Improved Grammar Compression in Constant Space

Reimi Tanaka
Department of Informatics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka
820-8502, Japan
r_tanaka@donald.ai.kyutech.ac.jp

Yoshimasa Takabatake
Kyushu Institute of Technology
takabatake@ai.kyutech.ac.jp

Tomohiro I
Kyushu Institute of Technology
tomohiro@ai.kyutech.ac.jp

Hiroshi Sakamoto
Kyushu Institute of Technology
hiroshi@ai.kyutech.ac.jp

Extended abstract of the 14th International Conference on Grammatical Inference
ICGI'18, September 5–7, 2018, Wrocław, Poland.

Abstract

Grammar compression is one of practical compression models based on CFG that is restricted to derives a single string deterministically. To minimize the CFG, the task of grammar compression is closely related to finding and removing duplicated patterns in the input string, and thus the process of finding a minimal grammar is simply considered to be the frequent pattern discovery. We demonstrate how to improve one of such algorithms.

Author Keywords

Grammar Compression; Online Algorithm; Approximation Algorithm; Frequent Pattern Discovery.

In the last decade, various grammar compression algorithms have been proposed intending the frequent pattern discovery in a small space. Almost grammar compression algorithms run in linear time in the input length, however, a huge working space is required for longer patterns. Basically, space-efficient grammar compressions are designed as online algorithms; Given a current grammar G_n for a current input string w and the next input symbol a, an online algorithm is required to construct the next G_{n+1} for wa without decompressing G_n. A frequent pattern mining algorithm proposed by [1] is one of these online algorithms placed in this framework.

However, one of the most successful grammar compressions called *Re-Pair* proposed by [2] has an inherent difficulty for online construction because a most frequent bigram XY must be found to be replaced by a symbol Z associated by the production rule $Z \to XY$. For the length n of the input string, a naive Re-Pair algorithm requires a priority queue in $\Omega(n)$ space containing of all bigrams according to the frequencies. Therefore, several practical implementations of Re-Pair maintain only top-k bigrams enlisting a technique for frequent item mining in stream data proposed by [3].

Lately, a space-efficient Re-Pair like algorithm, called Freq-Re-Pair, was proposed by [4] based on the above strategy preserving the compression ratio. Instead of storing all bigrams, Freq-Re-Pair maintains top-k bigrams with their *relative* frequencies with a fixed k. When a new bigram to be stored appears, the algorithm tries to resister it in the table where if there is no space for the new entry, the algorithm reduces all frequencies by one until a slot opens up. Although the exact ordering is lost, it is guaranteed that the table maintains top-k frequent bigrams in the string processed so far. Freq-Re-Pair is the three-path algorithm consisting of the processes: (i) computing top-k table T for the input string w, (ii) deciding the first position in w of any bigram in T taking account of overlap of bigrams, (iii) executing the replacement of all bigrams after the positions decided above, and the algorithm iterates this process until there is no frequent bigram.

Inspired by this study, we propose an improved Freq-Re-Pair in the following points of view: (1) One of overlapping bigrams XY or YZ in XYZ is replaced according to the relative frequency in the top-k table (Freq-Re-Pair does not takes account this information). (2) The process (ii) in Freq-Re-Pair is omitted for reducing the working space. We compare the performance of these two algorithms for different types of strings including both highly repetitive strings and ordinary strings of megabytes.

As shown in Table 1 for DNA sequences (Escherichia_Coli), our algorithm significantly improves the memory consumption preserving the compression ratio. The result for an ordinary string is shown in Table 2. Similarly to the case of highly repetitive strings, we can observe that our algorithm is more space-efficient than Freq-Re-Pair. Originally, Freq-Re-Pair was applied to learn linear models for classification, regression and feature extraction with various massive high-dimensional data. Besides these applications, an important our future work is an application to the frequent pattern mining and the comparison of the performance among other pattern mining algorithm.

Acknowledgements
This research is partially supported by KAKENHI (18K18111, 17H01791, 16K16009, 17H00762).

REFERENCES

1. S. Fukunaga, Y. Takabatake, T. I, H. Sakamoto. Online Grammar Compression for Frequent Pattern Discovery. In *ICGI*, pages 93–104, 2016.
2. N.J. Larsson and A. Moffat. Offline dictionary-based compression. *Proceedings of the IEEE*, 88(11): 1722–1732, 2000.
3. G.S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *VLDB*, pages 346–357, 2002.
4. Y. Tabei, H. Saigo, Y. Yamanishi and S. J. Puglisi. Scalable Partial Least Squares Regression on Grammar-Compressed Data Matrices. In *KDD*, pages 1875–1884, 2016.

	proposed			Freq-Re-Pair		
#top-k	10^5	10^6	10^7	10^5	10^6	10^7
compression time (s)	1075	861	825	820	1190	1165
compression ratio (%)	14.88	4.37	4.36	13.58	9.47	7.35
working space (MB)	8.71	88.14	300.12	46.68	157.90	904.38

Table 1: Performance analysis for highly repetitive string (Escherichia_Coli, 110MB).

	proposed			Freq-Re-Pair		
#top-k	10^5	10^6	10^7	10^5	10^6	10^7
compression time (s)	1838	2298	2459	1584	2103	2379
compression ratio (%)	21.14	16.53	12.86	20.24	15.81	12.19
working space (MB)	8.83	87.92	957.23	38.51	138.59	1100.09

Table 2: Performance analysis for ordinary text (English, 200MB).