Supplemental technical report

Anonymous

January 2020

This document contains supplemental technical information for the paper "A fast filtering algorithm for massive context free grammars."

1 Supplement to theoretical analysis of the TTF

Note that the below argument is simply a structural argument regarding how large a tree could be in the worst case, it doesn't actually show that any such grammar and vocabulary indexing exist which induce such a structure. However, in the full paper, we provide a construction showing that such a grammar and indexing does exist - we make no claims about the ubiquity of such grammars.

Theorem 1. *The worst-case tree construction for the TTF has $O(|P| \times |T|)$ nodes.*

Proof. There are only $|P|$ nodes, so by definition there can be at most $|P|$ leaf nodes - because there is never a node containing no rules.

In the worst case, each rule has been propagated down through all $|T|$ possible layers of the tree, meaning that in the worst case, there are $|T|$ parents for each of the $|P|$ leaf nodes. This yields a worst-case tree size of $|P| \times |T|$. $\square$

Theorem 2. *In the worst case, the sum of all of the calls to $filter\_rules()$ take $O(|P| \times |T|)$ steps.*

Proof. At worst, every rule undergoes a cal to $filter\_rules()$. A rule can never undergo such a rule more than once because only rules stored in visited leaves undergo such calls. Those leaves constitute a disjoint set of the set of productions. Thus at most $O(|P|)$ rules are filtered. Each such step takes $O(|T|)$ time because the unique elements of the RHS are checked to see if they are in A_s. There are at most $O(|T|)$ unique symbols in each rule. Thus the run time incurred by falls to $filter\_rules()$ is $O(|P| \times |T|)$. $\square$

Theorem 3. *In the worst case, the sum of all set unions takes $O(|P|)$ time.*

Proof. We never add the rules of one node, then add the nodes of one of its descendants. Clearly, all nodes which aren't in a descendant/ancestor relationship have disjoint rules, thus no rule is ever unioned into the result set twice. Thus at most $O(|P|)$ unions occur. $\square$

Theorem 4. *The worst-case run time for the TTF is $O(|P| \times |T|)$.*

Proof. The worst case run time is governed by the time to visit all of the nodes, the time taken to union all of the rules into the result set, and the time taken to filter the productions in the leaf nodes visited.

As noted in theorem 1, there are $O(|P| \times |T|)$ nodes. It takes constant time for each visit (ignoring time for filering rules and unioning them into the result set). Thus visiting nodes contributes $O(|P| \times |T|)$ cost overhead.

As noted in theorems 2 and 3, the other time contributions to the run time are $O(|P| \times |T|)$ and $O(|P|)$.

This sets the overall run time to $O(|P| \times |T| + |P| \times |T| + |P|) = O(|P| \times |T|)$. □

2 Supplement to experiment one

Experiment one was conducted by following the construction of the worst-case grammar spelled out in the analysis section of the full paper.

The construction begins with a small powerset grammars and then augments them by injecting many dummy symbols into the two longest rules in the initial powerset grammar.

The powerset grammar is constructed by starting with an initial vocabulary of size k and producing two productions for each set in the powerset of those k symbols, one production containing as RHS the symbols of that set in increasing lexicographic order, one with the symbols in reverse lexicographic order.

There are $2 \cdot (2^k - k - 1) + k$ productions in the powerset grammar. We then augment the powerset grammar by adding d symbols to the two productions in the initial grammar which contain all k symbols.

We produced 10 grammars in this fashion using $k \in [11, 12]$ and d between 20 and 100 thousand with step sizes of 20,0000. The stats for these grammars are in 1. Note for example, that the first grammar has $k = 11$ and $d = 20000$, accordingly $|T| = 20011$ and $|P| = 2 \cdot (2^{11} - 11 - 1) + 11 = 4083$.

$\|T\|$	$\|P\|$	$\|G\|$
20011	4083	62517
20012	8178	89140
40011	4083	102517
40012	8178	129140
60011	4083	142517
60012	8178	169140
80011	4083	182517
80012	8178	209140
100011	4083	222517
100012	8178	249140

Table 1: Statistics for the 10 grammars in experiment 1

We then conducted run time experiments by running the TTF, early-terminating TTF, and b-filter ten times each on either the two longest sentences in the language, or sentences randomly selected from the rest of the language.

We then performed regressions of those run times vs $|P| \times |T|$, $|G|$, and $|T|$.

The correlations for the run times in each category and the regression variables $|G|$ and $|P| \times |T|$ are shown in table 2. $|T|$ shows the best correlations with the run times across both median and worst-length experiments.

	$\|P\| \times \|T\|$	$\|G\|$	$\|T\|$
b_filter	.247	.484	.588
TTF	.442	.867	.955
early terminating TTF	.288	.37	.415

b_filter	.666	.916	.928
TTF	.606	.969	.984
early terminating TTF	.464	.795	.832

Table 2: R^2 values regressing worst-length (top) and median-length (bottom) input run times to grammar size parameters for the various filters

3 Supplement to experiment two

This experiment was basically similar as experiment one, except it produced powerset grammars with no extra augmentation. The vocab sizes were $|T| \in [15, 22]$. Grammar statistics are below, as are R^2 coefficients for the regression. Regression was only performed against $|G|$. The graphs in the initial submission to ACMSE were mislabeled to say $|P|$ on the x-axis. In this case $|G| \propto |T| \times |P|$, but $|T|$ is a small constant factor so the difference shouldn't be substantial. The R^2 values are in the table below. The graphs are reproduced below with the proper labels.

$\|T\|$	$\|P\|$	$\|G\|$
15	65519	491505
16	131054	1048560
17	262125	2228207
18	524268	4718574
19	1048555	9961453
20	2097130	20971500
21	4194281	44040171
22	8388584	92274666

Table 3: Statistics for the 8 grammars in experiment 2

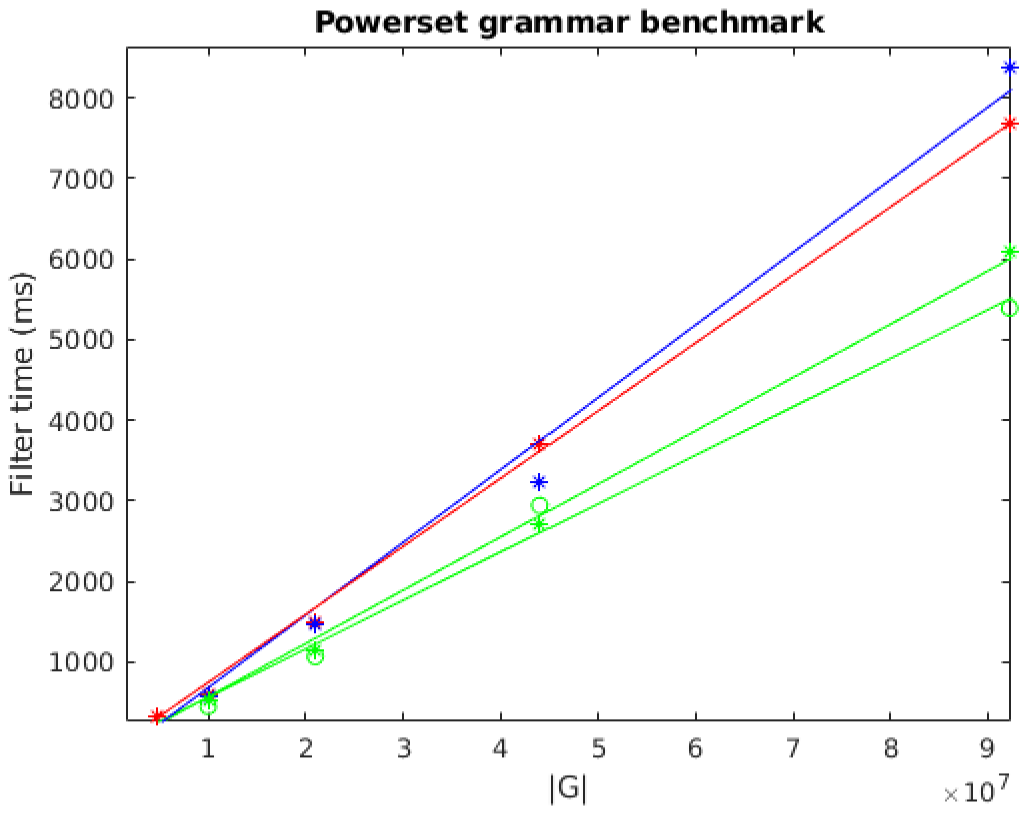

Figure 1: Filter time for inputs on a best-case grammar that produces a full binary tree: stars denote run time on worst string in the language, circles denote run time on median string in the language. Red is the naive TTF, green is b-filter, blue is TTF with early termination

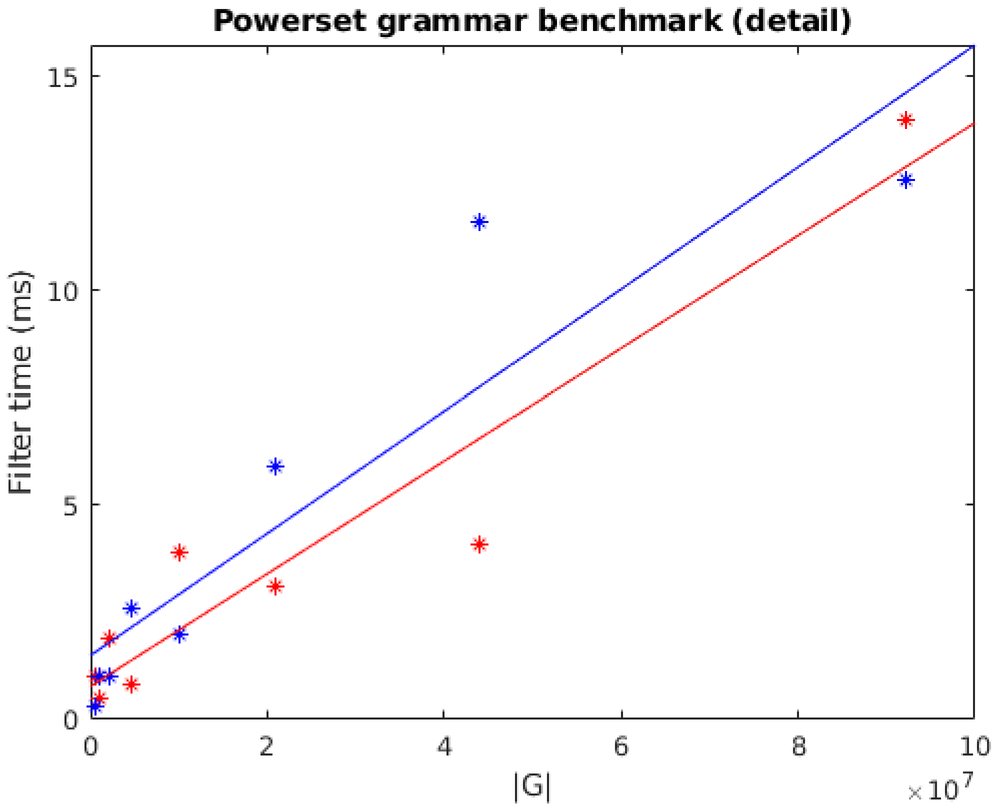

Figure 2: Detail showing linear dependence on $|G|$ for median-length run times of TTF and early terminating TTF

	$\lvert P\rvert$
b_filter	.998
TTF	.998
early terminating TTF	.992
b_filter	.994
TTF	.912
early terminating TTF	.856

Table 4: R^2 values regressing worst-length (top) and median-length (bottom) input run times to grammar size parameters for the various filters

4 Supplement to experiment three

The test CFG for this experiment was produced by converting a Systemic Functional Grammar [1] of English based on Edouard Hovy's SFG from the Penman project [2]. The resulting grammar has $|P| \approx 13,000,000$ and $|G| \approx 115$ million.

The resulting grammar has around 13 million rules. The total grammar size $|G|$, measured by the sums of the lengths of the right hand sides is $115,738,333$. Compare this to the two grammars under study by [?] which had $\approx 500,000$ rules each and total size 1 million and 12 million, respectively. The structure of the Halliday grammar is relatively flat and much of the rules arise from different ways of permuting otherwise equivalent RHSs. In particular, there are $8,975,867$ unique RHSs and $35,228$ unique sets of RHS elements. A consequence of this structure is that it is particularly well suited to both length-based filtering when input sentences are short (a basis for filtering not mentioned in [?] because their study permits grammars with ϵ-productions, while we ours does not) and b-filtering. Additionally, this structure makes the tree bear a structure much more similar to the grammar in experiment 2, than the one in experiment 1, in particular it should be closer to a full binary tree and should have few long right-branching chains.

Note that length-based filtering can not be performed on a grammar containing empty productions - a grammar where nonterminals don't necessarily yield at least one output symbol - because there is no certain relationship between the number of nonterminals on the RHS of a rule and the length of strings generable from it. The grammar (and its accompanying vocabulary) we have constructed is incomplete and can't handle all possible English constructions. For that reason, the performance is analyzed on a handpicked set of 10 sentences which can be found in the technical report on our github.

In order to get a sense of how many rules of this grammar are truly applicable for each input, we filter first by length, then by content, and finally, by removing nonterminals which were made unproductive or unreachable by the previous two steps of filtering. Below is a graph relating length of the input to the number of rules left after all stages of filtering.

As you can see, our grammars respond incredibly well to content filtering, achieving a factor of 100 reduction in grammar size on even the longest test sentence.

- the distributor registered this domain
- this domain was registered by the distributor
- this domain in the account was registered by the distributor
- this domain in the account was registered by the distributor in the domain
- the use of the nicknames in the decryption key is evidence that the account was the distributor of these samples
- the use of the nicknames of this domain in the decryption key is evidence that the account was the distributor of these samples
- the use of the nicknames of this domain in the decryption key in the record is evidence that the account was the distributor of these samples
- the use of the nicknames of this domain in the decryption key in the record of the activity is evidence that the account was the distributor of the software
- the use of the nicknames of this domain in the decryption key in the record of the activity is evidence that the account was the distributor of the software and that it was registered by the communication of the distributors
- the use of the nicknames of this domain in the decryption key in the record of the activity is evidence that the account was the distributor of the software and that it was registered by the communication of the distributors in these discussions

The asymptotic relationship shown in figure 5 between run time of the terminal-tree filter and the number of rules in the filtered subgrammar appears roughly linear for our example grammar, supporting the argument that, in expectation, the run time is roughly linear in the size of the subgrammar, though we lack a suffer from a small sample size for this language.

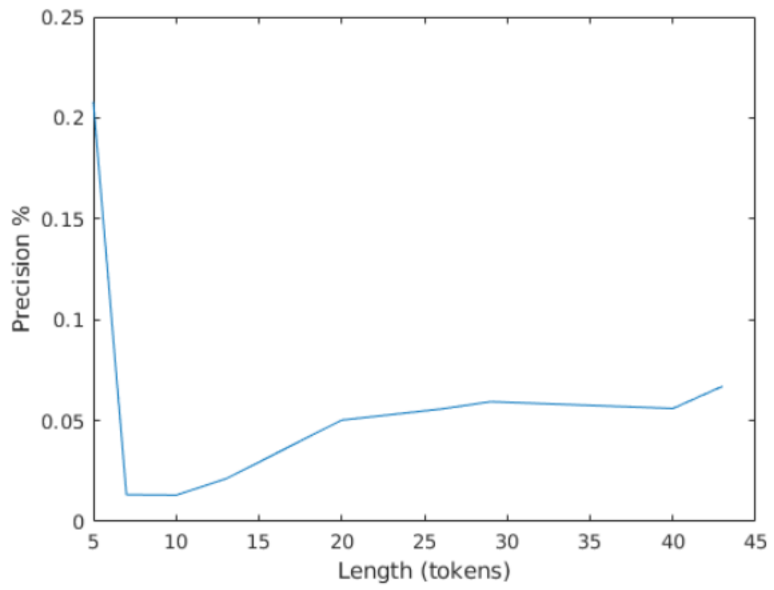

Figure 3: Precision expressed as % as measured by number of rules in the gold-standard divided by number of rules in the subgrammar.

References

[1] Michael Halliday and Christian M.I.M Matthiessen. *An Introduction to Functional Grammar*. Hodder Education, 2004.

[2] Eduard Hovy. The penman natural language project. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '90, page 430, USA, 1990. Association for Computational Linguistics.

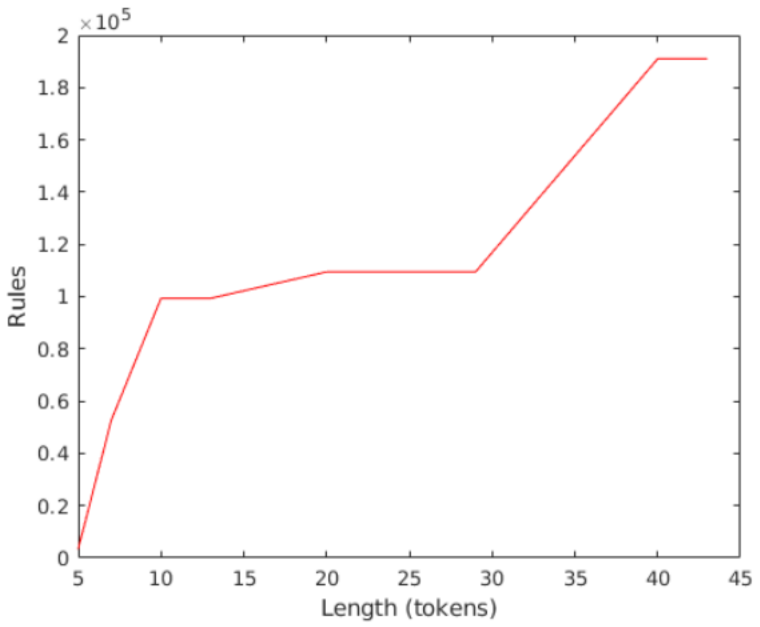

Figure 4: Number of rules resulting after content-filtering. b-filtering and our terminal-tree filtering accomplish close to 100 fold reduction of grammar size on sentences 45 words long

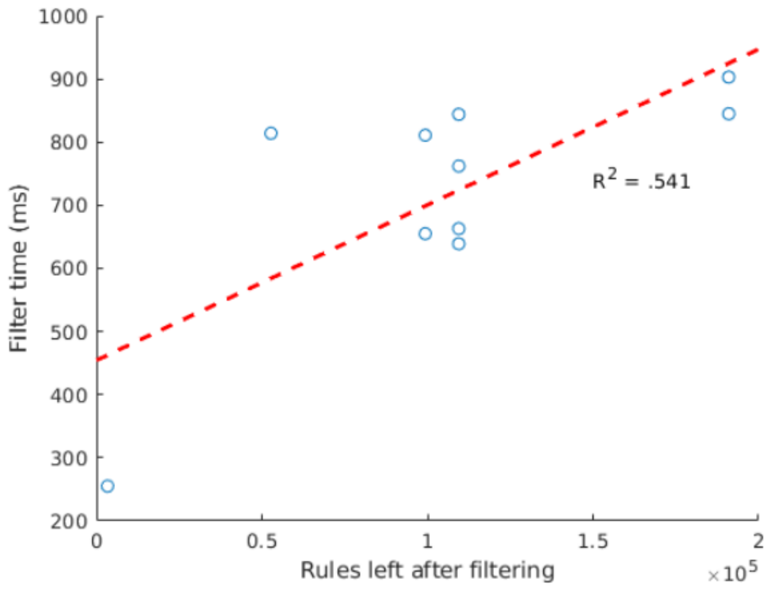

Figure 5: Regression of run time of the TTF to the number of productions in the filtered subgrammar for the test sentences on the English language CFG. We can see that run time is roughly linear in the number of productions counted-in.