Semgrex and Ssurgeon, Searching and Manipulating Dependency Graphs

John Bauer
HAI
Stanford University
horatio@cs.stanford.edu

Chloé Kiddon
Dept of Computer Science
Stanford University
chloe.kiddon@gmail.com *

Eric Yeh
SRI International
eric.yeh@sri.com

Alex Shan
Dept of Computer Science
Stanford University
azshan@stanford.edu

Christopher D. Manning
Linguistics & Computer Science
Stanford University
manning@stanford.edu

Abstract

Searching dependency graphs and manipulating them can be a time consuming and challenging task to get right. We document *Semgrex*, a system for searching dependency graphs, and introduce *Ssurgeon*, a system for manipulating the output of *Semgrex*. The compact language used by these systems allows for easy command line or API processing of dependencies. Additionally, integration with publicly released toolkits in Java and Python allows for searching text relations and attributes over natural text.

1 Introduction

With the rapid growth in languages supported by Universal Dependencies (Nivre et al., 2020), being able to easily and quickly search over dependency graphs greatly simplifies processing of UD datasets. Tools which allow for searching of specific relation structures greatly simplify the work of linguists interested in specific syntactic constructions and researchers extracting relations as features for downstream tasks. Furthermore, converting existing dependency treebanks from non-UD sources to match the UD format is valuable for adding additional data to Universal Dependencies, and doing this conversion automatically greatly reduces the time needed to add more datasets to UD. Accordingly, several tools have been developed for searching, displaying, and converting existing datasets.

In this paper, we describe *Semgrex*, a tool which searches for regex-like patterns in dependency graphs, and *Ssurgeon*, a tool to rewrite dependency graphs using the output of *Semgrex*. Both systems are written in Java, with Java API and command line tools available. In addition, there is a Python interface, including using displaCy (Honnibal et al., 2020) as a library to visualize the results of searches and transformation operations. The tools can be used programmatically to enable further processing of the results or used via the included command line tools. Furthermore, a web interface[1] shows the results of applying patterns to raw text.

Semgrex was released as part of CoreNLP (Manning et al., 2014). As such, it has been used in several projects (section 4). Several existing uses of *Semgrex* make use of it via the API, one of the strengths of the system, such as the OpenIE relation extraction of (Angeli et al., 2015).

More recently, we have added new pattern matching capabilities such as exact edge matching, a new python interface, and performance improvements. The previously unpublished *Ssurgeon* adds useful new capabilities for transforming dependency graphs.

Many of the features described here are similar to those in Grew-Match (Guillaume, 2021) and Semgrex-Plus (Tamburini, 2017). Similar to *Ssurgeon*, Semgrex-Plus uses *Semgrex* to find matches for editing, but *Ssurgeon* supports a wider range of operations. Compared with Grew-Match, *Semgrex* and *Ssurgeon* have the ability to start with raw text and search for dependency relations directly.

2 Semgrex

Semgrex and *Ssurgeon* are publicly released as part of the CoreNLP software package (Manning et al.,

*Chloé Kiddon is now at Google Research.

[1] https://corenlp.run/

Proceedings of the 21st International Workshop on Treebanks and Linguistic Theories (TLT, GURT/SyntaxFest 2023), pages 67 - 73

Attribute		
word	pos	lemma
ner	idx	upos

Table 1: Commonly used attributes of words

2014)[2]. *Semgrex* reads dependency trees from CoNLL-U files or parses dependencies from raw text using the associated CoreNLP parser.

Search patterns are composed of two pieces: node descriptions and relations between nodes. A search is executed by iterating over nodes, comparing each word to the node search pattern and checking its relationships with its neighbors using the relation search patterns.

2.1 Node Patterns

Dependency graphs of words and their dependency relations are represented internally using a directed graph, with nodes representing the words and labeled edges representing the dependencies.

When searching over nodes, the most generic node description is empty curly brackets. This search matches every node in the graph:

```
{}
```

Within the brackets, any attributes of the words available to *Semgrex* can be queried. For example, to query a person's name:

```
{word:Beckett}
```

It is possible to use standard string regular expressions, using the match semantics, within the attributes:

```
{word:/Jen.*/}
```

A node description can be negated. For example, this will match any word, as long as it does not start with Jen, Jenny, Jennifer, etc.

```
!{word:/Jen.*/}
```

Node descriptions can also be combined:

```
{word:/Jen.*/;tag:NNP}
```

See Table 1 for a list of commonly used word attributes. Note that `ner` may require a tool which provides NER annotations, such as (Manning et al., 2014) (Java) or (Qi et al., 2020) (Python), as those are typically not part of UD treebanks.

Relation	Meaning
A < B	A is the dependent of B
A > B	A is the governor of B
A << B	A is part of a chain of deps to B
A >> B	A is part of a chain of govs to B
A . B	idx(A) == idx(B) − 1
A .. B	idx(A) < idx(B)
A - B	idx(A) == idx(B) + 1
A -- B	idx(A) > idx(B)
A $+ B	C, A < C, B < C, idx(A) == idx(B) − 1
A $- B	C, A < C, B < C, idx(A) == idx(B) + 1
A $++ B	C, A < C, B < C, idx(A) < idx(B)
A $-- B	C, A < C, B < C, idx(A) > idx(B)
A >++ B	A gov B, idx(A) < idx(B)
A >-- B	A gov B, idx(A) > idx(B)
A <++ B	A dep B, idx(A) < idx(B)
A <-- B	A dep B, idx(A) > idx(B)

Table 2: Relations between words

2.2 Relation Patterns

Adding relations between nodes allows for searching over graph structures, making *Semgrex* a powerful search tool over dependency graphs. The relations used can be from any dependency formalism, although CoreNLP and Stanza both use Universal Dependencies by default.

The simplest relations are parent and child:

```
{word:Dep} < {word:Gov}
{word:Gov} > {word:Dep}
```

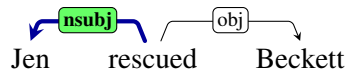
Many relations consider word order as well, such as the sister relations: + indicates the word on the left of the relation comes first, and − indicates the word on the right comes first:

```
{word:A} $+ {word:B}
{word:A} $- {word:B}
```

See Table 2 for a list of supported relations.

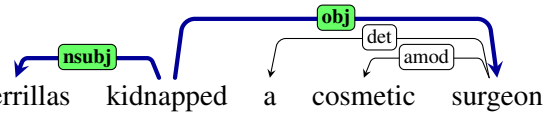
Relations can have labels, in which case the types on the edge between nodes must match:

```
{} <nsubj {}
```



A special token matches exactly at root, such as in this example from the UD conversion of EWT (Silveira et al., 2014):

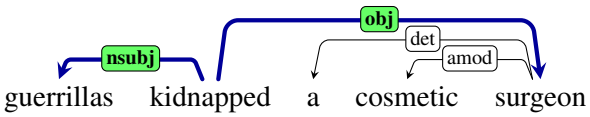
```
{$} > {}
```



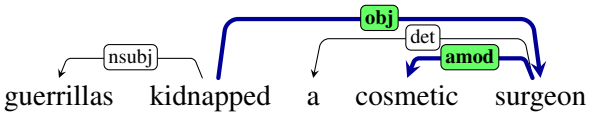
[2] https://stanfordnlp.github.io/CoreNLP/

Relations can be chained. Without parentheses, subsequent relations all apply to the same node; brackets denote that the later relations apply between the nodes in brackets as opposed to the head of the expression.

```
{} >nsubj {} >obj {}
```




```
{} >obj ({} >amod {})
```

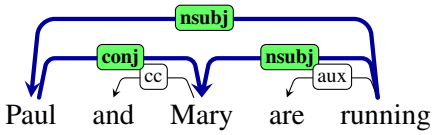


More advanced conjunction and disjunction operations are also possible. The JavaDoc[3] reference describes the complete syntax.

2.3 Named Nodes and Relations

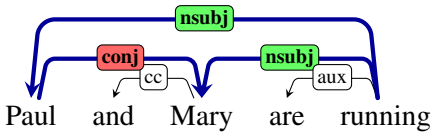
It is possible to name one or more nodes as part of a *Semgrex* pattern. This allows for relations between three or more nodes using backreferences. When a node is named in a backreference, it must be the exact same node as the first instance for the pattern to match.

```
{word:running}
  >nsubj ({} >conj {}=C)
  >nsubj {}=C
```



It is also possible to name the edges. This is useful when combined with *Ssurgeon*, which can manipulate an edge based on its name.

```
{word:running}
  >nsubj ({} >conj=conj {}=C)
  >nsubj {}=C
```



[3] https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/semgraph/semgrex/SemgrexPattern.html

2.4 Concrete Example

Here are a couple examples from a slot filling task using *Semgrex*. Both examples search for "son" or "daughter" in relation to possible family members. (Angeli et al., 2014)

This matches "John's daughter, Logan, . . . "

```
{lemma:/son|daughter|child/}
 >/nmod:poss/ {ner:PERSON}=ent
 >appos {ner:PERSON}=slot
```

This matches "Tommy, son of John, . . . "

```
{ner:PERSON}=slot
 >appos
   ({lemma:/son|daughter|child/}
    >/nmod:of/ {ner:PERSON}=ent)
```

2.5 Implementation Notes

Under the hood, the tool is built using JavaCC[4] to process input patterns.

The graphs are implemented as a collection of edges as relations, with nodes storing indices and the text information such as word, lemma, and POS. To represent edges to hidden copy nodes, nodes can be pointers to the same underlying data with a copy index on them. An example where this happens is *I went over the river and through the woods*, where the unstated *went* before *through the woods* is represented as a copy node.

Nodes are searched in topographical order if possible, and in index order if not, with the intention of making a canonical ordering on the search results.

3 Ssurgeon

Ssurgeon is an extension of *Semgrex* which includes rules to rewrite dependency graphs.

A pattern in *Ssurgeon* is a pattern for *Semgrex* with required named nodes and/or edges, depending on the edit type, along with a edit definition.

3.1 Edge Editing

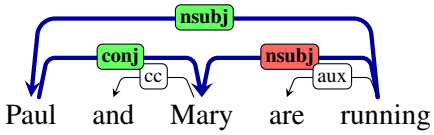
To add a new edge, the edit pattern must specify the governor, the dependent, and the edge type. The *Ssurgeon* pattern will add an edge to each match of the *Semgrex* pattern.

For example, in the previous "Paul and Mary are running" graph, the following would add the second $nsubj$ if it did not already exist. This would be useful for making enhanced dependencies, as basic UD has $conj(Paul, Mary)$ but not

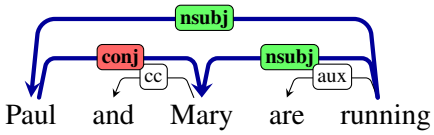
[4] https://javacc.github.io/javacc/documentation/grammar.html

$nsubj(running, Mary)$.) If the edge already exists, this rule does not add a duplicate edge.

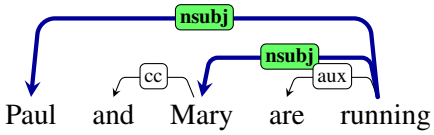
```
{word:running}=A
  >nsubj
  ({}=B >conj {}=C)

addEdge -gov A -dep C
        -reln nsubj
```



There are two mechanisms for deleting an edge. The first deletes an edge between two named nodes, and the second deletes a named edge. All edges which match the *Semgrex* pattern will be deleted.




```
{word:running}
  >nsubj {}=B
  >nsubj ({}=C !== {}=B)

removeEdge -gov B -dep C
           -reln conj
```



Alternatively, $removeNamedEdge$ removes a labeled edge:

```
{word:running}
  >nsubj {}=B
  >nsubj ({}=C >conj=conj {}=B)

removeNamedEdge -edge conj
```

There is also a mechanism for relabeling an edge, such as might be handy when mapping dependency graphs from one formalism to another:

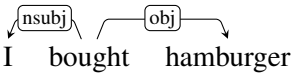
```
{word:running}
  >nsubj {}=B
  >nsubj ({}=C >conj=conj {}=B)

relabelNamedEdge
  -edge conj -reln dep
```

3.2 Node Editing

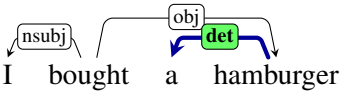
There are mechanisms to add a node, remove a subgraph starting from a node, and alter the content of a node.

To add a node, specify a position, a node to attach it to, and a relation. The position can be at the start or end of a sentence or relative to an existing node. For this modification, it is necessary to add a guard to the *Semgrex* expression, or it will enter an infinite loop of adding unlimited new nodes to the graph.

Note that in the following example, searching for a word with no *det* and adding a *det* to the node is already sufficient to prevent runaway nodes.




```
{word:bought}
  >dobj ({}=A !>det {})

addNode
  -word=a -reln det
  -gov A -position +A
```



4 Usage

Semgrex has been used for task-specific processing of academic work (Shah et al., 2018), news summarization (Li et al., 2016), text-to-scene generations (Chang et al., 2014), relation extraction (Chaganty et al., 2017), and processing medical documents (Profitlich and Sonntag, 2021).

Ssurgeon was used internally to simplify sentences using their dependencies as an extension to the textual entailment system in (Chambers et al., 2007).

The Java API and command line interfaces are part of the Java package CoreNLP (Manning et al., 2014)[5]. The python client is available via Stanza (Qi et al., 2020)[6] The code is actively maintained as of 2023, and suggestions for additional *Semgrex* relations, *Ssurgeon* operations, or other improvements are welcome at our github repo[7].

[5] https://stanfordnlp.github.io/CoreNLP/

[6] https://stanfordnlp.github.io/stanza/

[7] https://github.com/stanfordnlp/CoreNLP

4.1 Python Integration

Both *Semgrex* and *Ssurgeon* have Python APIs. This allows for operations on the results of Stanza (Qi et al., 2020) on natural language or using the API in Stanza to read and process existing UD datasets.

Here is an example of using *Semgrex* on the results of parsing an article on disease transmission with Stanza to find out what insect vectors transmit a disease. The search patterns here are abridged for readability.

```
EXPR = """
{word:/transmitted/} >/obl|advcl/
  ({word:/^(?!bite|biting|bites).*$/}=vector
   >/case|mark/ {word:/by|from|through/})
"""

def process_text(parser, text):
   doc = parser(text)
   results = semgrex.process_doc(doc, EXPR)
   facts = OrderedDict()
   for sentence_results, sentence in zip(results.result,
                                         doc.sentences):
      if sentence.text in facts:
         # already seen this exact sentence!
         # results will be exactly the same
         continue
      facts[sentence.text] = []
      for pattern_result in results.result:
         if len(pattern_result.match) == 0:
            continue
         for match in pattern_result.match:
            for named_node in match.node:
               new_fact = "  {}: {}".format(named_node.name,
                   sentence.tokens[named_node.index-1].text)
               if new_fact not in facts[sentence.text]:
                  facts[sentence.text].append(new_fact)
   return facts
```

Also included is a mechanism to display graphs as search results, thanks to an API call to displaCy (Honnibal et al., 2020).

5 Related Work

The analysis of dependency treebanks, especially Universal Dependencies, has a long history of using dependency searching and rewriting tools.

Constituency treebanks such as the Penn Treebank (Marcus et al., 1993) predate Universal Dependencies. To analyze such constituency datasets, Tgrep (Pito, 1993) and its successor Tgrep2 (Rohde, 2003) set the initial standard for searching tree structured data. Tregex and Tsurgeon (Levy and Andrew, 2006) extended the language and added functionality to rewrite constituency trees.

Semgrex was one of the earliest tools to address the problem of searching in dependency graphs. It was previous briefly described in a paper on entailment that was the first to use *Semgrex*, (Chambers et al., 2007), although that paper did not include *Ssurgeon* or fully explain the usage of *Semgrex*. *Semgrex* has been used by other research several times in the following years.

The authors of the UD_Italian-VIT (Alfieri and Tamburini, 2016) dataset used an extension of *Semgrex*, Semgrex-Plus (Tamburini, 2017), to convert the dependency form of VIT to Universal Dependencies. Semgrex-Plus adds edge creation, edge deletion, and word relabeling to a *Semgrex* result.

Also connected with UniversalDependencies are multiple search engines which allow for easier viewing of the treebanks. Tündra (Martens, 2013) allows for searching of a variety of treebanks using the TIGERSearch format (Lezius et al., 2002).[8] Additional treebanks not part of UD are included (Martens and Passarotti, 2014), although recent UD updates have not been incorporated.

Grew-Match (Guillaume, 2021)[9] hosts a website which allows for searching of existing UD and other dependency datasets. The interface is frequently updated, hosting the latest 2.11 treebanks as of January 2023.

(Heinecke, 2019) provides a web interface to backend parsers such as (Straka et al., 2016) and provides search, visual editing, and automatic pattern matching and replacement. However, it does not include a command line tool.

Other tools include UDeasy (Brigada Villa, 2022), which provides a graphical interface which allows for searching of UD treebanks or any other dependency formalism. spaCy reimplemented *Semgrex* as part of the 3.0 release, adding the DependencyMatcher tool (Honnibal et al., 2020).[10] UDapi (Popel et al., 2017) provides mechanisms for searching dependency graphs, parsing text, visualizing the graphs, and manipulating the graphs themselves. Odin is a rule-based event extraction framework over dependency structures (Valenzuela-Escarcega et al., 2016).

6 Conclusion

We have introduced *Semgrex* and *Ssurgeon*, flexible, simple systems for dependency matching and dependency tree manipulation.

7 Acknowledgements

We would like to thank the reviewers for their helpful feedback on this work.

[8] https://www.ims.uni-stuttgart.de/documents/ressourcen/werkzeuge/tigersearch/doc/html/QueryLanguage.html

[9] http://universal.grew.fr/

[10] https://spacy.io/api/dependencymatcher

References

Linda Alfieri and Fabio Tamburini. 2016. (Almost) automatic conversion of the Venice Italian Treebank into the merged Italian dependency treebank format. In *CLiC-it/EVALITA*.

Gabor Angeli, Sonal Gupta, Melvin Johnson, Christopher D. Manning, Julie Tibshirani, Jean Wu, and Sen Wu. 2014. Stanford's distantly supervised slot filling systems for KBP 2014.

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, Beijing, China. Association for Computational Linguistics.

Luca Brigada Villa. 2022. UDeasy: a tool for querying treebanks in conll-u format. In *Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-10)*, pages 16–19, Marseille, France. European Language Resources Association.

Arun Tejasvi Chaganty, Ashwin Paranjape, Jason Bolton, Matthew Lamm, Jinhao Lei, Abigail See, Kevin Clark, Yuhao Zhang, Peng Qi, and Christopher D. Manning. 2017. Stanford at TAC KBP 2017: Building a trilingual relational knowledge graph. In *Text Analysis Conference*.

Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine de Marneffe, Daniel Ramage, Eric Yeh, and Christopher D. Manning. 2007. Learning alignments and leveraging natural logic. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 165–170, Prague. Association for Computational Linguistics.

Angel X. Chang, Manolis Savva, and Christopher D. Manning. 2014. Learning spatial knowledge for text to 3D scene generation. In *EMNLP*.

Bruno Guillaume. 2021. Graph matching and graph rewriting: GREW tools for corpus exploration, maintenance and conversion. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 168–175, Online. Association for Computational Linguistics.

Johannes Heinecke. 2019. ConlluEditor: a fully graphical editor for universal dependencies treebank files. In *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest 2019)*, pages 87–93, Paris, France. Association for Computational Linguistics.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Roger Levy and Galen Andrew. 2006. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

Wolfgang Lezius, Hannes Biesinger, and Ciprian-Virgil Gerstenberger. 2002. Tigersearch manual.

Wei Li, Lei He, and Hai Zhuge. 2016. Abstractive news summarization based on event semantic link network. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 236–246, Osaka, Japan. The COLING 2016 Organizing Committee.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Scott Martens. 2013. Tündra: A web application for treebank search and visualization. In *Proceedings of The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12)*, pages 133–144, Sofia.

Scott Martens and Marco Passarotti. 2014. Thomas Aquinas in the TüNDRA: Integrating the index Thomisticus treebank into CLARIN-D. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 767–774, Reykjavik, Iceland. European Language Resources Association (ELRA).

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

Richard Pito. 1993. Tgrep user manual.

Martin Popel, Z. Žabokrtský, and Martin Vojtek. 2017. UDapi: Universal api for universal dependencies. In *UDW@NoDaLiDa*.

Hans-Jürgen Profitlich and Daniel Sonntag. 2021. A case study on pros and cons of regular expression detection and dependency parsing for negation extraction from german medical documents. technical report. *ArXiv*, abs/2105.09702.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Douglas Rohde. 2003. Tgrep2 user manual.

Sapan Shah, Dhwani Vora, B. P. Gautham, and Sreedhar Reddy. 2018. A relation aware search engine for materials science. *Integrating Materials and Manufacturing Innovation*, 7:1–11.

Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Chris Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 2897–2904, Reykjavik, Iceland. European Language Resources Association (ELRA).

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: Trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4290–4297, Portorož, Slovenia. European Language Resources Association (ELRA).

Fabio Tamburini. 2017. Semgrex-plus: a tool for automatic dependency-graph rewriting. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 248–254, Pisa,Italy. Linköping University Electronic Press.

Marco Antonio Valenzuela-Escarcega, Gus Hahn-Powell, and Mihai Surdeanu. 2016. Odin's runes: A rule language for information extraction. In *LREC*.