JMLR: Workshop and Conference Proceedings 57:79–92, 2016

The Generalized Smallest Grammar Problem

Payam Siyari PAYAMSIYARI@GATECH.EDU
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332, USA

Matthias Gallé MATTHIAS.GALLE@XRCE.XEROX.COM
Xerox Research Centre Europe
Meylan, France

Abstract

The Smallest Grammar Problem – the problem of finding the smallest context-free grammar that generates exactly one given sequence – has never been successfully applied to grammatical inference. We investigate the reasons and propose an extended formulation that seeks to minimize non-recursive grammars, instead of straight-line programs. In addition, we provide very efficient algorithms that approximate the minimization problem of this class of grammars. Our empirical evaluation shows that we are able to find smaller models than the current best approximations to the Smallest Grammar Problem on standard benchmarks, and that the inferred rules capture much better the syntactic structure of natural language.

Keywords: Smallest Grammar Problem, Grammatical Inference, Minimum Description Length, Structure Discovery, Substitutability, Unsupervised Parsing, Compression

1. Introduction

The Smallest Grammar Problem (SGP) is the optimization problem of finding a smallest context-free grammar that generates exactly a given sequence. As such, it has some superficial resemblance to Grammatical Inference, both because of the choice of model to structure the data (a formal grammar) and because the goal of identifying structure is explicitly called-out as a potential application of SGP and its extensions (Nevill-Manning and Witten, 1997; Charikar et al., 2005; Gallé, 2011; Siyari et al., 2016). However, concrete applications so far of the SGP to grammatical inference are either remarkably absent in the literature, or have been reported to utterly fail (Eyraud, 2006). The main reason for this is that the definition of the SGP limits the inferred models to be straight-line programs which have no generalization capacity. We present here an alternative definition which removes this constraint and design an efficient algorithm that achieves at the same time:

- smaller grammars than the state-of-the-art, measured on standard benchmarks.
- generalized rules which correspond to the true underlying syntactic structure, measured on the Penn Tree bank.

We achieve this by extending the formalism from simple straight-line grammars to non-recursive grammars. Our algorithm takes as input any straight-line grammar and infers additional generalization rules, optimizing a score function inspired both by the distributional hypothesis (Harris, 1954) and regularizing through the Minimum Description Length (MDL) principle. It is very efficient in practice and can easily be run on sequences of millions of symbols.

2. Related Work

We take inspiration from three related areas: the work around the Smallest Grammar Problem, concrete implementations of Harris substitutability theory (Harris, 1954) and the use of Minimum Description Length principle in grammatical inference.

The **Smallest Grammar Problem** was formally introduced and analyzed in Charikar et al. (2005), and provides a theoretical framework for the popular Sequitur algorithm (Nevill-Manning and Witten, 1997), as well as the work around grammar-based compression (Kieffer and Yang, 2000). It is defined as the combinatorial problem of finding a smallest context-free grammar that generates exactly the given input sequence. By reducing the expression power from Turing machines to context-free grammars it becomes a computable (although intractable) version of Kolmogorov Complexity. This relationship is also reflected in the interest of studying that problem: by finding smaller grammars, the hope is not only to find better compression techniques but also to model better the redundancies and therefore the structure of the sequence. The current state-of-the-art algorithms that obtain the smallest grammars on standard benchmarks are based on a search space introduced in Carrascosa et al. (2011a) which decouples the search of the optimal set of substrings to be used, and the choice of how to combine these in an optimal parsing of the input sequence. That parsing can be solved optimally in an efficient way, and the algorithms diverge in how they navigate the space of possible substrings to be used, trading off efficiency with a broader search. These algorithms include IRRMGP (Carrascosa et al., 2011b) – a greedy algorithm –, ZZ (Carrascosa et al., 2010) – a hill-climbing approach –, and MMAS-GA (Benz and Kötzing, 2013) – a genetic algorithm –. While their worst-case approximation ratio is hard to analyze, they perform empirically better than other algorithms with theoretical guarantees. Applications to grammar learning has been studied in Eyraud (2006, Chapter 3), which concludes that the failure to retrieve meaningful structure is due to the fact that `Sequitur` "est un algorithme dont le but est de compresser un texte à' l'aide d'une grammaire [. . .] Or la fréquence d'apparition d'un motif n'est pas une mesure permettant de savoir si ce motif est un constituant"[1].

Algorithms to find smaller grammars focus on *intrinsic* properties of substrings (occurrences, length). However, in grammatical inference a primary focus is the *context* in which a substring occurs in, a principle that traces back to Harris **substitutability theory** (Harris, 1954). Different classes of substituable languages have been defined, which all start from the intuition that if two words occur in the same context they should belong to the same semantic class (be generated by the same non-terminal). Occurrences of strings uwv and $uw'v$ are therefore a signal that the words w and w' are substituable one by the other (as in "The car is fast" and "The bike is fast", with w = car and w' = bike). In recent years,

1. "is an algorithm whose goal is to compress a text through a grammar [. . .] but the frequency of occurrences of a motif is not a signal to decide whether a motif is a constituent"

very good learnability results have been obtained with different variations of substituable languages (Yoshinaka, 2008; Clark and Eyraud, 2007; Luque and Infante-Lopez, 2010; Coste et al., 2012), and those insights have long been the basis of unsupervised learning algorithms applied to natural language text (Van Zaanen, 2000; Solan et al., 2005; Scicluna and De La Higuera, 2014).

The **Minimum Description Length** principle is a popular approach for model selection, and states that the best model to describe some data is the one that minimizes jointly the size of the model and the cost of describing the data given the model. This has been applied often in tools targeted to discover meaningful substructure, including grammatical inference (Cook and Holder, 1994; Keller and Lutz, 1997). In this context, the grammars obtained through the SGP are learning the data by heart as they do not perform any generalization. Instead of this, we propose to extend this model and to control its generalization capacity through MDL. This point is also our main divergence with Chirathamjaree and Ackroyd (1980) who also infer non-recursive grammars (although only linear grammars). While their use of the edit distance to decide which rules to re-use reflects the minimization intuition in MDL, this is not used when deciding when to generalize. It is therefore not straightforward how to measure the cost of deriving a particular string. Without such a cost, any comparison with algorithms inferring straight-line grammars seems unfair. In this paper, much of our attention is focused on how to derive non-recursive grammars that still are able to generate the input sequence exactly. While such a purist strategy may not be ideal to obtain the best empirical performance for structure inference, in this line of work we want to test the boundaries of such an approach and to be able to make fair comparisons with the results coming from the SGP.

3. Model

By focusing on grammars that generate a single sequence only, the SGP limits itself to straight-line grammars, which are context-free grammars which do neither branch nor loop. We propose here to relax the first of these constraints, allowing branching non-terminals. Such "non-recursive grammars" have found use in natural-language processing, where several applications have this characteristic (Nederhof and Satta, 2002), despite the fact that they are only able to generate finite languages.

Definition 1 *A non-recursive context free grammar is a tuple $G = \langle \Sigma, \mathcal{N}, S, \mathcal{P} \rangle$, with terminals Σ, non-terminals $\mathcal{N}$ starting symbol S and context-free production rules $\mathcal{P}$ such that for any $A, B \in \mathcal{N}$ not necessarily distinct, if B occurs in a derivation of A, then A does not occur in a derivation of B.*

Different from straight-line grammars, the language generated by non-recursive CFG can be larger than a single string, although it is always finite. In the spirit of the smallest grammar problem, we are still interested in encoding exactly one given string and have therefore to specify which of all the strings in the language is the encoded one.

The size of such a grammar with respect to a specific sequence s – which we will then try to minimize – will therefore be the sum of two factors: the size of the general grammar, and the cost of specifying s:

Definition 2 *Given a non-recursive context-free grammar $G = \langle \Sigma, \mathcal{N}, S, \mathcal{P} \rangle$, the size of G wrt s is defined as:*

$$|G|_s = \sum_{N \to \alpha \in \mathcal{P}} (|\alpha| + 1) + cost(s|G)$$

where *cost(s|G)* is the cost of expressing s given G, and should be expressed in the same unit that the grammar itself, namely symbols.

We are now ready to define our generalized version of the smallest grammar problem:

Definition 3 *Given a sequence s, a smallest non-recursive grammar is a non-recursive context-free grammar G such that $s \in L(G)$ and $|G|_s$ is minimal.*

Note that we do not put any restrictions on the language that the grammar could generate. A more general grammar (one that generates a larger language) may have less or shorter production rules, but the decoding may be more expensive. This is a standard trade-off in any MDL formulation. In the extreme case, where $|L(G)| = 1$ and $\text{cost}(s|G) = 0$, this reduces to the traditional smallest grammar problem. The goal of generalizing the definition is that by adding ambiguities, the grammar would end up having smaller size even with some amount of cost being added for resolving the ambiguities.

Before we define $cost(s|G)$, we need to introduce the mentioned type of ambiguity that when introduced, is potential to lead us to smaller grammars. From a compression perspective, we are interested in capturing more flexible patterns than just exact repeats. In that way, we are looking for non-terminals that generate words v and v', where both words are different although *similar*, so that disambiguating between them is cheap. Several such similarities have been defined in the domain of inexact pattern matching (Navarro, 2001), and a common practice is to start with *seeds*, which are exact repeats and then try to extend them to enlarge their support (number of occurrences) while minimizing the added differences. Such an idea has been used for instance for DNA compression (Chen et al., 2001; Gallé, 2011). The particular kind of inexact motif we focus on is based on insights from theoretical and experimental results in grammatical inference. We assume that v and v' share a common prefix and suffix, and that all the changes are contained in the middle. This is, $v = pws$ and $v' = pw's$, with $w \neq w'$. This corresponds to typical distributional approaches which look for words v, v' that occur in the same context, and has been applied as such similarly in ABL (Van Zaanen, 2000)

Van Zaanen (2002, Section 2.2) argues that replacing unequal parts leads to a smaller grammar than replacing equal parts. However, the analysis there does not take into account that replacing unequal parts adds ambiguity, which – from a lossless compression perspective – has to be disambiguated in order to retrieve the correct sequence. It is surprisingly hard to define an encoding in such a way that replacing such motifs results in grammars that are smaller than those that can be obtained by replacing exact repeats only, as has been reported previously (Dorr and Coste, 2014). In the remainder of this section we will describe several attempts of finding such encodings.

3.1. Algorithm

We propose to extend the greedy algorithm for inferring small straight-line grammars (Apostolico and Lonardi, 2000; Nevill-Manning and Witten, 2000) to take into account such inexact motifs. That algorithm (`Greedy`) chooses in each iteration the repeat that reduces the most the current grammar. For an exact repeat u, the gain $f(u, G)$ is the reduction in size of replacing all occurrences[2] of u in G by a new non-terminal N and adding a rule $N \to u$. By encoding the grammar in a single string, separating rules by special symbols[3], it can easily be shown that $f(u, G) = (|u| - 1)\ (\text{occ}_G(u) - 1) - 2$, where $\text{occ}_G(u)$ is the number of non-overlapping occurrences of u in G. Deducing such a formula for branching non-terminals is a bit more complicated, and depends strongly on the specific encoding used.

3.2. Encoding the Grammars

In order to provide a fair comparison with the straight-line grammars, we will model carefully the way the grammar is encoded. It should be done in such a way that the target sequence can be retrieved unambiguously from that encoding alone. This choice will then guide the optimization procedure to minimize it, and as we will see, it influences heavily the resulting grammar.

As a technical point we note that we take advantage of the sequential nature of the final encoding to sort and re-name the non-terminals conveniently. Before encoding the final grammar, the non-terminals are sorted by their depth in the parsing trees. For this, we define $\mathit{depth}_G(N)$ for $N \in \mathcal{N}$ as the maximal depth over all parse trees of all sequences $s \in L(G)$, and have the following:

Proposition 4 *If G is a non-recursive grammar, then $\mathit{depth}_G(N)$ is well-defined.*

which comes directly from the absence of recursion in these grammars.

We first choose to encode all the possible branchings sequentially, separated by a special separator symbol ($|$, where $| \notin (\Sigma \cup \mathcal{N})$).

3.2.1. Variable-length encoding

In the most general setting, we are interested in motifs of the form $u.^*v$, which match any substring of the form uwv with $w \in (\Sigma + \mathcal{N})^*$. Searching for such general motifs is computationally expensive but feasible, by considering all pairs of repeats.

As an example, consider the following sequence:

s = Alice was beginning to get very tired
Alice was getting very tired
Alice is very tired
Alice will be very tired
Alice was getting very tired

where we assume the alphabet to be the set of English words. Consider now the following grammar G_1 generating s:

2. Actually, a maximal set of non-overlapping occurrences.

3. So that the size of a grammar is just $\sum_{N \to \alpha \in \mathcal{P}} |\alpha| + 1$

$$\begin{aligned} S \rightarrow&\ N\ N\ N\ N\ N \\ O \rightarrow&\ \text{Alice } V \text{ very tired} \\ I \rightarrow&\ \text{was beginning to get} \mid \text{was getting} \mid \text{is} \mid \text{will be} \mid \text{was getting} \end{aligned} \tag{1}$$

which would be encoded as "$N_1N_1N_1N_1N_1$ # Alice N_2 is very tired # was beginning to get | was getting | is | will be | was getting #", where # is an end-of-rule separator and | the choice separator. Note that the expansion "was getting" is repeated twice. While this provides redundancy in this case, it is needed to ensure a unique decoding. An alternative solution, for instance, would be to provide a list of occurrences and spelling out each unique expansion only once. However, this does not end up with a better gain in the end, and – in addition – if this repetition represents a significant loss (because it occurs many times, or it is very long), it should be captured by a non-branching rule.

For such an encoding, *cost(s|G)* is the same for all s and is included in the encoding of the grammar. In this example, $|G_1| = 28$.

Unfortunately, this choice of general motifs and encoding proves to be unfit to compete against simple repeats. It can be shown that the reduction in the grammar size achieved by replacing one such motif is always bounded by the gain obtained by replacing both exact repeats u and v.

3.2.2. Fixed-length encoding

The reason for the lack of improvement with variable-length encoding is the additional overhead from the separator symbols (|). A standard strategy in data compression to get rid of separators is to focus on fixed-length words. We adapted this by restricting the inside part of the motif to be of fixed size. This is, we search for motifs of the form $u.^k v$, which match any substring of the form uwv with $w \in (\Sigma \cup \mathcal{N})^k$. While more restrictive in what they can capture, those motifs allow a more efficient encoding. An example grammar G_2 that represents sequence s and uses the knowledge of fixed-length motifs is:

$$\begin{aligned} S \rightarrow&\ \text{Alice was beginning to get very tired} \\ &\ O \ \ \text{Alice is very tired} \ \ \ O\ O \\ O \rightarrow&\ \text{Alice } I \text{ very tired} \\ I \rightarrow&\ \text{was getting} \mid \text{will be} \mid \text{was getting} \end{aligned} \tag{2}$$

which would be encoded as " Alice was beginning to get very tired N_1 Alice is very tired N_1 N_1 # Alice N_2 very tired # was getting | will be was getting". The separator symbol now has to be used only the first time, indicating the length of the expansion. As all right-hand sides of the same inside rule now have fixed-length, this information can be used to retrieve all production rules unambigously. The length of the expansion until that point, together with the number of occurrences of N_1 indicates the end of that rule without need of providing an additional end-of-rule symbol. The total size of G_2 is then 27.

We can now deduce the gain introduced by replacing a motif $u.^k v$ with non-terminals O and I:

$$\begin{array}{rcll} f(u.^k v, G) = & & \big((|u|+|v|)\ \mathrm{occ}_G(u.^k v)\big) & \text{gain in the sequence} \\ & - & \mathrm{occ}_G(u.^k v) & \text{new non-terminal} \\ & - & (|u|+|v|+1+1) & O \text{ rule + separator symbols} \\ & - & (1+1) & \text{separator symbols for } I \text{ rule} \\ = & & & \\ & & \big((|u|+|v|-1)\ (\mathrm{occ}_G(u.^k v)-1)\big)-5 & \end{array}$$

This results in the algorithm `NRGreedy`$_{\texttt{fix}}$ given in Alg. 1. $G_{w\mapsto N}$ refers to the grammar where all non-overlapping occurrences of the string w are replaced by the new non-terminal N and $N \to w$ is added to the productions. Similarly $G_{u.^k v\mapsto O,I}$ is the grammar where all non-overlapping realizations of $u.^k v$ are replaced by the new non-terminal O, and the rules are extended with $O \to uIv$ and $I \to w$ for all w such that uwv occurs in G.

While a smallest possible fixed-length non-recursive grammar is obviously smaller than a smallest straight-line grammar (because more general), our experiments (see Sect. 4) show that the final grammars obtained with `NRGreedy`$_{\texttt{fix}}$ are actually larger than those obtained by minimizing the size of straight-line grammars only.

3.3. Post-processing Algorithm

We finally report the results of a simple but effective method that starts from any straight-line grammar, and infers branching rules from it (Alg. 2). This is reminiscent of work done to generalize the output of the `Sequitur` algorithm (Nevill-Manning, 1996, Chapter 5).

The algorithm starts from any proposal for the smallest grammar problem. We then search for fixed-lengths motifs $u.^k v$, replace them greedily one by one until no further compression can be achieved starting with the one that achieves the highest compression. Note that, because the algorithm starts from a straight-line grammar with no positive-gain repeats left, all repeated left and right contexts are of length one[4], therefore reducing greatly the execution time.

Algorithm 1: Greedy algorithm `NRGreedy`$_{\texttt{fix}}$ to compute small non-recursive grammar generating s

```
Data: string $s$
Result: non-recursive grammar $G$ such that $s \in L(G)$
$G := \langle \Sigma(s), \{S\}, S, \{S \to s\} \rangle$;
while true do
    $w := \max_{w \in (\Sigma \cup \mathcal{N})^*} f(w, G)$;
    $u.^k v := \max_{u,v \in (\Sigma \cup \mathcal{N})^*, k \in \mathbb{N}} f(u.^k v, G)$;
    if $f(w,G) \le 0 \land f(u.^k v, G) \le 0$ then return $G$;;
    else if $f(w,G) > f(u.^k v, G)$ then
        $N$ is a fresh non-terminal;
        $G := G_{w \mapsto N}$;
    else
        $O, I$ are fresh non-terminals;
        $G := G_{u.^k v \mapsto O,I}$;
end
```

4. Which could of course be a non-terminal.

Algorithm 2: Post-Processing algorithm to compute small non-recursive grammar generating s

```
Data: string $s$, SGP algorithm sgp
Result: non-recursive grammar $G$ such that $s \in L(G)$
$G := \texttt{sgp}(s)$;
while true do
    $u.^k v := \max_{u,v \in (\Sigma \cup \mathcal{N})^*, k \in \mathbb{N}} f(u.^k v, G)$;
    if $f(u.^k v, G) \leq 0$ then
        return $G$;
    else
        $O, I$ are fresh non-terminals;
        $G := G_{u.^k v \mapsto O,I}$;
end
```

4. Experimental Results

We compared the effectiveness of our proposed algorithm with algorithms that approximate the smallest grammar problem in two areas. The first is the direct goal of SGP, namely, to find small grammars that encode the data. We show how the more expressive grammar can lead to consistently smaller grammars, and considerably so in sequences with a large number of fixed-size motifs. We also report results on qualitative measures of the obtained structure, using a linguistic corpus annotated with its syntactic tree structure.

4.1. Smaller Grammars

In this section, we compare the compression performance of our algorithm with four SGP solvers: `Greedy` (Nevill-Manning and Witten, 2000; Apostolico and Lonardi, 2000), IR-RMGP, ZZ (Carrascosa et al., 2011b) and *MMAS-GA* (Benz and Kötzing, 2013). We report the results on two datasets widely used in data compression: a DNA corpus[5] and the Canterbery corpus[6]. Details about these corpora are in Table 1.

The results on the final sizes are reported in Table 2. As pointed out before, we did not achieve to obtain smaller grammars by incorporating branching-rules inference during the main process (algorithm $\texttt{NRGreedy}_{\texttt{fix}}$). The final grammars were consistently larger than the simplest baseline (`Greedy`), often considerably so (see for instance `humhdab`, `alice29.txt`). However, the same idea of inferring fixed-motifs proved to be successful when applied as a post-processing. Moreover, this strategy can be applied to any straight-line grammar and can therefore be used after any SGP algorithm. Under the column #Ctx, we give the number of branching rules that are inferred. The number of occurrences of these rules is of course much higher in general.

While the reduction in size is small, it applies consistently throughout all the SGP algorithms we tried[7]. The better the original algorithm, the smaller the gain. While this

5. http://people.unipmn.it/~manzini/dnacorpus/historical/
6. http://corpus.canterbury.ac.nz/
7. We did not have access to the final grammars of *MMAS-GA*

sequence	$\|s\|$	$\|\Sigma(s)\|$	$\|\mathcal{R}(s)\|/\|s\|$
chmpxx	121,024	4	0.82
chntxx	155,844	4	0.77
hehcmv	229,354	4	1.46
humdyst	38,770	4	0.77
humghcs	66,495	4	13.77
humhbb	73,308	4	9.01
humhdab	58,864	4	1.21
humprtb	56,737	4	1.07
mpomtcg	186,609	4	1.36
mtpacga	100,314	4	0.97
vaccg	191,737	4	2.21

sequence	$\|s\|$	$\|\Sigma(s)\|$	$\|\mathcal{R}(s)\|/\|s\|$
alice29.txt	152,089	74	1.45
asyoulik.txt	125,179	68	1.22
cp.html	24,603	86	4.32
fields.c	11,150	90	5.03
grammar.lsp	3,721	76	3.43
kennedy.xls	1,029,744	256	0.08
lcet10.txt	426,754	84	2.00
plrabn12.txt	481,861	81	1.02
ptt5	513,216	159	194.74
sum	38,240	255	17.44
xargs.1	4,227	74	1.77

Table 1: Statistics of the the DNA corpus (left) and Canterbury (right). We report length, number of different symbols and the number of repeats normalized by length.

may point towards a convergence of the possible redundancy that can be extracted, it should be noted that our approach runs much faster than the more sophisticated algorithms (*ZZ*, *MMAS-GA*). Moreover, our best result in Table 2 become the new state-of-the-art in several cases, and we would expect an even better improvement if starting from the final grammars output by *MMAS-GA*.

We analyzed separately the huge difference in the gain obtained on the `kennedy.xlsx` file. This is a binary file, encoding a large spreadsheet (347×228, in Excel format) containing numerical values, many of which are empty. Most of the gains over the straight-line baselines seem to come from the way these numbers are getting encoded, with a common prefix and suffix and a fixed-length field for the specific value. These fields are therefore ideal candidates for our branching-rule inference. We were able to recreate those results by generating random Excel tables, obtaining improvements of 6 to 33% (relative to the original size of IRRMGP) depending on the number of non-zero entries the table had.

4.2. Better Structure

Following the original motivation for closing the gap between the structures found in SGP and the structures that are sought in grammatical inference, we evaluated the obtained branching rules by their capacity for unsupervised parsing. For this we benchmarked our method in the task of *unsupervised parsing*, the problem of retrieving the correct tree structure of a natural language text. We took the standard approach in the field, starting from the Part-of-Speech (POS) tags of the Penn Tree bank dataset (Marcus et al., 1994). Current *supervised* methods achieve a performance above 0.9 of F_1 measure (Vinyals et al., 2015). As expected, *unsupervised* approaches report worse performance, around 0.8 (Scicluna and De La Higuera, 2014). These are in general very computationally intense methods and performance is only reported on top of WSJ10, sentences of size up to 10. We diverge from that, reporting results on all 49 208 sentences[8]. For evaluation, we used precision over the set of brackets, together with the percentage of non-crossing brackets (Klein, 2005), a standard

8. Excluding sentence 1855, for which the EVALB evaluation tool we used had trouble processing.

Data	Grammar Size										
	`Greedy`	`NRGreedy`$_{\texttt{fix}}$	+Post	*#Ctx*	IRRMGP	+Post	*#Ctx*	ZZ	+Post	*#Ctx*	green MMAS-GA
chmpxx	28,704	**29,477** ↑	**28,534** ↓	64	27,683	**27,584** ↓	27	26,024	**26,024** =	0	25,882
chntxx	37,883	**38,212** ↑	**37,703** ↓	71	36,405	**36,285** ↓	25	33,942	**33,942** =	0	33,924
hehcmv	53,694	**54,451** ↑	**53,398** ↓	113	51,369	**51,242** ↓	30	-	-	-	48,443
humdyst	11,064	**11,166** ↑	**11,017** ↓	19	10,700	**10,680** ↓	6	10,037	**10,037** =	0	9,966
humghcs	12,937	**13,655** ↑	**12,908** ↓	14	12,708	**12,708** =	0	12,033	**12,023** ↓	5	12,013
humhbb	18,703	**18,893** ↑	**18,614** ↓	36	18,133	**18,060** ↓	20	17,026	**17,024** ↓	1	17,007
humhdab	15,311	**19,736** ↑	**15,242** ↓	27	14,906	**14,879** ↓	8	13,995	**13,995** =	0	13,864
humprtb	14,884	**17,122** ↑	**14,817** ↓	26	14,492	**14,451** ↓	11	13,661	**13,661** =	0	13,528
mpomtcg	44,175	**45,018** ↑	**43,930** ↓	89	42,825	**42,658** ↓	40	39,913	**39,911** ↓	1	39,988
mtpacga	24,556	**24,878** ↑	**24,408** ↓	58	23,682	**23,608** ↓	16	22,189	**22,189** =	0	22,072
vaccg	43,711	**44,261** ↑	**43,445** ↓	99	41,882	**41,778** ↓	29	-	-	-	39,369
alice29.txt	41,001	**50,777** ↑	**40,984** ↓	7	40,218	**40,218** =	0	37,702	**37,662** ↓	12	37,688
asyoulik.txt	37,475	**45,520** ↑	**37,464** ↓	4	36,910	**36,905** ↓	1	35,001	**34,953** ↓	16	34,967
cp.html	8,049	**8,310** ↑	**8,003** ↓	6	7,974	**7,971** ↓	1	7,768	**7,747** ↓	9	7,746
fields.c	3,417	**3,681** ↑	**3,380** ↓	7	3,385	**3,381** ↓	1	3,312	**3,285** ↓	13	3,301
grammar.lsp	1,474	**1,475** ↑	**1,458** ↓	2	1,472	**1,472** =	0	1,466	**1,462** ↓	1	1,452
kennedy.xls	166,925	-	**99,915** ↓	1,233	166,810	**98,479** ↓	1,174	166,705	**98,258** ↓	1,161	166,534
lcet10.txt	90,100	**115,625** ↑	**89,998** ↓	33	88,778	**88,750** ↓	9	-	-	-	87,086
plrabn12.txt	124,199	**165,122** ↑	**124,009** ↓	58	120,770	**120,760** ↓	2	-	-	-	114,960
ptt5	45,135	-	**45,118** ↓	7	44,129	**44,123** ↓	3	-	-	-	42,661
sum	12,207	**14,722** ↑	**11,761** ↓	52	12,127	**11,868** ↓	34	-	-	-	12,009
xargs.1	2,006	**2,092** ↑	**2,006** =	0	1,993	**1,990** ↓	1	1,973	**1,948** ↓	3	1,955

Table 2: Size of the final grammars obtained with the different algorithms. SGP algorithms (non-bold numbers) generate straight-line grammars, while `NRGreedy`$_{\texttt{fix}}$ and the **+Post** columns (bold numbers) infer non-recursive grammars. Green/Red down-/up-ward arrows show a reduction/increase in the grammar size with respect to the output of the reference algorithm in each section, and = shows no change. *#Ctx* is the number of branching rules detected by the post-processing algorithm. In the cases with -, either the final grammar or the output of ZZ algorithm was not available. The results for MMAS-GA are taken from Benz and Kötzing (2013).

practice for which we relied on the EVALB tool[9] which removes singleton and sentence-wide brackets. While precision is the percentage of correctly retrieved brackets, non-crossing brackets is the percentage over the retrieved brackets that do not contradict a gold bracket and give an idea on how not-incorrect the results are (as opposed to correct).

Our focus is on comparing the quality of the brackets of the branching rules with those of the non-branching rules. We furthermore distinguish the brackets covering a context, and the one covering an inside. For the rules $\{O \to \alpha I \beta, I \to \gamma_1 | \gamma_2\}$, the inside brackets cover γ_1 and γ_2, while the context rules cover $\alpha\gamma_1\beta$ and $\alpha\gamma_2\beta$. The number of context brackets is always the same as the number of inside brackets.

9. `nlp.cs.nyu.edu/evalb/`

As before, we are mainly interested in comparing the additional rules added by non-recursive grammars. The `Greedy` algorithm creates around 950 000 brackets, of which only 21% are correct. The proposed post-processing adds another 792 brackets, but with a much higher precision (50.48%) and mostly consistent (85%). In order to evaluate the sensitivity of these results, and to see if they generalize if more brackets are retrieved, we stopped the `Greedy` algorithm earlier: this creates larger grammars, with more options for the creation of branching rules in the post-processing stage. The final results are summarized in Fig. 1. Because of the small number of brackets (Fig. 1c) we do not report recall. While the numbers of correct and consistent brackets decreases with increasing number of branching rules, they do so very gently and are much higher than the accuracy of non-branching rules. Furthermore, a stark difference appears between context and inside brackets: while context brackets are much more often correct (reaching almost 60%), they are less consistent than inside brackets (which have a non-crossing percentage of 90%). These results get their whole meaning when compared to the brackets obtained by just considering the straight-line grammars. Their accuracy varies very little over the iterations, and is always extremely low (around 22%).

Finally, the drop around 20 000 iterations belongs to a point where highly frequent context patterns[10] stop being captured by branching rules and are modeled by repeats. This also means that the good performance at the end is not due to these easy to model constituents.

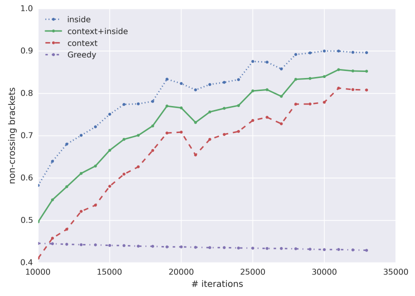
As said, reported values on unsupervised parsing of this dataset focuses on sentences of length up to 10, which only represents less than 10% of the total corpus. On these sentences, the context brackets obtain a precision of 80%, considerably higher than other reported results, although this is not a fair comparison as the number of retrieved brackets is low. But it is worth to highlight that parsing the longer sentences did not pose any problems at all in our (not optimized) implementations: in fact the algorithm was run on the concatenation of the overall set (over $1.3M$ tokens).

5. Conclusions

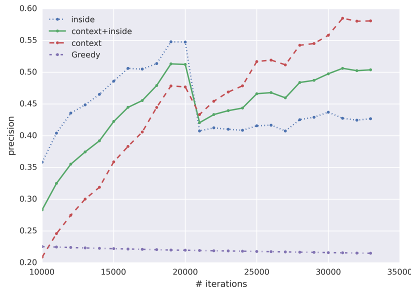
In this paper we provide a first step towards applying the results around the Smallest Grammar Problem for grammatical inference. We identify a probable reason for past failures, and show how to extend the work inferring small straight-line grammars towards non-recursive ones. Our starting point is the MDL principle, and faithful to this principle, we pay careful attention to the encoding of the final grammars to the point that the final search space is strongly constrained by the chosen encoding. This allows us to make a fair comparison with SGP algorithms on standard benchmarks used for that problem, as in both cases we allow to retrieve the original sequence unambiguously. Those algorithms consistently improve over the current state-of-the-art, substantially so in one case (`kennedy.xls` sequence). One direction of future work could focus on formalizing the phenomena exhibited by that sequence and where else it occurs.

With respect to the original motivation of structuring the sequences, the additional rules of our algorithm have a much higher precision than other methods for unsupervised parsing,

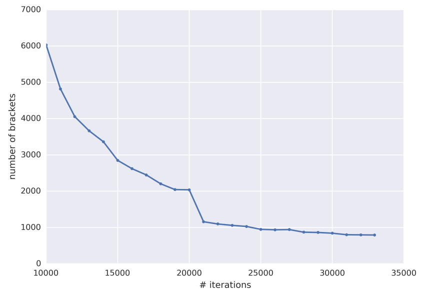
10. Most notably opening/closing quotation and parentheses



(*a*) Percentage of non-crossing brackets



(*b*) Bracketing precision



(*c*) Number of brackets of branching rules

Figure 1: Structuring accuracy of the proposed post-processing algorithm. x-axis is the number of iterations when we stopped the `Greedy` algorithm.

without explicitly trying to optimize for it. Recall, however, is much lower as very few rules are actually inferred. Nevertheless, we believe that our results show the potential of this generalized smallest grammar problem for this task and we are considering on how to build on the efficient algorithms developed in the field to capture more such rules.

References

A. Apostolico and S. Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE*, 88(11):1733–1744, 2000.

F. Benz and T. Kötzing. An effective heuristic for the smallest grammar problem. In *GECCO*, pages 487–494, 2013.

R. Carrascosa, F. Coste, M. Gallé, and G. Infante-Lopez. Choosing word occurrences for the smallest grammar problem. In *LATA*, pages 154–165, 2010.

R. Carrascosa, F. Coste, M. Gallé, and G. Infante-Lopez. The smallest grammar problem as constituents choice and minimal grammar parsing. *Algorithms*, 4(4):262–284, 2011a.

R. Carrascosa, F. Coste, M. Gallé, and G. Infante-Lopez. Searching for smallest grammars on large sequences and application to DNA. *JDA*, 11:62–72, 2011b.

M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

X. Chen, S. Kwong, and M. Li. A compression algorithm for DNA sequences. *IEEE Engineering in Medicine and Biology*, 20(4):61–66, 2001.

C Chirathamjaree and Martin H Ackroyd. A method for the inference of non-recursive context-free grammars. *International Journal of Man-Machine Studies*, 12(4), 1980.

A. Clark and R. Eyraud. Polynomial identification in the limit of context-free substitutable languages. *Journal of Machine Learning Research*, 8:1725–1745, 2007.

D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, pages 231–255, 1994.

F. Coste, G. Garet, and J. Nicolas. Local substitutability for sequence generalization. In *ICGI*, volume 21, pages 97–111, 2012.

F. Dorr and F. Coste. Compressing (genomic) sequences grammar inference. Technical report, INRIA, 2014.

R. Eyraud. *Inférence Grammaticale de Langages Hors-Contextes*. PhD thesis, U. Jean Monnet, 2006.

M. Gallé. *Searching for compact hierarchical structures in DNA by means of the Smallest Grammar Problem*. PhD thesis, Université Rennes 1, 2011.

Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

B. Keller and R. Lutz. Evolving stochastic context-free grammars from examples using a minimum description length principle. In *Workshop on Automata Induction, Grammatical Inference and Language Acquisition*, 1997.

J. C. Kieffer and E. Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.

D. Klein. *The Unsupervised Learning of Natural Language Structure*. PhD thesis, Stanford, 2005.

F. M. Luque and G. Infante-Lopez. PAC-learning unambiguous k,l-NTS languages. In *ICGI*, pages 122–134, 2010.

M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn Treebank: Annotating predicate argument structure. In *Workshop on Human Language Technology*, HLT '94, pages 114–119, 1994.

G. Navarro. A guided tour to approximate string matching. *CSUR*, 33(1):31–88, 2001.

M. Nederhof and G. Satta. Parsing non-recursive context-free grammars. In *ACL*, pages 112–119, 2002.

C. G. Nevill-Manning. *Inferring sequential structure*. PhD thesis, U of Waikato, 1996.

C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical strcture in sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7:67–82, 1997.

C. G. Nevill-Manning and I. H. Witten. On-line and off-line heuristics for inferring hierarchies of repetitions in sequences. *Proceedings of the IEEE*, 88(11):1745–1755, Nov 2000.

J. Scicluna and C. De La Higuera. PCFG induction for unsupervised parsing and language modelling. In *EMNLP*, pages 1353–1362, 2014.

P. Siyari, B. Dilkina, and C. Dovrolis. Lexis: An optimization framework for discovering the hierarchical structure of sequential data. In *KDD*, pages 1185–1194, 2016.

Z. Solan, D. Horn, E. Ruppin, and S. Edelman. Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences*, Jan 2005.

M. Van Zaanen. ABL: Alignment-based learning. In *COLING*, 2000.

M. Van Zaanen. *Bootstrapping Structure into Language: Alignment-Based Learning*. PhD thesis, University of Leeds, Leeds, UK, 2002.

O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. In *NIPS*, pages 2755–2763, 2015.

R. Yoshinaka. Identification in the limit of k,l-substitutable context-free languages. In *ICGI*, pages 266–279. Springer, 2008.