Re-Pair in Small Space**

Dominik Köppl[1] Tomohiro I[2] Isamu Furuya[3]
Yoshimasa Takabatake[2] Kensuke Sakai[2] Keisuke Goto[4]

[1]Kyushu University/JSPS, Japan, dominik.koeppl@inf.kyushu-u.ac.jp
[2]Kyushu Institute of Technology, Japan,
{tomohiro@, takabatake@, k_sakai@donald.}ai.kyutech.ac.jp
[3]Graduate School of IST, Hokkaido University, Japan, furuya@ist.hokudai.ac.jp
[4]Fujitsu Laboratories Ltd., Kawasaki, Japan, goto.keisuke@fujitsu.com

March 6, 2024

Among all known grammar compression schemes, *Re-Pair* [1] is probably the most studied one, featuring empirically observed high compression rates. It is computed by replacing the most frequent bigram of the input with a new non-terminal, recursing until no bigram occurs more than once. For finding the most frequent bigram in affordable time, most algorithms maintain large frequency tables. However, these frequency tables make it hard to compute Re-Pair on large scale data sets. As a solution for this problem we present, given a text of length n whose characters are drawn from an integer alphabet, an $\mathcal{O}(n^2)$ time algorithm computing Re-Pair in $n \lceil \lg \max(n, \tau) \rceil$ bits of working space including the text space, where τ is the number of terminals and non-terminals. Given that the characters of the text are drawn from a large integer alphabet with size $\sigma = \Omega(n)$, our algorithm works *in-place*. This is the first non-trivial in-place algorithm, as a trivial approach on a text T of length n would compute the most frequent bigram in $\Theta(n^2)$ time by computing the frequency of each bigram $T[i]T[i+1]$ for every integer i with $1 \leq i \leq n-1$, keeping only the most frequent bigram in memory. This sums up to $\mathcal{O}(n^3)$ total time, since there can be $\Theta(n)$ different bigrams considered for replacement by Re-Pair. To achieve our goal of $\mathcal{O}(n^2)$ total time, we study a trade-off algorithm finding the d most frequent bigrams in $\mathcal{O}(n^2 \lg d/d)$ time for a trade-off parameter d. We subsequently run this algorithm for increasing values of d by utilizing the space freed up when replacing a bigram with a non-terminal. We show that d increases so fast that we need to run it $\mathcal{O}(\lg n)$ times, which gives us $\mathcal{O}(n^2)$ time in total. Our major tools are appropriate text partitioning, elementary scans, and sorting steps. When $\tau = o(n)$, a different approach using word-packing and bit-parallel techniques becomes attractive, leading to an $\mathcal{O}(n \lg \log_\tau n \lg \lg \lg n / \log_\tau n)$ time algorithm. Our algorithm[1] can further be parallelized or used in external memory.

References

[1] N. J. Larsson and A. Moffat, "Offline dictionary-based compression," in *Proc. DCC*, 1999, pp. 296–305.

**A pre-print is available at http://arxiv.org/abs/1908.04933. This work is funded by the JSPS KAKENHI Grant Numbers JP18F18120 (Dominik Köppl), 19K20213 (Tomohiro I) and 18K18111 (Yoshimasa Takabatake), and the JST CREST Grant Number JPMJCR1402 including AIP challenge program (Keisuke Goto).

[1]We provide a naive implementation at https://github.com/koeppl/repair-inplace.