Report on "A Space Optimal Grammar Compression" by Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto

Swrajit Paul and Robert Manos

December 1, 2018

1 Problem statement/ motivation

There are many different instances of "highly-repetitive text collections", some examples are version controlled documents and source code, as well as genome sequences from similar species. These data sets are highly-compressible. As such big data balloons in size, there is a need for "fully online" and scalable methods of compression. In this paper fully online refers to the streaming model that has been dealt with in class and scalable as it sounds refers to an algorithm that really works on huge texts as they appear in the real world. The solution to this problem is grammar compression. A string can be derived deterministically and compressed if represented as a context free grammar (CFG), within in the context of highly repetitive text. There are a number of algorithms in this domain already but many of them have to operate globally on the whole text at once which makes it prohibitive to have everything in ram. It should also be noted that building the smallest context-free grammar to generate a target input string is known to be an NP-hard problem which can not even be approximated within a constant factor. The smallest grammar for a string is approximate offline in O(logN) and O(logN log N) approximate in the online setting [28]. In order for the algorithm to actually scale the amount of ram that is used needs to be reduced. The working space would ideally be some function of the compressed output because the compressed output grows much more slower as the input grows. To set the context of this paper, we are looking at reducing the working space of the algorithm to so called "succinct space" which is the encoded size plus some lower order terms. And to the best knowledge of the authors of this paper the only algorithm that does this "fully-online LCA (FOLCA) [28]" Folca does not accomplish the desired succinct working space as FOLCA requires encoding space plus the space needed for hash table. So the work of this paper is to optimize the working space of FOLCA by simulating its hash table. The authors claim to achieve a significant result from optimizing the compression as well as utilizing two techniques to speed up compression. They claim their algorithm SOLCA outperforms other state of the art algorithms in memory consumption but is multiple times slower.

2 Previous results

::: from paper 42 "Succinct grammar compression

The information-theoretic lower bound on the minimum number of bits needed to represent an SLP with n symbols is 2n + log n! + o(n).

The theorem presents the theoretic lower bound for the number of bits necessary to encode an SLP. The number of bits is some function of n, where $n = |\Sigma \cup V|$. The size of terminals and non terminals. It is immediately observable that O(n) could not suffice to encode such a grammar as that would only store the alphabet and variables. The other sizable part of the grammar that must be encoded is the production rules or how a subset of $\Sigma \cup V$ maps to itself. Naively, in terms of n, an NxN bit matrix could store the production rules for a maximum of $o(n^2)$ but this is a gross over allocation of space as production rules can only start with symbols in the set of variables and there

cannot be such a high degree of connections for an acyclic graph representing a grammar that deterministically derives a single string. Therefore the largest term in the theoretical lower bound O(logn!) equivalent to O(nlogn) feels about right because any algorithm should encode something akin to a mapping of logn variables onto n.

Starting from a CFG or SLP that deterministically derives our string. The set of productions rules P,we can be represented as a directed Acyclic Graph referred to as DAG, from a single source and to a single sink if a super sink s is used to connect all edges that represent production rules, into a DAG ending with s. For example $Z \rightarrow XY$ from the production rules P becomes to directed edges (Z,X) the left edge and (Z,Y) the right edge. Then all of such edges are connected into a graph ending with s. include diagram from axuilary paper It is necessary to decompose the directed graph into parts whose nodes have out degree one. Because there are production rules $Z \rightarrow XY$ nodes with in degree 2 can be slit into two in-branching trees. This is called spanning tree decomposition and is shown with figure 2 from the axilary paper. The TL and TR decomposition comes from the fact that when a tree is built from the production rules of a grammar using each symbol in a rule as a node, such as $Z \rightarrow XY$, there are now two arrows from Z to both X and Y respectively as each node is only one symbol in $\Sigma \cup V$. The proof of the theorem relies uses different sets operations and concepts. While there can be an equivalence to some CFG's that derive a target string by renaming variables that map with the same edge-degree, not all CFG's that derive a string have the same in-degree, the proof requires this canonical form of CFG called straight line program (SLP).

Definition 1: (Karpinsk-Rytter-Shinohara [18]) An SLP is a grammar compression over $\Sigma \cup$ V whose production rules are formed by either $X_i \rightarrow$ a or $X_k \rightarrow X_i X_j$, where a $\in \Sigma$ and $1 \leq$ i, j $<$ k $\leq |V|$.

T_n is the set of all possible ordered trees with n nodes.
T_R is the right of the spanning tree decomposition Admissibility: a grammar is admissible if it only derives a single string.
T is the set of of all possible ordered trees of n nodes for all possible positive values of n. S(n) is the set of all possible DAGs with n nodes and a single source/sink such that any internal node has exactly two children.

Proof: The proof states that for some σ the following relationship holds between the reduction in the size of the super set S(n) and the in-degree of trees in DAG(n) that bound the size of DAG(n) in S(n). This is important as it relates to the growth of any function that can encode an SLP

$|S(n)| / n^{\sigma} \leq |DAG(n)| \leq |S(n)|$ Let S(n, T) = {G $\in$ S(n) | $G = T \oplus T_R, T_R \in T_n$ }. $|S(n,T)|$ =(n $-$ 1)! for each T $\in T_n$ by induction on n $\geq$ 1. The rate in which the set grows is shown by adding T_L into G as a left tree and then by the hypothesis the number of G $\in$ S(n, T_L) is (n- 1)! The rate at which the set grows is confirmed by looking at on T'_L which is the rightmost expansion of T_L such that the rightmost node u is added as the rightmost child of node v in T_L which is then used for $G' \in$ S(n + 1, T'_L) with a left tree T'_L G' is built out of g by taking and edge from g (u,v) and add it as a left edge and to take edge(u,x)

and add it as a right edge Taking s as the source of G over v = s, each G' = G $\oplus$ (u, v)L $\oplus$ (u, x)R $\in$ S(n+1,T'_L) is an admissible graph and the number of such graphs is $n * |S(n, T'_L)| = n * (n-1) =$ n! by the inductive hypothesis.

3 Background

3.1 Notation

The sets Σ and V are finite sets of symbols. Σ is the set containing alphabet symbols and V is the set with Variables. The intersection of Σ and V is the null set. ($\Sigma \cup$ V =$\oslash$)Σ^* is the set containing all possible strings that can be derived from Σ, and Σ^q the set of all possible strings of length q over Σ. The size of set is denoted by $|S|$. $\sigma = |\Sigma|$, n = $|V|$, and N = $|S|$.

CFG and SLP : A special type of CFG(context free grammar) is proposed. CFG G = (, V, D, X_s) where V is a nite subset of X , D is a nite subset of V ($V \cup \Sigma$)*, and $X_s \in$ V is the start symbol. A grammar compression of a string S is a CFG that deterministically derives only S. A SLP(straight line program) is any production rule that takes the form of $X_k \rightarrow X_i X_j$, where $X_i X_j \in \Sigma \cup$ V, and $1 \leq$ i,j $<$ k $\leq$ n + σ.

4 Informal Results / Main Theorem

4.1 Modified Algorithm

FOLCA(Fully online grammar compression) uses dynamic tree structure and its $O(\log n / \log\log n)$ operation time cannot be improved much further because of its proven lower bound. Therefore there needs to be other ways to im The running time of FOLCA can be improved to $O(N/\alpha)$ Using the fact that FOLCA produces a well balanced grammar, therefore the POPPT has of at most 2logn. A new way to store the POSLP is proposed which will take $n \log(n+\sigma) + o(n \log(n+\sigma))$ space.

4.2 Reverse dictionary for inner variables

If there is an inner variable deriving XY , at least one of the following conditions holds, where v_X (resp. v_Y) is the corresponding node of X (resp. Y) in the POPPT:

(i) If a node has two children. v_X is a left child of its parent, and the parent has a right child (regardless of whether an internal node or leaf) representing Y , and

(ii) If a node has two children. v_Y is a right child of its parent, and the parent has a left child (regardless of whether an internal node or leaf) representing X. Therefore, D^{-1}(XY) can be looked up by a constant number of parent/child queries on B and access to L1. Only one of the conditions needs to be checked in order to determine the cases (i) and (ii); check (ii), if X $<$ Y , and check (i), otherwise.

Lemma 3. Let Z be an inner variable deriving $XY \in (V \cup \Sigma)$, and v_Z be the corresponding node of Z in the POPPT. If X < Y , the right child of v_Z is an internal node. Otherwise the left child of v_Z is an internal node. [, A Space-Optimal Grammar Compression]

Proof (Proof by contradiction) Assume that the right child of v_Z is a leaf node which represents Y. Since Z is inner variable according to the lemma, the left child of v_Z must represent X. If Y is larger than X and smaller than Z, the internal node corresponding to Y must be in the subtree rooted at the right child of v_Z , which contradicts the assumption. Assume that the left child of v_Z is a leaf (which represents X). Since Z is inner node, the right child of v_Z must be the internal node representing to Y . Since the internal node corresponding to X appears before the left child of v_Z , X < Y holds, therefore a contradiction.

Lemma 4. We can implement the reverse dictionary for inner variables that supports lookup in O(1) time.

4.3 Reverse dictionary for outer variables

Lemma 5. We can implement the reverse dictionary for outer variables to support lookup in O(log log n) expected time.

Proof For any 1 ≤ i ≤ n_0out the pair L2[i]L3[i] is the right-hand side of the i-th outer production rule (in post-order). Given i, we can compute the post-order number of the variable deriving L2[i]L3[i] by rank1(B,$select_0$01(B,i))+1. Hence, the task of our reverse dictionary is, given XY ∈ (V ∪ Σ)2, to return integer i such that L2[i]= X and L3[i]= Y, if such exists. If a phrase is found in the short suffix, the query is answered in O(1) expected time by using hash table. Thus, in what follows, we focus on the case where the answer is not found in the short suffix. By the GMRDS2 GB2 for L2[1,m0], we can compute in constant time, given an integer X, the range [i_X,j_X] in 2 such that the occurrences of X in L2 is represented by 2[i_X,j_X] in increasing order, namely, i_X = rank1(GB2,$select_0$(GB2,X)) + 1 and j_X = rank1(GB2,$select_0$(GB2,X +1)). Note that Y occurs in L3[i_X,j_X] (the occurrence is unique) iff there is an outer variable deriving XY. In addition, if k ∈ [i_X,j_X] is the occurrence of Y, then 2[k] is the post-order number of the variable we seek. Hence, the problem reduces to computing selectY (L3, rankY (L3, i_X 1) + 1), which can be performed in O(log log n) time by using the GMR for L3. [, A Space-Optimal Grammar Compression]

4.4 Access to the production rules of outer variables

Lemma 6. Given 1 ≤ i ≤ n_out, we can access L2[i]L3[i] in $O(\log \log n)$ time.

Proof If i > n_0out, L2[i]L3[i] is in the short suffixes. As we can afford to store L2[i]L3[i] in a plain array of O(n_out log n_out log log n_out)= o(n_out log n_out) bits of space, we can access it in O(1) time. If i ≤ n_0out, L2[i]L3[i] is represented by GMRs for L2[1,nout] and L3. Using GMRDS4 for L2[1,n_out], we can compute j = 1 2 [i] in O(log logn) time. Then, we can obtain L2[i] by

rank0(GB2,select1(GB2,j)) in O(1) time. In addition, L3[i] can be retrieved by accessing L3[j], which is supported in O(log logn) time by GMR for L3. [, A Space-Optimal Grammar Compression]

4.5 SOLCA

Theorem 7. Given a string of length N over an alphabet of size α, SOLCA computes a succinct POSLP of the string in O(N lg lg n) expected time using $n\log(n+\sigma)+o(n\log(n+\sigma))$ bits of working space. [, A Space-Optimal Grammar Compression]

Proof The input string processed online by SOLCA in the same way as FOLCA. When it compresses the string, it looks up the phrase in the reverse dictionary and if it doesn't exist then it adds the new variable to POSLP. Using the lemmas above(4 and 5), the compression and search for the variable in the reverse dictionary takes $O(\log\log n)$ time and $n\log(n+\sigma)+o(n\log(n+\sigma))$ bits of space.

5 Results

Results included in the experiment is not only FOLCA AND SOLCA but LZD and Re-Pair as a benchmark aswell as SOLCA+CRD, which uses a constant space reverse dictionary(CRD). The CRD suppots reverse dictionary query in constant time, while only using constant space to find items that are frequently reversed essentially caching them. There are two parts to this process of querying SOLCA+CRD, First checking for a given query X_iX_j in the CRD, and secondly if the query does not exist in the CRD only then check in SOLCA's reverse dictionary. It should be noted that this doesn't improve the worst case run time but it does offer some minor imporvment as querying the CRD is in constant time. The expermental results show this relationship between SOLCA and SOLA+CRD. For the experiment the CRD is implemented with a space restriction of 22MB, which is about the size of the cache on the experiment machine. The experiment machine had a terabyte of RAM. The size of the Wikipedia data set is 5GB and the size of the genome dataset is 3GB.

Table 3 Statistical information of input strings.

dataset	length of string (N)	alphabets (σ)	compression ratio (%)		
			SOLCA	LZD	Re-Pair
Wikipedia	5,368,709,120	210	3.65	3.46	0.62[9]
genome	3,273,481,150	20	41.38	36.34	9.05[10]

[1]

The memory consumption is compared for each different method. The data points in these graphs are every $5x10^8$. For the different variants of FOLCA the space is about $3n\log(n+\sigma)$. Because SOLCA and SOLCA+CRD maintain a compressed reversed dictionary their space usages is approximatly 1/3 that of FOLCA's, which is confirmed in these experimental results. What can also be seen by the expiement is the improvement in memory consumption by SOLCA and SOLCA+CRD. The construction time for each is also compared in Figure

4. There is an improvement in FOLCA to FOLCA+ by the use of the succient tree structure repressentation.

The authors conclude that their future work will be applying an improvement that has been made to FOLCA, a self-index, to SOLCA while maintaining optimal working space. This space is still open to improvement in compression time, but any alternative solutions would be confined to using similiar space requirement as this paper shows the space is already optimal so there is no immediate improvement possible there.

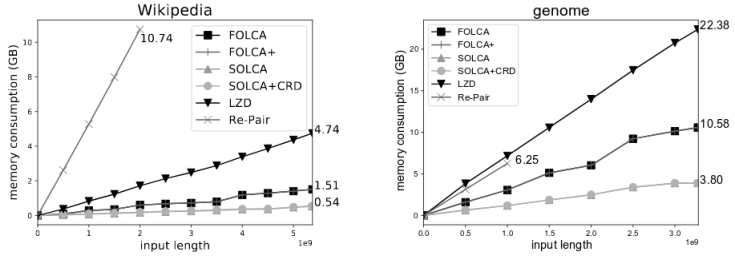

Figure 3 Working space for Wikipedia (left) and genome (right).

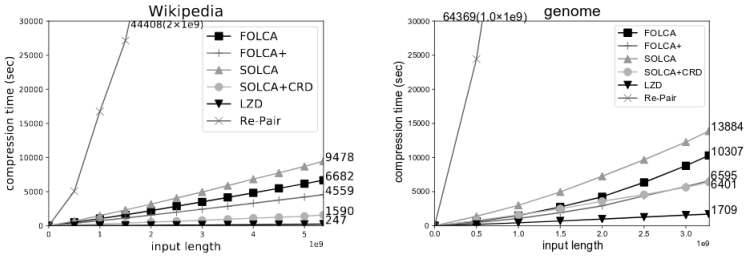

Figure 4 Compression time for Wikipedia (left) and genome (right). [1]

6 Reference

1. Yoshimasa Takabatake 1 , Tomohiro I 2 , and Hiroshi Sakamoto 3 A Space-Optimal Grammar Compression ESA 2017 http://drops.dagstuhl.de/opus/volltexte/2017/7864/pdf/LIPIcs-ESA-2017-67.pdf

2. Yasuo Tabei, Yoshimasa Takabatake, and Hiroshi Sakamoto. A succinct grammar com- pression. In Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013, Bad Herrenalb, Germany, June 17-19, 2013. Proceedings, pages 235246, 2013. doi: 10.1007/978-3-642-38905-4$_2$3.