Practical Repetition-Aware Grammar Compression

Isamu Furuya (Hokkaido University)

- We design RL-MR-RePair algorithm.
  - Grammar compression is a lossless data compression method that constructs a small-sized context-free grammar (CFG) that derives only the given text.
  - RePair [1], a grammar compression algorithm for the CFG scheme, achieves a high compression ratio despite its simple scheme.
  - Recently, a variant of RePair called MR-RePair [2] was proposed as a practical improvement.
  - Run-length CFG (RLCFG) [3,4] is an extension of CFG, which theoretically improves the effectiveness of CFG compression.
  - ▷ We design a novel compression algorithm for the RLCFG scheme called RL-MR-RePair, which follows RePair and MR-RePair for the CFG scheme.
- We propose POPPT+PGE.
  - Effective encoding methods for the grammar constructed by MR-RePair have not been discussed.
  - POPPT [5] is a partial parse tree of CFG and PGE [6,7] is an encoding method for a text.
  - ▷ We propose a practical bit-encoding scheme consisting of POPPT and PGE for MR-RePair and RL-MR-RePair.

Experimental results: Sizes of the files compressed by the gzip, bzip2, and RePair variants (the methods achieving the highest compression performance are highlighted in blue font). From top to bottom in each row (datasets) are listed the size (bytes), compression ratio (compressed file size)/(input file size)$\times 100$ (%), and the encoding method (`poppt+pge6` and `poppt+pge8` are the proposed methods).

	gzip	bzip2	RePair	MR-RePair	RL-MR-RePair
fib41	1,176,257 (0.4390)	14,893 (0.0056)	**50** **(0.0000)** `poppt+ible`	60 (0.0000) `poppt+ible`	60 (0.0000) `poppt+ible`
dna.001.1	28,486,029 (27.1664)	27,385,893 (26.1172)	**1,778,453** **(1.6961)** `ps+pge8`	1,894,870 (1.8071) `poppt+pge8`	1,889,630 (1.8021) `poppt+pge8`
sources.001.2	36,023,271 (34.3545)	34,619,138 (33.0154)	**2,324,485** **(2.2168)** `poppt+ible`	2,335,164 (2.2270) `poppt+pge8`	2,334,317 (2.2262) `poppt+pge8`
coreutils	49,920,838 (24.3182)	32,892,028 (16.0229)	5,451,520 (2.6556) `poppt+ible`	**5,106,577** **(2.4876)** `poppt+pge8`	5,106,824 (2.4877) `poppt+pge8`
einstein.en.txt	163,664,285 (34.9989)	24,157,362 (5.1660)	374,902 (0.0802) `poppt+pge6`	362,624 (0.0775) `poppt+pge8`	**362,372** **(0.0775)** `poppt+pge8`
influenza	10,636,889 (6.8710)	10,197,176 (6.5870)	4,137,727 (2.6728) `ps+pge8`	4,064,247 (2.6253) `poppt+pge8`	**4,025,295** **(2.6002)** `poppt+pge8`
para	116,073,220 (27.0399)	112,233,085 (26.1454)	11,710,363 (2.7280) `poppt+pge6`	11,269,822 (2.6254) `poppt+pge6`	**11,203,814** **(2.6100)** `poppt+pge6`
world_leaders	8,287,665 (17.6453)	3,260,930 (6.9428)	739,570 (1.5746) `poppt+pge6`	717,965 (1.5286) `poppt+pge8`	**707,450** **(1.5062)** `poppt+pge8`

Grammar compression algorithm: RL-MR-RePair

- RL-MR-RePair exetuces in $\mathcal{O}(n)$ expected time and words of space.
- Experimentally, RL-MR-RePair constructs smaller grammars for repetitive datasets than either RePair or MR-RePair.

Algorithm RL-MR-RePair

Input: T
Output: $G = \{V, \Sigma, s, R\}$

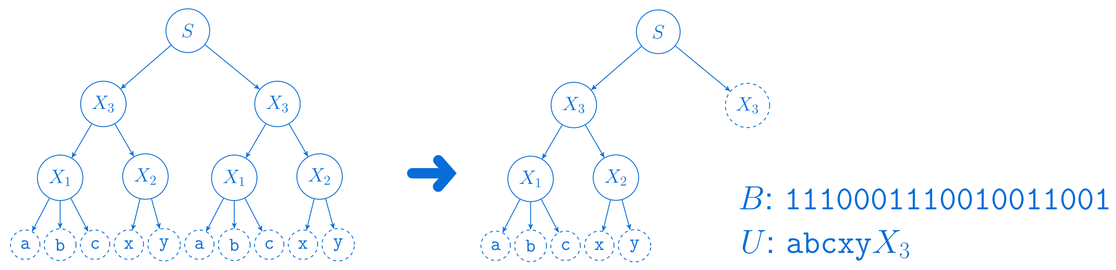
1: Replace each $a \in \Sigma$ in T with a new variable v_a and then add v_a to V and $v_a \to a$ to R.
2: **loop**
3: Find the most frequent maximal repeat r.
4: **if** $\#\mathrm{occ}_T(r) < 2$ **then**
5: Add $s \to T$ to R.
6: **return** G
7: **end if**
8: **if** $r = x^2$ with variable x **then**
9: Replace each run x^k with a new variable v_k and then add v_k to V and $v_k \to x^k$ to R.
10: **else**
11: **if** $|r| > 2$ and $r[1] = r[|r|]$ **then**
12: $r \leftarrow r[1..|r|-1]$
13: **end if**
14: Replace each r in T with a new variable v and then add v to V and $v \to r$ to R.
15: **end if**
16: **end loop**

- RL-MR-RePair constructs a RLCFG by the recursive procedure; it identifies the most frequent maximal repeat and replaces a substring with a new variable.
  - RL-MR-RePair searches run x^k and replaces that run if the most frequent maximal repeat is x^2.
  - Otherwise, it works similarly to MR-RePair, that is, it replaces the most frequent maximal repeat.

S															$S \to X_4X_4X_2$
X_4						X_4						X_2			$X_4 \to X_1X_3$
X_1		X_3				X_1		X_3				X_2			$X_3 \to \mathtt{b}^3$
X_1		b	b	b		X_1		b	b	b		X_2			$X_1 \to \mathtt{a}^3, X_2 \to \mathtt{a}^4$
a	a	a	b	b	b	a	a	a	b	b	b	a	a	a	a	

Bit-encoding scheme: POPPT+PGE

- The right-hand side of each run-length rule $v_i \to v_j{}^k$ is written as a symbol sequence $0kv_j$, where 0 is a special symbol.
  - ▷ In this representation, the RLCFG is treated as a CFG.
- A POPPT (post-order partial parse tree) is a partial parse tree whose internal nodes contain post-order variables.
  - From the (RL)CFG constructed by MR-RePair or RL-MR-RePair, we can generate a POPPT represinting the (RL)CFG.
  - ▷ We can encode POPPT P as a succinct representation comprising a bit sequence B and a text U [8].
    - B is built by traversing P in post-order and assigning c 0s and one 1 to a node with c children. Finally, a single 0 is inserted in B to represent the super node.
    - U stores the symbols of the leaves of P from left to right.



The partial parse tree of a (RL)CFG — POPPT — A succinct repetition

- PGE (packed gamma encoding) is an encoding method for a text.
  - PGE is expected to perform well when symbols in a text have similar values to their adjacent symbols.
- We apply PGE to encoding U of POPPT.
  - ▷ The experimental results confirmed the high compression performance of the encoding scheme.

[1] N.J. Larsson and A. Moffat: "Off-line dictionary-based compression," Proc. of IEEE, vol.88, no.11, pp.1722–1732, 2000.
[2] I. Furuya+: "MR-RePair: Grammar compression based on maximal repeats," DCC 2019, pp.508–517, 2019.
[3] A. Jeż: "Approximation of grammar-based compression via recompression," TCS, vol.592, pp.115–134, 2015.
[4] T. Nishimoto+: "Fully dynamic data structure for LCE queries in compressed space," MFCS 2016, pp.72:1–72:15, 2016.
[5] S. Maruyama+: "Fully-online grammar compression," SPIRE 2013, pp.218–229, 2013.
[6] P. Bille+: "Practical and effective Re-Pair compression," CoRR, vol.abs/1704.08558, 2017.
[7] N. Prezza: "rp: a space-efficient compressor based on the Re-Pair grammar." `https://github.com/nicolaprezza/Re-Pair`, Accessed: 2019-07-18.
[8] Y. Takabatake+: "Online pattern matching for string edit distance with moves," SPIRE 2014, pp.203–214, 2014.