Re-Pair

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

Definitions:
- $\Sigma$: integer alphabet of size $\sigma := n^{O(1)}$
- $T$: string on $\Sigma$ 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: $\lceil \log(n^2/2) \rceil$ bits

Related Work

Known algorithms computing Re-Pair in (expected) linear time:
- Space: Reference
  - $5n + 4n^2 + 4n^2 + \sqrt{n}$ words: Larsson and Moffat [4]
  - $12n + O(\rho)$ 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 $\sigma^\prime$: the number of non-terminals produced by Re-Pair
$\epsilon$: a constant with $0 < \epsilon \leq 1$
$\rho$: the maximum number of bigrams at any time

Our Contribution

A naive in-place algorithm takes $O(n^3)$ time since it
- needs $O(n^2)$ time finding the most frequent bigram, and
- may create up to $n$ non-terminals.

We improve this in the word RAM model with
- $O(n^2) \cap O(n^2 \lg \log n \log \log n \log n)$ time and
- $n \lceil \log \max(n, \tau) \rceil$ bits of working space including the text space, where $\tau$ is the total number of terminals and non-terminals.
- $T$ can be restored with $O(|\Sigma|)$ 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 $O(n \lg |\Sigma|)$ 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 $O(n^2 \log d / d)$ time
  - using $2d \lceil \log(n^2 / 2) \rceil + O(|\Sigma|)$ bits.

Pseudo Code

```
1 \k \leftarrow 0, i \leftarrow 0, f_0 \leftarrow O(1), T_0 \leftarrow T
2 \text{while highest frequency of a bigram in } T \text{ is } 1 \text{ do}
3 \quad F \leftarrow \text{frequency table of Tool 2 with } d = f_k
4 \quad f_k \leftarrow \text{minimum frequency stored in } F
5 \quad \text{while } F \neq \emptyset \text{ do}
6 \quad \quad \text{bc} \leftarrow \text{most frequent bigram stored in } F
7 \quad \quad T_{i+1} \leftarrow T, \text{replace } bc. X_{i+1}
8 \quad \quad \text{create rule } X_{i+1} \rightarrow bc
9 \quad \quad i \leftarrow i + 1
10 \quad \quad \text{introduce the } (i+1)\text{-th turn}
11 \quad \quad \text{remove all bigrams with frequency } < f_j \text{ from } F
12 \quad \quad \text{add new bigrams to } F \text{ having } X_{i+1} \text{ as a character and a frequency } \geq f_k
13 \quad \quad f_k \leftarrow f_k + \text{gained frequency space during } k\text{-th round}
14 \quad \quad k \leftarrow k + 1
15 \quad \quad \text{introduce the } (k+1)\text{-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 $f_k$ 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 $O(n)$ 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 $f_k = \Omega(n^\gamma)$ (Line 11).

There are $O(|\Sigma|)$ rounds since we can maintain all bigrams in the $O(|\Sigma|)$-th round ($f_k = \Theta(n)$ for $k = \Theta(|\Sigma|)$).

Tool 2: Computing $F$ for $k$-th round costs $O(n^2 \log f_k / f_k)$ time with $d = f_k$.

Total Time: $O\left(n^2 \sum_{i=0}^{k-1} \frac{1}{f_i^n} \right) = O(n^2)$

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 | a | a | c | a | a | a | b | c | a | b | ab:5 | ca:5 | ... | aa:3 |

| 8 | c | X1 | a | a | c | X1 | c | X1 | a | a | c | a | a | X1 | c | X1 | c | c | X1:4 | aa:3 | | |

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 $f_0 = 3$.

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 $f_0$. Obtain space for $D$ by aligning the compressed text $T_0$. (Line 11).
Row 4: Scan the text and copy each character preceding an occurrence of $X_1$ in $T_0$ to $D_0$.
Row 5: Sort all characters in $D_0$ lexicographically.
Row 6: Insert new bigrams (consisting of a character of $D$ and $X_1$) whose frequencies are $\geq f_0$.
Row 7: Symmetric to Row 4: Copy each character succeeding an occurrence of $X_1$ in $T_0$ to $D_0$, 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 $O(|\Sigma|)$ bits in $O(|\Sigma| \lg n \log n)$ time (time for popcount), where $\tau$ is the total number of terminals and non-terminals.
- Each turn takes $O\left(n \log |\Sigma| \lg n / \log n\right)$ amortized time.
- Tool 2 can run in $O\left(n^2 \log |\Sigma| \log n / \log n\right)$ time.

Total Time: $O\left(n^2 \sum_{i=0}^{k-1} \frac{1}{f_i^n} \right) = O\left(n^2 \log |\Sigma| \log n / \log n\right)$

References