system allows for adding other potentially relevant information as "alternative columns" (e.g. gene descriptions), which the user can individually keep or discard. After the user has successfully disambiguated, the full set of results can be downloaded as a comma-separated values (CSV) file for further downstream processing.

6.2 Implementation

The implementation of Bio-SODA UX consists of two main parts - the user web interface and the API. The web interface acts as a mediator between user and API. The API connects to the underlying Bio-SODA question answering system, which enables the natural language translation to SPARQL and query execution. The code for both of these parts is publicly available.

The user interface was developed using the jQuery library[10], the bootstrap framework[11] for graphical representations, and D3.js[12] for graph and data table visualization. Bio-SODA UX consists of three main parts, shown in Fig. 5: (1) the query input field for entering the users' natural language question; this part is also used to present the intermediate results to the user and to allow the user to disambiguate detected concepts via drop-down lists; (2) the node graph which displays the data model to the user, including where the candidate matches attach to the schema graph, providing some intuition of the data; (3) the data table on

[10] https://jquery.com
[11] https://getbootstrap.com
[12] https://d3js.org

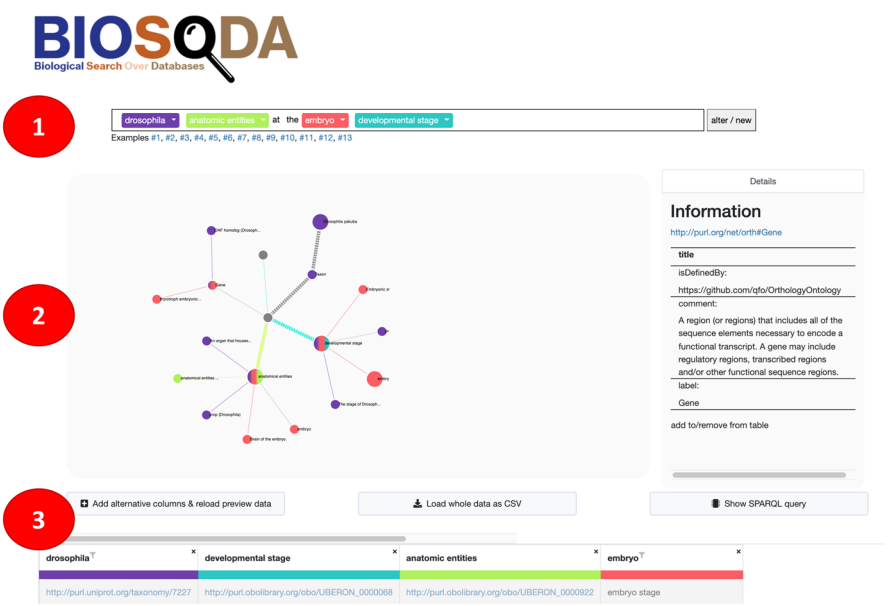

Fig. 5 Bio-SODA UX interface for knowledge graph exploration and query disambiguation. The three main components of the interface are: 1) an input field which also provides drop-downs with example candidate matches for each searched concept; 2) the fraction of the data model relevant to the question, shown in graph form; clicking on any node will display additional information in the "Details" box on the right; 3) the results table with options to extend with more attributes related to the concepts in the question

the bottom of the page, displaying the results corresponding to the current user selection.

The communication between the user interface and the Bio-SODA engine in the back-end is implemented via a series of API calls. We summarise here the main steps involved:

1. The front-end invokes the Bio-SODA algorithm when the user first inputs a natural language question. The back-end algorithm will respond to this initial call with the following information:

- ranked candidate matches for each concept in the original question. For each candidate match, the back-end will also indicate what is the class of the match (e.g. a *Gene*, *Protein* or *Anatomical Entity*) - which helps attach the candidate match to the graph displaying all candidates. The user will then have the option to select the best candidate match from the corresponding drop-down list, while also inspecting the data model graph and seeing how each candidate connects. Furthermore, the response from the back-end includes additional information for each candidate, such as a label or a description of the

RDF entry it matched. This information can be seen by clicking on the corresponding node displayed in the graph. Since for any given term, there can be many possible entries, in order to group results together, only one example candidate is provided for each class.
- a SPARQL query is returned and executed, corresponding to the best answer according Bio-SODA ranking algorithm. The results are displayed in the table on the bottom of the page and the SPARQL query itself can be inspected by clicking "Show SPARQL query".

2. When disambiguating, the user can select a different entry from the drop-down list of a given concept. This will trigger a new API call to the back-end, including the original question and all the current selections for each matched concept, such that the lookup and candidate ranking phases (see Sect. 5) in the Question Answering Pipeline can this time be skipped. The back-end will return a new SPARQL query and the corresponding results, according to the updated candidates selected by the user.

6.3 Use case

Here, we will focus on selected example use cases considering Bio-SODA UX applied over the bioinformatics database of gene expression Bgee.

6.3.1 Example 1

The user can first provide a natural language question, such as *"What are the drosophila anatomic entities at the embryo developmental stage?"*. In response, Bio-SODA UX returns the best-ranked answer in tabular form (see Fig. 6), while showing examples of candidate matches for each concept in graph form. The graph shows the connections between the chosen concepts and thus supports the user in understanding the underlying data model. For example, it can show how "anatomical entities" and "developmental stages" are linked through "gene expression conditions". The data table presents results matching the selected concepts and filters.

The user can analyze every matched concept, such as "drosophila" or "embryo", via the displayed graph. Furthermore, additional information, such as the URI of the matched node, its class, the property that matched and the Page Rank score of the node, can be displayed in the "Details" panel on the right side of the page, by clicking on the matched node in the graph. By default, the system chooses the best candidate match for every concept according to the Bio-SODA ranking algorithm (see Sect. 5). The chosen candidates are also emphasized in the graph in the form of larger nodes. However, alternatives are presented for every concept in the form of drop-down menus. For example, an *embryo* can refer to either an organ (anatomical entity) or a developmental stage. The user can disambiguate the intended meaning by inspecting the available options and choosing the correct candidate match from

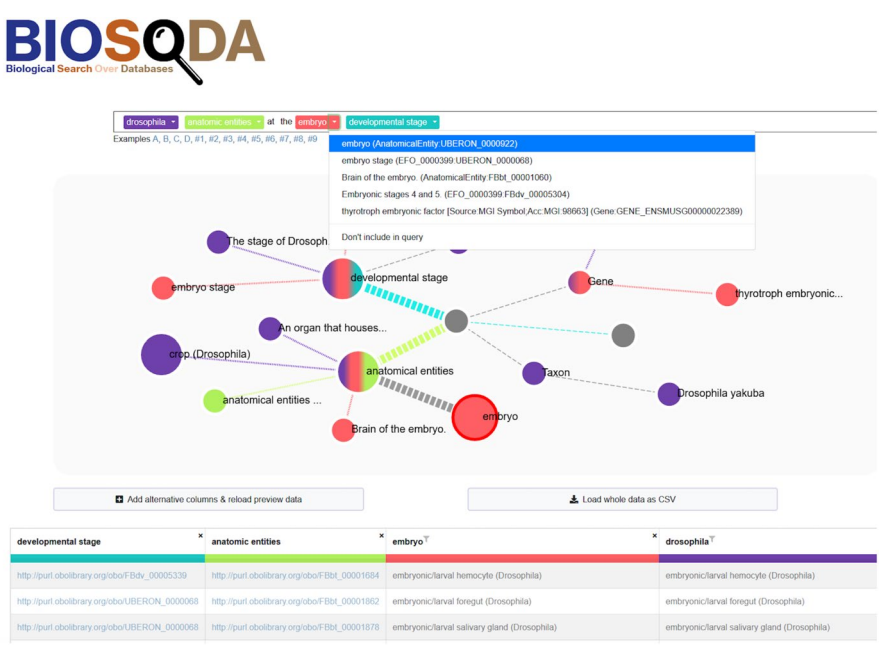

Fig. 6 Bio-SODA UX example use case for the question "drosophila anatomic entities at the embryo developmental stage"

the drop-down menu. The Bio-SODA UX supports the user by also assigning every concept a distinct color, which is then used throughout the disambiguation process.

In our example, the term "drosophila" can be matched, among other meanings, to an anatomical entity (i.e. an organ). This is because anatomical entities are sometimes explicitly annotated with the species name, e.g. "crop (Drosophila)" - an equivalent of the mammal stomach in the Drosophila (fly). However, the intended meaning of the concept in the original question is the name of a species (i.e., a *Taxon*), for example "Drosophila yakuba" or "Drosophila melanogaster" (the fruit fly).

By default, Bio-SODA ranks the crop higher, however, the ranking is not accurate since we want to receive a list of *all* anatomical entities from Drosophila flies, not only those explicitly *annotated* with the term "Drosophila". The users can in this case clarify their intended meaning by choosing the *Taxon* entry corresponding to the "Drosophila" concept, from the drop-down list that is highlighted in purple in the query in Fig. 6. A further disambiguation required in this case is to then select the "embryo" candidate match that corresponds to a *developmental stage* (i.e., "embryo stage", the second match in the drop-down shown in Fig. 6), as opposed to an anatomical entity (the top ranked candidate, chosen by default for this term). This can be selected from the drop-down menu of the term highlighted in red in the figure. The different candidates are also illustrated in the graph below (nodes colored in red), in order to facilitate understanding which match corresponds to the original intent of the query.

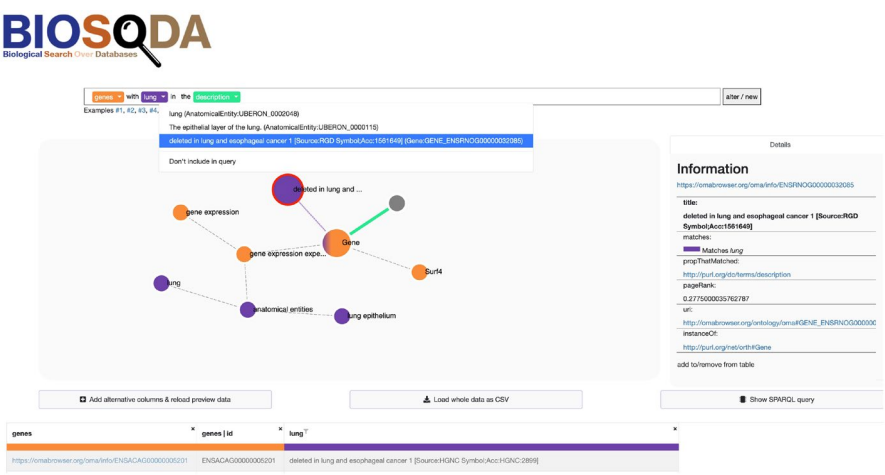

Fig. 7 Bio-SODA UX example use case for the question "genes with lung in the description"

6.3.2 Example 2

A second example, shown in Fig. 7, illustrates the response to the question *"What are the genes with lung in the description?"* This question reflects a user with more knowledge of the data model, since it requests information related to a property of Gene instances (i.e., those instances that explicitly contain the term *lung* in the gene description). By default, Bio-SODA ranks the candidate match *lung* corresponding to an anatomical entity (an organ) higher. This usage is much more common, which will reflect in the Page Rank score of the anatomical entity match *Lung*, therefore making it the top scored candidate for this term. However, this can lead to an incorrect or incomplete answer, as it would correspond to genes *expressed* in the lung organ, not those *annotated* with the term. Nevertheless, by inspecting the graph the user can easily clarify the intended meaning, by selecting the entry corresponding to an example of a gene description. This is the node attached to the *Gene* class, shown in the centre of Fig. 7.

In the drop-down list for the term *lung*, highlighted in purple in the original question, this is the third ranked candidate match. Hovering over the candidate matches will highlight the corresponding nodes in the graph with a red border, making it easier to understand where they attach in the original data model. Moreover, clicking on any of the nodes (e.g. the purple node attached to the Gene class, shown with a red border in Fig. 7) will also display more information about the candidate, including which class and property matched (i.e. the description, http://purl.org/dc/terms/description, of a Gene, http://purl.org/net/orth#Gene).

The user can optionally inspect the generated SPARQL statement by clicking on *Show SPARQL query* on the corresponding button in the page, which will also confirm the correct answer for a more advanced user, while helping a novice user better understand how to obtain information for this question via SPARQL queries.

Finally, the user can further find out more details about the retrieved genes by following the clickable links provided in the results table.

7 Experiments

In this section we evaluate the F1-score performance of Bio-SODA for translating natural language questions to SPARQL and compare it against state-of-the-art systems for querying RDF-based knowledge graphs. Note that we focus on top-performing open-source systems that are publicly available for testing and do not require training data [22].

In particular, we tested Sparklis [33], a generic query builder system for knowledge graphs[13]. Furthermore, we compared against GFMed [24] which was top ranked in the QALD4 biomedical challenge and specifically designed for this dataset. Apart from this, we use GFMed's publicly available grammar[14] to evaluate how the system performs outside of the official QALD4 biomedical dataset. In addition, we compared our approach against SQG [12], a system for query generation over knowledge graphs[15].

7.1 Datasets

Three datasets were considered for evaluating Bio-SODA, see Table 3. Importantly, all three are real-world, in-use datasets. For each dataset, we briefly highlight the specific challenges that need to be tackled in the context of designing a generic question answering system:

1. The *QALD4 biomedical* dataset is composed of Sider, DrugBank and Diseasome. This dataset includes several challenges such as multiple *Drug* classes and identical terms describing both *Disease* and *Side Effects* instances, which are connected via *owl:sameAs* properties.
2. The *bioinformatics* dataset is composed of the Bgee (gene expression) [41] and OMA (orthology) [42] RDF stores. Given the highly specialized domain information contained in these sources, a particularity of this dataset is that questions can include complex concepts which translate to long SPARQL query graphs. An added challenge deriving from this is that the same concepts can be connected through multiple equal-length paths with semantically different or even opposite meanings.
3. The *CORDIS* dataset of EU-funded projects. Although this dataset has a simpler schema, the challenge here is that questions can have a higher degree of ambiguity. In some cases, multiple interpretations are valid – for example, many terms are

[13] A live demo can be tested with any SPARQL endpoint at http://www.irisa.fr/LIS/ferre/sparklis/
[14] See http://cs-gw.utcluj.ro/~anca/GFMed/index.html
[15] Available at https://github.com/AskNowQA/SQG/

Table 3 Descriptions of the 3 public datasets used in our evaluation

Dataset	Sources	#Classes	#Triples	Size on Disk
QALD4-biomedical	Drugbank, Diseasome, Sider	12	0.69 M	200 MB
Bioinformatics	Bgee, OMA	37	430 M	30 GB
CORDIS	EU projects dataset	26	6.5 M	1 GB

reused often and in a variety of contexts, such as "*Big Data*". This can be either part of a project title, a topic or even an organization name. Therefore, identifying the query intent in some cases (*e.g. Show Big Data projects*) cannot be done without user disambiguation.

7.2 Queries

We have reused the official 50 queries of the QALD4 biomedical challenge[16]. We do not distinguish between training and test queries. Indeed, we report performance metrics for all systems we tested across the entire set of 50 queries. Given that the test set was also available to participants in the official challenge, we believe this to be a fair evaluation. We do not change the questions in the official challenge, not even in cases where we could identify mistakes in the question. Furthermore, as opposed to previous work using this benchmark [43], we do not materialize triples based on *owl:sameAs* statements and only use the exact dataset, as provided in the official benchmark.

For the bioinformatics dataset, in collaboration with domain experts, we created a benchmark of 30 queries, in increasing order of complexity, across two datasets, namely Bgee and OMA. The queries represent real information needs of domains experts within the field of gene expression and orthology, using the publicly available RDF data of Bgee[17] and OMA[18]. The average number of triple patterns per query here is 7 (not taking into account joint queries between the two sources, which have even higher complexity), with some questions jointly targeting 4 entities or more (*Gene*, *Species*, *Anatomical Entity*, *Developmental Stage*). In contrast, in existing benchmarks, such as LC-Quad [14], queries with only 2 entities are already considered complex.

In order to test Bio-SODA using an entirely different domain, using the CORDIS dataset of EU funded projects, we created a test set of 30 queries in increasing order of complexity. Given the relatively simple structure of this data model, the average number of triple patterns per query is close to that of existing KGQA benchmarks [14], with an average 2.3 triple patterns per query. However, the complexity stems from the usage of filters, literals in the query, as well as the higher degree of ambiguity.

[16] https://github.com/ag-sc/QALD/blob/master/4/data
[17] https://bgee.org/sparql
[18] https://sparql.omabrowser.org/sparql

Queries across the three datasets include aggregations, negations, and make extensive use of filters.

All questions, as well corresponding SPARQL queries, are available in the Evaluation folder of our GitHub repository[19].

7.3 Results

We use the standard evaluation metrics of precision (P), recall (R) and F1-score, macro-averaged over all questions in the dataset. For Bio-SODA in particular, although the system generates a ranked list of possible interpretations, we report numbers based on the top answer only (Precision@1). More precisely, these results are evaluated based on the top answer provided by the default ranking of the Bio-SODA system, without any user disambiguation involved. In the Bio-SODA UX interface, this corresponds to the answer (and corresponding SPARQL query) shown to the user by default in response to a given question, without any manual intervention from the part of the user. The results are presented in Table 4 and discussed in the following section. For easy accessibility to the Bio-SODA system, as well as reproducibility of the results, we also provide a demo page for each of the three datasets, available online (see Sect. 1).

We will now discuss the performance of each system in more detail.

GFMed shows the highest F1-score for the QALD4 dataset. However, it cannot (nor was it intended to) be used outside this dataset without rewriting the set of grammar rules that are strictly designed for question answering over specific releases of *Diseasome*, *Drugbank* and *Sider*. Hence, the F1-score for the bioinformatics dataset and the CORDIS datasets is 0.

SQG on the other hand, originally evaluated on the LC-Quad [14] benchmark, does not support complex multi-hop questions, nor filters or queries involving literals. "*Show me projects which started in* **2020**?" is an example of such a query, where *2020* is a numerical literal, as opposed to a linkable entity. While in the case of LC-Quad these limitations do not impact performance, all three datasets considered in our evaluation include such features, which explains the poorer performance of SQG: an F1-score of 0.42 in the case of QALD4, only 0.33 in the CORDIS dataset, and finally 0.16 in the case of the bioinformatics dataset. We note that these results are a theoretical best, since for SQG we assume perfect entity and property linking, leading to the highest performance it can achieve.

Finally, *Sparklis* is not a question answering system per-se, but rather a query builder, which helps users form the correct question by composing building blocks starting from examples of class names, properties, values etc. Therefore, in order to answer questions, we needed to rephrase them from the available building blocks *manually*. On the positive side, we found Sparklis to be a powerful system, because it enables building a rich variety of query types out-of-the-box.

[19] Evaluation in https://github.com/anazhaw/Bio-SODA/

Table 4 Performance of translating natural language questions to SPARQL. By considering a perfect user of the Sparklis tool, the minimum number of manual steps for composing a query (averaged over all queries) is shown between parentheses

Datasets and Systems	Precision	Recall	F1
Dataset 1: QALD4			
GFMed	1	0.99	0.99
SQG	0.42	0.42	0.42
Sparklis (5.5 steps/query)	0.88	0.88	0.88
Bio-SODA	0.61	0.60	0.60
Dataset 2: Bioinformatics			
GFMed	0	0	0
SQG	0.16	0.16	0.16
Sparklis	-	-	-
Bio-SODA	0.6	0.6	0.6
Dataset 3: CORDIS			
GFMed	0	0	0
SQG	0.33	0.33	0.33
Sparklis (6.2 steps/query)	1	1	1
Bio-SODA	0.66	0.66	0.66

To achieve this, only the SPARQL endpoint URL of the target RDF data store is required.

Using the query building methodology of Sparklis, 44 out of 50 questions in the QALD4 biomedical benchmark can be answered. Furthermore, all questions in the CORDIS dataset can also be answered. Although this result might seem surprising, recall that the major challenge of this dataset is disambiguation. The manual query building process in Sparklis addresses exactly this problem, provided that the user knows very well how the data are structured and semantically represented. Therefore, on the negative side, we found that the query building methodology requires precise understanding of the data model, especially if multiple classes have the same label, as is the case in QALD4.

For example, answering the question *"Which drugs might lead to strokes?"* requires knowing that the *Drugs* class to be used is the one in *Sider*, as opposed to the one in *Diseasome*. Furthermore, formulating questions in Sparklis is a manual and therefore time-consuming process. Even when making the strong assumption that the user has perfect knowledge of the data model, as well as of the features of Sparklis (for example, how to correctly formulate aggregations, which can be challenging), the minimal number of manual steps required to formulate questions is on average 5.5 interactions per question for QALD4 and 6.2 for CORDIS, with a maximum of 10 for the more complex questions. In most cases, the question resulting from composing the building blocks will be significantly different from a true natural language question.

We did not pursue this approach on the bioinformatics dataset, because complex concepts in this dataset (ortholog, paralog) cannot be expressed through the query building mechanism. More precisely, Sparklis does not support complex property paths.

Bio-SODA is a middle-ground between the generic, but manual approach of Sparklis, and the grammar-based approach of GFMed, which is not easily transferable to a new domain. More precisely, Bio-SODA achieves relatively good performance (around 0.6 F1-score) across the three datasets without requiring manual intervention. The only exception are two custom rules for the bioinformatics dataset, which help answer 4 out of 30 queries.

Although GFMed has the best results for QALD4, it cannot be used outside this dataset without a complete rewriting of the grammar rules. Sparklis also achieves better results on the two datasets tested, but with the big disadvantage that it is a manual approach, where the user must understand the data model in order to compose questions correctly. Our findings are further detailed in the Evaluation folder in our GitHub repository.

7.4 Impact of ranking algorithm

In this section we study the impact for our ranking algorithm on the performance of Bio-SODA. In particular, we conducted an ablation study to quantify the importance of ranking by PageRank score of candidate matches. For this purpose, we disable our ranking algorithm and instead use a simple string similarity-based ranking algorithm for candidate matches, returning the *overall* minimal subgraph as the top answer.

The results, displayed in Table 5, show that ranking makes a crucial difference, in particular for the CORDIS dataset. The reason for this is that for most of the keywords that describe metadata (such as class names, like *Project Topic* or *Subject Area*), there exists in the dataset a project whose acronym matches exactly. For example, there exist projects with acronyms such as *Topic*, *Area*, *Host*, *Code*, which are (according to string similarity only) classified as best matches for tokens in the original question. Constructing the *overall* minimal subgraph leads to wrong results in almost all cases, except for only 3 out of 30 questions, where there is no ambiguity. Note that adding *no other change* aside from considering PageRank scores in ranking enables answering 17 more queries out of 30 for this dataset.

7.5 Error analysis and remaining problems

In the QALD4 biomedical benchmark, Bio-SODA correctly answered 30 out of 50 questions with an additional 2 partially correct. We note that 1 question in QALD4

Table 5 Ablation study on the Bio-SODA performance of translating natural language questions to SPARQL: (a) SPARQL candidate query ranking with node centrality measure versus (b) traditional ranking approach with string similarity and overall minimal subgraph as top result

Dataset	(a) Correct with Bio-SODA Ranking	(b) Correct with String Similarity Ranking
QALD4	30/50	23/50
Bioinformatics	18/30	12/30
CORDIS	20/30	3/30

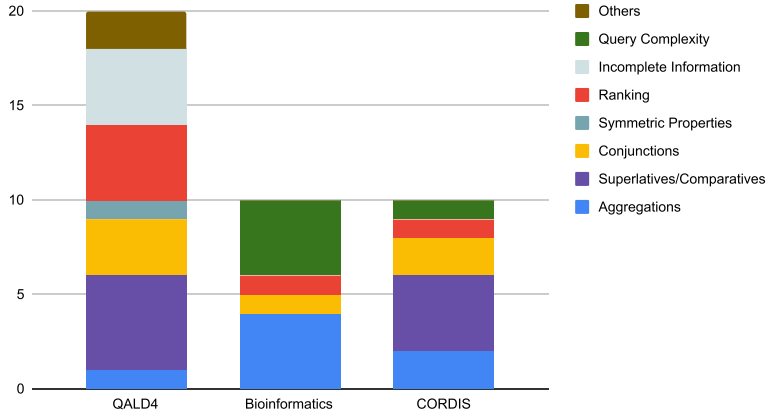

Fig. 8 Bio-SODA failure analysis. Out of the total 50 questions in the QALD4 biomedical benchmark, Bio-SODA cannot correctly answer 20. A further 12 out of 30 cannot be answered in the bioinformatics dataset, mainly due to query complexity (some queries having more than 10 triple patterns). Finally, on the CORDIS dataset 10 out of 30 queries cannot be answered, a large fraction of which include features currently unsupported in Bio-SODA: aggregations, comparatives, conjunctions etc

cannot be answered by Sparklis nor Bio-SODA due to missing label information. More precisely, the instance *http://www4.wiwiss.fu-berlin.de/diseasome/resource/genes/EDNRB* is the target of the question "Which genes are associated with Endothelin receptor type B?". However, the label *Endothelin receptor type B* is not assigned in the official dataset of the benchmark, nor can it be derived from the URI fragment, for example. Upon closer inspection, it becomes clear that the question is ill-formulated. Since *EDNRB* itself is a gene, the correct question should be "Which *diseases* are associated with EDNRB?". In total, we have found at least 4 out of 50 entries in the dataset to contain errors, either in the question formulation, or in the ground truth answer. These have already been discussed in previous studies [43].

An additional number of questions cannot be answered by Bio-SODA across the three datasets due to other reasons. We summarise them in Fig. 8, explained in the following:

- *Aggregations*. Our system currently does not support questions that require aggregations, such as *Count*, *Sum* etc. An example of such a question would be *Count the projects in the life sciences domain*. A possible solution to this would be to include pre-defined patterns or training a question classifier for this purpose.
- *Superlatives/Comparatives*. Another unsupported feature in the current prototype is the use of quantifiers (superlatives or comparatives). An example would be *Which drug has the highest number of side-effects?*
- *Conjunctions*. Conjunctive questions which involve multiple instances of the same class are not supported in the current prototype. An example of such a case is *List drugs that lead to strokes and arthrosis*. This limitation derives from our methodology in computing the minimal subgraph covering candidate matches,

which would require special handling for cases when multiple candidates of the same class are present in a question.

- *Properties with same domain and range*. Stemming from the same limitation mentioned above, these properties are currently not supported. In QALD4, the only instance of this is the *diseaseSubtypeOf* property, which has the *Disease* class as both domain and range. In the bioinformatics dataset we handle symmetric properties describing *ortholog* and *paralog* genes through custom rewrite rules.
- *Ranking*. One of the major sources of failure in our prototype remains ranking. In the QALD4 dataset, ranking problems affect 4 out of 50 queries. An example is: *What are the diseases caused by Valdecoxib?*. Here, the system cannot correctly choose *Drug - sideEffect - Side_Effect* over the alternative *Disease - possibleDrug - Drug*. The reason for this is that the *Disease* class matches exactly the term in the question, while the *Drug* class in *Diseasome* has a higher PageRank score than the one in *Sider*.

  A more complex corner-case is part of the bioinformatics dataset, *What are the genes with lung in the description?* The term *lung* is commonly used to refer to an *Anatomical Entity*. This is also reflected in the node importance of this match in the dataset. Therefore, the system cannot correctly determine that, in the context of this question, it should instead be considered part of the *description* property of a *Gene*. The correct candidate match scores very low, resulting in the correct answer also being ranked too low. However, through user disambiguation in the Bio-SODA UX, this question can also be correctly answered. The process is shown in Fig. 7 and a discussion is included in Sect. 6.3. A similar example from QALD4 is *Which drugs have bipolar disorder as indication?*, where *bipolar disorder* is matched against a *Disease* instead of a drug indication. In these cases user disambiguation, at the level of candidate matches, is an important component in solving the problem.
- *Incomplete information*. This problem affects mainly the results in the QALD4 dataset, more precisely 4 out of 50 queries. We have already covered the example of the question targeting the *EDNRB* gene, which lacks the correct label in the official dataset. We currently do not enrich the inverted index with synonyms or external information, which means questions must be formulated in terms of the available vocabulary of the dataset. However, this limitation could be addressed by indexing synonyms from external data sources. Additional three questions cannot be answered because they refer to URIs that do not have any class defined in the data, therefore the system cannot attach the candidate matches anywhere in the Schema Graph.

  An example is the *drugType* property, which can take two values, either http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugtype/experimental or http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugtype/approved. We believe a better modelling of the data should have provided, for example, either these as a *xsd:anyURI* datatype, given they are not used for any other purposes, or defined some class for both.
- *Query complexity (difficult queries)*. The bioinformatics dataset covers queries with high complexity, which are difficult to solve especially since they include

symmetric properties, with multiple instances of the same class, each filtered according to different conditions.

An example of such a question is: *Retrieve Oryctolagus cuniculus' proteins encoded by genes that are orthologous to Mus musculus' HBB-Y gene*. Here, the task is to retrieve *Gene* instances in a particular *Taxon* (species), namely the rabbit (*Oryctolagus cuniculus*), which are *orthologs* (symmetric property) of a second instance of *Gene*, labeled *HBB-Y*, in a different species, namely the mouse (*Mus musculus*). The resulting query has over 15 triple patterns, with 3 filters (the 2 species names plus the gene name).

- *Others*. Two questions in the QALD4 dataset have particular challenges, the first being a stemming error. In the question *Give me drugs in the gaseous state*, the term *gaseous* cannot be correctly stemmed to *gas*. The second type of error is due to unsupported *ASK* queries, *e.g. Are there drugs that target the Protein kinase C beta type?*. Here, Bio-SODA retrieves examples of such drugs, instead of the boolean *True*. However, we do not consider this a fundamental limitation and a question type classifier could be added in future work.

We report a more detailed analysis of all systems considered in this paper in the https://github.com/anazhaw/Bio-SODA/tree/master/Evaluation in our GitHub repository.

8 Lessons learned

Considering the challenges of question answering over knowledge graphs introduced in Sect. 3, we highlight the following design goals for natural language processing engines:

- *Generality*: The system should be easily adaptable to new datasets. In particular, the system should be able to answer questions in a new domain with minimal manual intervention and without relying on extensive training data, which is hard to obtain in many domains. Along this line, a desirable property is also the ability to cope with "real-world" datasets, dealing with incompleteness in the data, for example in the form of:
  - missing schema information (should be inferred from instance-level data);
  - missing labels (should be incorporated from URIs whenever meaningful);
- *Extensibility*: The system should easily work with multiple datasets (provided they are already semantically aligned—*i.e.*, data integration is a prior requirement). Many studies introduce possible approaches for data integration, including a recent approach for ontology-based data integration, covering one of the bioinformatics use cases presented in this paper [44].
- *Configurability*: The database owner must be able to specify which properties (*e.g.* labels, descriptions) should be searchable using the system. Our experience with real-world datasets showed that in general it is not desirable for *all* properties to be indexed and thus be searchable. As an example, in many cases, fields

in the queried data sources can be either redundant or too verbose. In bioinformatics, these are abstracts of papers that are assigned as values to an RDF property, whose length can therefore be up to 300 words. Similarly, in the CORDIS dataset, these are the abstracts of the EU projects. These cases should be handled through a dedicated approach, for example, based on classical information retrieval methods as discussed in [45].

- *Explainability*: The system should clearly guide the user through how a question was processed and interpreted. This starts from explaining which concepts were matched in relation to the original question, continuing with how these candidate matches are composed together in a query graph in order to provide the final SPARQL query. Finally, the query results should be understandable as well. Therefore, the projected variable names should also be meaningful.

9 Conclusions and outlook

In this paper we have introduced Bio-SODA, a question answering system for domain knowledge graphs, which we evaluated across three real-world datasets pertaining to different domains: biomedical, gene orthology and gene expression, and finally EU-funded projects. Our results have shown that Bio-SODA outperforms state-of-the-art systems that are publicly available for testing by a 20% F1-score improvement and more. The main advantage of Bio-SODA over existing open-source systems is that it can handle *complex, multi-triple pattern queries* without requiring user guidance and training data. Bio-SODA uses a novel ranking approach that takes into account both string and semantic similarity, as well as node centrality of candidate matches. Our experiments demonstrate that our ranking approach improves the quality of results, particularly in the context of datasets which can suffer from redundancy and imprecise labels.

We have also introduced Bio-SODA UX, a graphical interface allowing users to explore the underlying data models and disambiguate their questions dynamically. As future work, we plan to consider the users' feedback for learning to rank the best answer among resulting candidate queries. We also plan to evaluate the average number of disambiguation steps required for clarifying the semantics of user questions. As a long term direction for future research, we envision compiling a benchmark of cross-domain question-answer pairs, similarly to the Spider benchmark in the relational database world [46], which would enable research into refining pretrained KGQA models for new domains.

Acknowledgements We thank the Swiss National Science Foundation for funding (NRP 75, grant 407540_167149), Lukas Blunschi for the implementation of the SODA system for keyword search system over relational databases [47], on which our prototype is based, and Katrin Affolter for important contributions to the natural language processing pipeline in Bio-SODA. We thank Chiara Gabella and Séverine Duvaud from the SIB User Experience Team for their advice on the design of Bio-SODA UX.

Funding Open access funding provided by University of Lausanne.

References

1. Diefenbach, D., Both, A., Singh, K., Maret, P.: Towards a question answering system over the semantic web. Semantic Web Preprint **2018**, 1–19 (2018)
2. Zheng, W., Yu, J.X., Zou, L., Cheng, H.: Question answering over knowledge graphs: question understanding via template decomposition. In: Proceedings of the VLDB Endowment 11, pp. 1373–1386 (2018)
3. Vakulenko, S., Garcia, J.D.F., Polleres, A., de Rijke, M., Cochez, M.: Message Passing for Complex Question Answering over Knowledge Graphs. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 1431–1440 (2019)
4. Li, F., Jagadish, H.V.: Constructing an interactive natural language interface for relational databases. Proc. VLDB Endowm. **8**, 73–84 (2014)
5. Li, F., Jagadish, H.V.: Understanding natural language queries over relational databases. ACM SIGMOD Rec. **45**, 6–13 (2016)
6. Saha, D., Floratou, A., Sankaranarayanan, K., Minhas, U.F., Mittal, A.R., Özcan, F.: ATHENA: an ontology-driven system for natural language querying over relational data stores. Proc. VLDB Endowm. **9**, 1209–1220 (2016)
7. Brunner, U., Stockinger, K.: ValueNet: a natural language-to-SQL system that Learns from Database Information. International Conference on Data Engineering (ICDE) (2021)
8. Sakor, A., Singh, K., Vidal, M.-E.: An Entity and Relation Linking Framework over DBpedia, FALCON (2019)
9. Ferragina, P., Scaiella, U.: Tagme: on-the-fly annotation of short text fragments (by wikipedia entities). In: Proceedings of the 19th ACM international conference on Information and knowledge management, pp. 1625–1628 (2010)
10. Mendes, P.N., Jakob, M., García-Silva, A., Bizer, C.: DBpedia spotlight: shedding light on the web of documents. In: Proceedings of the 7th International Conference on Semantic Systems, pp. 1–8 (2011)
11. Olieman, A., Azarbonyad, H., Dehghani, M., Kamps, J., Marx, M.: Entity linking by focusing DBpedia candidate entities. In: Proceedings of the First International Workshop on Entity Recognition & Disambiguation, pp. 13–24 (2014)
12. Zafar, H., Napolitano, G., Lehmann, J.: Formal query generation for question answering over knowledge bases. In: European Semantic Web Conference. Springer, Berlin, pp. 714–728 (2018)
13. Singh, K., Lytra, I., Radhakrishna, A.S., Shekarpour, S., Vidal, M.-E., Lehmann, J.: No one is perfect: analysing the performance of question answering components over the dbpedia knowledge graph. arXiv preprint arXiv:1809.10044 (2018)
14. Trivedi, P., Maheshwari, G., Dubey, M., Lehmann, J.: Lc-quad: a corpus for complex question answering over knowledge graphs. In: International Semantic Web Conference. Springer, Berlin, pp. 210–218 (2017)
15. Dubey, M., Banerjee, D., Abdelkawi, A., Lehmann, J.: Lc-quad 2.0: a large dataset for complex question answering over wikidata and dbpedia. In: International Semantic Web Conference. Springer, Berlin. pp. 69–78 (2019)
16. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. VLDB J. **2019**, 1–25 (2019)
17. Unger, C., Forascu, C., Lopez, V., Ngomo, A.-C.N., Cabrio, E., Cimiano, P., Walter, S.: Question answering over linked data (QALD-4) (2014)
18. Hasnain, A., Mehmood, Q., Zainab, S.S., Saleem, M., Warren, C., Zehra, D., Decker, S., Rebholz-Schuhmann, D.: Biofed: federated query processing over life sciences linked open data. J. Biomed. Semant. **8**, 13 (2017)

19. Amer-Yahia, S., Koutrika, G., Braschler, M., Calvanese, D., Lanti, D., Lücke-Tieke, H., Mosca, A., de Farias, T.M., Papadopoulos, D., Patil, Y., et al.: INODE: building an end-to-end data exploration system in practice. ACM SIGMOD Rec. **50**, 23–29 (2021)
20. Sima, A.C., de Farias, T.M., Anisimova, M., Dessimoz, C., Robinson-Rechavi, M., Zbinden, E., Stockinger, K.: Bio-SODA: enabling natural language question answering over knowledge graphs without training data. In: 33rd International Conference on Scientific and Statistical Database Management, pp. 61–72 (2021)
21. Maheshwari, G., Trivedi, P., Lukovnikov, D., Chakraborty, N., Fischer, A., Lehmann, J.: Learning to rank query graphs for complex question answering over knowledge graphs. In: International Semantic Web Conference. Springer, Berlin, pp. 487–504 (2019)
22. Affolter, K., Stockinger, K., Bernstein, A.: A comparative survey of recent natural language interfaces for databases. VLDB J. **28**, 793–819 (2019)
23. Chakraborty, N., Lukovnikov, D., Maheshwari, G., Trivedi, P., Lehmann, J., Fischer, A.: Introduction to Neural Network based Approaches for Question Answering over Knowledge Graphs. arXiv preprint arXiv:1907.09361 (2019)
24. Marginean, A.: Question answering over biomedical linked data with grammatical framework. Semantic Web **8**, 565–580 (2017)
25. Hamon, T., Grabar, N., Mougin, F., Thiessard, F.: Description of the POMELO System for the Task 2 of QALD-2014. CLEF (Working Notes) **1212**, 28 (2014)
26. Hamon, T., Grabar, N., Mougin, F.: Querying biomedical linked data with natural language questions. Semantic Web **8**, 581–599 (2017)
27. Diefenbach, D., Giménez-Garcıa, J., Both, A., Singh, K., Maret, P.: Designing a portable Question Answering System over RDF data, QAnswer KG (2020)
28. Lukovnikov, D., Fischer, A., Lehmann, J., Auer, S.: Neural network-based question answering over knowledge graphs on word and character level. In: Proceedings of the 26th international conference on World Wide Web, pp. 1211–1220 (2017)
29. Deutch, D., Frost, N., Gilad, A.: Explaining Natural Language query results. VLDB J. **29**, 485–508 (2020)
30. Ngomo, A.-C.N., Bühmann, L., Unger, C., Lehmann, J., Gerber, D.: Sorry, I don't speak SPARQL: translating SPARQL queries into natural language. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 977–988 (2013)
31. Kokkalis, A., Vagenas, P., Zervakis, A., Simitsis, A., Koutrika, G., Ioannidis, Y.: Logos: a system for translating queries into narratives. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 673–676 (2012)
32. Diefenbach, D., Thalhammer, A.: Pagerank and generic entity summarization for rdf knowledge bases. In: European Semantic Web Conference. Springer, Berlin. pp. 145–160 (2018)
33. Ferré, S.: Sparklis: an expressive query builder for SPARQL endpoints with guidance in natural language. Semantic Web **8**(3), 405–418 (2017)
34. Diefenbach, D., Migliatti, P.H., Qawasmeh, O., Lully, V., Singh, K., Maret, P.: QAnswer: a Question Answering prototype bridging the gap between a considerable part of the LOD cloud and end-users. In: The World Wide Web Conference, pp. 3507–3510 (2019)
35. Paulheim, H., Bizer, C.: Type Inference on Noisy rdf Data. International Semantic Web Conference, pp. 510–525. Springer, Berlin (2013)
36. Kellou-Menouer, K., Kedad, Z.: Schema discovery in RDF data sources. In: International Conference on Conceptual Modeling. Springer, Berlin. pp. 481–495 (2015)
37. Redaschi, N., Consortium, U., et al.: Uniprot in RDF: Tackling data integration and distributed annotation with the semantic web. Nat. Preced., pp. 1–1 (2009)
38. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical Report, Stanford InfoLab (1999)
39. Distributional semantics resources for biomedical text processing: Moen, S.P.F.G.H., Ananiadou, T.S.S. Proc. LBM **2013**, 39–44 (2013)
40. Gkirtzou, K., Karozos, K., Vassalos, V., Dalamagas, T.: Keywords-to-sparql translation for rdf data search and exploration. In: International Conference on Theory and Practice of Digital Libraries. Springer, Berlin, pp. 111–123 (2015)
41. Bastian, F.B., Roux, J., Niknejad, A., Comte, A., Costa, S., Fonseca, S., De Farias, T.M., Moretti, S., Parmentier, G., De Laval, V.R., Rosikiewicz, M., et al.: The Bgee suite: integrated curated expression atlas and comparative transcriptomics in animals. Nucleic Acids Res. **49**, D831–D847 (2021)

42. Altenhoff, A.M., Train, C., Gilbert, K.J., Mediratta, I., de Farias, T.M., Moi, D., Nevers, Y., Radoykova, H.-S., Rossier, V., Vesztrocy, A.W., et al.: OMA orthology in 2021: website overhaul, conserved isoforms, ancestral gene order and more. Nucleic Acids Res. **49**, D373–D379 (2021)
43. Song, D, Schilder, F., Smiley, C., Brew, C., Zielund, T., Bretz, H., Martin, R., Dale, C., Duprey, J., Miller, T., et al.: TR discover: a natural language interface for querying and analyzing interlinked datasets. In: International Semantic Web Conference. Springer, Berlin, pp. 21–37 (2015)
44. Sima, A.C., de Farias, T.M., Zbinden, E., Anisimova, M., Gil, M., Stockinger, H., Stockinger, K., Robinson-Rechavi, M., Dessimoz, C.: Enabling semantic queries across federated bioinformatics databases. Database 2019 (2019)
45. Nadig, S., Braschler, M., Stockinger, K.: Database Search vs. Information Retrieval: A Novel Method for Studying Natural Language Querying of Semi-Structured Data. In: International Conference on Language Resources and Evaluation (LREC) (2020)
46. Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al.: Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint arXiv:1809.08887 (2018)
47. Blunschi, L., Jossen, C., Kossmann, D., Mori, M., Stockinger, K.: Soda: generating sql for business users. Proc. VLDB Endowm. **5**, 932–943 (2012)

Authors and Affiliations

Ana Claudia Sima[1] · Tarcisio Mendes de Farias[1,2,3] · Maria Anisimova[1,4] · Christophe Dessimoz[1,2,5,6] · Marc Robinson-Rechavi[1,3] · Erich Zbinden[1,4] · Kurt Stockinger[4]

1 SIB Swiss Institute of Bioinformatics, Lausanne, Switzerland

2 Department of Computational Biology, University of Lausanne, Lausanne, Switzerland

3 Department of Ecology and Evolution, University of Lausanne, Lausanne, Switzerland

4 ZHAW Zurich University of Applied Sciences, Zurich, Switzerland

5 Department of Genetics, Evolution, and Environment, University College London, London, UK

6 Department of Computer Science, University College London, London, UK