EasyChair Preprint

№ 883

Speeding up Natural Language Parsing by Reusing Partial Results

Michalina Strzyz and Carlos Gómez-Rodríguez

April 6, 2019

Speeding up Natural Language Parsing by Reusing Partial Results

Michalina Strzyz Carlos Gómez-Rodríguez

Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Research Group, Departamento de Computación
Campus de Elviña, s/n, 15071 A Coruña, Spain
{michalina.strzyz,carlos.gomez}@udc.es

Abstract. This paper proposes a novel technique that applies case-based reasoning in order to generate templates for reusable parse tree fragments, based on PoS tags of bigrams and trigrams that demonstrate low variability in their syntactic analyses from prior data. The aim of this approach is to improve the speed of dependency parsers by avoiding redundant calculations. This can be resolved by applying the predefined templates that capture results of previous syntactic analyses and directly assigning the stored structure to a new n-gram that matches one of the templates, instead of parsing a similar text fragment again. The study shows that using a heuristic approach to select and reuse the partial results increases parsing speed by reducing the input length to be processed by a parser. The increase in parsing speed comes at some expense of accuracy. Experiments on English show promising results: the input dimension can be reduced by more than 20% at the cost of less than 3 points of Unlabeled Attachment Score.

1 Introduction

Current state-of-art parsing algorithms are facing high computational costs, which can be an obstacle when applied to large-scale processing. For example, the BIST parser [1] reports speeds of around 50 sentences per second in modern CPUs, and even the fastest existing parsers, which forgo recurrent neural networks, are limited to a few hundred sentences per second [2]. While these speeds can be acceptable for small-scale processing, they are clearly prohibitive when the intention is to apply natural language parsing at the web scale. Thus, there is a need for approaches that can parse faster, even if this comes at some accuracy cost, as differences in accuracy above a certain threshold have been shown to be unimportant for some downstream tasks [3].

In this paper we propose a novel approach to improve the speed of existing parsers by avoiding redundant calculations. In particular, we identify fragments of the input for which a syntactic parse is known with high confidence from prior data, so that we can reuse the existing result directly instead of parsing the fragment again. This effectively reduces the length of the input being processed by the parser, which is a major factor determining parsing time.

We test a prototype of the approach where the reusable fragments are bigrams and trigrams, the matching criteria are based on part-of-speech tags, and the parsers to be optimized are linear-time transition-based dependency parsers. Note, however, that the technique is generic enough to be applied to any kind of parsing algorithm (including constituent parsers) and speed improvements are expected to be higher for higher-complexity parsers such as those based on dynamic programming, which have been the traditional target of pruning and optimization techniques [4, 5].

The aim of this approach is to significantly improve the parsing speed while keeping an acceptable accuracy level. This involves a trade-off between speed and accuracy, depending on how aggressively the technique is applied: more lenient confidence criteria for reusing fragments will lead to larger reductions in input length and thus faster parsing but at a cost to accuracy. We expect that the accuracy cost can be reduced in the future by using more fine-grained matching criteria (based on forms or lemmas) and augmenting training data so more fragments can be confidently reused.

2 Reuse of Partial Results

A natural language obeys the so-called Zipf's law, which shows how words are distributed with respect to their frequency. Namely, a language has a few high-frequency tokens and many uncommon tokens. This phenomenon is also present when investigating lemmas [6]. Similarly, n-gram phrases fit this distribution [7, 8]. Therefore, to some extent, repetitions of identical n-grams are likely to be found in large corpora.

This implies that a parser will probably encounter known n-grams in a text. Since most of the time repetitions of the same n-gram will have identical syntactic structure (e.g. the phrase "the 10 provinces of Canada" can be reasonably expected to always have the same internal structure), we can exploit this by reusing previous analyses for these known n-grams. More generally, this can be extended to similar n-grams ("the 50 provinces of Spain" can be expected to have the same analysis as the phrase above), which can be identified with templates: the two examples above can be captured by a template "the CD provinces of NNP" (where CD and NNP are the part-of-speech tags for numerals and proper nouns), associated with a partial syntactic analysis to be reused. This combination of template matching and result reuse can be seen as an instantiation of case-based reasoning, a cognitively-rooted problem-solving technique based on solving new problems by reusing solutions to similar past problems [9, 10].

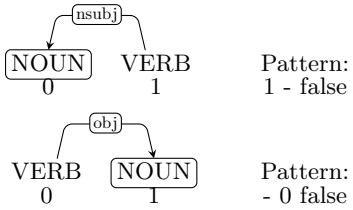
Given a template like in the example above, we implement case-based reasoning by examining the training set and counting the number of n-grams that match it, as well as the different syntactic analyses that they have been assigned. If the variability of these analyses is higher than a given threshold, then the template will not be useful. However, if matching n-grams exhibit the same syntactic structure the vast majority of the time, then we can create a rule assigning that

parse tree fragment to the template. At test time, we will reuse the parse tree fragment whenever we encounter an n-gram matching this rule.

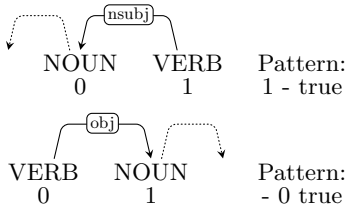
3 Generating the Templates

In this prototype we will use limited training data. Since the frequency of word pairs is higher than multi-word phrases [11] only templates consisting of bigrams or trigrams of PoS tags are considered in this implementation in order to not suffer excessively from sparsity problems. More detailed templates (such as those including lemmas or word forms) would require augmenting the training set.

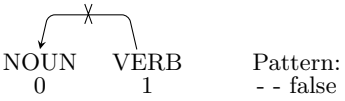
In order to calculate the level of confidence of a syntactic analysis for a given template, the following patterns for a bigram of PoS tags, as detailed in Figure 1, are taken into consideration.



(a) bigram with dependents with no descendants outside the scope



(b) bigram with a dependent pointing to a word outside the scope



(c) bigram with missing relation

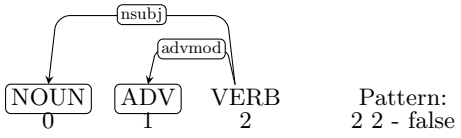
Fig. 1: Possible patterns for a bigram of PoS tags NOUN VERB and VERB NOUN used for calculating the confidence of a template. Each pattern denotes the index of each word's head ("-" in case of headless word) and whether a dependent is pointing to a word outside the scope of the bigram ("true") or not ("false").

To generate a template for a given PoS tag bigram, we count the frequency of each of these head patterns relative to the total frequency of the bigram in

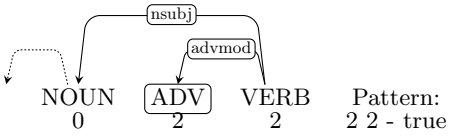
the training set. Then, we focus on the most frequent pattern for the bigram. If this pattern has a dependent pointing to a word outside the bigram (Figure 1b) or multiple unconnected roots (Figure 1c), then the bigram will be discarded for template generation. The reason is that our reuse approach is based on replacing a sentence fragment (for which the parse is extracted from a rule) with its head word. The parser will operate on this head word only, and the parsed fragment will then be linked with the resulting tree. To do this, reusable fragments must have a single head and no external material depending on their dependents.

If the dominant pattern is eligible according to these criteria, then we will consider it if its relative frequency (confidence) is above a given threshold (head threshold) and, since our parsing is labeled, if the most frequent dependency label involved in the pattern has, in turn, a relative frequency above a second threshold (label threshold).

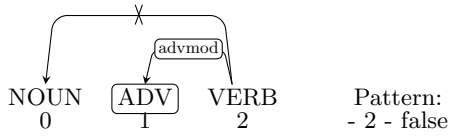
The confidence of heads and labels for a trigram is calculated analogously. The number of possible patterns of a trigram increases to a total of 19. As can be seen in Figure 2, a trigram can be a fully connected tree where the dependents have no descendants outside the scope (7 patterns) or where at least one of the dependents points to a word outside the scope (7 patterns) [1]. A trigram can also be not fully connected or not connected at all (5 patterns).



(a) trigram with dependents with no descendants outside the scope



(b) trigram with a dependent pointing to a word outside the scope



(c) trigram with missing relation with one of its components

Fig. 2: Some of the possible patterns for a trigram of PoS tags NOUN ADV VERB used for calculating the confidence of a template. Each pattern denotes the index of word's head ("-" in case of headless word) and whether a dependent is pointing to a word outside the scope of the trigram ("true") or not ("false").

[1] In this implementation we only consider projective trees for trigrams.

Similarly to the case of bigrams, only patterns for trigrams highlighted in Figure 2a that exceed predefined thresholds for confidence will be used as templates. If the input fragment matches a template, all dependents will be removed from the input to be processed by a parser. The patterns from Figure 2b and Figure 2c are discarded from being candidates for a template.

4 Experiments

4.1 Data and Evaluation

We conducted our experiments on the English treebank from Universal Dependencies (UD) v2.1 [12] and the results are evaluated with Unlabeled and Labeled Attachment Scores (UAS and LAS). The speeds are measured in tokens/second on CPU [2].

4.2 Model

During training time our model computes a set of rules that surpasses two thresholds: one for the dominant heads and the second for the dominant labels for a given n-gram. In our experiments the thresholds were set manually. Each template contains information about the n-gram's most likely head(s) and label(s) in order to automatically assign a syntactic analysis to the input that matches that template at parsing time.

As an example, Table 1 illustrates some templates that surpass the thresholds in the training set. The thresholds were set to 83% for confidence of head and 83% for confidence of label. This generated 141 unique templates, of which 97 had confidence of 100% for both thresholds but with low frequency.

Table 1: Example of templates generated during training time that surpass the predefined thresholds: 83% for confidence of the dominant head pattern and 83% for confidence of the dominant label pattern for a given n-gram of PoS tags.

Template	Dominant head pattern	Confidence of head pattern (%)	Dominant label pattern	Confidence of label pattern (%)
DET NOUN	1 - false	86.50	det - false	86.49
SCONJ PROPN VERB	2 2 - false	100	mark nsubj - false	100
AUX ADJ	1 - false	93.71	cop - false	93.64
ADV VERB	1 - false	93.17	advmod - false	91.31

[2] Intel Core i7-7700 CPU 4.2 GHz

The rules are applied to matching bigrams and trigrams on the training set, removing words other than the head.[3] This produces a reduced training set that no longer contains n-grams matching any of the templates. Our parser is then trained on this reduced training set. At parsing time templates are applied to the input, which is reduced in the same way, the parser is then ran on this shorter input and finally the parse tree fragments are attached to the resulting output. We verified experimentally that, as expected, a parser achieves better accuracy combined with our technique when trained on the reduced than on the entire training set.

4.3 Results

In our experiments we run the following dependency parsers on the remaining test set: transition-based BIST Parser [1] that uses bidirectional long short memory (BiLSTM) networks, as well as MaltParser [13] with the arc-eager and Covington transition systems.

Table 2 demonstrates the results after applying templates with threshold variation on the development set. In the experimental setup we use templates consisting of bigrams alone, trigrams alone or combined in order to compare their performance. While the specific parameters to choose depend on the speed-accuracy balance one wants to achieve, we selected the models where the thresholds for the dominant head and label pattern are 83-83 and 87-87 as our "optimal models", considering that they provide a reasonable trade-off between speed and accuracy. Thus, only these optimal thresholds were used afterwards when testing the technique on the development and test sets in order to investigate more in detail and to compare the accuracy and parsing speed. We use the notation $M_{\mathrm{z}}^{\mathrm{x,y}}$ where x indicates whether bigrams (2) were used, y trigrams (3) and z level of confidence for head and label pattern respectively.

Table 3 compares the performance of the parsers with and without applying our technique. The results are revealing in several ways. The experiments confirm that the more lenient confidence thresholds result in larger reductions of input length and thus faster parsing, but at the expense of accuracy. This applies to all parsers and indicates that our approach can be generic and applicable to diverse parsing algorithms. Moreover, the results show that it is feasible to reduce input size by more than 20% at the cost of less than 3 points of unlabeled attachment score.

5 Ongoing Work

One of the main weaknesses of the approach presented above is that the final templates generated during training are only based on adjacent words (i.e., words whose position indexes differ by 1). It does not take into account longer

[3] In case a bigram and trigram overlap, the n-gram with higher head confidence will be chosen and its dependents will be removed.

Table 2: Performance of MaltParser with arc-eager transition system and the % of the text reduction after applying templates with different thresholds, on the development set. The total number of words in the development set: 25150 and test set: 25097.

Setup	UAS (%)	LAS (%)	Word Reduction (%)
$M^{2,3}_{90-90}$	84.73	82.15	5.0
$M^{2,3}_{87-87}$	**84.38**	**81.64**	**8.3**
M^{3}_{85-85}	84.15	81.45	8.5
M^{2}_{85-85}	84.01	81.17	11.1
$M^{2,3}_{85-85}$	83.1	80.15	18.2
$M^{2,3}_{83-83}$	**82.95**	**80.03**	**20.7**
$M^{2,3}_{80-80}$	81.51	78.42	22.7
$M^{2,3}_{80-70}$	81.2	77.71	24.2

dependency arcs that could be captured by a bigram or trigram after removing the intervening dependents with shorter arcs. This approach can be improved by iteratively finding new templates and recalculating their confidence based on the outcome of applying the preceding template and removing the dependents it captured. In this way, an n-gram would capture a longer arc after intervening words have been removed. In this new approach, the order of applying templates generated during training time is crucial, instead of treating templates as a *bag-of-rules* that match an n-gram from the input.

We performed a preliminary experiment where new templates are generated based on the outcome of applying preceding templates and removing dependents. We believe it can be beneficial to localize noun phrases first in the input sentence, because they cover vast part of sentences. We look at the distance between PoS tags in an n-gram where the head is a noun. Some PoS tags show tendency to appear closer to a noun than others. We give priority to templates that capture PoS tags closest connected to a noun. In subsequent steps, we generate templates that show the highest confidence at each iteration, and add them to a list that is applied in order. This technique applied on MaltParser with the arc-eager transition system reduces the dev set by almost 21% with UAS of 82.14 and a LAS of 79.20.

6 Conclusion

We have obtained promising results where the input length can be reduced by more than 20% at the cost of less than 3 points of UAS. We believe that our work can be a starting point for developing templates that in the future can

Table 3: Performance of BIST Parser and MaltParser with the arc-eager and Covington transition systems and after applying templates compared with the baseline. The reported parsing speed (tokens/sec) only refers to the runtime of the dependency parser on the entire data set (baseline) or remaining text (that was passed to the parser after extracting the fragments captured by the templates) excluding the time needed to run the technique which is already negligible and will be optimized in the future versions.

Parser	Data Set	Setup	UAS (%)	LAS (%)	Word Reduction (%)	Tokens/Sec	Speed-up Factor
BIST Parser	dev	**baseline**	**88.07**	**86.08**	**NA**	1818 ± 51	NA
		$M^{2,3}_{87-87}$	86.94	84.49	8.3	2006 ± 20	1.10x
		$M^{2,3}_{83-83}$	85.20	82.34	20.7	2328 ± 34	1.28x
	test	**baseline**	**87.64**	**85.65**	**NA**	1860 ± 36	NA
		$M^{2,3}_{87-87}$	86.11	83.63	8.7	2006 ± 83	1.08x
		$M^{2,3}_{83-83}$	84.79	82.21	20.7	2291 ± 39	1.23x
MaltParser arc-eager	dev	**baseline**	**85.07**	**82.65**	**NA**	17387 ± 711	NA
		$M^{2,3}_{87-87}$	84.38	81.64	8.3	18118 ± 860	1.04x
		$M^{2,3}_{83-83}$	82.95	80.03	20.7	19758 ± 588	1.14x
	test	**baseline**	**84.58**	**82.00**	**NA**	17748 ± 801	NA
		$M^{2,3}_{87-87}$	83.32	80.44	8.7	18626 ± 459	1.05x
		$M^{2,3}_{83-83}$	82.02	79.13	20.7	19286 ± 889	1.09x
MaltParser Covington	dev	**baseline**	**83.99**	**81.65**	**NA**	16121 ± 581	NA
		$M^{2,3}_{87-87}$	82.84	80.24	8.3	16500 ± 819	1.02x
		$M^{2,3}_{83-83}$	81.69	78.92	20.7	18210 ± 484	1.13x
	test	**baseline**	**83.68**	**81.34**	**NA**	16009 ± 1035	NA
		$M^{2,3}_{87-87}$	82.75	80.02	8.7	16561 ± 629	1.03x
		$M^{2,3}_{83-83}$	81.36	78.6	20.7	17395 ± 1407	1.09x

significantly speed up parsing time by avoiding redundant syntactic analyses at the minimal expense of accuracy.

The present study has investigated two approaches. In the first technique, which is the main focus of the paper (Section 4.2) templates were treated as a *bag-of-rules* that have to exceed predefined thresholds for the dominant head and label pattern for a given PoS tag n-gram, prioritizing ones with the highest confidence. In the second approach (Section 5) more importance is given to the order in which templates should be applied. However, the second technique is still in a preliminary stage, and requires some refinement. Research into solving this problem is already in progress. To further our research we plan to use both PoS tags and lemmas in our templates. The sparsity problem in finding n-grams involving lemmas will be tackled by augmenting training data with parsed sentences.

While we have tested our approach on transition-based dependency parsers, it is worth noting that the technique is generic enough to be applied to practically any kind of parser. Since fragment reuse is implemented as pre and postprocessing step, it works regardless of the inner working of the parser. As the technique reduces the input length received by the parser, speed gains can be expected to be larger on parsers with higher polynomial complexity, like those based on dynamic programming. The same idea would also be applicable to other grammatical representations, for example in constituent parsing, by changing the reusable fragments to the relevant representation (e.g. subtrees of a constituent tree).

Further studies will need to be undertaken in order to show the results of the approach when applied on other kinds of parsers, and on other languages different from English.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the TELEPARES-UDC project (FFI2014-51978-C2-2-R) and the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia (ED431B 2017/01). We gratefully acknowledge NVIDIA Corporation for the donation of a GTX Titan X GPU.

References

1. Kiperwasser, E., Goldberg, Y.: Simple and accurate dependency parsing using bidirectional LSTM feature representations. TACL **4** (2016) 313–327
2. Straka, M., Straková, J.: Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, Association for Computational Linguistics (2017) 88–99
3. Gómez-Rodríguez, C., Alonso-Alonso, I., Vilares, D.: How important is syntactic parsing accuracy? an empirical evaluation on rule-based sentiment analysis. Artificial Intelligence Review (2017) 1–17
4. Bodenstab, N., Dunlop, A., Hall, K., Roark, B.: Beam-width prediction for efficient context-free parsing. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1. HLT '11, Stroudsburg, PA, USA, Association for Computational Linguistics (2011) 440–449
5. Vieira, T., Eisner, J.: Learning to prune: Exploring the frontier of fast and accurate parsing. Transactions of the Association for Computational Linguistics **5** (2017) 263–278
6. Baroni, M.: Distributions in text. In: Corpus Linguistics: An international handbook-Volume 2, Mouton de Gruyter (2009) 803–821
7. Ha, L.Q., Sicilia-Garcia, E.I., Ming, J., Smith, F.J.: Extension of Zipf's law to words and phrases. In: Proceedings of the 19th international conference on Computational linguistics-Volume 1, Association for Computational Linguistics (2002) 1–6
8. Ha, L.Q., Hanna, P., Ming, J., Smith, F.: Extending Zipfs law to n-grams for large corpora. Artificial Intelligence Review **32** (2009) 101–113
9. Richter, M.M., Aamodt, A.: Case-based reasoning foundations. The Knowledge Engineering Review **20** (2005) 203207
10. Hüllermeier, E.: Case-Based Approximate Reasoning. Volume 44 of Theory and Decision Library. Springer (2007)
11. Smith, F., Devine, K.: Storing and retrieving word phrases. Information Processing & Management **21** (1985) 215–224
12. Nivre, J., et al.: Universal dependencies 2.1 (2017) LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics ('UFAL), Faculty of Mathematics and Physics, Charles University.
13. Nivre, J., Hall, J., Nilsson, J.: Maltparser: A data-driven parser-generator for dependency parsing. In: Proceedings of LREC. Volume 6. (2006) 2216–2219