Re-Pair Achieves High-Order Entropy [*]

Gonzalo Navarro
Dept. of Computer Science
University of Chile
Chile
gnavarro@dcc.uchile.cl

Luís Russo
INESC-ID
Tech. University of Lisbon
Portugal
lmsrusso@gmail.com

Abstract

Re-Pair is a dictionary-based compression method invented in 1999 by Larsson and Moffat. Although its practical performance has been established through experiments, the method has resisted all attempts of formal analysis. In this paper we show that Re-Pair compresses a sequence $T[1,n]$ over an alphabet of size σ and k-th order entropy H_k, to at most $2H_k + o(n \log \sigma)$ bits, for any $k = o(\log_\sigma n)$.

1 Introduction

Re-Pair is a dictionary-based compression method invented in 1999 by Larsson and Moffat [11, 12]. It has been successfully used in different scenarios related to word-based text compression [17], searching compressed text [8], compression of Web graphs [4], and suffix array compression [7], to give some examples.

Re-Pair can be tought of as a grammar-based compressor [9] as it discovers a grammar that generates the text. Re-Pair is a remarkably simple member of this family, with a very simple heuristic rule to build the grammar. Moreover, it decompresses very fast, and this is an important part of its appeal. As shown by the authors, Re-Pair achieves competitive compression ratios (albeit there are compressors that perform better). However, no theoretical guarantee is given in the original papers, and the method has resisted all attempts of analysis over the years.

In this paper we present the first analysis of Re-Pair, finally proving that it achieves high-order compression. More precisely, given a text $T[1,n]$ over an alphabet of size σ, Re-Pair achieves at most $2nH_k(T) + o(n \log \sigma)$ bits of space, where H_k is the k-th order entropy of T, taken either in the classical information-theory sense over an ergodic source [5], or in the sense of empirical entropy [13]. Note that, unless $k = 0$, the condition on k implies that the alphabet must be small, $\log \sigma = o(\log n)$.

Previous analyses we are aware of only proved very coarse properties of its optimality as a grammar-based compressor [3].

[*]Funded in part by a grant from Yahoo! Research Latin America.

2 Re-Pair

Re-Pair operates by repeatedly finding the most frequent pair of symbols in a sequence and replacing it with a new symbol, until every pair appears only once. More precisely:

1. It identifies the most frequent pair ab in T
2. It adds a rule $R \rightarrow ab$ to a dictionary, where R is a new symbol that does not appear in T.
3. It replaces every occurrence of ab in T by symbol R (as far as possible, e.g. one cannot replace both occurrences of aa in aaa).
4. It iterates until every pair in T appears once.

Re-Pair can be implemented in linear time and space [12]. There are several techniques to achieve further compression, for example by representing the rules in compressed form [12, 7], but those are not relevant for this paper.

3 Analyzing Re-Pair

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 us call $C = c_1c_2 \ldots c_p$ the current sequence, mixing original symbols (terminals) and newly created symbols (nonterminals). Hence we call p the length of C (measured in symbols) and d the size of the dictionary (measured in entries, each formed by 2 symbols). In the beginning, $C = T$, $p = n$ and $d = 0$. 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 us call $expand(c_i)$ the sequence of terminals that symbol c_i represents in T ($expand(c_i) = c_i$ if c_i is already terminal). Hence $T = expand(c_1) \cdot expand(c_2) \ldots expand(c_p)$ at any stage. Each $expand(p_i)$ is called a *phrase* and the partition is called a *parsing* of T. We will also denote $expand(XY) = expand(X) \cdot expand(Y)$.

The number of different symbols in the sequence after d steps is at most $\sigma + d$, and thus we would need $\lceil \log(\sigma + d) \rceil$ bits to represent each symbol (by log we mean $\log_2$ in this paper). For simplicity we will make the pessimistic assumption that each dictionary cell, as well as each symbol in the compressed text, will be stored using $\lceil \log n \rceil$ bits. This pessimistic, as shown by the next lemma.

Lemma 1 *At any step of the process, it holds $\sigma + d \leq n$.*

Proof. We show that at least d repeated symbols are identified in T when forming a dictionary of size d. If we form a rule $A \rightarrow ab$ (a and b terminals) we have actually identified (at least) two repeated symbols in T. Let us count only the repetition of b. Instead of counting the repetition of a, we will count those of nonterminal A just formed, as if it were a terminal. This way, every formed rule (no matter whether it is formed by two terminals or nonterminals) removes at least one repetition. Hence if we form d rules there are at least d repeated symbols in T and hence $\sigma \leq n - d$. $\square$

We will speak of "integers" to denote symbols stored using $\lceil \log n \rceil$ bits. The size of the compressed data, at any step of the process, is therefore upper bounded by $(p + 2d)\lceil \log n \rceil$ bits (recall that each dictionary entry occupies 2 integers). The following lemma is rather obvious but important.

Lemma 2 *The size of the compressed data, $p + 2d$ integers, does not increase along the process.*

Proof. We recall that Re-Pair chooses the most frequent pair of symbols at each iteration. Let b be the highest frequency at some particular step. This means that a new entry (using 2 integers) will be added to the dictionary, while the text will be shortened by b integers. The process continues as long as $b \geq 2$, so $+2 - b \leq 0$ and hence $p + 2d$ never increases from one step to the next. $\square$

It is also clear that b cannot increase along the process, as shown in the next lemma.

Lemma 3 *The frequency of the most repeated pair does not increase from one step to the next.*

Proof. Assume the most frequent pair formed the rule $X \rightarrow AB$, and after the replacement, the most frequent pair CD has more occurrences than X. If CD does not contain X, then it existed before X was created and hence AB was not the most frequent pair. If it contains X, it cannot be more frequent than X alone. $\square$

We will also make use of the following lemma.

Lemma 4 *Let XY and ZW be two different consecutive pairs of (terminal or nonterminal) symbols in C at any stage of the compression. Then $expand(XY) \neq expand(ZW)$.*

Proof. We prove, more generally, that XY and ZW cannot expand to the same sequence of terminals and nonterminals, that is, they cannot become equal at any point of the expansion process. Assume XY and ZW expand to a phrase of the same length ℓ, otherwise the lemma follows trivially. We proceed by induction on ℓ. The base case is $\ell = 2$, that is, zero expansion steps took place, and then the lemma follows trivially. Let us now assume that $\ell > 2$ and both XY and ZW expand to phrase α, so that, along the compression process, one occurrence of α is parsed into XY and another occurrence is parsed into ZW. Let AB be the first pair of symbols within α that is replaced by the compression process. Since the replacement ocurrs in both occurrences of α in the text, it produces a shorter phrase α' in both places. By the inductive hypothesis, α' cannot be parsed into XY in one occurrence and into

ZW in another occurrence. □

Finally, we make heavy use of the following theorem, proved by Kosaraju and Manzini.

Theorem 1 *[10] Let $y_1 \dots y_t$ denote a parsing of the string $T[1,n]$ over an alphabet of size σ, such that each phrase y_i appears at most b times. For any $k \geq 0$ we have*

$$t \log t \quad \leq \quad nH_k(T) \ + \ t\log(n/t) \ + \ t \log b \ + \ \Theta(t(1 + k \log \sigma))$$

Proof. *See Lemma 2.3 in [10]. The latter term appears as $\Theta(kt + t)$ in there, but in their appendix (where the lemma is proved) the right term becomes apparent. The results holds either if we interpret H_k as the empirical entropy [13] or as the classical entropy of a stationary ergodic sequence.* □

We are now ready to state our main result.

Theorem 2 *Let $T[1,n]$ be a text over an alphabet of size σ and having k-th order (classical or empirical) entropy $H_k(T)$. Then, compression algorithm Re-Pair achieves a representation using at most $2nH_k(T) + o(n \log \sigma)$ bits for any $k = o(\log_\sigma n)$ (which implies $\log \sigma = o(\log n)$ unless $k = 0$).*

Proof. Since the highest frequency is nonincreasing (Lemma 3), let us consider the process at the step d where we have replaced every pair with frequency more than $b = \log^2 n$, and now the most frequent pair has frequency at most b. We will analyze the size of the sequence plus dictionary, $(p + 2d)\lceil \log n \rceil$ bits, at this point, knowing that the size cannot grow in further steps of the process (Lemma 2).

Since every replacement up to this point chose a pair appearing more than b times, it follows that the text has decreased by at least $b + 1$ symbols at each step, and hence at most $n/(b+1)$ steps might have been carried out up to now. Therefore, the size of the dictionary at this step is limited by $d < n/b$. Hence $2d\lceil \log n \rceil < 2(n/b)(\log(n) + 1) = O(n/\log n) = o(n)$ is well within the sublinear part of the space occupancy the theorem promises.

Let us now focus on the remaining term, $p\lceil \log n \rceil$. At this step, no consecutive pair of symbols in the current sequence C appears more than b times. In particular, if we choose c_1c_2, c_3c_4, ..., no pair will appear more than b times. Let us then consider the following parsing: $T = \mathit{expand}(c_1c_2) \cdot \mathit{expand}(c_3c_4) \dots \mathit{expand}(c_{2i-1}c_{2i}) \dots$. Because of Lemma 4, no phrase in this expansion appears more than b times either. The parsing contains $t = \lceil p/2 \rceil \geq p/2$ phrases (if p is odd, the last phrase will be just $\mathit{expand}(c_p)$, and it cannot be equal to any other phrase: either it is of length 1, or there is a rule $c_p \rightarrow XY$ to which we can apply Lemma 4 against any other $c_{2i-1}c_{2i}$).

We are almost ready to apply Theorem 1, yet we have $p\lceil \log n \rceil \leq 2t\lceil \log n \rceil$ and our bound is on $t \log t$. To move forward, we must focus on the interesting case of sufficiently large t. If $t\lceil \log n \rceil \leq n/\log n$, then $p\lceil \log n \rceil \leq 2n/\log n = o(n)$, in which case the theorem is trivially true (as all the space falls within the $o(n \log \sigma)$ term).

The interesting case is $t\lceil \log n \rceil > n/\log n$, and therefore $\log n \geq \log t \geq \log n - O(\log\log n)$. Hence $t\lceil \log n\rceil \leq t\log n + O(t) \leq t\log t + O(t\log\log n)$. Also, $\log(n/t) < \log(\log^2 n) = O(\log\log n)$. Finally, since $b = \log^2 n$ we also have $t\log b = O(t\log\log n)$, so we obtain the following from Theorem 1:

$$\begin{aligned} t\lceil \log n \rceil &\leq t\log t + O(t\log\log n) \\ &\leq nH_k(T) + O(t\log\log n) + O(t(1+k\log\sigma)) \\ &= nH_k(T) + O(t(\log\log n + k\log\sigma)) \end{aligned}$$

and therefore

$$\begin{aligned} t &\leq \frac{nHk(T)}{\lceil \log n\rceil - O(\log\log n + k\log\sigma)} \\ t\lceil \log n \rceil &\leq nH_k(T)\left(1 + O\left(\frac{\log\log n + k\log\sigma}{\log n}\right)\right) \\ &\leq nH_k(T) + O\left(\frac{n(\log\log n + k\log\sigma)}{\log_\sigma n}\right) \end{aligned}$$

which is $nH_k(T) + o(n\log\sigma)$ provided $k = o(\log_\sigma n)$ (and hence either $k = 0$ or $\log\sigma = o(\log n)$).

Thus $(p + 2d)\lceil \log n\rceil \leq 2t\lceil \log n\rceil + 2(n/b)\lceil \log n\rceil = 2nH_k(T) + o(n\log\sigma) + o(n)$, and we have proved our theorem. □

4 Conclusions

We have presented the first analysis of Re-Pair, which shows that it achieves k-th order entropy with a penalty factor of 2. Apart from upper-bounding the performance of the classic Re-Pair compression algorithm, our analysis gives some intriguing clues. We note that we have used $b = \log^2 n$ to prove our theorem, yet any $b = \log^t n$ for any constant $t > 1$ would do (more precisely, the limits are $b = \omega(\log n)$ and $\log b = o(\log n)$). The fact that a small b achieves our same upper bound shows that weaker versions of Re-Pair, which instead of choosing the best (i.e., most frequent) pair, choose a "good enough" pair, are likely to perform well (e.g. [4]). On the other hand, the fact that such a large b also achieves the same upper bound explains in part the success of techniques that limit the size of the dictionary (by preempting the compression process) [7].

We point out, however, some limitations of our analysis which deserve further research:

1. The factor 2 could explain why Re-Pair does not perform as well as other k-th order compressors such as PPM. However, we are not sure this upper bound is tight. To be sure one should find a class of sequences which achieve the 2 factor over a sufficiently small alphabet (ideally binary).

2. The analysis holds for sufficiently small k compared to σ, more precisely for $k = o(\log_\sigma n)$. In particular, unless $\log \sigma = o(\log n)$, the analysis holds only for $k = 0$. The main tool used in our proof [10] seems not to give useful results for $\log \sigma = \Theta(\log n)$. The fact that Re-Pair performs so well on large alphabets [17, 7, 4], therefore, remains largely unexplained. Still, there are some limits to what can be achieved for these large alphabets: In [6] it is shown that, as soon as $\sigma^{k+1/c-3\varepsilon} = \Omega(n)$, it is not possible to represent T using $cnH_k(T) + \varepsilon n \log \sigma$ bits of space. It is likely that k-th order entropy is not a good model to analyze the behavior of compression algorithms in those cases. Still, it should be possible to obtain results related to $H_k(T)$ for $k \leq \alpha \log_\sigma n$, for constant $0 < \alpha < 1$, as achieved using the Burrows-Wheeler Transform, for example [2, 13, 14].

3. It could be possible to extend our analysis to other compression methods that operate by textual substitution, see for example [1, 16]. Some of those, however, are on-line (e.g. Sequitur [15]), in the sense that they choose the phrases to replace as they process the text. This does not match well with our analysis, which assumes that the most frequent pair overall is chosen. Our usage of parameter b, however, suggests that on-line variants that replace phrases that appear at least b times are likely to be tractable with our analysis technique.

References

[1] A. Apostolico and S. Lonardi. Off-line compression by greedy textual substitution. *Proc. IEEE*, 88(11):1733–1744, 2000.

[2] M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[3] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[4] F. Claude and G. Navarro. A fast and compact Web graph representation. In *Proc. SPIRE*, LNCS 4726, pages 105–116, 2007.

[5] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley, 1991.

[6] T. Gagie. Large alphabets and incompressibility. *Information Processing Letters*, 99(6):246–251, 2006.

[7] R. González and G. Navarro. Compressed text indexes with fast locate. In *Proc. CPM*, LNCS 4580, pages 216–227, 2007.

[8] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage systems: a unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003.

[9] J. Kieffer and E. Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46:737–754, 2000.

[10] R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM Journal on Computing*, 29(3):893–911, 1999.

[11] J. Larsson and A. Moffat. Off-line dictionary-based compression. In *Proc. DCC*, pages 296–305, 1999.

[12] J. Larsson and A. Moffat. Off-line dictionary-based compression. *Proc. IEEE*, 88(11):1722–1732, 2000.

[13] G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.

[14] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.

[15] C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

[16] C. Nevill-Manning and I. Witten. On-line and off-line heuristics for inferring hierarchies of repetitions in sequences. *Proc. IEEE*, 88(11):1745–1755, 2000.

[17] R. Wan. *Browsing and Searching Compressed Documents*. PhD thesis, Dept. of Computer Science and Software Engineering, University of Melbourne, 2003. `http://eprints.unimelb.edu.au/archive/00000871`.