Interactive Text Graph Mining with a Prolog-based Dialog Engine

Paul Tarau and Eduardo Blanco

Department of Computer Science and Engineering
University of North Texas
paul.tarau@unt.edu, eduardo.blanco@unt.edu

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, integrate sentence identifiers as graph nodes and after ranking the graph we take advantage of the implicit semantic information that dependency links 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

Logic programming languages have been used successfully for inference and planning tasks on restricted domain natural language processing tasks [1–4] but not much on open domain, large scale information retrieval and knowledge representation tasks. On the other hand, deep learning systems are very good at basic tasks ranging from parsing to factoid-trained question answering systems, but still taking baby steps when emulating human-level inference processing on complex documents [5, 6]. Thus, a significant gap persists between neural and symbolic processing in the field.

A motivation of our work is to help fill this gap by exploring synergies between the neural, graph based and symbolic ecosystems in solving a practical problem: building a dialog agent, that, after digesting the content of a text document (e.g., a story, a textbook, a scientific paper, a legal document), enables the user to interact with its most relevant content.

We will start with a quick overview of the tools and techniques needed. Building a state-of-the art Natural Language Processing system requires interaction with multi-paradigm components as emulating their functionality from scratch could easily become a 100-person/years project. In our case, this means integrating a declarative language module, focusing on high level text mining, into the Python-based **nltk** ecosystem, while relying on the Java-based Stanford CoreNLP toolkit for basic tasks like segmentation, part-of-speech tagging and parsing.

Overview of the System Architecture Fig. 1 summarizes the architecture of our system. The Stanford parser is started as a separate server process to which the Python 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 handles also the call to OS-level spoken-language services, when activated.

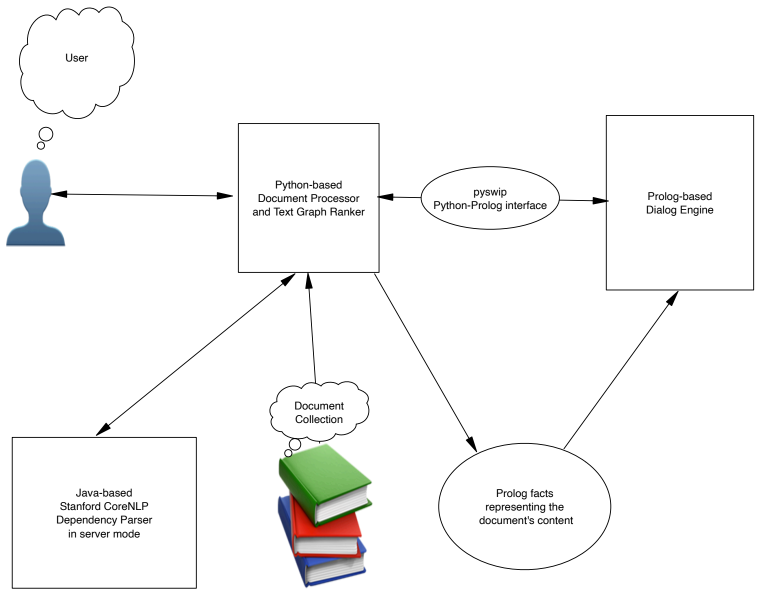

Fig. 1. System Architecture

Today's dependency parsers [7–9], among which the neurally-trained Stanford dependency parser [7] stands out, produce highly accurate dependency graphs and part of speech tagged vertices. Seen as edges in a text graph, they provide, by contrast to collocations in a sliding window, "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 the holistic view they shed on the interconnections between text units that act as recommenders for the most relevant ones, as well as the comparative simplicity of the algorithms. At close to 3000 citations and a follow-up of some almost equally as highly cited papers like [10] the

TextRank algorithm [11, 12] 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. With help from the labels marking the edges of a dependency graph and the part of speech tags associated to its nodes, we will 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, interaction 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 doors for 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 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 puts in context the main ideas of the paper and justifies some of the architecture choices we have made. Section 5 overviews related work and background information. Section 6 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[1]. We start with the building and the ranking of the text graph. Then, we overview the summary, keyphrase and relation extraction and the creation of the Prolog database that constitutes the logical model of the document, to be processed by the dialog engine.

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 [13].

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.

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

We use unique sentence identifiers and unique lemmas[2] 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 redirect the dependency edges toward nouns with subject and object roles, as shown for a simple short sentence in Fig. 2 as *"about"* edges.

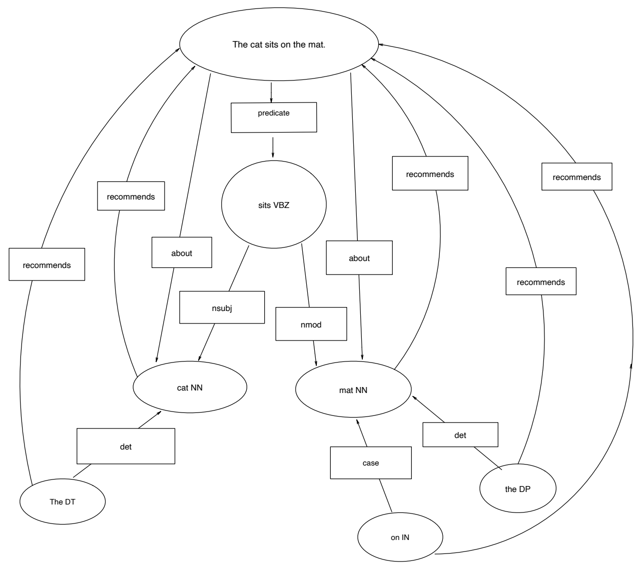

Fig. 2. Dependency graph of a simple sentence with redirected and newly added arrows

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 sentences recommend verbs with predicate function from where their recommendation spreads to nouns relevant as predicate arguments (e.g., having subject or object roles).

By using the PageRank implementation of the **networkx** toolkit[3], 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 [4], is shown in Fig. 3.

[2] 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.

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

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

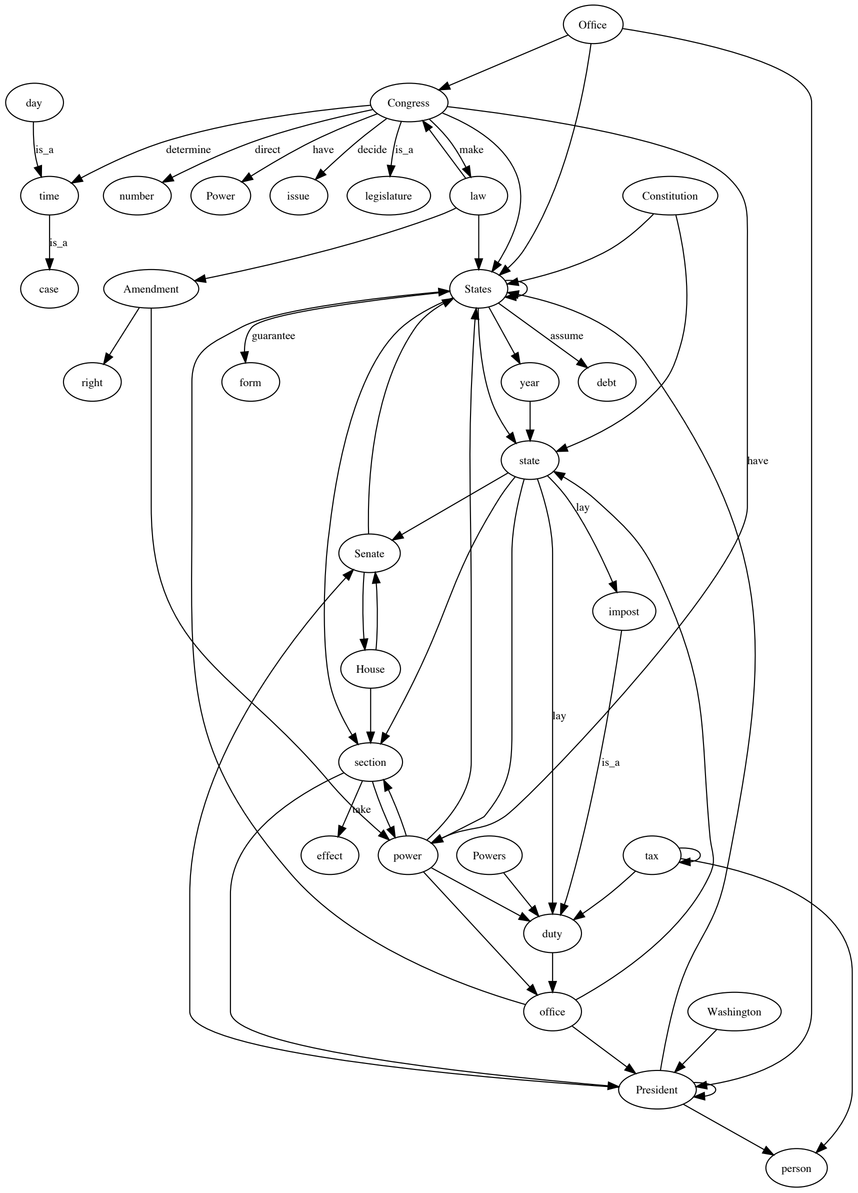

Fig. 3. Text graph of the highest ranked words in the U.S. Constitution

2.2 Pre- and post-ranking graph refinements

The algorithm induces a form of automatic stopword 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 focussed 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.

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 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 [14] hypernyms and hyponyms[5]. Similarly, "`part_of`" relations are extracted using meronyms and holonyms[6]. 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

[5] More general and, respectively, more specific concepts.

[6] Concepts corresponding to objects that are part of, and, respectively, have as part other objects.

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.

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, 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. `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

They provide a relational view of a document in the form of a 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 [15], abductive reasoners [16] or via Answer Set Programming systems [17]. 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 [4, 18, 3].

We have applied this process to the *Krapivin document set* [19], a collection of **2304** research papers annotated with the authors' own keyphrases and abstracts.

The resulting 3.5 GB *Prolog dataset*[7] is made available for researchers in the field, interested to explore declarative reasoning or text mining mechanisms.

3.2 The Prolog interface

We use as a logic processing tool the open source SWI-Prolog system[8] [20] that can be called from, and can call Python programs using the `pyswip` adaptor[9]. 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 paper sizes of 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[10]. 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".

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.

[7] `http://www.cse.unt.edu/~tarau/datasets/PrologDeepRankDataset.zip`
[8] `http://www.swi-prolog.org/`
[9] `https://github.com/yuce/pyswip`
[10] And also speak them out if the `quiet` flag is off.

Short-time 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.

Sentence selection Answer sentence selection happens by a combination of several interoperating algorithms:

– use of *personalized PageRank* [21, 22] 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 currently 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.

3.5 Interacting with the dialog engine

The following example shows the result of a query on the US Constitution document.

```
>>> 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 dec-
laration , 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 con-
tinue to discharge the same as Acting President ; otherwise , the President shall resume
the powers and duties of his office .
```

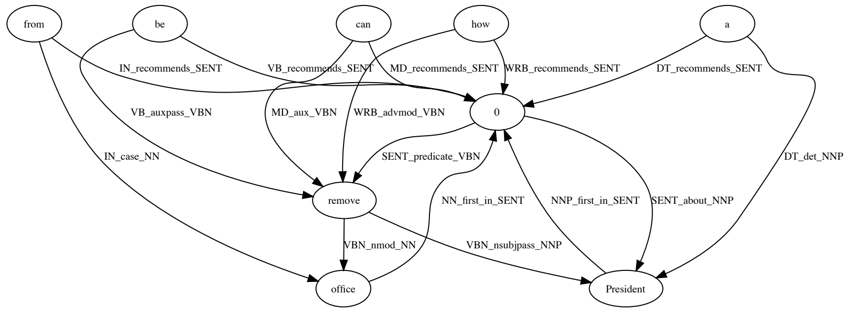

Fig. 4. Graph of query on the U.S. Constitution

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).
```

Our next example uses an ASCII version of Einstein's 1920 book on relativity, retrieved from the Gutenberg collection[11] 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 .

[11] `https://www.gutenberg.org/files/30155/30155-0.txt`

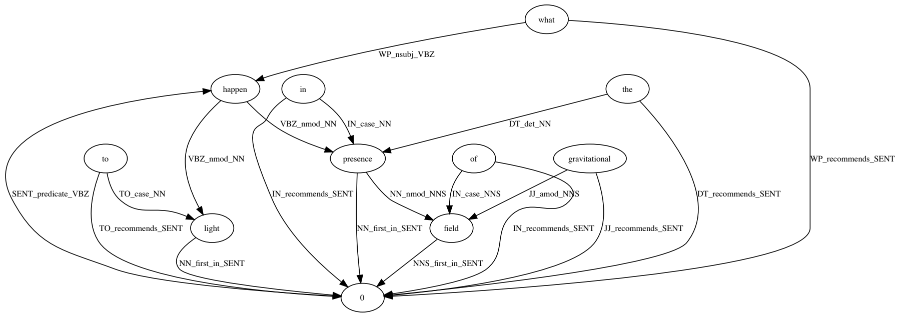

Fig. 5. Graph of query on Einstein's book on Relativity

The query graph is shown in Fig. 5. 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[12], 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.

4 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 like also 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

[12] `https://www.tesla.com/sites/default/files/model_3_owners_manual_north_america_en.pdf`

already Turing-complete), but also that modalities, beliefs, sentiments, hypothetical and contrafactual judgements 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 [23].

5 Related work

Dependency parsing The Stanford neural network based dependency parser [7] is now part of the Stanford CoreNLP toolkit[13], which also comes with part of speech tagging, named entities recognition and co-reference resolution [13]. Its evolution toward the use of Universal Dependencies [24] makes tools relying on it potentially portable to over **70** languages covered by the Universal Dependencies effort [14].

Of particular interest is the connection of dependency graphs to logic elements like predicate argument relations [25]. The mechanism of automatic conversion of constituency trees to dependency graphs described in [9] provides a bridge allowing the output of high-quality statistically trained phrase structure parsers to be reused for extraction of dependency links.

We analyze dependency links and POS-tags associated to their endpoints to 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 In TextRank [11, 12] keyphrases are using a co-occurrence relation, controlled by the distance between word occurrences: two vertices are connected if their corresponding lexical units co-occur within a sliding window of 2 to 10 words. Sentence similarity is computed as content overlap giving weights on the links that refine the original PageRank algorithm [26, 27]. TextRank needs elimination of stop words and reports best results when links are restricted to nouns and adjectives. In [10] several graph centrality measures are explored and [28] offers a comprehensive overview on graph-based natural language processing and related graph algorithms. Graph-based and other text summarization techniques are surveyed in [29] and more recently in [30]. Besides ranking, elements like coherence via similarity with previously chosen sentences and avoidance of redundant rephrasings are

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

shown to contribute to the overall quality of the summaries. *The main novelty of our approach in this context is building the text graph 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, with [8] pointing out to their role as a generic interface between parsers and relation extraction systems. In [31] several models grounded on syntactic patterns are identified (e.g., subject-verb-object) that can be mined out from dependency graphs. Of particular interest for relation extraction facilitated by dependency graphs is the shortest path hypothesis that prefers relating entities like predicate arguments that are connected via a shortest paths in the graph [32]. To facilitate their practical applications to biomedical texts, [33] extends dependency graphs with focus on 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 [34], in fact a prequel to the TextRank paper [11]. 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 [35, 8, 36].

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, but extensions are planned to integrate Wikipedia content, via the `dbpedia` database[15] and to extract more elaborate logic relations using a Prolog-based semantic parser like Boxer [37].

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 [1–4], although recent work like [18] 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.

6 Conclusions

The key idea of the paper has evolved from our search for synergies between symbolic AI and emerging machine-learning based natural language processing tools. 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

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

on heterogeneous, but syntactically and semantically meaningful text units (words and sentences) resulting in a web of interleaved links, mutually recommending 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 interest. Its applications range from assistive technologies to visually challenged people, voice interaction with user manuals, teaching from K-12 to graduate level and interactive information retrieval from complex technical or legal documents.

Last but not least, we have used our system's front end to generate the Prolog dataset at `http://www.cse.unt.edu/~tarau/datasets/PrologDeepRankDataset.zip`, derived from more than 2000 research papers and made it available to other researchers using logic programming based reasoners and content mining tools.

Acknowledgment

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

References

1. Lierler, Y., Inclezan, D., Gelfond, M.: Action languages and question answering. In Gardent, C., Retoré, C., eds.: IWCS 2017 - 12th International Conference on Computational Semantics - Short papers, Montpellier, France, September 19 - 22, 2017, The Association for Computer Linguistics (2017)
2. Inclezan, D., Zhang, Q., Balduccini, M., Israney, A.: An ASP methodology for understanding narratives about stereotypical activities. TPLP **18**(3-4) (2018) 535–552
3. Mitra, A., Clark, P., Tafjord, O., Baral, C.: 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 (2019) 3003–3010
4. Inclezan, D.: Restkb: A library of commonsense knowledge about dining at a restaurant. In Bogaerts, B., Erdem, E., Fodor, P., Formisano, A., Ianni, G., Inclezan, D., Vidal, G., Villanueva, A., Vos, M.D., Yang, F., eds.: Proceedings 35th International Conference on Logic Programming (Technical Communications) , Las Cruces, NM, USA, September 20-25, 2019. Volume 306 of Electronic Proceedings in Theoretical Computer Science., Open Publishing Association (2019) 126–139
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. CoRR **abs/1706.03762** (2017)
6. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR **abs/1810.04805** (2018)
7. Chen, D., Manning, C.: 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 (2014) 740–750

8. Adolphs, P., Xu, F., Li, H., Uszkoreit, H.: Dependency Graphs as a Generic Interface between Parsers and Relation Extraction Rule Learning. In Bach, J., Edelkamp, S., eds.: KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings. Volume 7006 of Lecture Notes in Computer Science., Springer (2011) 50–62
9. Choi, J.D.: Deep Dependency Graph Conversion in English. In: Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories. TLT'17, Bloomington, IN (2017) 35–62
10. Erkan, G., Radev, D.R.: LexRank: Graph-based Lexical Centrality As Salience in Text Summarization. J. Artif. Int. Res. **22**(1) (December 2004) 457–479
11. Mihalcea, R., Tarau, P.: TextRank: Bringing Order into Texts. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2004), Barcelona, Spain (July 2004)
12. Mihalcea, R., Tarau, P.: An Algorithm for Language Independent Single and Multiple Document Summarization. In: Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP), Korea (October 2005)
13. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In: Association for Computational Linguistics (ACL) System Demonstrations. (2014) 55–60
14. Fellbaum, C.: WordNet, An Electronic Lexical Database. The MIT Press (1998)
15. Schulte, C.: Programming constraint inference engines. In Smolka, G., ed.: Proceedings of the Third International Conference on Principles and Practice of Constraint Programming. Volume 1330 of Lecture Notes in Computer Science., SchloßHagenberg, Austria, Springer-Verlag (October 1997) 519–533
16. Denecker, M., Kakas, A.C.: Abduction in logic programming. In: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I, London, UK, UK, Springer-Verlag (2002) 402–436
17. Schaub, T., Woltran, S.: Special Issue on Answer Set Programming. KI **32**(2-3) (2018) 101–103
18. Olson, C., Lierler, Y.: Information extraction tool text2alm: From narratives to action language system descriptions. In Bogaerts, B., Erdem, E., Fodor, P., Formisano, A., Ianni, G., Inclezan, D., Vidal, G., Villanueva, A., Vos, M.D., Yang, F., eds.: Proceedings 35th International Conference on Logic Programming (Technical Communications) , Las Cruces, NM, USA, September 20-25, 2019. Volume 306 of Electronic Proceedings in Theoretical Computer Science., Open Publishing Association (2019) 87–100
19. Krapivin, M., Autayeu, A., Marchese, M.: Large Dataset for Keyphrases Extraction. Technical Report DISI-09-055, DISI, Trento, Italy (May 2008)
20. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming **12** (1 2012) 67–96
21. Haveliwala, T.H.: Topic-sensitive pagerank. In: Proceedings of the 11th International Conference on World Wide Web. WWW '02, New York, NY, USA, ACM (2002) 517–526
22. Haveliwala, T., Kamvar, S., Jeh, G.: An analytical comparison of approaches to personalizing pagerank. Technical Report 2003-35, Stanford InfoLab (June 2003)
23. Tarau, P., Blanco, E.: Dependency-based text graphs for keyphrase and summary extraction with applications to interactive content retrieval. ArXiv **abs/1909.09742** (2019)
24. de Marneffe, M.C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., Manning, C.D.: Universal Stanford dependencies: A cross-linguistic typology. In: Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), Reykjavik, Iceland, European Languages Resources Association (ELRA) (May 2014) 4585–4592

25. Choi, J.D., Palmer, M.: Transition-based Semantic Role Labeling Using Predicate Argument Clustering. In: Proceedings of the ACL 2011 Workshop on Relational Models of Semantics. RELMS '11, Stroudsburg, PA, USA, Association for Computational Linguistics (2011) 37–45
26. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project (1998)
27. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems **30**(1–7) (1998) 107–117 http://citeseer.nj.nec.com/brin98anatomy.html.
28. Mihalcea, R.F., Radev, D.R.: Graph-based Natural Language Processing and Information Retrieval. 1st edn. Cambridge University Press, New York, NY, USA (2011)
29. Nenkova, A., McKeown, K.R.: A Survey of Text Summarization Techniques. In Aggarwal, C.C., Zhai, C., eds.: Mining Text Data. Springer (2012) 43–76
30. Allahyari, M., Pouriyeh, S.A., Assefi, M., Safaei, S., Trippe, E.D., Gutierrez, J.B., Kochut, K.: Text Summarization Techniques: A Brief Survey. CoRR **abs/1707.02268** (2017)
31. Stevenson, M., Greenwood, M.: Dependency Pattern Models for Information Extraction. Research on Language & Computation **7**(1) (March 2009) 13–39
32. Bunescu, R.C., Mooney, R.J.: 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, Stroudsburg, PA, USA, Association for Computational Linguistics (2005) 724–731
33. Peng, Y., Gupta, S., Wu, C., Shanker, V.: An extended dependency graph for relation extraction in biomedical texts. In: Proceedings of BioNLP 15, Association for Computational Linguistics (2015) 21–30
34. Mihalcea, R., Tarau, P., Figa, E.: PageRank on Semantic Networks, with application to Word Sense Disambiguation. In: Proceedings of The 20st International Conference on Computational Linguistics (COLING 2004), Geneva, Switzerland (August 2004)
35. Li, W., Zhao, J.: TextRank Algorithm by Exploiting Wikipedia for Short Text Keywords Extraction. 2016 3rd International Conference on Information Science and Control Engineering (ICISCE) (2016) 683–686
36. Mihalcea, R., Csomai, A.: Wikify!: Linking Documents to Encyclopedic Knowledge. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management. CIKM '07, New York, NY, USA, ACM (2007) 233–242
37. Bos, J.: Open-domain semantic parsing with boxer. In Megyesi, B., ed.: Proceedings of the 20th Nordic Conference of Computational Linguistics, NODALIDA 2015, May 11-13, 2015, Institute of the Lithuanian Language, Vilnius, Lithuania, Linköping University Electronic Press / ACL (2015) 301–304