Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

Graph Based Pattern Classification for NLU and Slot Filling: Approach and Analysis

Julian Eggert, Johane Takeuchi

2022

Preprint:

Copyright ACM 2022. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in NLPIR '22: Proceedings of the 2022 6th International Conference on Natural Language Processing and Information Retrieval, https://doi.org/10.1145/3582768.3582770.

© Julian Eggert 2022. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in the ACM DL conference proceedings, http://dx.doi.org/10.1145/3582768.3582770.

Graph Based Pattern Classification for NLU and Slot Filling: Approach and Analysis

Julian Eggert
julian.eggert@honda-ri.de
Honda Research Institute Europe
Offenbach/Main, Germany

Johane Takeuchi
johane.takeuchi@jp.honda-ri.com
Honda Research Institute Japan
Wako, Japan

ABSTRACT

In Natural Language Understanding, semantic parsing refers to the task of extracting meaningful words from text, with slot filling being a special case. At the same time, syntactic parsing deals with the identification of syntactic structure in the text. In this paper, we analyze to which extent syntactic structure can be used for semantic parsing. For this purpose, we represent the syntactic structure of sentences from an annotated database as graphs, incrementally store a large number of graph prototypes in a knowledge base, and approach the slot filling problem as a graph matching problem. We analyze how this approach scales with the number of prototypes and show that it provides a lightweight framework that can be efficiently used to support semantic parsing tasks.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Knowledge representation and reasoning**; *Semantic networks.*

KEYWORDS

Slot Filling, Pattern Classification, Syntactic Parse Trees (SPT), Semantic Parsing, Graph Databases, Subgraph Matching, Partial Graph Match, Natural Language Understanding (NLU)

ACM Reference Format:
Julian Eggert and Johane Takeuchi. 2022. Graph Based Pattern Classification for NLU and Slot Filling: Approach and Analysis. In *Proceedings of the 6th International Conference on Natural Language Processing and Information Retrieval (NLPIR'22).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Natural Language Processing (NLP), and in particular, Natural Language Understanding (NLU), is a crucial component supporting a broad set of Artificial Intelligence (AI) functions such as automated reasoning, question answering, dialog generation or text content analysis. It is also the basis for many applications involving AI agents (e.g. automatic chat bots) and robotic assistants that interact with humans via language (speech or text) as an interface. In NLU, the term "understanding" is much debated, but it basically means "making sense of", in a way that a system can utilize the NLP for adjusting or improving its behavior, and in particular its interaction (language-based, but also others) with the human. The purpose of NLU is therefore "semantic parsing", i.e., to transform a language input semantically closer to a representation that can support the behavioral response of an AI system. One way to approach this is to identify structures ("slots") in a text utterance that carry special semantic importance. These structures can be labeled (by humans) in the text with a "semantic type" and then be used for a supervised classification system. With such a system at hand, the task is then to identify the slots in a given text ("slot labeling / slot detection"), or, correspondingly, extract the text snippets that correspond to a certain type of slot ("slot filling"). For an overview of current joint slot filling and sentence categorization ("itent detection") approaches, see e.g. [11].

However, starting from scratch, because of the variability and complexity of human language, such slot classification systems require large amounts of training data. To alleviate this problem, for a decent generalization over text variations, priors containing general knowledge about the structure of language can be included. This might be in the form of pretrained large language models as researched lately, see e.g. the models from OpenAI called "gpt2" and "gpt3" (Generative Pretrained Transformer) [7], that comprise correlation statistics about the appearance of words in texts and are used very successfully for a large variety of NLU related tasks [2]. An alternative are structural decompositions of the text based on syntactic analysis. In this paper, we propose to leverage general knowledge about the structure of language as given by syntactic models. The NLU task of slot filling can then be seen as analogous to identifying certain elements of the syntatic structure, and we propose to achieve this using graph-based pattern matching techniques. In the subsequent sections we explain our approach in detail and analyze its slot filling and generalization capabilities.

The idea of using the information from syntactic decomposition has been used in different contexts. E.g. for knowledge graph based question answering systems, a graph isomorphism detection allows to map NLU queries to a set of query templates [10]. In other works, syntactic structures have been manually inspected and classified into characteristic patterns in preprocessing steps [9], or they have been used to find a direct match between language and knowledge base structures [1].

To our knowledge, however, the performance of syntactic decompositions in combination with graph-based approaches for generalized slot filling has not been analyzed systematically before.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
NLPIR'22, December 16–18, 2022, Bangkok, Thailand
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 APPROACH

2.1 Syntactic Parse Trees

In NLU tasks, a frequent approach to understanding the meaning of the words (i.e., their semantics) in a sentence occurs by analysing typical components in terms of types of words and their dependencies. One way is by using so-called Syntactic Parsing or Dependency Parsing, which is the task of mapping the words of a sentence to a syntactic structure. The most widely used syntactic structures have the form of a tree with made up of nodes and branches. For visualization, the trees are often shown upside-down, see Fig. 1 for an example of such a syntactic parse tree resulting from a simple sentence.

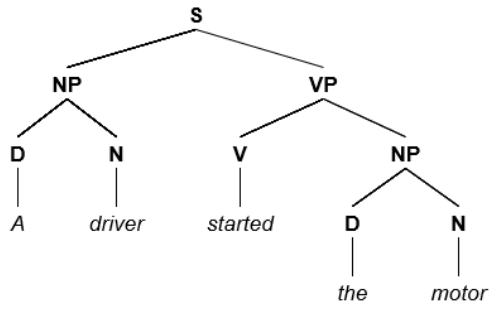

Figure 1: Decomposition of a simple sentence into a tree with syntactic components. S=Sentence, NP=Noun phrase, VP=Verb phrase, D=Determiner, N=Noun, V=Verb.

Syntactic parse trees (SPT) therefore reflect the structure of a sentence in terms of syntactic components and dependencies. They can be used to represent "typical types of sentences", in the sense that all sentences of one type will share the same syntactic structure although they might be composed of totally different words.

It has been suggested that humans, to understand the meaning of natural language sentences, first process the syntactic structure of a sentence and then use this to infer the meaning. For some evidence of brain structures supporting this idea see e.g. [5].

To extract meaning from sentences, we exploit the fact that there is a correlation between typical syntactic structures and categories and the role a word plays to convey certain semantic in a sentence. For example, verbs are important syntactic elements in a sentence because they provide a frame to connect other elements like objects or subjects. In terms of semantics, as e.g. important for an artificial intelligent agent that has to make sense of a sentence, verbs most often convey information about an action that can be executed by an actor to modify the state of an object when it is applied on it.

Since slot filling is the task of finding meaningful or important component categories of a sentence, it is of no surprise that there is also a correlation between relevant slots and syntactic structures. The idea is therefore to exploit this correlation and use the SPT structures to identify slots.

However SPT's have been developed mainly with linguistic categories and applications in mind, like e.g. grammar checking, and implemented with mixtures of parsing and machine learning algorithms. The results are not unique; a given sentence can have multiple parse trees, both from the purely syntactic but also from the semantic viewpoint, with different approaches developed for further disambiguation. This ambiguity introduces potential errors in the classification of sentence components and in all subsequent semantic analysis. Therefore it is unclear to which extent SPT's are suited for appropriately extracting slot categories.

Let's have a look at a specific decomposition of a more complex sentence, as used later for the slot filling task experiments.

```
"I'm looking for an expensive restaurant in the new
    addenbrookes district."
```

Here, the SPT is composed by the syntactic parts-of-speech categories for each word, and the dependency types between the words. Using e.g. spacy [4, 8], we can visualize the resulting tree together with the original sentence as follows.

2.2 Representing Prototypical Sentences as Graphs

As introduced in the last subsection, SPT's are a practical way to represent syntactic structures of sentences. Basically, the decomposition as shown in Fig. 2 is a labeled graph (labeled because there are different types of nodes and relationships), so that we can use graph algorithms to anaylze the graphs, e.g. graph similarity algorithms to compare graphs with each other.

The basic idea is to store a number of prototypical SPT's as graphs in a knowledge base, and then to see if the SPT graph of a new sentence matches any of the already known prototypes. The prototypical SPT's would be stored together with the additional slot information. In case that there is a match between a prototype and the new sentence graph, an evaluation of the graph elements would establish a correspondence between the elements to identify the slot elements from the new graph.

In its simplest form, this works by exact matches, i.e., in order to be recognized a new sentence would have to have exactly the same syntactic structure as one of the prototypes. Generalization occurs by means of word variation, so that any sentence that is a syntactically equivalent transformation would also match. In this sense, the two sentences

```
"I'm looking for an expensive restaurant in the new
    addenbrookes district"
"Tom was searching for an affordable cafe in the old city
     area."
```

would both be recognized, since they share the same syntactic structure, and therefore they have the same SPT resp. the same labeled graph.

In fig. 3, we show the prototypical graph corresponding to the SPT of a sentence like "I'm looking for a cheap coffee shop in the city center area". In addition to the visualized nodes and links from the parse tree, the nodes for "cheap", "shop" (representing the "coffee shop" compound) and "area" (representing the "city center area" compound) are tagged with their corresponding slot category labels.

In fig. 4, we finally show an excerpt of the prototype knowledge base used for pattern retrieval and slot filling.

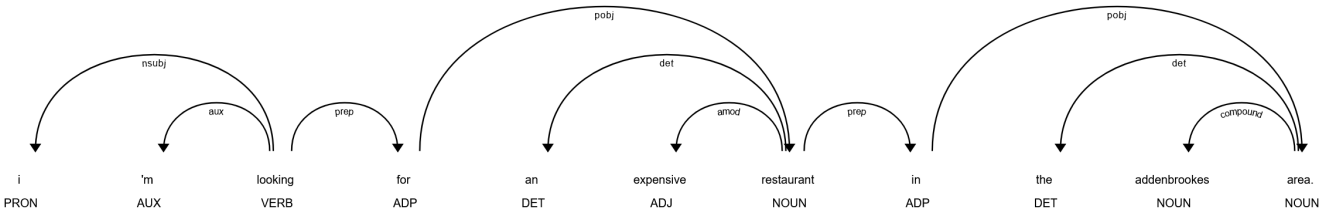

Figure 2: Syntactic parse-tree decomposition of a sample sentence as used in the experiments. Each token/word is sorted into syntactic "part-of-speech" categories. In addition dependencies between the tokens is indicated as arrows connecting tokens.

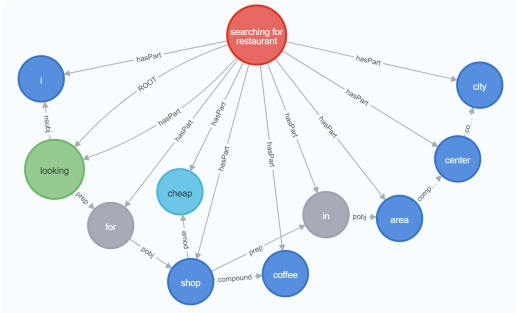

Figure 3: A graph pattern prototype for the sentence "I'm looking for a cheap coffee shop in the city center area" as it is represented in the KB. The red node represented the pattern hub, and all tokens are connected to it via "hasPart" links. The verb (green node) is the root of the SPT, and connects to all other components like nouns, pronouns and proper nouns (blue nodes) or adjective and adverbs (cyan node). In addition, nodes corresponding to slots are tagged with their slot category labels.

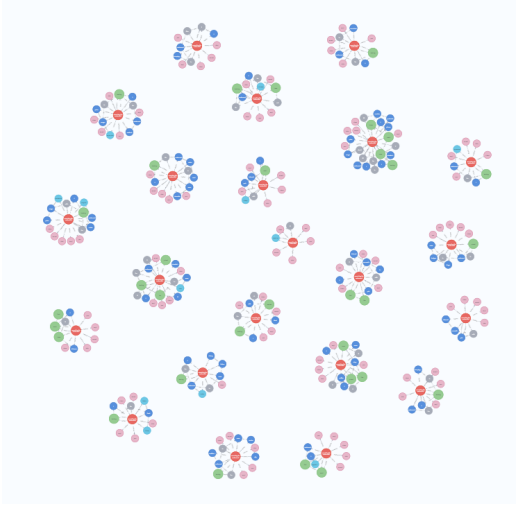

Figure 4: An excerpt of the knowledge base, consisting of prototype patterns for all the different syntactic structures from the learned sentence samples.

2.3 Partial Graph Matching

Even if we do not look at single words, but at the syntactic structure level, the number of different patterns that can be generated is still prohibitively large. This is because every single word or order variation would have to be covered by a separate prototype.

To alleviate this, we introduce 3 further extensions into our approach, however the approach also works without them. As a first extension, we simplify the sentences syntactically, removing single filler words (like "yes", "um", "hey", "hi" and similar), and other syntactically negligible structures. This can be done in a transparent way by transforming the simplified sentences back to the full sentences after the slot identification.

As a second extension, we reduce the complexity of the syntactic components of the SPT. In particular, we group together the components that fulfill a similar purpose and appear in the same parts of a SPT, like nouns, pronouns and proper nouns, into a single syntactical group.

As a further extension we allow partial graph matching for our prototypes. This means that a graph in the KB will also serve as a prototype to new sentences with a syntactic structure that matches partially, i.e., where the new sentence is a subgraph of the prototype. Consider e.g. the 2 sentences:

```
"I am looking for a restaurant in the city"
"I am actually looking for a restaurant in the city
    center district"
```

A straightforward prototype extraction would have to store dedicated prototypes for each of the sentences. With partial graph matching, the first sentence structure is a subgraph of the second sentence structure, because the additional words ("actually", "center district") are basically adjectival, adverbial or other modifier to the main structure that expresses as additional leafs or subtrees on the prototype graph. For partial graph matching, we require that all syntactic structures (all its nodes and relationships) of the smaller graph must be present in the larger graph.

The extensions, and in particular the partial graph matching, allow a single prototype to represent several sentence structures. To reduce the number of prototypes further, we included this into the prototype learning scheme in the sense that subgraph-compatible prototypes are fused together into a single prototype. This occurs by the following learning procedure:

```
1. For a new pattern (i.e., a sentence with slot
    annotation), check if it matches with any of the
    patterns in the KB
```

```
2. If not, include the new pattern into the KB by storing
     its labeled graph.
3. Check back if any of the other previously existing
    patterns qualifies as a subgraph of the newly
    inserted graph
4. If yes, remove the previously existing pattern(s)
```

2.4 Retrieval Ambiguity and Slot Filling

Regardless of the number of prototypes, the matching of the SPT of a new sentence to the prototypical patterns is ambiguous. Consider the case of e.g. a very compact sentence like "find a restaurant", we immediately notice that due to its simplicity this matches a large number of prototypes, i.e., the pattern matching is ambiguous [1]. In this case, we have to decide from which pattern to extract the slot information.

For disambiguation, we have implemented the following algorithm which

```
1. Collect the set of prototype patterns that match to a
    sentence
2. For each slot category and slot value, each matching
    prototype provides one vote
3. For each slot category, the slot value with the
    largest number of votes is returned.
```

3 EVALUATION

3.1 Data

For the evaluation we used an available data set with slot annotations from a restaurant-finding dialog domain. The NLU task is to understand human sentences that ask for restaurants in the vicinity. The slot categories comprise a "restaurant type" (slot label "restaurant", e.g. normal restaurant, pub, coffee shop, etc.), the type of food served ("food": e.g. international, mexican, sea food, fast food ...), the price range ("pricerange": cheap, expensive, moderate, any, etc.) and the area (slot label "area") where the restaurant is located.

The data set consists of 1095 different, manually annotated sentences. Annotations can comprise a single word or a group of contiguous words. The annotations were added to the sentences in the spacy annotation style with square braquets indicating the slot and round braquets the slot category. Examples are shown below:

```
"I'm looking for a [vegetarian](food) [restaurant](type)
    in the [girton area](area)"
"Yes I'm searching for a [coffee shop](type) serving [any
     kind of food](food) at a [cheap](pricerange) price
    range."
```

3.2 Annotation Preprocessing

In the last section, we have explained how single nodes in the graph representation for a prototypical sentences can be tagged to with slot category labels. Since each node in the graph corresponds to a SPT token this means that basically only single tokens (usually corresponding to single words) can be annotated with slot information and retrieved accordingly. This is in mismatch with the human annotation of a group of contiguous words as a single slot, therefore we have to preprocess the slot annotation in a specific way.

[1]We analyze this "retrieval ambiguity" in sections 3.3 and 3.4

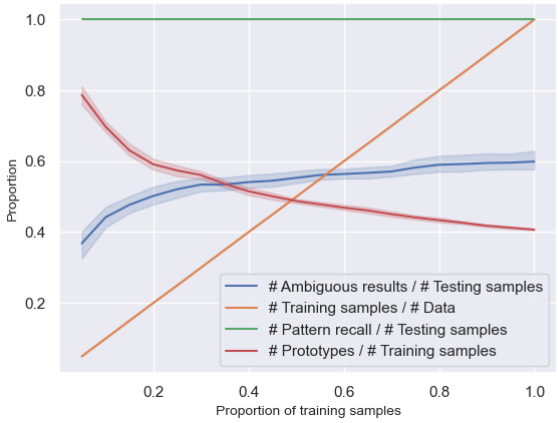

Figure 5: Pattern storage and retrieval properties of the proposed approach. Knowledge bases (kbs) of different sizes are trained with a subset of the 1095 sentences with annotated slots. The KB is then tested with syntactically equivalent sentences without annotation. On average, one prototype serves to represent about more than 2 training samples (red curve). At the same time, ambiguity during recall increases (blue curve). All curves are plotted with confidence intervals gained from multiple independent simulation runs.

One observation was that the group-based slot annotation usually correlated with syntactic groupings, i.e., human annotators considered groupings as meaningful if these were also compounds of the corresponding SPT's. For example, typical groups where given by expressions like "any type of food" or "in the addenbrookes area". In the SPT's, these groups correspond to compounds formed by a subtree of the SPT of the sentence. E.g., in figure 2, the last 4 words on the right, "in the addenbrookes area", form a modifier compound to the noun "restaurant" that can be extracted starting from the preposition "in" and following the relationships in a forward manner. In this sense, the word "in" is the root word of this compound, and if we annotate this word with the corresponding slot category we are able to retrieve the entire compound from the attached subtree.

To enable with the annotation of compounds, we therefore proceeded according to the following algorithm:

```
1. Search for the root word of each annotation compound
2. Reduce the annotation to this root word
3. Learn prototypes with the root word slot labels
4. During slot recognition, retrieve the slot root words
5. After that, expand the root words back to compounds
```

3.3 Training and Testing

The knowledge base was trained with the annotated sentences with partial graph matching enabled, so that a single prototype can serve to match also all shorter patterns and sentences that a pattern might contain a valid subgraphs.

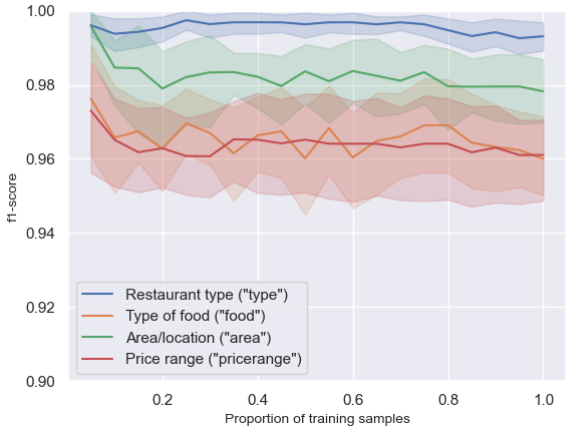

Figure 6: Confirmation of the slot filling performance of the proposed approach, plotted for the different slot categories for increasing number of training samples. All curves are plotted with confidence intervals gained from multiple independent simulation runs.

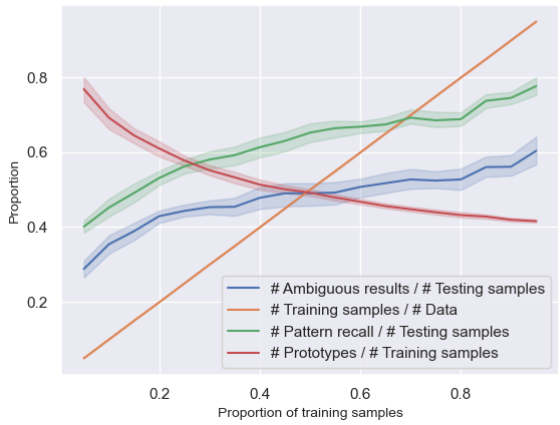

Figure 7: Pattern storage and retrieval properties of the proposed approach. Different knowledge bases (kbs) were trained with a subset of increasing size of the 1095 sentences with annotated slots. The KBs are then tested with previously unseen sentences without annotation.

For the evaluations, 10 runs were computed for each KB parameter setting, evaluating each run with independently subsampled data.

Our first question is whether we can use graph prototypes to first store slot annotations on prototypical sentences and then to retrieve them back using syntactically equivalent sentences. Here we are particularly interested in ambiguities introduced by correlations and overlaps between the graphs generated for the different sentences.

For this purpose, we learned a KB from subsets of the 1095 annotated sentences, and used a syntactically fully equivalent subset to retrieve the data. The size of the subset varied from 0 to 1095 (x-axis, proportion of training samples from 0 to 1).

The results are shown in figure 5.

It shows the general indicators for the KB. With increasing size of the KB, new patterns are incorporated step by step, but at a decreasing rate, so that at the beginning there is nearly one prototype per training pattern, but at the end on average each prototype represented more than 2 training patterns (red line). At the same time, the retrieval ambiguity [2] increases, however at a sublinear pace (blue line). The pattern recall (green line) serves as control, indicating that during retrieval there is at least one match for every tested pattern.

The slot filling performances are shown in fig. 6 in terms of their f1-score. Here we see that retrieval is good but not perfect. Since all tested patterns are matched to patterns in the KB (previous figure, green line), the errors are due to the ambiguity (previous figure, blue line). I.e., if more that one pattern prototype matches, the subsequent slot retrieval procedure sometimes chooses the wrong pattern prototype for the slot filling. An additional source for small errors is also the SPT procedure.

From fig. 6, we can also see how the different slot categories vary in terms of their recognition performance. Syntactically "easy" structures like the restaurant type, which always corresponds to the object of a sentence, lead to very high recognition rates whereas syntactically more variable and more complex constructs are more difficult to recognize from their syntactic structure.

As a conclusion of this section, we can see that slot filling by SPT pattern storage and retrieval works well, that the number of stored prototypes can be reduced by introducing partial graph matching and that the ambiguities by matching with multiple prototypes still allow for a successful slot identification.

3.4 Generalization Performance

Our next research question is now concerned with the overall performance of the system, i.e., its generalization to previously unseen sentences with possibly different syntactic structures. In this case, this will largely depend on the coverage of the space of all syntactic patterns relevant for the task by the learned prototypes.

For this purpose, we used disjoint subsets of training and testing samples taken from the annotation dataset. For each parameter set, we aggregated the results gained from running the training and testing on 10 different knowledge bases and evaluated the results as a function of the training to test ratio.

In fig. 7 we show again the general system properties. The overall number of prototypes per training sample and the ambiguity in the recall is similar to that of fig. 5. The pattern recall, however, is now largely in sync with the ambiguity curve.

The slot filling performance is shown in fig. 8, again given by the f1-score. In comparison with fig. 5 we see however, that the overall performance curves are similar to the pattern recall, an a closer

[2]Retrieval ambiguity is given by the number of testing samples that matches to more than one prototype pattern.

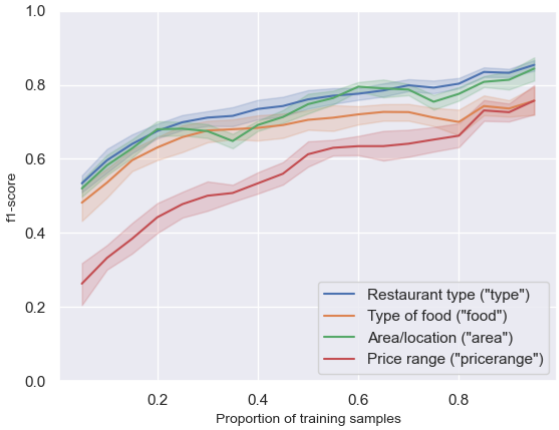

Figure 8: Generalization performance of the proposed approach, tested with previously unseen sentences, plotted for the different slot categories for increasing number of training vs. testing samples.

inspection reveals that the recall error dominates also the slot filling performance. This means that the majority of errors occurs because of missing matching prototypes, and not by subsequent errors during pattern or slot ambiguity resolution. This also means that with larger knowledge bases containing more prototypes with better coverage of all possible sentences, generalization performance can increase further.

3.5 Implementation Details

For simplicity and scaling purposes, we have used an available graph database for the implementation of the pattern prototype knowledge base and the pattern matching processes.

We implemented the approach with a Neo4j [6] database accessed via python. The pattern matching processes were written in Cypher [3] as a query language, which allowed to concentrate in a declarative way on describing what we wanted to retrieve. The SPT was generated via spacy [8].

The training times for the complete dataset of 1095 annotated sentences were approximately 180 sec on a single-core Intel i5 computer. Slot recognition times were 535 sec from cold start and 40 sec after graph database cache warming up. We did not perform about any kind of optimization of the approach so far.

4 DISCUSSION

By construction, the approach has a guaranteed generalization capability that goes beyond the available data set, since sentences with syntactic structures contained in one of the prototype patterns will also evaluate correctly, always assuming that the SPT generation does not introduce further errors. We did not include this generalization performance in our analysis since it is straightforward, for practical purposes, however, this largely extends the applicability range of the approach.

Overall, the approach generalizes on several different levels. First, it generalizes over word variation as indicated above. Second, it generalizes based on the partial graph matching, meaning that e.g. sentence variations compatible with subgraphs of KB prototypes are also covered correctly. And third, it generalizes over sentence variation by the introduced syntactic simplification and syntactic component grouping, resulting in an overall reduced number of prototypical patterns.

A beneficial property of the approach is also that it is a fully incremental system. At each presentation of a new sentence, it can be evaluated if this is compatible with its set of prototypes in the KB, enabling an insertion of additional properties during runtime. By this way, the KB can be extended further as required.

From the analysis of section 3.4, extrapolating the performance curves we conclude that the main reason for a limitation of the performance is the number of prototypes extracted from the data. It seems that in our case the resulting about 440 prototypical patterns (from the full dataset) are insufficient for a coverage that can be considered to be complete. An incremental extension of the system to the missing patterns would be very beneficial here.

Concluding, we have shown that a graph-based approach which leverages the set of common, prototypical syntactic structures encountered in labeled dataset of natural text expressions can be used efficiently for semantic parsing tasks. In particular for slot filling we have shown how to create an incremental knowledge base of prototypes which a novel sentence can be compared against. The presented approach is, particularly compared to the current developments based on large-language models, very lightweight in terms of implementation and recall, and it provides direct incremental capabilities in case that the set of prototypes has to be expanded.

REFERENCES

[1] Kinjal Basu, Sarat Chandra Varanasi, Farhad Shakerin, Joaquín Arias, and Gopal Gupta. 2021. Knowledge-driven Natural Language Understanding of English Text and its Applications. https://arxiv.org/abs/2101.11707
[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. https://arxiv.org/abs/2005.14165
[3] Cypher. 2022. Cypher documentation. https://neo4j.com/developer/cypher
[4] M. Honnibal and I. Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. (2017).
[5] Alessandro Lopopolo, Antal van den Bosch, Karl Magnus Petersson, and Roel M. Willems. 2020. Distinguishing Syntactic Operations in the Brain: Dependency and Phrase-Structure Parsing. *Neurobiology of Language* 2 (2020), 152–175.
[6] Neo4j. 2022. Neo4j documentation webpage. https://neo4j.com/docs
[7] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.
[8] spaCy. 2022. spaCy webpage. https://spacy.io
[9] Nadine Steinmetz, Ann-Katrin Arning, and Kai-Uwe Sattler. 2019. From Natural Language Questions to SPARQL Queries: A Pattern-based Approach. In *BTW 2019*, Torsten Grust, Felix Naumann, Alexander Böhm, Wolfgang Lehner, Theo Härder, Erhard Rahm, Andreas Heuer, Meike Klettke, and Holger Meyer (Eds.). Gesellschaft für Informatik, Bonn, -, 289–308.
[10] Daniel Vollmers, Rricha Jalota, Diego Moussallem, Hardik Topiwala, Axel-Cyrille Ngonga Ngomo, and Ricardo Usbeck. 2021. Knowledge Graph Question Answering Using Graph-Pattern Isomorphism. In *Studies on the Semantic Web*. IOS Press, -. https://doi.org/10.3233/ssw210038
[11] H. Weld, X. Huang, S. Long, J. Poon, and S. C. Han. 2021. A survey of joint intent detection and slot-filling models in natural language understanding. https://doi.org/10.48550/ARXIV.2101.08091