Interactive Text Graph Mining with a Prolog-based Dialog Engine

Paul Tarau

Dept. of Computer Science and Engineering
University of North Texas
1155 Union Circle, Denton, Texas 76203, USA
paul.tarau@unt.edu

Eduardo Blanco

Dept. of Computer Science and Engineering
University of North Texas
1155 Union Circle, Denton, Texas 76203, USA
eduardo.blanco@unt.edu

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

On top of a neural network-based dependency parser and a graph-based natural language processing module we design a Prolog-based dialog engine that explores interactively a ranked fact database extracted from a text document.

We reorganize dependency graphs to focus on the most relevant content elements of a sentence and integrate sentence identifiers as graph nodes. Additionally, after ranking the graph we take advantage of the implicit semantic information that dependency links and WordNet bring in the form of subject-verb-object, "is-a" and "part-of" relations.

Working on the Prolog facts and their inferred consequences, the dialog engine specializes the text graph with respect to a query and reveals interactively the document's most relevant content elements.

The open-source code of the integrated system is available at `https://github.com/ptarau/DeepRank`.

Keywords: *logic-based dialog engine, graph-based natural language processing, dependency graphs, query-driven salient sentence extraction, synergies between neural and symbolic text processing.*

1 Introduction

This is an extended and improved version of our PADL'20 paper (Tarau and Blanco 2020).[1]

Logic programming languages have been used successfully for inference and planning in natural language processing tasks restricted to narrow domains (Lierler et al. 2017; Inclezan et al. 2018; Mitra et al. 2019; Inclezan 2019). Their success, however, is limited in open-domain large-scale information extraction and knowledge representation tasks. On the other hand, deep learning systems are good at basic tasks ranging from parsing to factoid question answering, but they are still taking baby steps emulating human-level inference on complex documents (Vaswani et al. 2017; Devlin et al. 2018). Thus, a significant gap persists between neural and symbolic approaches in the field.

[1] Selected by the reviewers of PADL'20 and the program chairs Ekaterina Komendantskaya and Yanhong Annie Liu to be submitted to the Rapid Publications track of the journal *Theory and Practice of Logic Programming*.

The work presented here aims at filling this gap. We explore synergies between neural, graph-based and symbolic approaches to solve a practical problem: building a dialogue agent. This agent digests a text document (e.g., a story, a textbook, a scientific paper, a legal document) and enables the user to interact with the most relevant content.

We will start with a quick overview of the system, including the main tools and techniques of each module. Our system builds upon state-of-the-art natural language processing tools, and couples them with a declarative language module focusing on high-level text mining. We integrate the modules in the Python-based nltk ecosystem (Bird and Loper 2004), and rely on the Java-based Stanford CoreNLP toolkit (Manning et al. 2014) for basic natural language processing tasks such as sentence boundary detection, tokenization, part-of-speech tagging and parsing.

Overview of the System Architecture

Fig. 1 summarizes the architecture of our system. The Stanford CoreNLP dependency parser is started as a separate server process to which the Python-based text processing module connects as a client. It interfaces with the Prolog-based dialog engine by generating a clausal representation of the document's structure and content as well as the user's queries. The dialog engine is responsible for handling the user's queries for which answers are sent back to the Python front-end, which also handles calls to OS-level spoken-language services, when activated.

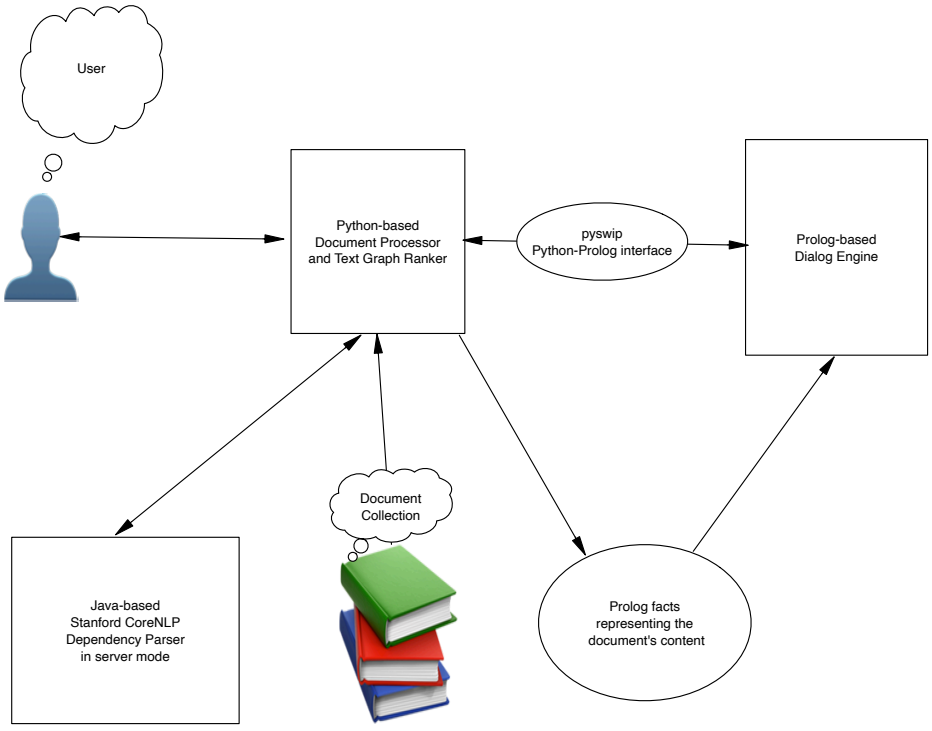

Fig. 1. System Architecture

State-of-the-art dependency parsers (Chen and Manning 2014; Adolphs et al. 2011; Choi

2017), among which the neural Stanford dependency parser (Chen and Manning 2014) stands out, produce highly accurate dependency graphs. The vertices in these graphs are words and their part-of-speech tags, and labeled edges indicate the syntactic heads of words (e.g., subject, direct object). In contrast to collocations in a sliding window, dependency graphs provide "distilled" building blocks through which a graph-based natural language processing system can absorb higher level linguistic information.

Inspired by the effectiveness of algorithms like Google's PageRank, recursive ranking algorithms applied to text graphs have enabled extraction of keyphrases, summaries and relations. Their popularity continues to increase due to their holistic view on the interconnections between text units, which signal the most relevant text units. Additionally, these algorithms are comparatively simpler. At more that 3100 citations and a follow-up of other highly cited papers (Erkan and Radev 2004), the TextRank algorithm (Mihalcea and Tarau 2004; Mihalcea and Tarau 2005) and its creative descendants have extended their applications to a wide variety of document types and social media interactions in a few dozen languages.

While part of the family of the TextRank descendants, our graph-based text processing algorithm will use information derived from the dependency graphs associated to sentences. We leverage part-of-speech tags assigned to vertices (words) and edge labels (syntactic dependencies between words) in dependency graphs in order to extract rank-ordered facts corresponding to content elements present in sentences. We pass these to logic programs that can query them and infer new relations, beyond those that can be mined directly from the text.

Like in the case of a good search engine, interactions with a text document will focus on the most relevant and semantically coherent elements matching a query. With this in mind, the natural feel of an answer syntactically appropriate for a query is less important than the usefulness of the content elements extracted: just sentences of the document in their natural order.

We will also enable spoken interaction with the dialog engine, opening the door to the use of the system via voice-based appliances. Applications range from assistive technologies to visually challenged people, live user manuals, teaching from K-12 to graduate level classes, and interactive information retrieval from complex technical or legal documents.

The paper is organized as follows. Section 2 describes the graph-based Natural Language Processing module. Section 3 describes our Prolog-based dialog engine. Section 4 shows interaction examples with several document types. Section 5 puts in context the main ideas of the paper and justifies some of the architecture choices we have made. Section 6 overviews related work and background information. Section 7 concludes the paper.

2 The graph-based Natural Language Processing module

We have organized our Python-based textgraph processing algorithm together with the Prolog-based dialog engine into a unified system.[2] We start with the building and the ranking of the text graph. Then, we overview the summary, keyphrase and relation extraction components, and the creation of the Prolog database that constitutes the logical model of the document, to be processed by the dialog engine.

[2] Our implementation is available at `https://github.com/ptarau/DeepRank`.

2.1 Building and ranking the text graph

We connect as a Python client to the Stanford CoreNLP server and use it to provide our dependency links via the wrapper at `https://www.nltk.org/` of the Stanford CoreNLP toolkit (Manning et al. 2014).

Unlike the original TextRank and related approaches that develop special techniques for each text processing task, we design a unified algorithm to obtain graph representations of documents, that are suitable for keyphrase extraction, summarization and interactive content exploration.

We use unique sentence identifiers and unique lemmas[3] as nodes of the text graph. As keyphrases are centered around nouns and good summary sentences are likely to talk about important concepts, we will need to reverse some links in the dependency graph provided by the parser, to prioritize nouns and deprioritize verbs, especially auxiliary and modal ones. Thus, we (a) redirect the dependency edges toward nouns with subject and object roles, as shown for a simple short sentence in Fig. 2, and (b) add *"about"* edges from the sentences they occur in.

We also create *"recommend"* links from words to the sentence identifiers and back from sentences to verbs with *predicate* roles to indirectly ensure that sentences recommend and are recommended by their content. Specifically, we ensure that (a) sentences recommend verbs with predicate function, and (b) their recommendation spreads to nouns that are predicate arguments (e.g., having subject or object roles).

By using the PageRank implementation of the **networkx** toolkit,[4] after ranking the sentence and word nodes of the text graph, the system is also able to display subgraphs filtered to contain only the highest ranked nodes, using Python's `graphviz` library.

An example of text graph, filtered to only show word-to-word links, derived from the U.S. Constitution,[5] is shown in Fig. 3.

2.2 Pre- and post-ranking graph refinements

The algorithm induces a form of automatic stop word filtering, due to the fact that our dependency link arrangement ensures that modifiers with lesser semantic value relinquish their rank by pointing to more significant lexical components. This is a valid alternative to explicit "leaf trimming" before ranking, which remains an option for reducing graph size for large texts or multi-document collections as well as helping with a more focused relation extraction from the reduced graphs.

Besides word-to-word links, our text graphs connect sentences as additional dependency graph nodes, resulting in a unified keyphrase and summary extraction framework. Note also that, as an option that is relevant especially for scientific, medical or legal documents, we add `first_in` links from a word to the sentence containing its first occurrence, to prioritize sentences where concepts are likely to be defined or explained.

Our reliance on graphs provided by dependency parsers builds a bridge between deep neural network-based machine learning and graph-based natural language processing enabling us to often capture implicit semantic information.

[3] A lemma is a canonical representation of a word, as it stands in a dictionary, for all its inflections e.g., it is **"be"** for "is", "are", "was" etc.

[4] `https://networkx.github.io/`

[5] Available as a text document at: `https://www.usconstitution.net/const.txt`

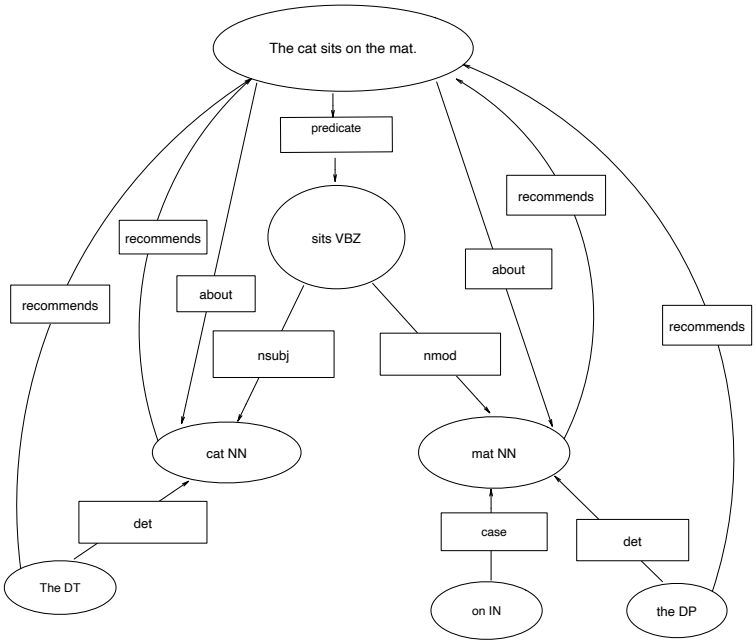

Fig. 2. Dependency graph of a simple sentence with redirected and newly added arrows

2.3 Summary and keyword extraction

As link configurations tend to favor very long sentences, a post-ranking normalization is applied for sentence ranking. After ordering sentences by rank we extract the highest ranked ones and reorder them in their natural order in the text to form a more coherent summary.

We use the parser's compound phrase tags to fuse along dependency links. We design our keyphrase synthesis algorithm to ensure that highly ranked words will pull out their contexts from sentences, to make up meaningful keyphrases. As a heuristic, we mine for a context of 2-4 dependency linked words of a highly ranked noun, while ensuring that the context itself has a high-enough rank, as we compute a weighted average favoring the noun over the elements of its context.

2.4 Relation extraction

We add subject-verb-object facts extracted from the highest ranked dependency links, enhanced with "is-a" and "part-of" relations using WordNet via the `nltk` toolkit. We plan in the future

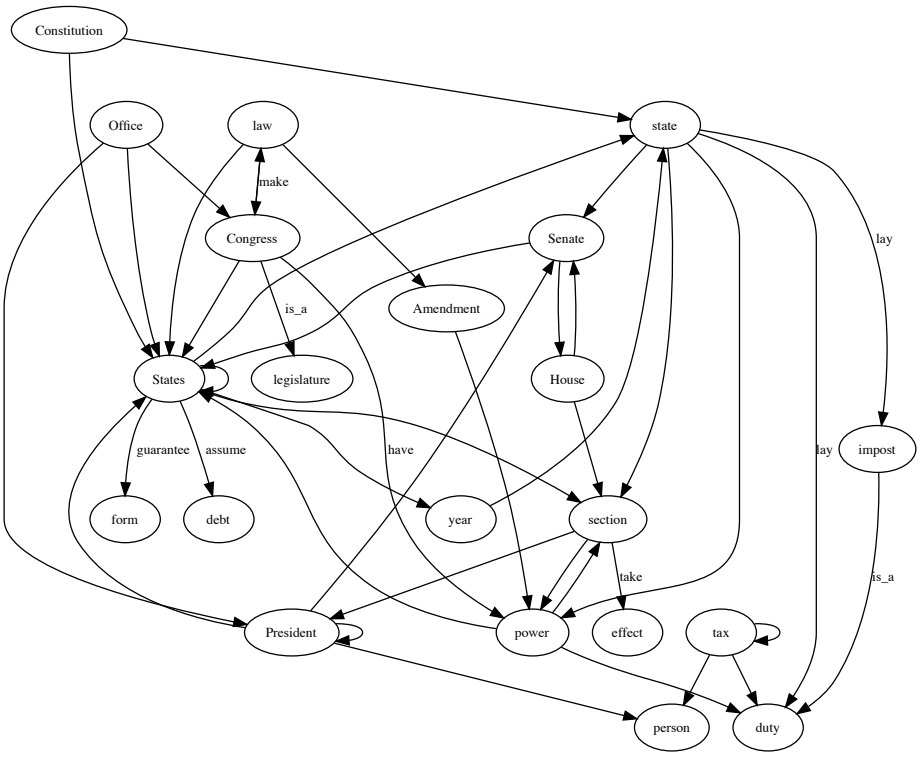

Fig. 3. Text graph fragment connecting the highest ranked words in the U.S. Constitution

to also generate relations from conditional statements identified following dependency links and involving negations, modalities, conjuncts and disjuncts, to be represented as Prolog rules. Subject-verb-object (SVO) relations are extracted directly from the dependency graph and an extra argument is added to the triplet marking the number of the sentence they originate from.

"Is-a" relations are extracted using WordNet (Fellbaum 1998) hypernyms and hyponyms.[6] Similarly, "`part_of`" relations are extracted using meronyms and holonyms.[7] As a heuristic that ensures that they are relevant to the content of the text, we ensure that both their arguments are words that occur in the document, when connecting their corresponding synsets via WordNet relations. By constraining the two ends of an "is-a" or "part-of" edge to occur in the document, we avoid relations derived from synsets unrelated to the document's content. In fact, this provides an effective word-sense disambiguation heuristic.

[6] More general and, respectively, more specific concepts.
[7] Concepts corresponding to objects that are part of, and, respectively, have as part other objects.

3 The Prolog-based dialog engine

After our Python-based document processor, with help from the Stanford dependency parser, builds and ranks the text graph and extracts summaries, keyphrases and relations, we pass them to the Prolog-based dialog engine.

3.1 Generating input for post-processing by logic programs

Once the document is processed, we generate, besides the dependency links provided by the parser, relations containing facts that we have gleaned from processing the document. Together, they form a Prolog database representing the content of the document.

To keep the interface simple and portable to other logic programming tools, we generate the following predicates in the form of Prolog-readable code, in one file per document:

1. `keyword(WordPhrase).` – the extracted keyphrases
2. `summary(SentenceId,SentenceWords).` – the extracted summary sentences sentence identifiers and list of words
3. `dep(SentenceID,WordFrom,FromTag,Label,WordTo,ToTag).` – a component of a dependency link, with the first argument indicating the sentence they have been extracted
4. `edge(SentenceID,FromLemma,FromTag,RelationLabel,ToLemma,ToTag).` – edge marked with sentence identifiers indicating where it was extracted from, and the lemmas with their POS tags at the two ends of the edge
5. `rank(LemmaOrSentenceId,Rank).` – the rank computed for each lemma
6. `w2l(Word,Lemma,Tag).` – a map associating to each word a lemma and a tag, as found by the POS tagger
7. `svo(Subject,Verb,Object,SentenceId).` – subject-verb-object relations extracted from parser input or WordNet-based `is_a` and `part_of` labels in verb position
8. `ner(SentId,ListOfNamedEntityPairs)` extracted from Named Entity Recognizer (`NER`) annotations
9. `sent(SentenceId,ListOfWords).` – the list of sentences in the document with a sentence identifier as first argument and a list of words as second argument

These predicates provide a relational view of a document in the form of a fact database that will support the inference mechanisms built on top of it.

The resulting logic program can then be processed with Prolog semantics, possibly enhanced by using constraint solvers (Schulte 1997), abductive reasoners (Denecker and Kakas 2002) or via Answer Set Programming systems (Schaub and Woltran 2018). Specifically, we expect benefits from such extensions for tackling computationally difficult problems like word-sense disambiguation (WSD) or entailment inference as well as domain-specific reasoning (Inclezan 2019; Olson and Lierler 2019; Mitra et al. 2019).

We have applied this process to the *Krapivin document set* (Krapivin et al. 2008), a collection of **2304** research papers annotated with the authors' own keyphrases and abstracts.

The resulting 3.5 GB *Prolog dataset*[8] is made available for researchers in the field, interested to explore declarative reasoning or text mining mechanisms.

[8] `http://www.cse.unt.edu/~tarau/datasets/PrologDeepRankDataset.zip`

3.2 The Prolog interface

We use as a logic processing tool the open source SWI-Prolog system[9] (Wielemaker et al. 2012) that can be called from, and can call Python programs using the `pyswip` adaptor.[10] After the adaptor creates the Prolog process and the content of the digested document is transferred from Python (in a few seconds for typical scientific papers with 10-15 pages), query processing is realtime.

3.3 The user interaction loop

With the Prolog representation of the digested document in memory, the dialog starts by displaying the summary and keyphrases extracted from the document.[11] One can see this as a "mini search-engine", specialized to the document, and, with help of an indexing layer, extensible to multi-document collections. The dialog agent associated to the document answers queries as sets of salient sentences extracted from the text, via a specialization of our summarization algorithm to the context inferred from the query.

As part of an interactive *read/listen, evaluate, print/say* loop, we generate for each query sentence, a set of predicates that are passed to the Prolog process, from where answers will come back via the `pyswip` interface. The predicates extracted from a query have the same structure as the database representing the content of the complete document, initially sent to Prolog.

3.4 The answer generation algorithm

Answers are generated by selecting the most relevant sentences, presented in their natural order in the text, in the form of a specialized "mini-summary". We will next overview our query answering algorithm, with examples to follow in section 4.

3.4.1 Query expansion

Answer generation starts with a query-expansion mechanism via relations that are derived by finding, for lemmas in the query, WordNet hypernyms, hyponyms, meronyms and holonyms, as well as by directly extracting them from the query's dependency links. We use the rankings available both in the query and the document graph to prioritize the highest ranked sentences connected to the highest ranked nodes in the query.

3.4.2 Short-term dialog memory

We keep representations of recent queries in memory, as well as the answers generated for them. If the representation of the current query overlaps with a past one, we use content in the past query's database to extend query expansion to cover edges originating from that query. Overlapping is detected via shared edges between noun or verb nodes between the query graphs.

[9] `http://www.swi-prolog.org/`
[10] `https://github.com/yuce/pyswip`
[11] And also speak them out if the `quiet` flag is off.

3.4.3 Answer sentence selection

Answer sentence selection is performed with a combination of several interoperating algorithms:

- use of *personalized PageRank* (Haveliwala 2002; Haveliwala et al. 2003) with a dictionary provided by highest ranking lemmas and their ranks in the query's graph, followed by reranking the document's graph to specialize to the query's content
- matching guided by SVO-relations
- matching of edges in the query graph against edges in the document graph
- query expansion guided by rankings in both the query graph and the document graph
- matching guided by a selection of related content components in the short-term dialog memory window

Matching against the Prolog database representing the document is implemented as a size constraint on the intersection of the expanded query lemma set, built with highly ranked shared lemmas pointing to sentences containing them. The set of answers is organized to return the highest-ranked sentences based on relevance to the query and in the order in which they appear in the document.

We keep the dialog window relatively small (limited to the highest ranked 3 sentences in the answer set, by default). Relevance is ensured with help from the rankings computed for both the document content and the query.

3.4.4 Personalized PageRank

Using *personalized PageRank* (Haveliwala 2002; Haveliwala et al. 2003) can be seen as a specialization of the document graph with respect to the query. The personalization dictionary is also used to implicitly redirect flow, otherwise stuck in the sink nodes of the text graph, to content related to the query. The predicate `query_pers_sents` is generated for each query and then passed to Prolog, where it is used to prioritize the answers computed by the answer search algorithms.

3.4.5 Matching guided by SVO-relations

SVO-facts inferred from syntactic dependencies and from WordNet relations have a 4-th argument, indicating the sentence number where they have been found. Thus walking over them in a transitive closure computation (limited to at most K inference steps) allows us to collect a path made of the sentences and relations used at each step, suggesting possibly interesting candidate answers.

The predicate `tc/7` implements a `K`-step limited transitive closure computation also controlled by a set of relations available and a loop checking mechanism that avoids revisiting the same nodes repeatedly. We believe that it also shows the expressiveness of a declarative programming pattern in handling a fairly complex set of requirements in a clear and compact form. Our computation is exposed via the interface predicate `tc/5`.

The predicate `tc(K,A,Rels,C,Res)` *holds if we can get from word* `A` *to word* `C` *in at most* `K` *steps using any relation in the set* `Rels` *and returning in* `Res` *the number of steps left, the path followed and a sentence number in the document, possibly relevant as an answer.*

```
%% tc(+K,+A,+Rels,?A,-Res)
tc(K,A,Rels,C,Res):-tc(A,Rels,C,[],K,_,Res).
```

```
tc(A,Rels,C,Xs,SN1,N2,Res) :-
  succ(N1,SN1),
  member(Rel,Rels),
  call_svo(A,Rel,B,Id),
  not(memberchk(B-_,Xs)),
  tc1(B,Rels,C,[A-Rel|Xs],Id,N1,N2,Res).

tc1(B,_Rels,B,Xs,Id,N,N,res(N,Id,Xs)):-
  % Id must be a known sentence Id!
  nonvar(Id).
tc1(B,Rels,C,Xs,_,N1,N2,Res) :-
   tc(B,Rels,C,Xs,N1,N2,Res).
```

Note that loop checking is achieved by keeping a path of elements the form Word-Relation and the the "_" variable in the definition of tc/5 ensures that paths of length *up to* `K` are are returned. We also accommodate `SVO` relations originating from a *domain-specific ontology*, for which the sentence identifier is left as an *unbound logical variable*, provided that at the end of the available inference steps an actual sentence of the document is returned, a requirement ensured with the `nonvar/1` test in the first clause of the predicate `tc1/8`. If the system detects the presence of additional `svo/4` facts from a domain specific ontology (consulted using a plugin mechanism), the single step predicate `call_svo/4` will extend its scope over such additional `SVO` relations.

3.4.6 Using Named Entity Recognition

Named Entity Recognition (NER) provides person, location, time, etc. annotations that provide additional relations, usable for inference. When NER is available, matching directed by **wh**-words like *where*, *when*, *who*, questions trigger scanning the named entity database, also sent to Prolog.

For a document on the CDC COVID-19 status, the system extracts `NER` relations like

```
ner(86, [(0, ('March', 'DATE')),
           (1, ('10', 'DATE')),
           (2, ('CDC', 'ORGANIZATION')),
           (3, ('infection', 'CAUSE_OF_DEATH'))]).
```

saying that sentence `86` is talking about named entities hinting at events on March 10 when the CDC, an organization, has identified life-threatening infections.

3.4.7 Using **wh**-*word replacement with logic variables*

Edges in the query originating or targeting **wh**-words get replaced with logic variables with labels also corresponding to expected syntactic roles (e.g., `nsubj` for who).

Besides matching them against corresponding edges in the document we also take advantage of the named entity relation `ner/2` from which we derive specialized answer predicates.

```
who(KWs,SentId):-wh(['PERSON','ORGANIZATION','TITLE'],KWs,SentId).

where(KWs,SentId):-
  wh(['LOCATION','CITY','COUNTRY','STATE_OR_PROVINCE'],KWs,SentId).
```

```
many(KWs,SentId):-wh(['NUMBER', 'ORDINAL', 'MONEY'],KWs,SentId).

when(KWs,SentId):-wh(['DATE','TIME','DURATION'],KWs,SentId).
```

The `wh/3` predicate will scan the corresponding `ner/2` facts against a match, to be also validated by ensuring that answers have a high enough personalized PageRank.

4 Interacting with the dialog engine

We will next show interaction examples with several document types, with focus on key aspects of our question-answering algorithms.

The following example shows the result of a query on the US Constitution document, with edges marked by POS-tags of the two nodes aggregated with the label of the dependency link connecting them.

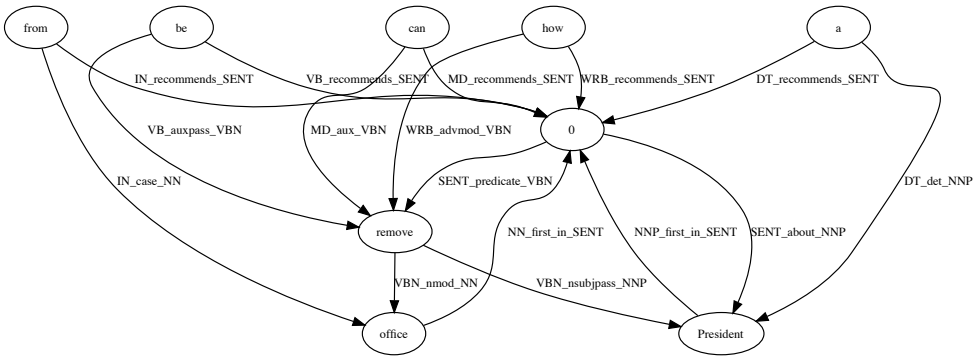

Fig. 4. Graph of a query on the U.S. Constitution

>>> talk_about('examples/const')
?-- **How can a President be removed from office?**
59 : In Case of the Removal of the President from Office , or of his Death , Resignation , or Inability to discharge the Powers and Duties of the said Office , the same shall devolve on the Vice President , and the Congress may by Law provide for the Case of Removal , Death , Resignation or Inability , both of the President and Vice President , declaring what Officer shall then act as President , and such Officer shall act accordingly , until the Disability be removed , or a President shall be elected .
66 : Section 4 The President , Vice President and all civil Officers of the United States , shall be removed from Office on Impeachment for , and Conviction of , Treason , Bribery , or other high Crimes and Misdemeanors .
190 : If the Congress , within twenty one days after receipt of the latter written declaration , or , if Congress is not in session , within twenty one days after Congress is required to assemble , determines by two thirds vote of both Houses that the President is unable to discharge the powers and duties of his office , the Vice President shall continue to discharge the same as Acting President ; otherwise , the President shall resume the powers and duties of his office .

Note the relevance of the extracted sentences and resilience to semantic and syntactic variations (e.g., the last sentence does not contain the word "remove"). The dependency graph of the query is shown in Fig. 4. The clauses of the `query_rank/2` predicate in the Prolog database corresponding to the query are:

```
query_rank('President', 0.2162991696472837).
query_rank('remove', 0.20105324712764877).
query_rank('office', 0.12690425831428373).
query_rank('how', 0.04908035060099132).
query_rank('can', 0.04908035060099132).
query_rank('a', 0.04908035060099132).
query_rank('be', 0.04908035060099132).
query_rank('from', 0.04908035060099132).
query_rank(0, 0.0023633884483800784).
```

The impact of Personalized PageRank can be seen by comparing the cloud-map for the document (see Fig. 5) with the cloud-map of the document specialized to the query (see Fig. 6). Note the increased emphasis in Fig. 6 of some relevant actors (President, Senate) as well as the relevant concepts related to removal from office of a President (impeachment, profit, disability, judgment).

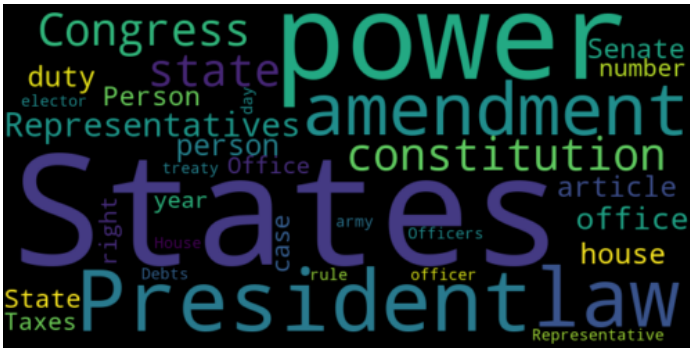

Fig. 5. Word-cloud of U.S. Constitution before specialization w.r.t. to query

Fig. 6. Word-cloud of U.S. Constitution after query-driven personalized PageRank

Our next example uses an ASCII version of Einstein's 1920 book on relativity, retrieved from

the Gutenberg collection[12] and trimmed to the actual content of the book (250 pages in `epub` form).

```
>>> talk_about('examples/relativity')
?-- What happens to light in the presence of gravitational fields?
```

611 : In the example of the transmission of light just dealt with , we have seen that the general theory of relativity enables us to derive theoretically the influence of a gravitational field on the course of natural processes , the laws of which are already known when a gravitational field is absent .

764 : On the contrary , we arrived at the result that according to this latter theory the velocity of light must always depend on the co-ordinates when a gravitational field is present .

765 : In connection with a specific illustration in Section XXIII , we found that the presence of a gravitational field invalidates the definition of the coordinates and the time , which led us to our objective in the special theory of relativity .

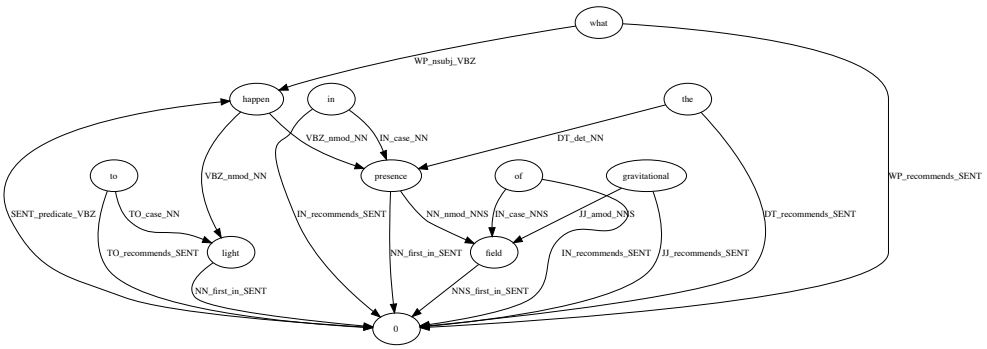

Fig. 7. Graph of query on Einstein's book on Relativity

The query graph is shown in Fig. 7. After the less than 30 seconds that it takes to digest the book, answers are generated in less than a second for all queries that we have tried. Given the availability of spoken dialog, a user can iterate and refine queries to extract the most relevant answer sentences of a document.

On an even larger document, like the Tesla Model 3 owner's manual,[13] digesting the document takes about 60 seconds and results in 12 MB of Prolog clauses. After that, query answering is still below 1 second.

```
>>> talk_about('examples/tesla')
?-- How may I have a flat tire repaired?
```

3207 : Arrange to have Model 3 transported to a Tesla Service Center , or to a nearby tire repair center .

3291 : Note : If a tire has been replaced or repaired using a different tire sealant than the one available from Tesla , and a low tire pressure is detected , it is possible that the tire sensor has been damaged .

The highly relevant first answer is genuinely useful in this case, given that Tesla Model 3's do not have a spare tire. Being able to use voice queries while driving and in need of urgent technical information about one's car, hints towards obvious practical applications of our dialog engine.

Directly querying news articles with potentially urgent information is another application. The

[12] `https://www.gutenberg.org/files/30155/30155-0.txt`
[13] `https://www.tesla.com/sites/default/files/model_3_owners_manual_north_america_en.pdf`

following interaction is based on the US-CDC guidelines on the COVID-19 pandemics (as posted on march 17, 2020).

```
>>> talk_about('examples/covid')
?-- How does the COVID-19 virus spread in a community?
25 : Pandemics happen when a new virus emerges to infect people and can spread between people sustainably .
26 : Because there is little to no pre-existing immunity against the new virus , it spreads worldwide .
47 : Three U.S. states are experiencing sustained community spread .
```

The highest ranked lemmas after running personalized PageRank show the emphasis on key lemmas coming from the query as well as some of the overall important related concepts in the document.

```
query_pers_words('virus', 0.1279800580708069).
query_pers_words('spread', 0.08076646367741867).
query_pers_words('covid', 0.06794580306202914).
query_pers_words('community', 0.05973841781795769).
query_pers_words('CoV', 0.01846080235270702).
query_pers_words('response', 0.012416829623738527).
query_pers_words('influenza', 0.012240656425061098).
query_pers_words('culture', 0.012084585427826857).
query_pers_words('illness', 0.01129131793978254).
query_pers_words('people', 0.009611356400409315).
query_pers_words('healthcare', 0.008738409714154799).
query_pers_words('risk', 0.00859884680785955).
```

The evolution between the cloud-map of the document and the cloud-map after the reranked document is shown in Fig. 8 and Fig. 9.

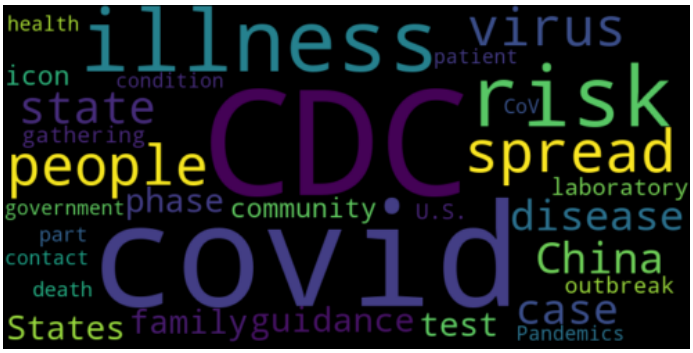

Fig. 8. Word-cloud of the CDC COVID-19 document before specialization w.r.t. to query

The following examples show the effect if the **wh-**words triggering answers derived from the `ner/2` facts.

```
>>> talk_about('examples/covid')
?-- Who did declare the outbreak an emergency?
On March 13 , the President of the United States declared the COVID-19 outbreak a national emergency .
27 : On March 11 , the COVID-19 outbreak was characterized as a pandemic by the WHO .
```

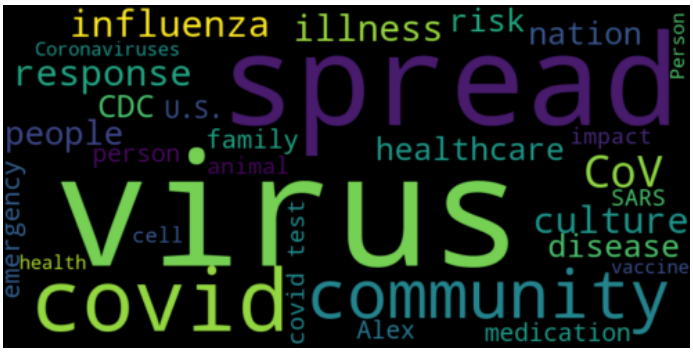

Fig. 9. Word-cloud of the CDC COVID-19 document after query-driven personalized PageRank

?-- **When did the outbreak become a pandemic?**

4 : On March 11 , WHO publicly characterized COVID-19 as a pandemic .

22 : COVID-19 Now a Pandemic A pandemic is a global outbreak of disease .

27 : On March 11 , the COVID-19 outbreak was characterized as a pandemic by the WHO .

5 Discussion

Ideally, one would like to evaluate the quality of natural language understanding of an AI system by querying it not only about a set of relations explicitly extracted in the text, but also about relations inferred from the text. Moreover, one would also like to have the system justify the inferred relations in the form of a proof, or at least a sketch of the thought process a human would use for the same purpose. The main challenge here is not only that theorem-proving logic is hard, (with first-order classical predicate calculus already Turing-complete), but also that modalities, beliefs, sentiments, hypothetical and counterfactual judgments often make the underlying knowledge structure intractable.

On the other hand, simple relations, stated or implied by text elements that can be mined or inferred from a ranked graph built from labeled dependency links, provide a limited but manageable approximation of the text's deeper logic structure, especially when aggregated with generalizations and similarities provided by WordNet or the much richer Wikipedia knowledge graph.

Given its effectiveness as an interactive content exploration tool, we plan future work on packaging our dialog engine as a set of Amazon Alexa skills for some popular Wikipedia entries as well as product reviews, FAQs and user manuals.

Empirical evaluation of our keyphrase and summarization algorithms will be subject to a different paper, but preliminary tests indicate that both of them match or exceed Rouge scores for state of the art systems (Tarau and Blanco 2019).

6 Related work

Dependency parsing

The Stanford neural network based dependency parser (Chen and Manning 2014) is now part of the Stanford CoreNLP toolkit,[14] which also comes with part of speech tagging, named entity recognition and co-reference resolution (Manning et al. 2014). Its evolution toward the use of Universal Dependencies (de Marneffe et al. 2014) makes systems relying on it potentially portable to over **70** languages covered by the Universal Dependencies effort. [15]

Of particular interest is the connection of dependency graphs to logic elements like predicate argument relations (Choi and Palmer 2011). The automatic conversion of constituency trees to dependency graphs proposed by Choi (2017) provides a bridge allowing the output of high-quality statistically trained phrase structure parsers to be reused for extraction of dependency links.

In this context, our novel contribution is that we analyze dependency links and part-of-speech tags associated to their endpoints in order to build a unified document graph from which we extract SVO relations. By redirecting links to focus on nouns and sentences we not only enable keyphrase and summary extraction from the resulting document graph but also facilitate its use for query answering in our dialog engine.

Graph based Natural Language Processing

TextRank (Mihalcea and Tarau 2004; Mihalcea and Tarau 2005) extracts keyphrases using word co-occurrence relations controlled by the distance between words: two vertices are connected if their corresponding lexical units co-occur within a sliding window ranging from 2 to 10 words. Sentence similarity is computed as content overlap giving weights to the links that refine the original PageRank algorithm (Page et al. 1998; Brin and Page 1998). TextRank needs elimination of stop words and obtains best results when links are restricted to nouns and adjectives. Erkan and Radev (2004) explore several graph centrality measures, and Mihalcea and Radev (2011) offer a comprehensive overview of graph-based natural language processing and related graph algorithms. Graph-based and other text summarization techniques are surveyed by Nenkova and McKeown (2012) and more recently by Allahyari et al. (2017). Besides ranking, elements like coherence via similarity with previously chosen sentences and avoidance of redundant rephrasings are shown to contribute to the overall quality of the summaries.

The main novelty of our approach in this context is building text graphs from dependency links and integrating words and sentences in the same text graph, resulting in a unified algorithm that also enables relation extraction and interactive text mining.

Relation Extraction

The relevance of dependency graphs for relation extraction has been identified in several papers. Among others, Adolphs et al. (2011) point out to their role as a generic interface between parsers and relation extraction systems. Stevenson and Greenwood (2009) identify several models grounded on syntactic patterns (e.g., subject-verb-object) that can be mined out from depen-

[14] `https://stanfordnlp.github.io/CoreNLP/`
[15] `https://universaldependencies.org/`

dency graphs. Of particular interest for relation extraction facilitated by dependency graphs is the shortest path hypothesis that prefers relating entities like predicates and arguments that are connected via a shortest path in the graph (Bunescu and Mooney 2005). To facilitate their practical applications to biomedical texts, Peng et al. (2015) extend dependency graphs with richer sets of semantic features including "is-a" and "part-of" relations and co-reference resolution.

The use of ranking algorithms in combination with WordNet synset links for word-sense disambiguation goes back as far as Mihalcea et al. (2004), which is in fact a prequel to TextRank (Mihalcea and Tarau 2004). With the emergence of resources like Wikipedia, a much richer set of links and content elements has been used in connection with graph-based natural language processing (Li and Zhao 2016; Adolphs et al. 2011; Mihalcea and Csomai 2007).

We currently extract our relations directly from the dependency graph and by using one step up and one step down links in the WordNet hypernym and meronym hierarchies. We plan extensions to integrate Wikipedia content via the `dbpedia` database,[16] and to extract more elaborate logic relations using a Prolog-based semantic parser like Boxer (Bos 2015).

Logic Programming Systems for Natural Language Processing

A common characteristic of Prolog or ASP-based NLP systems is their focus on closed domains with domain-specific logic expressed in clausal form (Lierler et al. 2017; Inclezan et al. 2018; Mitra et al. 2019; Inclezan 2019), although recent work (e.g., Olson and Lierler (2019)) extracts action language programs from more general narratives.

As our main objective is the building of a practically useful dialog agent, and as we work with open domain text and query driven content retrieval, our focus is not on precise domain-specific reasoning mechanisms. By taking advantage of the Prolog representation of a document's content, we use reasoning about the extracted relations and ranking information to find the most relevant sentences derived from a given query and the recent dialog history.

7 Conclusions

The key idea of the paper has evolved from our search for synergies between symbolic AI and emerging natural language processing tools built with machine learning techniques. It is our belief that these are complementary and that by working together they will take significant forward steps in natural language understanding. We have based our text graph on heterogeneous but syntactically and semantically meaningful text units (words and sentences) resulting in a web of interleaved links. These links mutually recommend each other's highly ranked instances. Our fact extraction algorithm, in combination with the Prolog interface, has elevated the syntactic information provided by dependency graphs with semantic elements ready to benefit from logic-based inference mechanisms. Given the standardization brought by the use of *Universal Dependencies*, our techniques are likely to be portable to a large number of languages.

The Prolog-based dialog engine supports spoken interaction with a conversational agent that exposes salient content of the document driven by the user's interests. Its applications range from assistive technologies to visually challenged people, voice interaction with user manuals, teaching from K-12 to graduate-level classes, and interactive information retrieval from complex technical or legal documents.

[16] `https://wiki.dbpedia.org/`

Last but not least, we have used our system's front end to generate a Prolog dataset derived from more than 2000 research papers. We make this dataset available to other researchers using logic programming based reasoners and content mining tools.[17]

Acknowledgment

We are thankful to the anonymous reviewers of **PADL'2020** for their careful reading and constructive suggestions.

References

ADOLPHS, P., XU, F., LI, H., AND USZKOREIT, H. 2011. Dependency Graphs as a Generic Interface between Parsers and Relation Extraction Rule Learning. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings*, J. Bach and S. Edelkamp, Eds. Lecture Notes in Computer Science, vol. 7006. Springer, 50–62.

ALLAHYARI, M., POURIYEH, S. A., ASSEFI, M., SAFAEI, S., TRIPPE, E. D., GUTIERREZ, J. B., AND KOCHUT, K. 2017. Text Summarization Techniques: A Brief Survey. *CoRR abs/1707.02268*.

BIRD, S. AND LOPER, E. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Association for Computational Linguistics, Barcelona, Spain, 214–217.

BOS, J. 2015. Open-domain semantic parsing with boxer. In *Proceedings of the 20th Nordic Conference of Computational Linguistics, NODALIDA 2015, May 11-13, 2015, Institute of the Lithuanian Language, Vilnius, Lithuania*, B. Megyesi, Ed. Linköping University Electronic Press / ACL, 301–304.

BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems 30,* 1–7, 107–117. http://citeseer.nj.nec.com/brin98anatomy.html.

BUNESCU, R. C. AND MOONEY, R. J. 2005. A Shortest Path Dependency Kernel for Relation Extraction. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. HLT '05. Association for Computational Linguistics, Stroudsburg, PA, USA, 724–731.

CHEN, D. AND MANNING, C. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 740–750.

CHOI, J. D. 2017. Deep Dependency Graph Conversion in English. In *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories*. TLT'17. Bloomington, IN, 35–62.

CHOI, J. D. AND PALMER, M. 2011. Transition-based Semantic Role Labeling Using Predicate Argument Clustering. In *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics*. RELMS '11. Association for Computational Linguistics, Stroudsburg, PA, USA, 37–45.

DE MARNEFFE, M.-C., DOZAT, T., SILVEIRA, N., HAVERINEN, K., GINTER, F., NIVRE, J., AND MANNING, C. D. 2014. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Languages Resources Association (ELRA), Reykjavik, Iceland, 4585–4592.

DENECKER, M. AND KAKAS, A. C. 2002. Abduction in logic programming. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*. Springer-Verlag, London, UK, UK, 402–436.

DEVLIN, J., CHANG, M., LEE, K., AND TOUTANOVA, K. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR abs/1810.04805*.

ERKAN, G. AND RADEV, D. R. 2004. LexRank: Graph-based Lexical Centrality As Salience in Text Summarization. *J. Artif. Int. Res. 22,* 1 (Dec.), 457–479.

[17] `http://www.cse.unt.edu/~tarau/datasets/PrologDeepRankDataset.zip`

FELLBAUM, C. 1998. *WordNet, An Electronic Lexical Database*. The MIT Press.

HAVELIWALA, T., KAMVAR, S., AND JEH, G. 2003. An analytical comparison of approaches to personalizing pagerank. Technical Report 2003-35, Stanford InfoLab. June.

HAVELIWALA, T. H. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web*. WWW '02. ACM, New York, NY, USA, 517–526.

INCLEZAN, D. 2019. Restkb: A library of commonsense knowledge about dining at a restaurant. In Proceedings 35th International Conference on *Logic Programming* (Technical Communications) , Las Cruces, NM, USA, September 20-25, 2019, B. Bogaerts, E. Erdem, P. Fodor, A. Formisano, G. Ianni, D. Inclezan, G. Vidal, A. Villanueva, M. D. Vos, and F. Yang, Eds. Electronic Proceedings in Theoretical Computer Science, vol. 306. Open Publishing Association, 126–139.

INCLEZAN, D., ZHANG, Q., BALDUCCINI, M., AND ISRANEY, A. 2018. An ASP methodology for understanding narratives about stereotypical activities. *TPLP 18,* 3-4, 535–552.

KRAPIVIN, M., AUTAYEU, A., AND MARCHESE, M. 2008. Large Dataset for Keyphrases Extraction. Tech. Rep. DISI-09-055, DISI, Trento, Italy. May.

LI, W. AND ZHAO, J. 2016. TextRank Algorithm by Exploiting Wikipedia for Short Text Keywords Extraction. *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*, 683–686.

LIERLER, Y., INCLEZAN, D., AND GELFOND, M. 2017. Action languages and question answering. In *IWCS 2017 - 12th International Conference on Computational Semantics - Short papers, Montpellier, France, September 19 - 22, 2017*, C. Gardent and C. Retoré, Eds. The Association for Computer Linguistics.

MANNING, C. D., SURDEANU, M., BAUER, J., FINKEL, J., BETHARD, S. J., AND MCCLOSKY, D. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60.

MIHALCEA, R. AND CSOMAI, A. 2007. Wikify!: Linking Documents to Encyclopedic Knowledge. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*. CIKM '07. ACM, New York, NY, USA, 233–242.

MIHALCEA, R. AND TARAU, P. 2004. TextRank: Bringing Order into Texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*. Barcelona, Spain.

MIHALCEA, R. AND TARAU, P. 2005. An Algorithm for Language Independent Single and Multiple Document Summarization. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*. Korea.

MIHALCEA, R., TARAU, P., AND FIGA, E. 2004. PageRank on Semantic Networks, with application to Word Sense Disambiguation. In *Proceedings of The 20st International Conference on Computational Linguistics (COLING 2004)*. Geneva, Switzerland.

MIHALCEA, R. F. AND RADEV, D. R. 2011. *Graph-based Natural Language Processing and Information Retrieval*, 1st ed. Cambridge University Press, New York, NY, USA.

MITRA, A., CLARK, P., TAFJORD, O., AND BARAL, C. 2019. Declarative question answering over knowledge bases containing natural language text with answer set programming. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*. AAAI Press, 3003–3010.

NENKOVA, A. AND MCKEOWN, K. R. 2012. A Survey of Text Summarization Techniques. In *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Springer, 43–76.

OLSON, C. AND LIERLER, Y. 2019. Information extraction tool text2alm: From narratives to action language system descriptions. In Proceedings 35th International Conference on *Logic Programming* (Technical Communications) , Las Cruces, NM, USA, September 20-25, 2019, B. Bogaerts, E. Erdem, P. Fodor, A. Formisano, G. Ianni, D. Inclezan, G. Vidal, A. Villanueva, M. D. Vos, and F. Yang, Eds. Electronic Proceedings in Theoretical Computer Science, vol. 306. Open Publishing Association, 87–100.

PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. 1998. The PageRank Citation Ranking: Bringing Order to the Web. Tech. rep., Stanford Digital Library Technologies Project.

PENG, Y., GUPTA, S., WU, C., AND SHANKER, V. 2015. An extended dependency graph for relation extraction in biomedical texts. In *Proceedings of BioNLP 15*. Association for Computational Linguistics, 21–30.

SCHAUB, T. AND WOLTRAN, S. 2018. Special Issue on Answer Set Programming. *KI 32,* 2-3, 101–103.

SCHULTE, C. 1997. Programming constraint inference engines. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming*, G. Smolka, Ed. Lecture Notes in Computer Science, vol. 1330. Springer-Verlag, SchloßHagenberg, Austria, 519–533.

STEVENSON, M. AND GREENWOOD, M. 2009. Dependency Pattern Models for Information Extraction. *Research on Language & Computation 7,* 1 (Mar.), 13–39.

TARAU, P. AND BLANCO, E. 2019. Dependency-based text graphs for keyphrase and summary extraction with applications to interactive content retrieval. *ArXiv abs/1909.09742*.

TARAU, P. AND BLANCO, E. 2020. Interactive Text Graph Mining with a Prolog-based Dialog Engine. In *Practical Aspects of Declarative Languages - 22th International Symposium, PADL 2020, New Orleans, USA, January 20-21, 2020, Proceedings*, E. Komendantskaya and Y. A. Liu, Eds. Lecture Notes in Computer Science, vol. 12007. Springer, 3–19.

VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. 2017. Attention is all you need. *CoRR abs/1706.03762*.

WIELEMAKER, J., SCHRIJVERS, T., TRISKA, M., AND LAGER, T. 2012. SWI-Prolog. *Theory and Practice of Logic Programming 12*, 67–96.

Appendix: Response to the conference reviews, along and identification of the new material

Response to reviewers' comments

Reviewer 1

>>> Regarding improvements, some more technical detail would certainly be welcome, as well as perhaps some small examples illustrating what each step does, i.e., not just the input-output behavior and the graphs. Also, some more detail on the experimental evaluation would be very welcome. The paper states that preliminary results show that the system is better than state of the art, but then leaves the reader wanting, when it says that these results are the subject of another paper.

We have added explanations focusing on the question answering part and referred to our paper (at https://arxiv.org/pdf/1909.09742.pdf) for the evaluation of the summarization and keyword extraction, not directly related to this paper. We have added a new example with queries on a CDC COVID-19 document and discussed the action of our Prolog predicates on the resulting answers.

>>> - line 29: you may want to add refs to the deep learning-based query-answering systems you are referring to.
- the text in the figures is too small to read in a printed version.

Several references have been added, and we have zoomed our computer-generated figures to the highest possible size that fits on the TPLP page. We have also replaced some of the graphviz figures by generating new ones (e.g., the US constitution graph) with fewer nodes resulting implicitly in larger fonts.

Reviewer 2

>>> - Some linguistic terms are not defined, but may not be known to the reader, it could be helpful to introduce them before using them, including: hypernyms, hyponyms, meronyms, holonyms.

Added definitions/explanations for hypernyms, holonyms, etc.

>>> - On lines 187-189, the authors suggest the possibility of using constraint solvers, abductive reasoning, or answer set programming. However, it is not clear whether the programs that result from the generation ever warrant the use of such systems, or if Prolog is sufficient, what the difference or benefit would be from such an exercise.

We have discussed differences with related work based on ASP systems in the related work subsection on "Logic Programming Systems for Natural Language Processing". By releasing the dataset readable for ASP, Datalog, ILP or CP systems sharing basic syntax with Prolog, we hope that experiments with such systems will in the future reveal how alternative declarative reasoning models will perform on the same dataset.

>>> - In Section 3, it is not clear what the second argument of dialog_about stands for although that is nor particularly relevant, it should either be removed, or the possible values should be explained to the user.

Fixed, by adding `dialog_about` *specializing the predicate for the case when a file argument for providing the questions is not needed, given the interactive dialogue.*

>>> - In general, throughout the paper, commas after connectives could help readability (e.g., then, thus, etc.). - In Section 2.4, is-a and part-of are inconsistently styled (sometimes roman, sometimes tt) - On line 150, there is an extra apostrophe before Is-a. - In Section 3.1, in the enumerated list, predicates are inconsistently styled (sometimes roman, sometimes tt). - On line 289, "less that" -> "less than"

All fixed.

>>> - On line 297, the question is not well-formed (perhaps "How may I..."?) although not sure if that is intentional.

Changed query to "How may I have a flat tire repaired?", with no impact on the answers returned, given that high ranked nouns and verbs dominate in our answer search algorithms.

Reviewer 3

>>> The paper presents a Prolog-based dialogue engine able to answer queries in natural language. This is a very challenging and interesting application which deserves being presented at PADL. The work is rather mature and builds upon a number of well-established technologies. An implementation and some test data sets are provided, which make the experiments replicable. I only recommend to comment more on the use of declarative programming in this application domain.

We have focussed our addition of new content on describing the details of our declarative Prolog-based algorithms, covering aspect as the use of logic variables when answering wh-word queries against relations involving Named Entities.

>>> - ll. 178, 179, 181, 183: please use the same font for predicates

Done.

>>> - l. 188: bibliographic references should be added also fro constraint solvers and abductive reasoners

Done.

>>> - footnote 7: please use link environment

We have used the hyperlink environment for all http links end ensured they are all clickable from the PDF file.

New Material

We focussed on adding details of our declarative programming answer computation algorithms and more detailed examples.

- added details on Personalized PageRank (subsection 3.4.4)
- added details on the use of SVO-relations, including the Prolog code of one of the reasoning mechanisms (subsection 3.4.5)
- added details on using NER (Named Entity recognition) (subsection 3.4.6)
- added details on the use of NERs for answering **wh**-word queries (subsection 3.4.7)
- added new COVID-19 example, showing the impact on answers for the above features (page 14)
- updated references and added several new ones