Answering End-User Questions, Queries and Searches on Wikipedia and its History

Maurizio Atzori
University of Cagliari
atzori@unica.it

Shi Gao
UCLA
gaoshi@cs.ucla.edu

Giuseppe M. Mazzeo
UCLA
mazzeo@cs.ucla.edu

Carlo Zaniolo
UCLA
zaniolo@cs.ucla.edu

Abstract

Knowledge bases (KBs) encoded using RDF triples deliver many benefits to applications and programmers that access the KBs on the web via SPARQL endpoints. In this paper, we describe and compare two user-friendly systems that seek to make the universal knowledge of Web KBs available to users who neither know SPARQL, nor the internals of the KBs. We first describe CANaLI, that lets people enter Natural Language (NL) questions and translates them into SPARQL queries executed on DBpedia. CANaLI removes the ambiguities that are often present in NL communication by requiring the use of a Controlled NL and providing on-line knowledge-driven question-completion that shows alternate correct interpretations. While CANaLI is a very powerful NL system, which placed first in the 2016 competition on Question Answering over Linked Data QALD-6, even more powerful user-friendly interfaces are available to users who enter questions and queries on web-browsers. In particular, the SWiPE system provides a wysiwyg interface that lets users specify powerful queries on the Infoboxes of Wikipedia pages in a query-by-example fashion. Thus, in addition to those supported in CANaLI, we now have queries with (i) complex aggregates, (ii) structured conditions combined with keyword-based searches, and (iii) temporal conditions on Cliopedia, a historical knowledge base that captures the evolution of Wikipedia entities and properties. These systems demonstrate that semi-curated web document corpora and their KBs are making possible the seamless integration through user-friendly interfaces of (i) NL question answering, (ii) structured DB queries, and (iii) information retrieval. These were once viewed as distinct functions supported by different enabling technologies.

1 Introduction

Knowledge bases (KBs) are playing a crucial role in many applications, such as text summarization and classification, opinion mining, semantic search, and question answering systems. In recent years, several projects have been devoted to creating such KBs of general nature or specialized focus: [23, 34, 35, 36, 37, 39, 41, 42]. Of particular interest are KBs that are closely connected with curated or semicurated document corpora, such as Wikipedia. In fact, most of Wikipedia pages use an Infobox to summarize key properties and values of the entities (subjects) described in the pages. Now, DBpedia [43] has harvested and organized the information contained in those Infoboxes into a KB using RDF [22] and provides a SPARQL endpoint for accessing the information. The success of the encyclopedic Wikipedia and its associates KBs have motivated the start of thousands

Copyright 2016 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of related projects that cover more specialized domains [38], and provide RDF KBs accessible via SPARQL or other structured query languages.

However, the great majority of web users are neither familiar with SPARQL nor with the internals of the KBs and are thus denied access to these KBs and the many benefits they offer. Therefore, the design of user-friendly interfaces to deliver the riches of web KBs to all users has emerged as a challenging research priority of great technical interest and practical significance. The importance of the topic has inspired a large body of previous research and the launching of an annual competitions on Question Answering over Linked Data (QALD). In this paper, we describe the Natural Language (NL) QA system CANaLI [25], that placed first in the 2016 competition and also supports a KB-driven query completion function that assists users in formulating their questions.

While NL QA interfaces often provide the most comfortable communication medium for casual users, other friendly interfaces are available through web browsers. In particular, the SWiPE system provides a wysiwyg interface [2] where *by-example structured queries* (BEStQ) are entered as follows: (i) the user selects an example page activating its Infobox, (ii) the user can now insert conditions into the relevant fields of the Infobox, and (iii) these conditions are translated into a SPARQL query that is executed on the DBpedia KB. We will discuss this system in some depth since it allows users to ask powerful queries and questions that are currently not available in NL QA systems, including (a) queries requiring complex conditions and aggregates, (b) questions combining structured queries with keyword searches, and (c) questions and queries on the history of the KB. We will underscore the significance of these new types of questions and queries which suggest important new goals for research and CANaLI's extensions.

2 The Design of CANaLI

The importance and the difficulty of NL QA is witnessed by the large amount of research efforts devoted to this problem [12, 13, 28]. Answering questions posed in NL requires to tackle several non-trivial sub-problems, such as deriving the syntactic structure of the question, associating the phrases of the question with the resources of the KB, and resolving the ambiguities that are quite common in these two tasks (inasmuch as different concepts can be represented using the same phrase and several syntactical relationships are possible between the constituents of the sentence). Ambiguity resolution is indeed a very hard problem, sometimes even for humans[1], and the domain knowledge, usually available to the speakers but not to the system, is fundamental in order to disambiguate the query intention.

We designed our NL system, CANaLI [25], with the objective of avoiding ambiguities. This goal was achieved by combining the use of a *controlled natural language* (CNL) interface with an interactive autocompleter that guides the user in typing questions that are consistent with the underlying KB.

The use of a CNL [20] reduces the possibility of ambiguities in interpreting the syntactical relationships between the constituents of the sentence, by limiting the syntactical forms that can be accepted. The key challenge of these systems is to be able to accept a language that is formal enough to be interpreted by machines with high accuracy, and, at the same time, natural enough to be readily acquired by people as an idiomatic version of their NL. The CNL used by CANaLI, besides being flexible enough to support the typical questions that users may want to pose over a KB, enables a very desirable question autocompletion function: although it is very popular in Web search engines, autocompletion is a novelty for CNL systems. Moreover, while autocompletion in search engines is based on the popularity of searches, the CANaLI autocompletion strategy is based on the underlying KB. This feature allows people to feel more comfortable while entering the questions, since they are (i) guided in following the grammar accepted by CANaLI whereby users will only need minimal knowledge of the CNL accepted by the system before they can start using it, and (ii) made aware that alternative interpretations are possible for a question. Thus, while the main benefit of (i) is improving interaction with the CNL interface, (ii) would be desirable in any NL system to forewarn the users of potential ambiguities in their questions.

[1]This problem can be exemplified by the sentence: "I saw the man on the hill with a telescope".

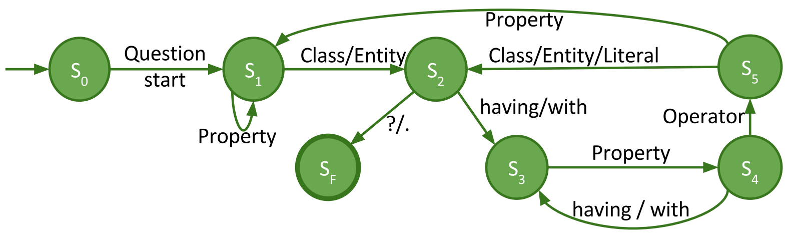


Figure 1: The main states and transitions of the finite-state automaton used by CANaLI

2.1 Answering Simple Questions

Figure 1 depicts the main states and transitions of the finite-state automaton used by CANaLI[2]. We provide an intuition of the operation of CANaLI by means of some simple examples. Let us consider the question: "Who is the spouse of Barack Obama?". The automaton is initially in the state S_0, ready to accept tokens representing the *question start*. The token "Who is the" represents a valid way to start a question; thus the automaton accepts it, and moves to the state S_1, where it is ready to accept a token representing either a *property*, or an *entity*, or a *class*. In our example, the user enters "spouse", that is a *property* recognized by CANaLI. Thus, the system loops back to S_1, ready to accept another *property*, *entity*, or *class*. In our example, the user enters "Barack Obama" that, being an *entity* with the *property* "spouse," is accepted by the system. In general, in order to be consistent with the underlying KB (*DBpedia*, in our example), the user must enter entities that have a spouse, otherwise the system will stop the user from progressing any further. However, to reach this 'no progress' point the user must have ignored the set of previously generated suggestions, shown as valid completions of the user's input just under the input window: if the user selects and clicks on any such completion its text is added to the input window. Going back to our example, the input of the token "Barack Obama" triggers the transition to S_2, where a range of new input tokens can be accepted, including the *question mark*, which marks the end of the question, and launches the translation of the sequence of accepted tokens to a SPARQL query followed by its execution.

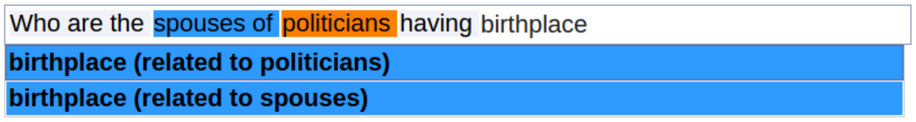
Let us now consider another example: "What is the alma mater of the spouse of Barack Obama?". In this case, at S_1, the user would input the *property* "alma mater", whereby the system loops back to S_1, where it accepts the second *property*: "spouse". CANaLI accepts this 'chain' of properties because it is consistent with the underlying KB, which contains cases of "spouses" with an "alma mater". Now, the system is still in S_1, where the question is completed like in the previous example. These two simple examples show how the four basic states S_0, S_1, S_2, and S_F, support a large set of very common 'factual' questions asked by everyday users[3].

More complicated but nevertheless common questions are those adding constraints, i.e., query conditions. For instance, assume that the user wants to ask[4]: "Who are the spouses of politicians having birthplace equal to United States?". After the input of the fragment "Who are the spouses of" has taken the automaton to S_1, the token "politicians" is accepted as a *class* with "spouse" as a valid *property*, and this moves the automaton to S_2. In S_2, CANaLI can accept "having", and other uninterpreted connectives used as syntactic sugar, to move to S_3, where it will accept only a *property* related to a previously accepted token. In this case, "birthplace" can be accepted since "spouses" have this *property*. This example illustrates the ambiguities that beset all NL interfaces, no matter how sophisticated their parser is. In fact, also "politicians" have "birthplace" as a valid *property*. CANaLI tackles this problem by means of its interactive autocompletion system, that displays in real time all alternative interpretations that are compatible with the underlying KB. For instance, in the case at hand, CANaLI will display the two alternatives shown in Figure 2.

[2]The system response, indeed, is based on the context provided by the question typed so far and the underlying ontology, rather than just the current state and last token as a finite-state automaton would.

[3]Indeed, the factual questions asked most frequently on the web are definition questions (e.g., What is the EU?), that are even simpler.

[4]This provides a good example of the broken but effective English now supported by CANaLI.


Who are the spouses of politicians having birthplace

birthplace (related to politicians)

birthplace (related to spouses)

Figure 2: The autocompleter of CANaLI suggesting properties that can be related to spouses/politicians and having thelast word typed by the user, i.e., “birthplace”, in their label

Once the user has explicitly made a choice by clicking on the result corresponding to her intention, the automaton moves to the state S_4, that expects an operator of some kind. Thus, the user can input “equal to”, and the automaton moves to state S_5, that accepts the right-hand side of the constraint. In general, the right hand side of a constraint can be an element of the KB or a *literal*, as long as it is a valid value for the previously accepted *property*. In our example, the *entity* “United States” can be accepted, since it is a valid value for the *property* “birthplace”. Now that the automaton is in S_2 again, the user can specify more constraints, or input the question mark, ending the question.

2.2 Experimental evaluation

A popular set of QA benchmarks has evolved as result of the annual QALD (Question Answering over Linked Data) challenge [40]. The benchmarks consist of sets of NL questions, each associated with its gold standard formulation in SPARQL, that produces answers against which the accuracy of the answers returned by the system under test are evaluated. The evaluation uses precision, recall and the F-measure that combines the two as: $F = (2 \times Precision \times Recall) \div (Precision + Recall)$. CANaLI entered the 2016 contest, QALD-6. On DB-pedia questions CANaLI ranked first with an F-measure of 88%—i.e., 13% above the second-placed system, and with wider difference over the other competitors, including those that will be discussed in Section 5. CANaLI also performed well on previous QALD testbeds, including those based on MusicBrainz [39] KB and the three biomedical KBs: DrugBank [35], Diseasome [42], and SIDER [41]. In fact on these testbeds CANaLI performed as well or better than other systems in terms of precision, recall and F-measure.

2.3 Interacting Using a Controlled Natural Language

Given that CANaLI proved so accurate, it is natural that one should wonder about the extent in which restrictions imposed by the CNL makes it less user friendly than a full natural language interface. To answer this question, we show in Table 1 a sample of questions taken from previous QALD challenges, and their rephrased forms that were accepted by CANaLI.

Clearly, in all these questions, a reasonable rephrasing effort is demanded to the user. However, we found that such a rephrasing becomes quite natural once users understands that CANaLI expect questions on concepts or properties of concepts, expressed as nouns—plus optional specifications on properties that the concepts have. For instance, “Who is the creator of Captain America” is a question about a noun and thus accepted by CANaLI, whereas “who created Captain America” is not (see Q1). Likewise, “What is the height of Michael Jordan” is proper rephrasing for “how tall is Michael Jordan” (Q2), and “What is the date of Halloween” is the one for “when is Halloween” (Q3); likewise “What is the number of students of a University” is the rephrasing needed for “how many students does a University have” (Q4). Q4 presents an apparent difficulty, given by the actual name (used in DBpedia) of the Free University of Amsterdam. The autocompleter, in this case, helps the user by suggesting the correct denomination of the university. Q5 shows how constraints must be specified by using the properties. In this case, the “German lakes” are those lakes “having country Germany”. The same limitation holds on questions that require counting. Thus, in Q6, in order to know “how often Nicole Kidman married”, we need to count the number of values taken by this talented actress’ property “spouse”. Q7 shows an example of questions with two constraints. The preposition “by”, quite common in natural language, is indeed ambiguous,

	Original question	Question rephrased for CANaLI
Q1	Who created the comic Captain America?	Who is the creator of Captain America?
Q2	How tall is Michael Jordan?	What is the height of Michael Jordan?
Q3	When is Halloween?	What is the date of Halloween?
Q4	How many students does the Free University in Amsterdam have?	What is the number of students of VU University Amsterdam?
Q5	Which rivers flow into a German lake?	What are the rivers flowing into lakes having country Germany?
Q6	How often did Nicole Kidman marry?	What is the count of spouse of Nicole Kidman?
Q7	Give me all books by William Goldman with more than 300 pages.	Give me the books having author William Goldman and number of pages greater than 300.
Q8	Does Abraham Lincoln's death place have a website?	Is there a website of death place of Abraham Lincoln?
Q9	Who is the youngest Darts player?	Who is the darts player with the greatest birth date?
Q10	Give me all actors who were born in Paris after 1950.	Give me the actors born in Paris having birth date greater than 1950-12-31.

Table 1: Questions taken from QALD-3 and QALD-4 challenges and their rephrasing for CANaLI

and only the background knowledge of the fact that "William Goldman" is an author (and not and editor, for instance) can disambiguate its corresponding meaning. The same principle holds for Yes/No questions, like Q8. The user must ask if a concept there exists, in this case the property "website" of the property "death place" of "Abraham Lincoln". Q9 gives an idea of how superlatives must be rephrased. In this case, the "youngest" player is a player with the "greatest birth date". Q10, instead, shows how constraints on dates must be rephrased (the autocompleter guides in typing dates in the standard format `yyyy-MM-dd`).

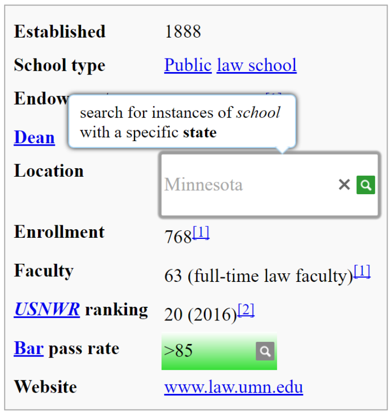
This analysis allows us to conclude that most questions require some reformulation in order to be accepted by CANaLI. However, we observe that (i) users quickly gain the needed skills once they start working with the system, and (ii) the introduction synonyms for concepts in the KB, and the addition of few simple rules to extend the grammar recognized by the system can go a long way toward making the interface of the system more natural, while still avoiding ambiguities. While these improvements are likely to confirm CANaLI as a leading CNL for DBpedia and, RDF KBs in general, the SWiPE system discussed next, can supports advanced queries requiring (i) complex aggregates, (ii) a mixture of structured and keyword conditions, and (iii) historical queries, i.e., queries that pose difficult research challenges for NL systems.

3 SWiPE: Query by Example on Wikipedia

The very Infoboxes that have been the source of the knowledge stored in DBpedia can be turned into active forms on which powerful queries can be specified using the By-Example Structured Query (BEStQ) approach of SWiPE [2, 3].

As a running example, let us suppose a user from Minneapolis is looking for a Law School with some desired requirements. She can start her search from the Wikipedia page of any Law school containing the Infobox: for instance the *University of Minnesota Law School* page. Then, by clicking on the SWiPE *bookmarklet*[5] the original Wikipedia page will be refreshed, showing the same page with the Infobox turned into an active form that allows the user to enter the desired conditions (as shown in Fig. 3(a)), plus a personalized toolbar showing at the bottom of the page (called SWiPE *bottom bar*) with additional features. In our case, the user wants to find Law schools located in New York with a high bar pass rate. She will then enter $>$`85%` as a first constraint for the *Bar pass rate* and `New York` in the *Location* field (replacing the original values in the Infobox). It is

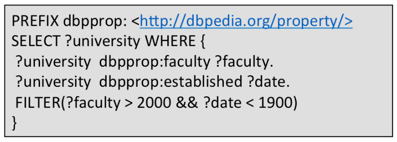
[5]A special browser bookmark that actively interacts with the current web page



Established	1888
School type	Public law school
Endow	
Dean	
Location	Minnesota
Enrollment	768[1]
Faculty	63 (full-time law faculty)[1]
USNWR ranking	20 (2016)[2]
Bar pass rate	>85
Website	www.law.umn.edu

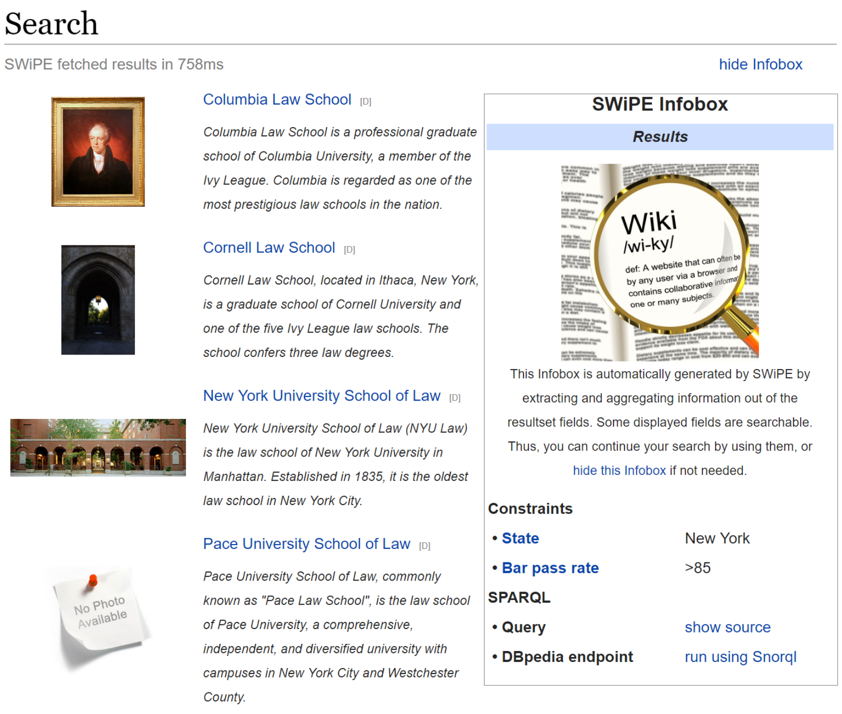
search for instances of *school* with a specific **state**

(a)



```
PREFIX dbpprop: <http://dbpedia.org/property/>
SELECT ?university WHERE {
 ?university  dbpprop:faculty ?faculty.
 ?university  dbpprop:established ?date.
 FILTER(?faculty > 2000 && ?date < 1900)
}
```

(c)



Search

SWiPE fetched results in 758ms

hide Infobox

Columbia Law School [D]

Columbia Law School is a professional graduate school of Columbia University, a member of the Ivy League. Columbia is regarded as one of the most prestigious law schools in the nation.

Cornell Law School [D]

Cornell Law School, located in Ithaca, New York, is a graduate school of Cornell University and one of the five Ivy League law schools. The school confers three law degrees.

New York University School of Law [D]

New York University School of Law (NYU Law) is the law school of New York University in Manhattan. Established in 1835, it is the oldest law school in New York City.

Pace University School of Law [D]

Pace University School of Law, commonly known as "Pace Law School", is the law school of Pace University, a comprehensive, independent, and diversified university with campuses in New York City and Westchester County.

No Photo Available
Wiki
/wi-ky/
def: A website that can often be by any user via a browser and contains collaborative information one or many subjects

SWiPE Infobox

Results

This Infobox is automatically generated by SWiPE by extracting and aggregating information out of the resultset fields. Some displayed fields are searchable. Thus, you can continue your search by using them, or hide this Infobox if not needed.

Constraints

• **State**	New York
• **Bar pass rate**	>85

SPARQL

• **Query**	show source
• **DBpedia endpoint**	run using Snorql

(b)

Figure 3: A QBE query on a university Infobox (a), its results (b) and a SPARQL query generated by SWiPE (c)

worth noting that the Location field in Wikipedia is recognized to correspond to the *state* property in DBpedia, as shown in the popup tooltip in Fig. 3(a) just before overwriting the original value "Minnesota" with the user constraint "New York". SWiPE also features an autocompletion-like functionality to help the user with the proper spelling for values[6]. The user may also choose to sort results based on some fields, for instance the *US-NWR ranking*. Upon clicking on the green *lens* button available in each field and also in the Bottom Bar, SWiPE converts this two-line BEStQ specification into the equivalent (22-line long) SPARQL query. This is made possible by our on-the-fly algorithm that matches Infobox labels and values to the appropriate DBpedia RDF properties (`http://dbpedia.org/property/state` and `http//dbpedia.org/ontology/barPassRate` in the case at hand), greatly simplifying the user search experience by removing the burden of manually finding the correct properties and property names in the underlying KB. Finally, the SPARQL query is executed against our instance of DBpedia, on a Virtuoso server; results are then reformatted by SWiPE in a Wikipedia-like layout and presented to the user as shown in Fig. 3(b).

Since the original SWiPE prototype presented in [2], a number of improvements and new features have been added, including joins, aggregates, and the keyword-based retrieval capabilities of free-text search engines. Joins allow to input constraints on a different page w.r.t. the example pages used to start the search. For instance, in the previous example, the user may want to find only Law Schools belonging to universities established before 1900 and with a faculty size above 2000. These constraints will be entered on another example page, related to a University, and results will be filtered accordingly. The first page of the returned results also shows an Infobox that summarizes the query, and provides links to meta-information, such as the SPARQL query used, which for the example at hand is shown in Fig. 3(c).

Another example that we have frequently used [5] is that of finding cities with certain properties. A mobile

[6]This is an important feature to avoid unexpected empty results for constraints such as "USA" not matching DBpedia values stored as "United States"

version has been also developed [5], simplifying geolocation searches by featuring a touch interface for with location-based ranges queries. For that, our user only needs to find in Wikipedia the page of a familiar city, and then replace in its Infobox the existing values of the desired properties with conditions that specify the query. This is the well-known Query-By-Example (QBE) approach that is credited with bringing ease-of-use to relational databases and is even more desirable here, since it shields the users from having to discover the internal names and organization of DBpedia. It is also quite powerful, since it supports the specification of queries involving joins and aggregates.

Moreover, the user can still enter text conditions in the standard search box of Wikipedia to find the pages that satisfy both the structured query and the standard keyword-based retrieval made popular by web search engines. Our experiments show that this combination yields very powerful queries producing selective high-precision answers [3]. In order to detail the combination of keyword-based and structure, in addition to the two conditions previously entered in the Infobox, our prospective law student may also enter the words "`ivy league`" in the search box shown in the SWiPE bottom bar. As a result, SWiPE will provide the list of all Wikipedia pages satisfying both the structured conditions in the Infobox and those in the search box. Therefore, the last two entries in Figure 3(b) will no longer be in the answer, since 'New York University' and 'Pace University' are not in the Ivy League. This simple example clarifies the dramatic improvements in precision and recall delivered by BEStQ searches with respect to traditional keyword-only searches. In fact we claim that the combination of structured queries and keyword based searches routinely delivers accuracy levels (i.e., F-measure values) that are very hard to achieve with keywords alone, even when long-tail keywords and related optimization techniques are used. For instance, by using the following keyword combination "*Ivy League Law Schools New York bar pass rate,*" major search engines return hundreds of thousands of results, while Wikipedia gets a total of 53 answers, including some very surprising ones such as '*Christmas*', '*List of the Cosby Show episodes*' and '*List of Batman: The Brave and the Bold Character*', each of which contains all the specified keywords but is totally unrelated to the topic of interest.

In addition to supporting structured and keyword searches combined, the BEStQ approach supports well sorting, aggregate queries and historical queries that are not easily supported in CANaLI. While aggregate queries are simply implemented using an approach similar to that used in QBE, support for historical queries required nontrivial temporal extensions, that are described in [27] and summarized in the next section.

4 Cliopedia and SPARQL^T: Managing the History of KBs

As the real world evolves and the KBs are updated, the history of entities and their properties becomes of great interest. Table 2 shows the edit history of some attributes in Wikipedia Infoboxes. Such updates are quite common in many properties: e.g., on the average each value in the population property of the city pages is updated more than 7 times. This is not specific to Wikipedia, but also happens in other knowledge repositories such as Yago [23] and GovTrack [37]. Thus, users need query tools of comparable power and usability to explore such histories and flash-back to the past.

There is in fact a growing recognition of the importance of managing and querying the evolution history of knowledge bases in the technical literature, and several approaches [14, 21, 29, 30] have been proposed to support the queries on temporal RDF datasets.

In [8, 9] we proposed a vertically integrated system, RDF-TX (RDF Temporal eXpress), that efficiently supports the data management and query evaluation of large temporal RDF datasets while simplifying the temporal queries for SPARQL programmers and consequently, for end-user interfaces facilitating the expression of the same queries. To support the queries over the evolution history of KBs, we proposed efficient storage and index schemes for temporal RDF triples using multiversion B+ tree [4] and implemented a query engine which achieves fast query evaluation by taking advantage of comprehensive indices. We also built a query optimizer that generates efficient join orders using a cost-based model that exploit statistics collected on temporal RDF graphs. Thus,

Category	Property	Average Number of Updates
Software	Release	7.27
Player	Club	5.85
Country	GDP(PPP)	11.78
City	Population	7.16

Table 2: Statistics of Wikipedia Infobox Edit History

in RDF-TX, we provide a general and scalable solution for the problem of managing and querying massive temporal RDF by providing:

- A user-friendly BEStQ interface to view and query the history of knowledge base. Users can specify queries by entering simple conditions using a point-based temporal model. The system then translates these queries into equivalent SPARQL^T queries described next.
- SPARQL^T, a temporal extension of the structured query language SPARQL which simplifies the expression of temporal joins and eliminates the need for temporal coalescing.
- An efficient main memory system RDF-TX for managing temporal RDF data and evaluating SPARQL^T queries using a multiversion B+ tree (MVBT). This stores indexed temporal RDF triples in combination with an effective delta encoding scheme to reduce the storage overhead. RDF-TX also features a query optimizer that generates efficient join orders using the statistics of temporal RDF graphs.

4.1 Temporal RDF and By-Example Temporal Queries

Temporal RDF Model. RDF KBs can be presented as (subject, predicate, object) triples, which work well for static information, but not for the evolution history. For that we instead use the Temporal RDF model [14] that extends the RDF Graph with temporal elements. In the temporal RDF model, each (s, p, o) triple is annotated with a time element, producing the quadruplet (s, p, o, t). Table 3 shows the temporal RDF triples for Wikipedia Infobox of *San Diego*.

Consider the fact that Bob Filner served as the mayor of San Diego from December 4, 2012 to August 30, 2013. This fact can be represented as: ⟨*San Diego*, *mayor*, *Bob Filner*⟩:[*12/04/2012 . . . 08/30/2013*], while [*12/04/2012 . . . 08/30/2013*] represents all the days between 12/04/2012 and 08/30/2013. DAY provides the basic granularity in out temporal model.

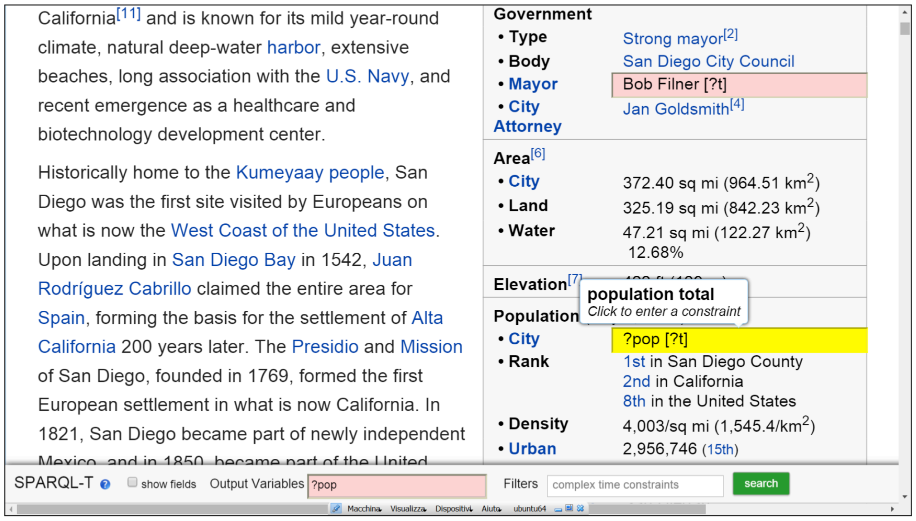
Note that we adopt the point-based temporal model that dovetails with our BEStQ interface and simplifies the expression of temporal queries at the logical level; however at the physical level we retain the interval representation for efficiency reasons. Queries on the point-based model can be easily mapped into equivalent ones on the interval-based model for execution.

By-Example Temporal Queries. The Wikipedia Infobox of San Diego, clearly shows the current mayor, but not the past mayors, nor it allows us to find who was the mayor at certain date, nor to find the total population of the city when Bob Filner was mayor. To provide answers to these and similar questions, we have developed a system that extends Wikipedia Infobox with historical information and extend SPARQL and SWiPE with the ability of asking such temporal queries.

Our system support historical Infoboxes, where once a field is selected a pull down menu opens showing the history of that field. Thus, by selecting the mayor field in the Infobox the user actually sees its history with the last four mayors of San Diego. But in addition to viewing former values in the history of the entity properties, our user might enter query conditions in the fields of the Infobox that are identified as active fields by their highlighted backgrounds. Then, say that our user who saw the previous mayor in the pull-down menu of San Diego, now wants to find its population at the time when Bob Filner served as the mayor. For that, the user

Predicate	Object	Timestamp
Mayor	Bob Filner	12/04/2012 ... 08/30/2013
	Todd Gloria	08/31/2013 ... 03/02/2014
	Kevin Faulconer	03/03/2014 ... *now*
Population	1322553	12/19/2012 ... 10/01/2013
	1307402	10/02/2013 ... 04/29/2014
	1345895	04/30/2014 ... 05/21/2015
	1381069	05/22/2015 ... *now*

Table 3: Temporal RDF Triples for *San Diego*


California[11] and is known for its mild year-round climate, natural deep-water harbor, extensive beaches, long association with the U.S. Navy, and recent emergence as a healthcare and biotechnology development center.

Historically home to the Kumeyaay people, San Diego was the first site visited by Europeans on what is now the West Coast of the United States. Upon landing in San Diego Bay in 1542, Juan Rodríguez Cabrillo claimed the entire area for Spain, forming the basis for the settlement of Alta California 200 years later. The Presidio and Mission of San Diego, founded in 1769, formed the first European settlement in what is now California. In 1821, San Diego became part of newly independent Mexico, and in 1850, became part of the United

Government	
• Type	Strong mayor[2]
• Body	San Diego City Council
• Mayor	Bob Filner [?t]
• City Attorney	Jan Goldsmith[4]
Area[6]	
• City	372.40 sq mi (964.51 km²)
• Land	325.19 sq mi (842.23 km²)
• Water	47.21 sq mi (122.27 km²) 12.68%
Elevation[7]	
Population	
• City	?pop [?t]
• Rank	1st in San Diego County 2nd in California 8th in the United States
• Density	4,003/sq mi (1,545.4/km²)
• Urban	2,956,746 (15th)

Figure 4: What was the population of San Diego when Bob Filner was the mayor?

can use the page of any city, including San Diego which is shown in Figure 4. Since all the fields are editable, the user enters "Bob Filner" in the *Government Mayor* and variable "?pop" in *City Population* and these two Infoboxes are set with the same temporal variable "?t" to indicate the temporal join.

Then our system generates and executes a SPARQL^{T} query, as discussed later. Again, it is important to remember that this query could have been entered in the Infobox of any city, and indeed since in our query conditions "San Diego" is not specified, our query will return the population of every city where Bib Filner served as the mayor.

4.2 SPARQL^{T} Query Language

To query the temporal RDF graphs, we propose a temporal extension of SPARQL called SPARQL^{T}. A SPARQL query consists of triple query patterns $\{s\ \ p\ \ o\}$, which corresponds to the RDF model $\{subject\ \ predicate\ \ object\}$. Similarly, SPARQL^{T} query is a set of SPARQL^{T} query patterns $\{s\ \ p\ \ o\ \ t\}$ where t refers to the temporal element in temporal RDF model. SPARQL^{T} supports all kinds of temporal queries and Allens operations [1]. Temporal selection queries that retrieve information about a previous snapshot of the KB can be easily expressed in SPARQL^{T} using one query pattern, as shown in Example 1.

EXAMPLE 1. When did Bob Filner served as the mayor of San Diego.
SELECT [?t]
{**San_Diego mayor Bob_Filner ?t**}

More complex queries often use temporal joins; in SPARQL^T joins are expressed by query patterns that share the same temporal element. For finding the population of San Diego when Bob Filner was mayor we write:

EXAMPLE 2. Find the population of San Diego when Bob Filner served as the mayor of San Diego.
SELECT ?pop [?t]
{**San_Diego mayor Bob_Filner ?t .**
San_Diego population ?pop ?t . }

SPARQL^T also introduces a set of built-in functions to faciliate the expression of complex temporal conditions, such as TSTART and TEND. Because of lack of space, we refer our reader to [9] for more query examples and complete semantics.

4.3 RDF-TX Query Engine

Previous works [14, 21, 29, 30] rely on relational databases/RDF engines to store and query temporal RDF triples, which results in complex and inefficient evaluation for temporal queries. Moreover, the indices for accelerating the processing of temporal queries proposed in the past only support a limited set of temporal queries. These limitations have been overcome by RDF-TX, which integrates indexing and query evaluation as follows:

Storage and Index. We choose in-memory MVBT indices to store temporal RDF and support fast SPARQL^T query evaluation. MVBT is a general temporal index that supports fast data update and range search. However, it comes with the large storage overhead, which is resolved in RDF-TX by applying delta compression between index entries.

Query Evaluation. Given a SPARQL^T query, we first parse the query and generate an execution plan in which every SPARQL^T query pattern is converted into a query pattern (k, i) on MVBT to retrieve all the temporal RDF triples with keys in range k and intervals overlapping interval i.

Optimization. In complex queries with multiple query patterns, inefficient execution plans may lead to large intermediate results and significantly slow down execution. Thus we introduce a query optimizer which use statistical estimations to find efficient join orders for complex queries. We introduced a temporal aggregate index, CMVSBT, which provides fast statistics estimation with a small storage overhead. Then, using CMVSBT, the optimizer employs the bottom-up dynamic programming strategy [26] to find the cost-optimal query plans.

5 Related Work

The problem of supporting user-friendly interfaces to DBpedia and other KBs has been widely recognized as important and attracted many research approaches, which because of space limitations we can only mention in a very succinct and incomplete fashion.

Approaches such as *exploratory browsing* [17] and *faceted search* [15, 16] allow users to formulate complex selection conditions, whereas SWiPE can also support joins and aggregates [2]. The NL approach is the one that, so far, has received most attention [12, 13, 20, 28]. These systems are remarkable, not only because of their number, but also because they use techniques that are quite diverse and tested at very different levels of F-measure on QALD testbeds. Among such systems we find **Xser** [32], **gAnswer** [33], **CASIA** [19], **Aqqu** [18], **Intui3** [6], **RTV** [10], besides **Squall2sparql** [7] where the sentences must be annotated by users, and **GFMed** [24] which is a CNL system specialized for the biomedical domain.

There has also been a significant amount of previous work on historical KBs. For instance, Gutierrez et al. [14] extended the RDF model with time elements and several approaches [11, 21, 29, 30] have been proposed to support the queries on temporal RDF datasets. Most previous works employ relational databases and RDF engines to store temporal RDF triples and rewrite temporal queries into SQL/SPARQL for evaluation. The languages proposed in those works use an interval-based temporal model which leads to complex expressions for temporal queries, e.g., those requiring joins and coalescing [31]. At the physical level, previous approaches exploit indexes such as tGrin [29] to accelerate the processing of simple temporal queries, but they do not explore the use of general temporal indices and query optimization techniques.

6 Conclusions and Further Work

In this paper, we have described systems that let non-programmers access RDF KBs using either NL or friendly by-example interfaces. These systems integrates into simple and powerful framework the functions of question answering, query computation, and keyword-based document searching, which were traditionally viewed as separate functions supported by different technologies. Our current work focuses on improving the naturalness of CANaLI by careful addition of synonyms, while making sure that no ambiguity is introduced. Extensions to include keywords in NL questions, and to support historical questions are also under investigation. Combining structured Infobox conditions with CNL queries represents a topic for longer term research.

7 Acknowledgements

This research was supported in part by a 2015 Google Faculty Research Award.

References

[1] J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Commun. ACM*, Vol. 26(11):832–843, 1983

[2] M. Atzori and C. Zaniolo. SWiPE: Searching WikiPedia by Example. *WWW (Companion Volume)*, 309–312, 2012.

[3] M. Atzori and C. Zaniolo. Expressivity and Accuracy of By-Example Structured Queries on Wikipedia. *WETICE*, 239–244, 2015.

[4] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An Asymptotically Optimal Multiversion B-tree *VLDB*, 264–275, 1996

[5] A. Dessi, A. Maxia, M. Atzori and C. Zaniolo. Supporting semantic web search and structured queries on mobile devices. *Semantic Search over the Web (SSW 2013), VLDB 2013 Workshops*, 5:1–5:4, 2013.

[6] C. Dima. Answering Natural Language Questions with Intui3. *Working Notes for CLEF 2014 Conference*, 1201–1211, 2014

[7] S. Ferré. SQUALL: The expressiveness of SPARQL 1.1 made available as a controlled natural language. *Data Knowledge Engineering*, Vol. 94(B): 163–188, 2014

[8] S. Gao, M. Chen, M. Atzori, J. Gu, and C. Zaniolo. SPARQLT and its User-Friendly Interface for Managing and Querying the history of RDF knowledge base. *ISWC*, 2015

[9] S. Gao, J. Gu, and C. Zaniolo. RDF-TX: A Fast, User-Friendly System for Querying the History of RDF Knowledge Bases. *EDBT*, 269–280, 2016

[10] C. Giannone, V. Bellomaria, and R. Basili. A HMM-based Approach to Question Answering against Linked Data. *Working Notes for CLEF 2013 Conference*, 2013

[11] F. Grandi. T-SPARQL: a TSQL2-like Temporal Query Language for RDF. *GraphQ*, 21–30, 2010

[12] B. F. Green Jr., A. K. Wolf, C. Chomsky, and K. Laughery. Baseball: An Automatic Question-answerer. *Western Joint IRE-AIEE-ACM Computer Conference*, 1961.

[13] P. Gupta and V. Gupta. A Survey of Text Question Answering Techniques. *International Journal of Computer Applications*, Vol. 53(4):1–8, 2012

[14] C. Gutierrez, C. A. Hurtado, and A. A. Vaisman. Introducing Time into RDF. *TKDE*, Vol 19(2):207–218, 2007

[15] J. Guyonvarch and S. Ferré. Scalewelis: a Scalable Query-based Faceted Search System on Top of SPARQL Endpoints. *Working Notes for CLEF Conference* , 2013

[16] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted Wikipedia Search. *Int. Conf. on Business Information Systems*, 2010

[17] L. Han, T. Finin, and A. Joshi. Schema-free structured querying of DBpedia data. *CIKM*, 2012

[18] B. Hannah and H. Elmar. More Accurate Question Answering on Freebase. *CIKM*, 1431–1440, 2015

[19] S. He, Y. Zhang, K. Liu, and J. Zhao. CASIA@V2: A MLN-based Question Answering System over Linked Data. *Working Notes for CLEF 2014 Conference*, 1249–1259, 2014.

[20] T. Kuhn. A Survey and Classification of Controlled Natural Languages. *CoRR*, abs/1507.01701, 2015

[21] E. Kuzey and G. Weikum. Extraction of Temporal Facts and Events from Wikipedia. *TempWeb*, 25–32, 2012

[22] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, Vol. 6(2):167–195, 2015

[23] J .Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia *Artif. Intell.*, 28–61, 2013

[24] A. Marginean. GFMed: Question Answering over BioMedical Linked Data with Grammatical Framework. *Working Notes for CLEF 2014 Conference*, 1224–1235, 2014

[25] G. M. Mazzeo and C. Zaniolo. Answering Controlled Natural Language Questions on RDF Knowledge Bases. *EDBT*, 608-611, 2016

[26] G. Moerkotte and T. Neumann. Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. *VLDB*, 930–941, 2006

[27] H. Mousavi, M. Atzori, S. Gao, and C. Zaniolo. Text-Mining, Structured Queries, and Knowledge Management on Web Document Corpora. *SIGMOD Record 43(3)*, 48–54, 2014

[28] S. R. Petrick. Natural Language Based Computer Systems. *IBM J. Res. Dev.*, Vol. 20(4):314–325, 1976

[29] A. Pugliese, O. Udrea, and V. S. Subrahmanian. Scaling RDF with Time. *WWW*, 605–614, 2008

[30] J. Tappolet and A. Bernstein. Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. *ESWC*, 308–322, 2009

[31] D. Toman. Point vs. Interval-based Query Languages for Temporal Databases. *PODS*, 58–67, 1996

[32] K. Xu, Y. Feng, and D. Zhao. Answering Natural Language Questions via Phrasal Semantic Parsing. *Working Notes for CLEF 2014 Conference*, 1260–1274, 2014

[33] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over RDF: a graph data driven approach. *SIGMOD*, 313–324, 2014

[34] `http://www.cyc.com/platform/opencyc`

[35] `http://www.drugbank.ca/`

[36] `http://www.geonames.org/`

[37] `https://www.govtrack.us/`

[38] `http://linkeddata.org/`

[39] `http://musicbrainz.org/`

[40] `http://www.sc.cit-ec.uni-bielefeld.de/qald`

[41] `http://sideeffects.embl.de/`

[42] `http://wifo5-03.informatik.uni-mannheim.de/diseasome/`

[43] `http://wiki.dbpedia.org/`