Optimizing Dependency Parsing Throughput

Albert Weichselbraun and Norman Süsstrunk
Department of Information, University of Applied Sciences Chur, Pulvermühlestrasse 57, Chur, Switzerland

Keywords: Natural Language Processing, Dependency Parsing, Performance Optimization, Throughput.

Abstract: Dependency parsing is considered a key technology for improving information extraction tasks. Research indicates that dependency parsers spend more than 95% of their total runtime on feature computations. Based on this insight, this paper investigates the potential of improving parsing throughput by designing feature representations which are optimized for combining single features to more complex feature templates and by optimizing parser constraints. Applying these techniques to *MDParser* increased its throughput four fold, yielding *Syntactic Parser*, a dependency parser that outperforms comparable approaches by factor 25 to 400.

1 INTRODUCTION

Dependency parsing is considered a key technology for improving natural language processing. Although information extraction tasks such as named entity linking, named entity recognition, opinion mining and coreference resolution would benefit from dependency parsing, throughput has been one of the mayor obstacles towards its deployment for big data and Web intelligence applications.

Therefore, researchers have put more attention to parsing throughput in recent years which led to significant speed improvements. The latest version of the Stanford parser (Chen and Manning, 2014), for instance, is able to process on average more than 900 sentences per second, a number which is even exceeded by MDParser (Volokh, 2013) with a throughput of approximately 5200 sentences per second.

In contrast to related work which primarily focuses on the parsing and feature selection strategy, this paper puts its emphasis on feature extraction and the optimization of parsing constraints. The suggested methods, therefore, complement related work, i.e. they can easily be applied on top of them.

A major motivator for applying dependency parsing is its potential to enable a more sophisticated text processing for information extraction tasks such as opinion mining. Qiu et al., for example, define syntactic rules that draw upon dependency trees to extract opinion targets from text documents. Afterwards, their approach propagates the value from opinion-bearing words to their targets based on a set of predefined dependency rules, connecting the targets with further terms within the sentence (Qiu et al., 2011). Targets can transfer their sentiment to other terms which can either be new sentiment terms or targets. While Qiu et al. focused on identifying unknown sentiment terms, Gindl et al. apply this approach to the extraction of sentiment aspects and targets (Gindl et al., 2013). Poria et al. use dependency parsing for decomposing text into concepts based on the dependency relations between clauses. Their approach is domain-independent and extracts concepts with an accuracy of 92%, obtains a precision of 85% for sentiment analysis, and a precision of 63% for emotion recognition (anger, sadness, disgust, fear, surprise and joy), therefore, outperforming approaches which solely relied on N-grams or part-of-speech tagging (Poria et al., 2014).

These improvements and the potential to facilitate more advance text understanding for information extraction and text mining tasks in big data environments have been major drivers for developing and evaluating the methods discussed in this paper.

The rest of this paper is organized as follows: Section 2 presents an overview of related work. Section 3 then provides an analysis of the dependency parsing process, addresses areas for improving the parser's throughput, and performs isolated evaluations of how these improvements impact the parsing speed. We then present a detailed benchmark and evaluation of our approach in Section 4 as well as a discussion of the obtained results. The paper closes with an outlook and conclusion in Section 5.

2 RELATED WORK

Most current dependency parsers are either graph-based or transition-based (Nivre and McDonald, 2008). Graph-based dependency parsers learn a model for scoring dependency graphs from training sentences and choose the highest scoring dependency graph for unknown sentences. This strategy requires the parser to create and score a large number of dependency graphs which is a time consuming task. Transition-based parsers, in contrast, only optimize the transitions between parser states, i.e. the decision on whether to create a dependency with the label l between two nodes n_i and n_j. They derive the dependency graph by performing such transitions for all words in the sentence using a parsing strategy which determines the set of possible transitions. Popular parsing strategies are based on the transition system by Nivre for projective and non-projective parsing (Nivre, 2009), or on the Covington algorithm (Covington, 2001). Parsers which are based on Nivre's transition system have a parsing complexity of $O(n)$ for projective, and $O(n^2)$ for non-projective parsing (Nivre, 2008). Since the fraction of non-projective dependencies is usually very low even non-projective dependency parsers that use Nivre's transition system have a close to linear time complexity (Nivre, 2009). The Covington algorithm, in contrast, has a time complexity of $O(n^2)$ in the worst case (Volokh and Neumann, 2012).

Clearparser, which is now part of the ClearNLP project (https://github.com/clearnlp), extends Nivre's algorithm (Nivre and McDonald, 2008) with a transition which determines whether the parser continues with projective or non-projective parsing, cutting the average parsing speed by 20% compared to the original implementation (Choi and Palmer, 2011).

Although the importance of the parsing strategy for parser throughput is well researched and undisputed, feature extraction seems to play an even more important role in optimizing dependency parsing since it has not received much attention yet. Recent research by He et al. suggests that many state of the art dependency parsers spend most of their time in the feature extraction step (He et al., 2013). Chen and Manning observed that more than 95% of the parser's runtime is consumed by feature extraction, a number which is even surpassed by the baseline parser used by Bohnet (Chen and Manning, 2014; Bohnet, 2010). The baseline evaluated in his paper uses the architecture of McDonald and Pereira and spends 99% of its total runtime on feature extraction (McDonald and Pereira, 2006).

He et al. address this problem by improving feature selection. Their parsing framework dynamically selects features for each edge, achieving average accuracies comparable to parsers which use the full feature set with less then 30% of the feature templates (He et al., 2013).

Chen and Manning, in contrast, use dense features which embed high dimensional sparse indicator features into a low-dimensional model (Chen and Manning, 2014). This approach does not only reduce feature extraction cost but also addresses common problems of traditional feature templates such as sparsity, and incompleteness (i.e. the selected feature templates do not cover every useful feature combination) (Chen and Manning, 2014).

3 METHOD

3.1 Feature Design

Syntactic Parser mitigates the problem of feature sparseness by performing a frequency analysis of lexical word forms, replacing words with a very low frequency with the label `unknown`. Optimizing the feature design by replacing infrequent word forms with the `unknown` label did not only improve the parser's throughput and memory consumption but also positively affected the parsing accuracy since it enabled the classifier to learn strategies for handling infrequent features.

We also systematically scanned for feature templates which do not significantly improve parsing performance and identified four combined features used by MDParser as candidates for removal from the parsing process.

3.2 Feature Representation

Parsers tend to spend a significant amount of time in the feature extraction and generation process. MDParser generates new features by combining elementary string features such as part-of-speech tags, word forms and dependencies with feature identifiers that indicate the feature type (e.g. `pi`, `pip1`, `pj`, etc.). Depending on the feature type between two (single feature) and four (ternary feature) string concatenations are required to generate the corresponding feature. Volokh and Neumann (Volokh and Neumann, 2012) analyzed this problem and came to the conclusion that computing these combination requires a significant amount of the total parsing runtime. They also considered transforming string features into binary ones but suspected that the transformation might be too time intensive to provide an overall benefit (Volokh and Neumann, 2012).

Based on this analysis, an efficient feature representation would not only need to be more efficient for combining features but is also required to minimize the number of transformations.

MDParser transforms String features to integer values since the used liblinear classifier operates on numerical values. This step requires a total of 27 transformations for every word, since MDParser computes 27 different features templates. Syntactic Parser, in contrast, transforms elementary features to integer values before the features are combined, requiring only three transformations (part-of-speech, word form and dependency) for every word in a sentence.

Syntactic Parser represents its numerical features n as 64-bit integers (Java long type) which consist of (i) an 8 bit value indicating the feature type, and (ii) one or more of the following elementary features: word forms (20 bit), part-of-speech tag (16 bit) or dependency label (16 bit). The feature type is always stored in the last eight bits, while the position of elementary features depends on the feature type.

We, therefore, replace the mapping $\mathcal{M} : \mathcal{F} \to \mathbb{N}$ of parser feature strings $f_i \in \mathcal{F}$ to integer values $n \in \mathbb{N}$ with three mappings that translate word forms $w_i \in \mathcal{W}$ ($\mathcal{M}_{wordform} : \mathcal{W} \to \{0, ..., 1048575\}$), part-of-speech tags $pos_i \in \mathcal{P}$ ($\mathcal{M}_{pos} : \mathcal{P} \to \{0, ..., 65535\}$) and dependency labels $d_i \in \mathcal{D}$ ($\mathcal{M}_{dep} : \mathcal{D} \to \{0, ...65535\}$) to integer values. The numerical feature value n is then derived by performing bit operations on these elementary features.

Figure 1 illustrate the binary encoding for the features wf_i, m_3, and m_6. Combining and encoding features is very fast, since it only involves bit operations on 64 bit values which can be performed directly in the CPU's registers. We use hash maps which have a time complexity of $O(1)$ to translate between the string features extracted from the input text and their internal numerical representation. Another potential optimization is replacing these maps with hash values obtained by calls to String's `hashCode()` method or to high throughput non-cryptographic hash functions such as Murmur3 [1]. Our experiments showed that Java's hash maps are very effective and no significant speed improvement could be obtained by replacing hash maps with the hash functions mentioned above.

[1]https://code.google.com/p/smhasher/

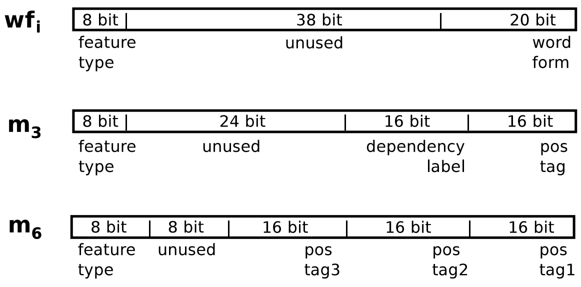

Figure 1: Binary feature encoding for single (wf_i), binary (m_3) and ternary features (m_6).

3.3 Constraint Checking

The dependency parser needs to ensure that the created dependency tree does not contain single heads, reflexive nodes, improper roots, or cycles. MDParser and Syntactic Parser perform projective parsing and, therefore, also eliminate non-projective dependencies. The parser tests for every combination of two nodes, whether one of the above conditions is met, before it even considers a dependency between these nodes. This strategy has considerable speed and accuracy benefits, since these tests are less expensive than querying the machine learning component for the existence of a dependency.

Considering that (i) the test for cycles and projectiveness traverse the dependency tree[2] and are, therefore, much more expensive than tests for single heads, reflexive nodes and improper roots, and that (ii) tests are called $\frac{n \cdot (n-1)}{2}$ times for a sentence with n words, optimizing the order of these tests has the potential to considerably improve the performance of the dependency parsing algorithm. This is also reflected in the performance improvement obtained by changing the original test sequence (single heads, reflexive nodes, cycles, improper roots, projectiveness) to an optimized one (single heads, reflexive nodes, improper roots, cycles, projectiveness). We also consider that projectiveness is a symmetric property and, therefore, compute it only once rather than twice (for the potential dependency $n_i \to n_j$ and for the dependency $n_j \to n_i$).

The results summarized in Table 1 show a performance boost of factor three for numerical features and factor four for combining numerical features with optimized constraint checking over the original implementation. The computations required for optimizing

[2]Nivre introduces an algorithm that does not require traversing the dependency tree but rather performs these checks in almost linear time (proportional to the inverse of the Ackermann function) by using suitable data structures to keep track of the connected components of each node (Nivre, 2008).

the feature representation have been performed on an Intel Core i3-2310M CPU at 2.10 GHz using a single thread and the OpenJDK virtual machine in version 1.7.0_75 on the Ubuntu 14.04 operating system. The presented results have been computed as the average of three runs which have been performed after parsing of 50,000 sentences to warm up the virtual machine. In all experiments the standard deviation of the runtime between the experiments has been below 1.5%.

Table 1: Comparison of the feature encoding speed for string and numerical feature representations.

feature representation	**number of sentences**	**processing time (ms)**
string	1,000	16
	10,000	161
	100,000	1,577
	1,000,000	16,827
numerical	1,000	6
	10,000	55
	100,000	532
	1,000,000	5,414
numerical &	1,000	4
optimized	10,000	41
constraints	100,000	405
checking	1,000,000	4,077

3.4 Parser Model

Based on performance evaluations with different model optimization strategies, we decided to adopt the strategy deployed by MDParser. MDParser divides the parser model into several smaller models. For every distinct part-of-speech tag feature (pi-feature), a separate model is trained. All feature sets that are generated in one transition are grouped together according to this pi-feature. Table 2 illustrates the structure of this composite parser model. This separation has the following advantages:

- The models are comparably small and more specific which leads to faster predictions and better accuracy.
- Every model can be optimized according to the pi feature. This is important since the pi-feature considerably influences the optimal prediction for the next parser step.

We also perform a final optimization step on the trained model which eliminates features with a weight of zero. This step reduces the model size and improves the estimation speed, since only a few percentages of the total feature set gets non-zero weights due to the high number of sparse features.

Table 2: Composite parser model based on the part-of-speech tag (pi-feature).

Feature pi	**Model**
pi=VB	model1
pi=NN	model2
...	...

4 EVALUATION

4.1 Evaluation Setting

The experiments in this section compare Syntactic Parser's accuracy and throughput with four publicly available, well known dependency parsers: (i) MaltParser[3], a transition based-parser which supports multiple transition systems, including arc-standard and arc-eager (Nivre et al., 2010), (ii) MSTParser[4], a two-stage multilingual dependency parser created by McDonald et al. (McDonald et al., 2006), (iii) the latest version of the Stanford dependency parser (Chen and Manning, 2014), and (iv) MDParser which has been the target of the optimizations described in this paper.

The evaluations have been performed on a single thread of a Ubuntu 14.04 LTS server with two 16 core Intel Xeon E5-2650 CPUs at 2.0 GHz and 128 GB RAM using the 64-bit Java HotSpot in version 1.8.0_31 and uses the universal dependency treebank v2.0[5], a publicly available collection of treebanks with syntactic dependency annotations. The evaluation framework ran all parsers in their default configuration using the train files enclosed in the corpora for training and the test files for testing accuracy and throughput. We did not perform any optimizations such as running *MaltOptimizer*, optimizing command line parameters or the feature design for any of these parsers (including syntactic parser).

4.2 Results

Table 3 compares the parsers' throughput and accuracy for the English universal dependencies corpus. The experiments report parsing accuracy in terms of correctly assigned head (unlabeled attachment score; UAS) and the fraction of heads and types which have been correctly identified (labeled attachment score; LAS). Bold number indicate the best results for a

[3]maltparser.org

[4]www.seas.upenn.edu/ strctlrn/MSTParser

[5]code.google.com/p/uni-dep-tb/

Table 3: Throughput and accuracy on the universal dependency treebank.

Parser	**Throughput in sent/sec**	**UAS (LAS) per language in %** en	de	fr	es
Syntactic	22,448	**85.3 (82.3)**	78.9 **(73.5)**	79.3 **(75.4)**	82.2 (78.6)
MDParser	5,240	83.9 (80.6)	78.6 (72.2)	79.5 (74.9)	82.1 (78.0)
Stanford	909	82.1 (79.6)	76.6 (71.3)	78.8 (74.5)	80.7 (76.8)
Malt (arc-eager)	628	85.0 (82.2)	78.9 (72.8)	79.1 (75.1)	82.4 (78.8)
Malt (arc-standard)	691	84.8 (82.2)	78.2 (72.4)	79.6 (75.2)	**83.0 (79.3)**
MST	38	84.3 (80.6)	**80.7** (73.1)	**79.8** (74.4)	82.8 (78.0)

particular experiment. The results show that Syntactic Parser provides a significantly higher throughput than any other parser. The differences between MDParser and Syntactic Parser are particularly interesting since Syntactic Parser is based on MDParser and, therefore, employs the same parsing algorithm, indicating that a more than four-fold improvement has been solely achieved by applying the optimizations discussed in this paper. The evaluation also demonstrates that these enhancements did not come at the cost of accuracy, on the contrary, Syntactic Parser outperforms MDParser in three out of four evaluations according to the UAS measure and in all experiments for the LAS metric.

The results presented in Table 3 indicate that MaltParser, MSTParser and Syntactic Parser are very close in terms of accuracy. Syntactic Parser provides the best UAS for English, MaltParser outperforms its competition for Spanish, and MST performs best for German and French. Syntactic Parser also yields the best LAS for English, French and German. MDParser does particularly well for French dependencies, but provides considerably lower scores for the other languages. These differences have been caused by the changes to the feature set and the introduction of a placeholder label for low frequency terms in Syntactic Parser (Section 3.1).

5 OUTLOOK AND CONCLUSIONS

Many big data information extraction approaches still apply lightweight natural language processing techniques since more sophisticated methods do not yet meet their requirements in terms of throughput and scalability. The presented research addresses this problem by optimizing the feature representation and constraint handling of dependency parsers yielding significant speed improvements.

The main contributions of this work are (i) suggesting a method to address common performance bottlenecks in dependency parsers (ii) introducing *Syntactic Parser*[6], a dependency parser which has been optimized based on the techniques presented in this paper, and (iii) conducting comprehensive experiments which outline the performance impact of the proposed techniques, and compare the created parser to other state of the art approaches.

The presented methodology yielded Syntactic Parser, a dependency parser which provides state of the art parsing accuracy and excels in throughput and performance. Implementing the proposed optimizations increased the parsing performance more than four-fold and also yielded a better overall accuracy when compared to MDParser although both parsers implement the same parsing strategy. Syntactic Parser clearly outperforms all other parsers in terms of throughput although it yields a comparable UAS and LAS at the English, French, German and Spanish universal dependency treebank.

ACKNOWLEDGEMENTS

The research presented in this paper has been conducted as part of the COMET project (www.htw-chur.ch/comet), funded by the Swiss Commission for Technology and Innovation (CTI), and the Radar Media Criticism Switzerland project (medienkritik.semanticlab.net/about) funded by the Swiss National Science Foundation (SNSF).

REFERENCES

Bohnet, B. (2010). Very High Accuracy and Fast Dependency Parsing is Not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 89–97,

[6]A Web demo of Syntactic Parser is available at demo.semanticlab.net/syntactic-parser. A runnable version of the parser may be obtained from the authors for scientific purposes.

Stroudsburg, PA, USA. Association for Computational Linguistics.

Chen, D. and Manning, C. (2014). A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Choi, J. D. and Palmer, M. (2011). Getting the Most out of Transition-based Dependency Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 687–692, Stroudsburg, PA, USA. Association for Computational Linguistics.

Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *In Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Gindl, S., Weichselbraun, A., and Scharl, A. (2013). Rule-based opinion target and aspect extraction to acquire affective knowledge. In *First WWW Workshop on Multidisciplinary Approaches to Big Social Data Analysis (MABSDA 2013)*, Rio de Janeiro, Brazil.

He, H., III, H. D., and Eisner, J. (2013). Dynamic Feature Selection for Dependency Parsing. In *Empirical Methods in Natural Language Processing (EMNLP 2013)*.

McDonald, R., Lerman, K., and Pereira, F. (2006). Multilingual Dependency Parsing with a Two-Stage Discriminative Parser. In *Tenth Conference on Computational Natural Language Learning (CoNLL-X)*.

McDonald, R. and Pereira, F. (2006). Online learning of approximate dependency parsing algorithms. In *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006))*, volume 6, pages 81–88.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computer Linguistics*, 34(4):513–553.

Nivre, J. (2009). Non-projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 351–359, Stroudsburg, PA, USA. Association for Computational Linguistics.

Nivre, J. and McDonald, R. (2008). Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio. Association for Computational Linguistics.

Nivre, J., Rimell, L., McDonald, R., and Gómez-Rodríguez, C. (2010). Evaluation of Dependency Parsers on Unbounded Dependencies. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 833–841, Stroudsburg, PA, USA. Association for Computational Linguistics.

Poria, S., Agarwal, B., Gelbukh, A., Hussain, A., and Howard, N. (2014). Dependency-based semantic parsing for concept-level text analysis. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing*, number 8403 in Lecture Notes in Computer Science, pages 113–127. Springer Berlin Heidelberg.

Qiu, G., Liu, B., Bu, J., and Chen, C. (2011). Opinion Word Expansion and Target Extraction through Double Propagation. *Computational Linguistics*, 37(1):9–27.

Volokh, A. (2013). *Performance-oriented dependency parsing*. PhD Thesis, Saarland University.

Volokh, A. and Neumann, G. (2012). Dependency Parsing with Efficient Feature Extraction. In Glimm, B. and Krüger, A., editors, *KI 2012: Advances in Artificial Intelligence*, number 7526 in Lecture Notes in Computer Science, pages 253–256. Springer Berlin Heidelberg.