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, [2]Kyushu Institute of Technology, [3]Graduate School of IST, Hokkaido University, [4]Fujitsu Ltd., Kawasaki

Re-Pair

- Grammar compression: replace recursively bigram with highest frequency
- ☺ High compression ratio in practice
- ☹ Computation needs a lot of memory

Definitions:

Σ : integer alphabet of size $\sigma := n^{\mathcal{O}(1)}$
T : string on Σ of length n
bigram : string of length 2
bigram frequency : number of all *non-overlapping* occurrences of a bigram in T
Cost of storing a bigram with its frequency : $\lceil \lg(n\sigma^2/2) \rceil$ bits

Related Work

Known algorithms computing Re-Pair in (expected) linear time:

Space	Reference
$5n + 4\sigma^2 + 4\sigma' + \sqrt{n}$ words	Larsson and Moffat [4]
$12n + \mathcal{O}(p)$ bytes	González et al. [3]
$(1+\epsilon)n + \sqrt{n} + n$ words	Bille et al. [2]
$(1+\epsilon)n + \sqrt{n}$ words	Bille et al. [1]

where

σ' : the number of non-terminals produced by Re-Pair
ϵ : a constant with $0 < \epsilon \leq 1$
p : the maximum number of bigrams at any time

Our Contribution

A naive in-place algorithm takes $\mathcal{O}(n^3)$ time since it

- needs $\mathcal{O}(n^2)$ time finding the most frequent bigram, and
- may create up to n non-terminals.

We improve this in the word RAM model with

- $\mathcal{O}(n^2) \cap \mathcal{O}(n^2 \lg \log_\tau n \lg\lg\lg n / \log_\tau n)$ time and
- $n \lceil \lg \max(n, \tau) \rceil$ bits of working space including the text space, where τ is the total number of terminals and non-terminals.
- T can be restored with $\mathcal{O}(\lg n)$ additional bits of working space.

For that, we use the following tools:

Tool 1 An array of length n can be sorted in-place in $\mathcal{O}(n \lg n)$ time [5].
Tool 2 With Tool 1, given an integer $d \in [1..n]$, we can compute the frequencies of the d most frequent bigrams
- in $\mathcal{O}(n^2 \lg d / d)$ time
- using $2d \lceil \lg(\sigma^2 n/2) \rceil + \mathcal{O}(\lg n)$ bits.

Pseudo Code

```
1  k ← 0, i ← 0, f_0 ← O(1), T_0 ← T
2  while highest frequency of a bigram in T is > 1 do
3    F ← frequency table of Tool 2 with d := f_k
4    t_k ← minimum frequency stored in F
5    while F ≠ ∅ do                                  ▷ during the i-th turn
6      bc ← most frequent bigram stored in F
7      T_{i+1} ← T_i.replace(bc, X_{i+1})           ▷ create rule X_{i+1} → bc
8      i ← i + 1                                     ▷ introduce the (i+1)-th turn
9      remove all bigrams with frequency < t_k from F
10     add new bigrams to F having X_{i+1} as a character and a
       frequency ≥ t_k
11   f_{k+1} ← f_k + gained frequency space during k-th round
12   k ← k + 1                                       ▷ introduce the (k+1)-th round
```

Description of the Pseudo Code

- Our algorithm works in rounds and turns.
- A round has multiple turns.
- At the start of the k-th round (after Line 2):
  1. Compute the frequency table F with f_k entries using Tool 2.
  2. Fix a threshold t_k equal to the minimum frequency in F (Line 4).
- During the i-th turn create a new non-terminal X_{i+1} (Line 7):
  1. Replace the most frequent bigram stored in F, and
  2. Update F (remove infrequent bigrams, add new bigrams containing X_{i+1}).
- Each turn takes $\mathcal{O}(n)$ amortized time.
- A round ends if F becomes empty (Line 5).
- Terminate when all remaining bigrams have a frequency < 2 (Line 2).

We can show that there is a constant $\gamma > 1$ such that $f_k = \Omega(\gamma^k)$ (Line 11).

- There are $\mathcal{O}(\lg n)$ rounds since we can maintain all bigrams in the $\mathcal{O}(\lg n)$-th round ($f_k = \Theta(n)$ for $k = \Theta(\lg n)$).
- Tool 2: Computing F for k-th round costs $\mathcal{O}((n^2 \lg f_k)/f_k)$ time with $d = f_k$.
- Total Time: $\mathcal{O}\left(n^2 \sum_{k=0}^{\lg n} \frac{k}{\gamma^k}\right) = \mathcal{O}(n^2)$

Example of the First Turn

T and F are stored in entries 1 to 21 and in entries 22 to 24, respectively.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	c	a	b	a	a	c	a	b	c	a	b	a	a	c	a	a	a	b	c	a	b	ab:5	ca:5	aa:3
2	c	X_1		a	a	c	X_1		c	X_1		a	a	c	a	a	X_1		c	X_1		ab:0	ca:1	aa:3
																	D							
3	c	X_1	a	a	c	X_1	c	X_1	a	a	c	a	a	X_1	c	X_1								aa:3
4	c	X_1	a	a	c	X_1	c	X_1	a	a	c	a	a	X_1	c	X_1	c	c	c	a	c			aa:3
5	c	X_1	a	a	c	X_1	c	X_1	a	a	c	a	a	X_1	c	X_1	a	c	c	c	c			aa:3
6	c	X_1	a	a	c	X_1	c	X_1	a	a	c	a	a	X_1	c	X_1							cX_1:4	aa:3
7	c	X_1	a	a	c	X_1	c	X_1	a	a	c	a	a	X_1	c	X_1	a	c	a	c			cX_1:4	aa:3
8	c	X_1	a	a	c	X_1	c	X_1	a	a	c	a	a	X_1	c	X_1	a	a	c	c			cX_1:4	aa:3
9	c	X_1	a	a	c	X_1	c	X_1	a	a	c	a	a	X_1	c	X_1							cX_1:4	aa:3
																							F	

D : temporary character array counting bigrams containing X_1

Row 1: The highest frequency is 5 (due to ab and ca). The lowest frequency represented in F is $t_0 := 3$.

During Turn 1, our algorithm proceeds as follows (cf. Lines 6 to 10):

Row 2: Choose ab as a bigram to replace with a new non-terminal X_1. Replace every occurrence of ab with X_1 while decrementing frequencies in F accordingly to the neighboring characters of the replaced occurrence.

Row 3: Remove from F every bigram whose frequency falls below the threshold t_0. Obtain space for D by aligning the compressed text T_1.

Row 4: Scan the text and copy each character preceding an occurrence of X_1 in T_1 to D.

Row 5: Sort all characters in D lexicographically.

Row 6: Insert new bigrams (consisting of a character of D and X_1) whose frequencies are $\geq t_0$.

Row 7: Symmetric to Row 4: Copy each character *succeeding* an occurrence of X_1 in T_1 to D, then proceed as in Rows 5 to 6 (cf. Rows 8 and 9).

Broadword Approach

- We can search a bigram and replace its occurrences in a broadword of $\mathcal{O}(\log_\tau n)$ bits in $\mathcal{O}(\lg\lg\lg n)$ time (time for popcount), where τ is the total number of terminals and non-terminals.
- Each turn takes $\mathcal{O}(n \lg\lg\lg n / \log_\tau n)$ amortized time.
- Tool 2 can run in $\mathcal{O}(n^2 \lg\lg\lg n / \log_\tau n)$ time.
- Total Time: $\mathcal{O}\left(n^2 \sum_{k=0}^{\lg n} \min\left(\frac{k}{\gamma^k}, \frac{\lg\lg\lg n}{\log_\tau n}\right)\right) = \mathcal{O}\left(\frac{n^2 \lg \log_\tau n \lg\lg\lg n}{\log_\tau n}\right)$

References

[1] P. Bille, I. L. Gørtz, and N. Prezza. Practical and effective Re-Pair compression. *arXiv 1704.08558*, 2017.
[2] P. Bille, I. L. Gørtz, and N. Prezza. Space-efficient Re-Pair compression. In *Proc. DCC*, pages 171–180, 2017.
[3] R. González, G. Navarro, and H. Ferrada. Locally compressed suffix arrays. *ACM Journal of Experimental Algorithmics*, 19(1), 2014.
[4] N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proc. DCC*, pages 296–305, 1999.
[5] J. W. J. Williams. Algorithm 232 - heapsort. *Communications of the ACM*, 7(6):347–348, 1964.