Re-Pair Achieves High-Order Entropy

Gonzalo Navarro, Luís Russo

Department of Computer Science, University of Chile, Chile
Department of Computer Science, University of Lisbon, Portugal

Abstract

Re-Pair is a dictionary-based compression method invented in 1999 by Larsson and Moffat [LM99, LM00]. Although its practical performance has been established through experiments, the method has resisted all attempts of formal analysis. We show that Re-Pair compresses a sequence of length \( n \) over an alphabet of size \( \sigma \) and \( k\)-th order entropy \( H_k \), to at most \( 2nH_k + o(n \log \sigma) \) bits, for any \( k = o(\log n) \).

Introduction

Re-Pair is a dictionary-based compression method invented in 1999 by Larsson and Moffat [LM99, LM00]. As shown by the authors, Re-Pair achieves competitive compression ratios (albeit there are compressors that perform better). However, no theoretical guarantees are given in the original papers, and the method has resisted all attempts of analysis over the years.

We present the first analysis of Re-Pair, proving that it achieves high-order compression. More precisely, given a text \( T \), let \( C = (c_0, \ldots, c_t) \) be the current sequence, mixing original symbols (terminals) and newly created symbols (nonterminals). Hence we call \( p \) the length of \( C \) (measured in symbols) and \( t \) the size of the dictionary (measured in entries, each formed by 2 symbols). In the beginning, \( C = T, p = n \) and \( t = 0 \). At each step, \( d \) grows by 1 and \( p \) decreases at least by 2. Hence \( d \) also signals the number of compression steps already executed.

Let \( d \) be the number of compression steps. We call \( \exp_d(C) \) the sequence of terminals that symbol \( c_i \) represents in \( T \) (\( \exp_d(c_i) = c_i \) if \( c_i \) is already terminal). Hence \( T = \exp_0(C) \), \( T = \exp_1(C) \), \( \exp_2(C) \), and \( \exp_3(C) \) at any step. Each \( \exp_d(c_i) \) is called a phrase and the partition is called a parsing of \( T \). We will denote a parsing of the string \( T[i..j] \) over an alphabet of size \( \sigma \) as \( \exp_d(T[i..j]) \).

We start with a text \( T[1..n] \) over an alphabet \( \sigma \leq n \). Re-Pair compression proceeds in a sequence of steps, each step creating a new dictionary entry. At an arbitrary step of the process, let \( T[i..j] \) be the current sequence, mixing original symbols (terminals) and newly created symbols (nonterminals). Hence \( p = n - d \). We show that \( \exp_d(C) \) has \( \leq 2 \log(n/d) \) bits to represent each symbol (by \( e \log \) we mean \( e \log \sigma \) in this paper).

We make heavy use of the following lemmas, proved by Kosaraju and Manimi.

**Theorem 1.** Let \( y_1 \ldots y_{t+1} \) denote a parsing of the string \( T[i..j] \) over an alphabet of size \( \sigma \), such that each phrase \( y_i \) appears at most \( h \) times. For any \( k \geq 0 \) we have

\[
\log(t) \leq nH_k(T) + \log(n/t) + k \log(1 + \log(t))
\]

We are now ready to state our main result.

**Theorem 2.** Let \( T[1..n] \) be a text over an alphabet of size \( \sigma \) and having \( k\)-th order entropy \( H_k \). Then, compression algorithm Re-Pair achieves a representation using at most \( 2nH_k + o(n \log \sigma) \) bits for any \( k = o(\log n) \) (which implies \( \log \sigma = o(\log n) \) unless \( k = o(n) \).

Proof We study \( p + 2d \) when the most frequent pair occurs at most \( h = \log^2 \) times. This is achieved in at most \( n/h + 1 \) stages, hence \( \log(n/h) + 1 \) steps, hence \( \log(n/h) \leq 1 \) bits, hence \( \log(n/h) = o(1) \). Consider the parsing \( \exp_d(c_i) \), \( \exp_d(c_i) \), \( \ldots \) of \( \sigma = p/2 \) strings that do not appear more than \( h \) times and apply lemma 4. Further, \( \delta \) increases, especially when \( (\log n/h) > h \), which is the bound for \( \log n/h \).

Although it is not the case, imagine we are at the last step in our example. In that case part of the explanation in the previous paragraph can represented schematically as:

\[
\begin{align*}
E & \rightarrow a \\
B & \rightarrow a A \\
C & \rightarrow A d \\
D & \rightarrow B c \\
E & \rightarrow a D \\
\end{align*}
\]


References


