Quick Semantic Selections in Word Processors

A senior project submitted in partial fullfilment of the requirements
for the degree of Bachelor of Science
Computer Science
Yale College

Hristo Staykov

Spring 2021

Abstract

Text selection in word processors is ordinarily done using a pointing device or the keyboard arrow cluster, and can be precisely defined on a character basis. However, such precision is unnecessary if user thinks of the text as a string of words or sentences. It can impose a burden when the user is trying to select a complete sentence, but has to point to the exact start and end of the desired selection. This can be particularly troublesome on mobile devices with limited screen space.

This project explores how tools from the NLP community can aid the use in editing text by guiding their selections. There already exist shortcuts for selecting single words or the entirety of the text, but, to the author's knowledge, there are no tools which allow for quick selection of sentences, embeded sentences, and constituents in natural language.

This project implements a simple text editor that processes the user's text using NLP tools, and proposes interactions which can be used to easily select meaningful constituents of a text.

1 Introduction

Most text editing interface in use today derives from the Gypsy editor developed in the 1970s, where text is selected by dragging the mouse [**tesler2012personal**].

Modern operating systems offer additional shortcuts to aid with selection, such as double-clicking to select a word and triple-clicking to select a paragraph.

However, for any other type of selection, the user has to rely extensively on a pointing device or the keyboard arrow cluster. Consider the following passage, where the user selection is shown in grey:

> I rode my bike one last time though the maze of small streets and alleys, stopped beside the waffle shop, and flipped the kickstand.

Suppose we want to select the phrase the maze of small streets and alleys. This can be done in one of two ways:

- Using the keyboard, move the cursor to the beginning of the desired selection, and, using the ⇧+→ keys, select to the end.
- Using the mouse, move the cursor to the start of the selection, press the left button, and drag to the end of the selection, and release.

With either method, it is possible to miss or add an extra character, and create selections such as he maze of small streets and alleys, .

Observe that the current selection (streets) is already contained within the desired final selection. Moreover, the desired selection happens to form a *constituent* in the syntax tree of the sentence [1].

We propose several user interactions to exploit that structure, described in sec. 3.

1.1 Previous work

Modal code editors like `vim` and `kakoune` support selecting so-called *"text objects"*, such as a line of code, or characters enclosed between delimiters such as `(` and `)` [**vimobject**][2]. More recently, `tree-sitter`[**atom_treesitter**] has been used alongside such editors to parse the code being edited and provide more intelligent selections.

However, the approach taken in modal text editors does not translate well to editing natural language:

- Natural language is not as structured as source code, and cannot be parsed as easily in real time. Punctuation alone is not enough to delimit sentences.
- The keyboard shortcuts for creating such selections are not discoverable by the average user.

Parsing of natural language is can be realised through the use of *constituency parsers* and *dependency parsers.* Such parsers exist for various languages and can reach parsing speeds up to hundreds of sentences in a second (See sec. 2.2).

1.2 Constituency parses

2 Implementation

2.1 Overview

We implement a desktop application split into a *back end*, which handles parsing, and a *front end*, which is a simple text editor that leverages the parser and employs gestures and shortcuts to expose the text structure to the user.

[1] That is, it can easily replaced by another noun phrase, such as "campus", to form the sentence "I rode my bike one last time through campus, stopped beside. . . "

[2] Kakoune documentation on Github: https://github.com/mawww/kakoune/blob/master/doc/pages/keys.asciidoc#inner-object

2.2 Back end and choice of parser

We use an existing implementation of a constitency parser. We experimented with a few constituency parsers for this applciation.

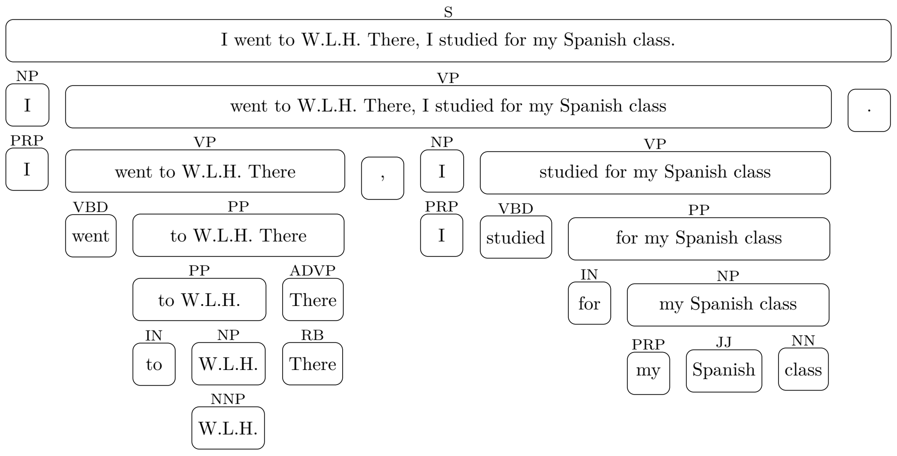
2.2.1 Stanford CoreNLP

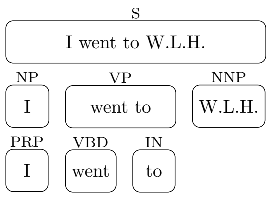
First, we looked into Stanford's CoreNLP package, which is a Java program that can run as a server locally on the user's computer. It encodes results in a protobuffer, and we were able to easily create an API for CoreNLP in Swift (See [**SwiftCoreNLP**]).

However, the PCFG Parser included in CoreNLP [**pcfg**] proved too slow for this application, and the author regrets not looking at the date the paper for that parser was published. While CoreNLP does support a newer, neural network parser [**stanfordShiftReduce**], we did not experiment further with it.

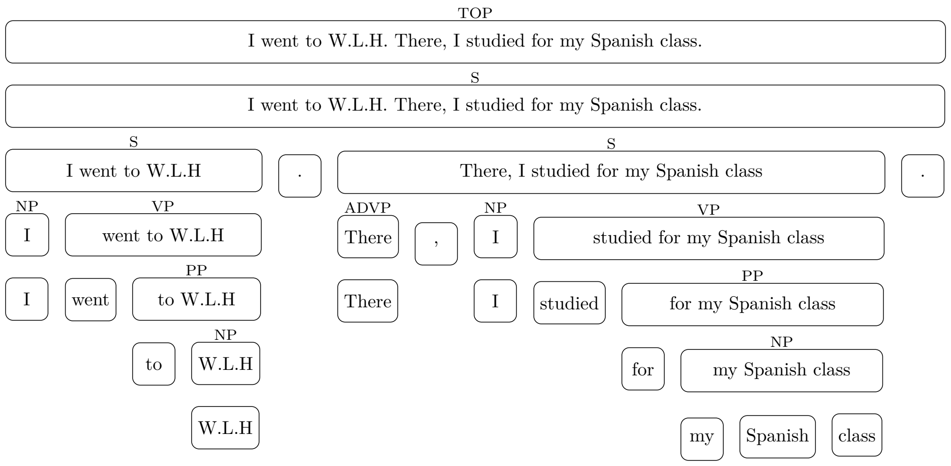
2.2.2 Berkeley Neural Parser

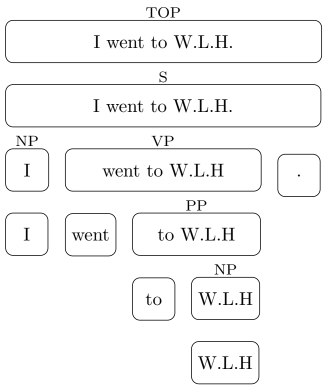
Berkeley's neural network parser [**berkeleyNeural**] uses neural networks and runs in near-linear time.





3 User interactions


TOP
I went to W.L.H. There, I studied for my Spanish class.
S
I went to W.L.H. There, I studied for my Spanish class.
S
I went to W.L.H
.
S
There, I studied for my Spanish class
.
NP
I
VP
went to W.L.H
ADVP
There
,
NP
I
VP
studied for my Spanish class
I
went
PP
to W.L.H
There
I
studied
PP
for my Spanish class
to
NP
W.L.H
for
NP
my Spanish class
W.L.H
my
Spanish
class



4 Syntax highlighting

5 Abstract

In word processors, the user manipulates the selection using the mouse or the keyboard. That way, they can precisely control the selection with character precision. However, the user might be more likely to think of the selection as a string of words or sentences, rather than a string of characters. The precision of the mouse can impose a burden when the user is trying to select a complete sentence, but has to point to the exact start and end of that sentence. This can be particularly troublesome on some mobile devices with limited screen space.

For my senior project, I would like to explore how tools from the NLP community can be used to augment the text editing experience. The output of a constituency parser can be used to quickly select the phrase or the sentence the user is editing. Consider the following sentence:

> I went to the White House in Washington and interviewed the President.

The parse tree of that sentence provides information on how to intelligently expand the selection: it is more likely that the user might want to select White House as opposed to the White. Then, using keyboard shortcut, the mouse wheel, or another interaction, the user can change the selection to White House, the White House in Washington, or went to the White House in Washington.

6 Previous work

Modal text editors like `vim` and `kakoune` support selecting so-called *"text objects"*, such as a line of code, or characters enclosed between delimiters such as `(` and `)` [3][4]. More recently, `tree-sitter`[5] has been used alongside such editors to parse the code being edited and provide more intelligent selections.

However, the approach taken in modal text editors does not translate to editing natural language:

- Natural language is not as structured as source code, and cannot be parsed as easily in real time.
- The keyboard shortcuts for creating such selections are not discoverable by the average user.

Parsing of natural language is can be realised through the use of *constituency parsers* and *dependency parsers*. Such parsers exist for various languages and can reach parsing speeds up to hundreds of sentences in a second [6].

7 Objectives and deliverables

In this project, I hope to:

- Use a constituency parser to parse text input in real-time. nlike `tree-sitter`, which can be run on long source files with reasonable speed, parsing natural language will involve caching and stitching segments of text to compensate for the latency of the NLP pipeline.
- Create a simple text editor that integrates with a constituency parser.
- Experiment with different user interactions that can be used to interact with the underlying semantic information.

[3]VIM REFERENCE MANUAL, cursor-motions navigation https://vimhelp.org/motion.txt.html#object-select

[4]Kakoune documentation on Github: https://github.com/mawww/kakoune/blob/master/doc/pages/keys.asciidoc#inner-object

[5]https://tree-sitter.github.io/tree-sitter/

[6]Danqi Chen and Christopher D Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. Proceedings of EMNLP 2014

A mock-up of what selecting text using the constituency tree can be found at https://hristost.github.io/semantic-selections-js-demo/. This demo uses a hard-coded text and parse tree, and navigation is done using the keyboard.

8 Milestones

8.1 February

1. **Research**:

- Research open-source constituency parsers and decide on the processing pipeline.
- Decide on the form of the proposed text editor: will it be a web application? A desktop application? When deciding, it should be considered how the final product can be easily distributed to users for testing, noting that we also have to package the NLP model.
- Decide on a programming language and GUI framework to be used throughout the project.

8.2 March

2. **Parsing**:

- Embed the processing pipeline into a project, or implement an interface to a NLP server such as Stanford's CoreNLP. Process the output of the pipeline to a tree of constituents that includes the range of each constituent in the text.

3. **Text editing**:

- Integrate a simple text field with the parser, and run the parser on every change.
- Allow the user to select constituents as shown in the mockup.

4. **Caching**:

- Implement a caching mechanism that will allow for parsing only segments of the text that have changed. That will allow for editing longer texts without a percievable delay.
- Design an algorithm to predict what chunks of the text can be parsed independently to produce the same tree as when parsed in context.

8.3 April

5. **User interface**:

- Experiment with different interactions to let the user select text: keyboard shortcuts, mouse, trackpad, etc.
- Visualise which neighbouring constituents can be selected.

- Package the text editor in a distributable application.

6. **Evaluation**:

- Design a small study to answer the following questions:
  - Is the use of constituents for selections intuitive to the user?
  - Can the user easily learn the new interactions?
  - Would the test group prefer to use such interactions in their everyday life?
  - Is the constutency parse predictable to the user?

I expect to be writing the final report concurrently with the described milestones.