entropy

MDPI

Article

Prediction and Evaluation of Zero Order Entropy Changes in Grammar-Based Codes

Michal Vasinek * and Jan Platos

Department of Computer Science, FEECS, VSB-Technical University of Ostrava, 17. listopadu 15/12172, Ostrava 708 33, Czech Republic; jan.platos@vsb.cz

* Correspondence: michal.vasinek@vsb.cz; Tel.: +420-597-323-971

Academic Editor: Raúl Alcaraz Martínez

Received: 30 January 2017; Accepted: 10 May 2017; Published: 13 May 2017

Abstract: The change of zero order entropy is studied over different strategies of grammar production rule selection. The two major rules are distinguished: transformations leaving the message size intact and substitution functions changing the message size. Relations for zero order entropy changes were derived for both cases and conditions under which the entropy decreases were described. In this article, several different greedy strategies reducing zero order entropy, as well as message sizes are summarized, and the new strategy MinEnt is proposed. The resulting evolution of the zero order entropy is compared with a strategy of selecting the most frequent digram used in the Re-Pair algorithm.

Keywords: data compression; grammars; entropy; transformations; context; Re-Pair

1. Introduction

Entropy is a key concept in the measurement of the amount of information in information theory [1]. From the data compression perspective, this amount of information represents the lower limit of the achievable compression of some information source. Due to the well-known work by Shannon [2], we know that using less bits than the amount given by entropy to represent a particular message or process would necessarily lead to the loss of some information and as a consequence our inability to properly recover the former structure of the message.

This work is focused on the study of entropy in data compression, and therefore, our discussion will be restricted to only finite messages. These finite messages are formed by symbols, and in this perspective, the entropy can be understood as the lowest number of bits needed on average to uniquely represent each symbol in a message. There are messages for which the evaluation of entropy can be a very hard task, and so, we are often forced to satisfy ourselves with some approximation of entropy.

The simplest approximation is the one based on the probability distribution of symbols in a particular message. In this case, the symbols are viewed as independent entities, and their mutual relationships are not taken into account. A better approximation of entropy is based on the conditional probabilities when we also take into account how symbols follow each other. We can also approximate entropy by computing bits per byte ratio of message encoded by state of the art data compression algorithms.

In this article, we study entropy at the level of independent symbols. This approximation of entropy is often called zero order entropy. There are two major data compression algorithms in use that compress messages almost to the rate given by zero order entropy: Huffman [3] and arithmetic [4] coding. The zero order entropy can be computed using the Shannon equation:

$$H(X) = -\sum_{x \in \Sigma} p(x) \log p(x) \tag{1}$$

where X stands for a random variable representing the probability distribution of symbols in the input message m and $p(x)$ is the probability of symbol x from alphabet Σ. The expected length of the code for the particular symbol x is given by $-\log p(x)$. If the expected length of the code is multiplied by its probability, we obtain the average number of bits needed to represent any symbol $x \in \Sigma$. The size of the message using the expected lengths of codes is given as a product of the length of the message $|m|$ measured as a number of symbols and zero order entropy:

$$|m|^H = |m|H(X) \tag{2}$$

When we refer to the term entropic size of the message, we always mean the quantity given by Equation (2), and it will be denoted using a superscript as $|m|^H$. We study how the entropic size of the message evolves when all occurrences of some m-gram are substituted for some other n-gram and vice versa. We study two such substitutions: transformations and compression functions. Transformations replace n-grams of the same length. Transformation leaves the message size intact, but since the probability of symbols changes, the value of zero order entropy also has to change. Compression functions replace m-grams for n-grams, where $m > n$, and leave the message size smaller, but the change in zero order entropy also occurs.

The main idea behind the concept of transformations is the following: consider Huffman coding; more probable symbols are encoded by shorter or at least by the same length prefix codes than the lower probability ones, if the symbol β is more probable than the symbol γ, but in the context of some symbol α, γ is more frequent than β, then if these symbols following α are exchanged, the longer codes used for encoding γ will instead be encoded by shorter codes representing the encoding of β. Under this assumption, it is possible to pre-process data so that the frequency of more frequent symbols increases and the frequency of less frequent symbols decreases.

1.1. Notation and Terminology

- The alphabet of the input message m of the size $|m|$ is denoted by Σ and its size by $|\Sigma|$.
- Greek symbols are used to denote variables representing symbols in the input message. For instance, suppose two digrams $\alpha\alpha$, $\alpha\beta$ and the alphabet $\Sigma = \{0, 1\}$. Then, $\alpha\alpha \in \{00, 11\}$ and $\alpha\beta \in \{00, 01, 10, 11\}$.
- When we refer to the term entropy, we always mean Shannon's entropy defined by Equation (1).
- All logarithms are to base two.
- Any quantity Q_i with a subscript $i \in N$ denotes consecutive states of the quantity between substitutions. For instance, a quantity Q_0 is a value of the quantity before any substitution is applied, and Q_1 is a value of the quantity after some substitution is applied.

1.2. Order of Context and Entropy

1.2.1. Zero Order Context

When all symbols are interpreted as independent individual entities and no predecessors are taken into consideration, such a case is called zero order context. Zero order entropy is then computed as Shannon's entropy of a random variable given by the probabilities of symbols in the input message.

1.2.2. N-th Order Context

In a case where the probability distribution of symbols following a particular fixed length prefix w is taken into consideration, then if the length of the prefix is N, then the order of context is N, and the N-th order entropy is computed as Shannon's entropy of the conditional distribution of symbols following all different prefixes w_i.

2. Previous Work

The class of algorithms dealing with exchanges of different n, m-grams are called grammar-based algorithms. Their purpose is to provide a set of production rules inferring the content of the message. Using the Chomsky hierarchy, we identify two classes of formal grammars used in data compression: context-free grammars (CFG) and context-sensitive grammars (CSG). Context transformations presented in Section 3 belong to the CSG class; meanwhile, compression functions belong to the CFG class. The problem of the search for the most compact context-free grammar representation of a message is NP-hard, unless $P = NP$ [5]. Instead of searching for the optimal solution, many heuristic and greedy algorithms were proposed.

CFGs for data compression were first discussed by Nevill-Manning [6] followed by the proposal of the SEQUITUR algorithm [7]. SEQUITUR reads the sentence in the left-right manner so that each repeated pattern is transformed into a grammar rule. The grammar is utilized in such a way that the two properties are fulfilled: a digram uniqueness (no pair of adjacent symbols appear more than once in the grammar and a rule utility); every rule is used more than once. Kiefer and Yang [8] were the first who addressed data compression using CFGs from the information theoretic perspective; they showed that the LZ78 [9] algorithm can be interpreted as a CFG and that the proposed BISECTION algorithm forms a grammar-based universal lossless source code. BISECTION repeatedly halves the initial message into unique phrases of length 2^k, where k is the integer.

In the work of Yang and He [10], the context-dependent grammars (CDG) for data compression were introduced. In CSG, the context is present in both sides of production rules; meanwhile in CDG, the context is defined only on the left side of the production rule.

One of the first concepts in greedy grammar-based codes was the byte pair encoding (BPE) [11]. The BPE algorithm selects the most frequent digram and replaces it with some unused symbol. The main weakness of this approach is that the algorithm is limited to an alphabet consisting only of byte values. The concept of byte pair encoding was later revised, and the limitation on the alphabet size used was generalized independently by Nakamura and Murashima [12] and by Larsson and Moffat [13]; the resulting approach is called Re-Pair [13]. Re-Pair stands for recursive pairing, and it is a very active field of research [14,15]. It iteratively replaces the most frequent digrams with unused symbols until there is no digram that occurs more than once.

Unlike BPE that codes digrams using only byte values, Re-Pair expects that the symbols of the final message will be encoded using some entropy coding algorithm. Approaches derived from Re-Pair are usually greedy, since each iteration of the algorithm is dependent on a successful search of the extremal value of some statistical quantity related to the input message. The study of the Re-Pair from the perspective of ergodic sources is discussed in [16,17]. Neither BPE nor Re-Pair compress the message into the least possible size, but they rather form a trade-off between message and dictionary sizes. Re-Pair-like algorithms are off-line, in the sense that they need more than one pass through the input message; meanwhile, SEQUITUR incrementally builds the grammar in a single pass. The Re-Pair algorithm is an algorithm with $O(n)$ time complexity; it is easy to implement using linked lists and a priority queue. Further, it was shown in [18] that it can compress an input message of length n over an alphabet of size $|\Sigma|$ into at most $2H_k + o(n \log |\Sigma|)$ bits, where H_k is k-th order entropy.

Our recent studies were focused on a special class of grammar transforms that leave the message size intact [19,20]. In the present paper, the class of grammar transformations is extended with a novel concept of higher order context transformation [21]. We shall provide examples of transformations and the evaluation of entropy resp. entropic size reduction to the class of grammar compression algorithms, and we compare the evolution of entropy, entropic size and the resulting number of dictionary entries for Re-Pair and our version of Re-Pair, called MinEnt, which is based on the selection of the pair of symbols reducing the entropic size of the message the most. Re-Pair finds application in areas such as searching in compressed data [22], compression of suffix arrays [23] or compression of inverted indexes [24], to name a few. These areas are also natural application fields for MinEnt. From the

perspective of the number of passes through the message, the approaches discussed in this paper belong to off-line algorithms.

3. Transformations and Compression Algorithms

In this section, we will describe and evaluate several invertible transformations T and substitution functions F so that for any two consecutive states of the message, m_0 and m_1, before and after application of T or F, the following relation holds:

$$|m_1|^H < |m_0|^H \tag{3}$$

The measure of the size of the message by the entropic size of the message is preferred, since using the arithmetic coding, one can achieve a compression rate very close to the zero order entropy, and so, the size $|m|^H$ is in theory accessible. Further, it allows the comparison of two distinct substitutions when their resulting sizes measured by the number of symbols are equal. The derivation of equations for the computation of $|m_1|^H$, resp. $\Delta|m|^H = |m_0|^H - |m_1|^H$, are provided in Section 4.

3.1. Transformations

Consider transformation, where we replace all occurrences of some symbol β for some symbol γ and vice versa; such a transformation is called a symmetry transformation, because it does not modify any measurable quantities related to the amount of information. The information content is changed when the replacement is taken in the context of an other symbol α. Such a transformation corresponds to the exchange of all digrams $\alpha\beta$ for $\alpha\gamma$ and vice versa. In this section, several different forms of transformation are distinguished and briefly described. Some properties of transformations and their proofs can be found in Appendix A.

3.1.1. Context Transformation

The concept of context transformations was first proposed in [25], and the results were presented in [19]. It is the simplest transformation that assumes a pair of digrams beginning with the same symbol when one of the digrams is initially missing in the input message.

Definition 1. *Context transformation (CT) is a mapping* $CT(\alpha\beta \to \alpha\gamma, w) : \Sigma^n \to \Sigma^n$ *that replaces all digrams* $\alpha\beta$ *for* $\alpha\gamma$*, where* $p(\alpha, \gamma) = 0$ *and* $\beta \neq \gamma$*.* Σ *is the alphabet of the input message* w*, and* n *is the length of* w*.*

Let $CT_{\leftarrow}$ be the context transformation applied from the end of the message to the beginning and $CT_{\rightarrow}$ in the opposite direction. The context transformation $CT_{\rightarrow}$ is an inverse transformation of $CT_{\leftarrow}$. The proof of this property with an explanation for why it is the only pair of the function and its inverse is left to Appendix A. The application of two consecutive context transformations and their inverse functions is presented in the following example:

Example 1.

$$\begin{aligned}
&abcdabacd|CT_{\leftarrow}(ab \to aa)\\
&aacdaaacd|CT_{\leftarrow}(cd \to cc)\\
&aaccaaacc|CT_{\rightarrow}(cc \to cd)\\
&aacdaaacd|CT_{\rightarrow}(aa \to ab)\\
&abcdabacd|
\end{aligned}$$

3.1.2. Generalized Context Transformation

Context transformations were restricted in cases where one of the digrams was missing in the input message. This restriction is removed by the introduction of the generalized context transformations first proposed in [20].

Definition 2. *Generalized context transformation (GCT) is a mapping* $GCT(\alpha\beta \leftrightarrow \alpha\gamma, w) : \Sigma^n \rightarrow \Sigma^n$ *that exchanges all occurrences of a digram* $\alpha\beta$ *by a digram* $\alpha\gamma$ *and vice versa.* Σ *is the alphabet of the input message* w*, and* n *is the length of* w*.*

Example 2.

$$aabcabab|GCT_{\leftarrow}(ab \leftrightarrow aa)$$
$$abacaaaa|GCT_{\rightarrow}(aa \leftrightarrow ab)$$
$$aabcabab$$

Meanwhile, both transformations CT and GCT swap occurrences of two different digrams beginning with the same symbol; they differ in the way they are applied and how the inverse transformation is formed. GCT can be applied in both directions, and the inverse transformation GCT^{-1} is always applied in the opposite direction, than the forward transformation direction. The algorithm based on the CT and GCT works as follows:

1. Find and apply transformation T so that the change of the entropic size $\Delta|m|^H = |m_0|^H - |m_1|^H$ is maximal.
2. Repeat Step 1 until no transformation can decrease the entropic size of the message.

It is also possible to define a transformation and its inverse so that all symbols constituting replaced pairs differ, for instance $ab \leftrightarrow cd$; such a transformation is called generic transformation GT. In this article, we have not proposed algorithms based on GT, but because the set of all generalized context transformations is a subset of a set of generic transformations, the proof of the inverse transformation existence is the same for both GCT and GT. The reader can find the proof in Appendix A.

3.1.3. Higher Order Context Transformation

Every time we apply any generalized context transformation GCT, we acquire knowledge about the positions of two distinct digrams in the message. We can either discard this knowledge or we can try to build on it. In the following definition, we define a transformation that is applied over positions where some other transformation was applied before:

Definition 3. *Let* $P(w, m)$ *be a set of positions of the first symbol following the sub-message* w *in the message* m *and* $w[i] \neq w[0]$*,* $i > 0$*. If* $\beta, \gamma \neq w[0]$*, then the higher order context transformation (HOCT) is a mapping* $HOCT(w\beta \leftrightarrow w\gamma, m, P(w, m)) : \Sigma^n \rightarrow \Sigma^n$ *that exchanges all sub-messages* $w\beta$ *for sub-messages* $w\gamma$ *and vice versa.*

The restriction that the sub-message w has to satisfy is $w[0] \neq w[i]$, where $i > 0$ is closely related to the existence of the inverse transformation to $HOCT$. The properties related to the $HOCT$ and their proofs are left to Appendix A.

Let $O = |w|$ be the size of the sub-message w from Definition 3, then O is an order of $HOCT$. Any $GCT(\alpha\beta \leftrightarrow \alpha\gamma)$ is then the first order $HOCT(\alpha\beta \leftrightarrow \alpha\gamma, m, P(\alpha, m))$. Given that we just before applied some transformation $m_1 = HOCT_1(w\beta \leftrightarrow w\gamma, m, P(w, m))$, we can decide to collect the positions of either $w_1 = w\beta$ or $w_2 = w\gamma$, collect the distribution of symbols in $P(w_i, m)$ and apply another $HOCT(w_i\rho \leftrightarrow w_i\varphi, m_1, P(w_i, m))$. In this sense, $HOCT$ is not used only to interchange different

sub-messages, but it also allows one to proceed with some other transformation *HOCT* of a higher order. The application of two consecutive *HOCT* transformations is presented in the following example:

Example 3.

$$\begin{aligned} &abcdabcd|HOCT(ab \leftrightarrow ad, P(a,m) = \{1,5\}) \\ &adcdadcd|HOCT(adc \leftrightarrow add, P(ad,m) = \{2,6\})) \\ &adddaddd| \end{aligned}$$

The *HOCT* transformation is a recursive application of *GCT* in the context of some prefix w. The steps of the algorithm are outlined as follows:

1. Find and apply $HOCT(\alpha\beta \leftrightarrow \alpha\gamma)$ over the set of positions $P(\alpha)$, so that the change of entropic size $\Delta|m|^H = |m_0|^H - |m_1|^H$ is maximal and $\Delta|m|^H > Lim$.
2. If the frequency of $\alpha\beta$ resp. $\alpha\gamma$ is larger than one, then repeat Step 1 over the set of positions $P(\alpha\beta)$ resp. $P(\alpha\gamma)$, i.e., positions where *HOCT* from Step 1 was applied; otherwise, repeat Step 1 over positions $P(\alpha)$ or return if no more *HOCT* passes the entropic size reduction conditions.

The algorithm above is iteratively called for symbols sorted from the most frequent one to the least frequent one. The *Lim* variable can be used to restrict transformations whose entropic size reduction is too small, so they cannot be efficiently stored in the dictionary.

3.2. Compression Functions

In the preceding section, we described three types of transformations that leave message size intact. In this section, we will focus on a description of two approaches in the replacement of digrams for a new symbol. First, we describe basic principles of the well-known algorithm Re-Pair, and then, we will propose a modification of Re-Pair called MinEnt.

3.2.1. Re-Pair

The main idea behind the Re-Pair algorithm is to repeatedly find the most frequent digram and replace all of its occurrences with a new symbol that is not yet present in the message. The algorithm can be described in the following steps:

1. Select the most frequent digram $\alpha\beta$ in message m.
2. Replace all occurrences of $\alpha\beta$ for new symbol γ.
3. Repeat Steps 1 and 2 until every digram appears only once.

In Step 2 of the algorithm, the pair $\alpha\beta$ together with a new symbol γ are stored in a dictionary. The implementation details of the Re-Pair algorithm are left to Section 3.2.2 regarding the proposed MinEnt algorithm.

3.2.2. MinEnt

The MinEnt algorithm proposed in this article is derived from the Re-Pair algorithm. The main difference is in Step 1, where instead of the selection of the most frequent digram, we select a digram that minimizes $|m_1|^H$ from Equation (3):

1. Select digram $\alpha\beta$ in message m_0 so that the change of entropic size $\Delta|m|^H = |m_0|^H - |m_1|^H$ is maximal.
2. Replace all occurrences of $\alpha\beta$ for new symbol γ.
3. Repeat Steps 1 and 2 until every digram appears only once.

More precisely, let $m_1 = MinEnt(m_0, \alpha\beta \rightarrow \gamma)$ be the application of Step 1 and Step 2 of the MinEnt algorithm, then digram $\alpha\beta$ fulfills:

$$\arg\min_{\alpha,\beta\in\Sigma_0} |MinEnt(m_0, \alpha\beta \rightarrow \gamma)|^H \tag{4}$$

where Σ_0 is the alphabet of the message m_0. To demonstrate the difference between Re-Pair and MinEnt, consider the following example:

Example 4.

$$m_0 = aababcdcdb$$

The entropic size of m_0 *is* $|m_0|^H = 19.71$ *bits. There are two non-overlapping digrams that occur twice: ab and cd.*

$$(m_0, ab \rightarrow e) = aeecdcdb$$

$$(m_0, cd \rightarrow e) = aababeeb$$

Based on the Re-Pair algorithm, we do not know which digram should be preferred, because both have the same frequency. In the MinEnt case, we can compute $|m_1|^H$ for both cases, yielding $|m_1|^H_{ab} = 18$ bits and $|m_1|^H_{cd} = 12.49$ bits, and so, the replacement $cd \rightarrow e$ will be the preferred one.

The MinEnt and the Re-Pair strategies of digram selection are evaluated using the algorithm described in [13]. In the initialization phase of the algorithm, the input file is transformed into the linked list, and each input byte is converted into the unsigned integer value. In the next step, the linked list is scanned, and the frequencies and positions of all digrams are recorded. Frequencies of digrams, resp. the change of the entropic size of the message measured in bytes, are used as indices for the priority queue. The size of the queue is limited to the maximal frequency, resp. in the case of the MinEnt algorithm, the maximum entropic size decrease.

The algorithm iteratively selects the digram with the highest priority, replaces all occurrences of the digram with the newly-introduced symbol, decrements counts of neighboring digrams and increments counts of newly-introduced digrams. In the case of the MinEnt algorithm, we have to recompute the change of the entropic size of all digrams in the priority queue. We restrict the number of recomputed changes of the entropic size to the top 20 digrams with the highest priority, so that the time complexity of this additional step remains $O(1)$. Both algorithms are accomplished in $O(n)$ expected time; see [13] for details. The memory consumption is larger in the MinEnt case, because each digram has to be assigned with the additional quantity: the value of the change of the entropic size of the message.

3.3. Discussion of the Transformation and Compression Function Selection Strategies

To demonstrate the behavior of aforementioned algorithms, we proposed strategies for the selection of transformations and compression algorithms. We compared the evolution of the entropy of the alphabet, the entropic size of the message and the final size of the message given as the sum of the entropic size of the message and the upper boundary on the size of the dictionary (Section 3.3.1). The following strategies are being compared:

- *GCT*: selection of the generalized context transformation so that the decrease of entropy is maximal.
- *HOCT*: selection of the higher order context transformation so that the decrease of entropy is maximal in the context of prefix w.
- Re-Pair: selection of the most frequent digram and its replacement with an unused symbol.
- MinEnt: selection of the most entropic size reducing digram and its replacement with an unused symbol.

3.3.1. The Upper Boundary on the Dictionary Entry Size

All transformations and compression functions are usually stored as an entry in a dictionary. To be able to compare the effectiveness of transformations, we selected the worst case entropy of each symbol, given by $\log |\Sigma_i|$, where Σ_i is an alphabet and subscript i denotes the number of applied transformations.

In the *GCT* and *HOCT* strategies, the size of the alphabet will be constant, unless some symbols were completely removed, then the size of the alphabet decreases. Re-Pair and MinEnt algorithms, which introduce new symbols, have an increasing alphabet size. The upper boundary on the resulting size of each dictionary entry $|D|$ for *GCT* and *HOCT* transformations is defined as:

$$|D| = 3 \log |\Sigma_0|$$

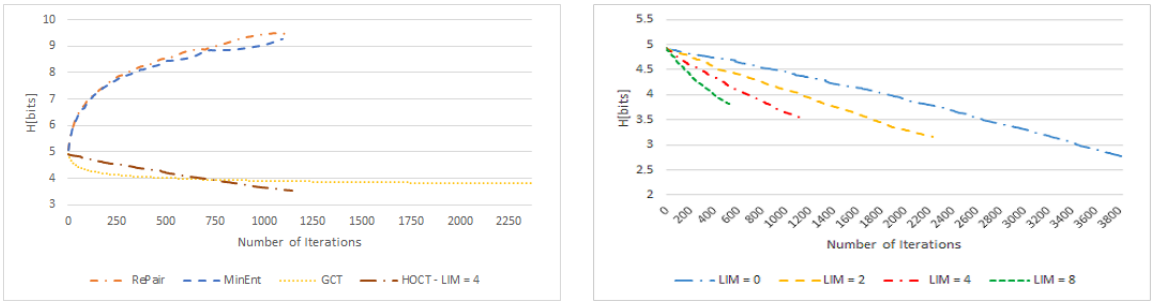
where $|\Sigma_0|$ is the size of the initial alphabet. The Larsson and Moffat [13] version of the Re-Pair introduces several efficient ways of dictionary encoding: the Bernoulli model, literal pair enumeration and interpolative encoding. In our experiments with Re-Pair and MinEnt, we used interpolative encoding to encode dictionary.

3.3.2. Comparison of the Alphabet's Entropy Evolution

Even though the transformations and compression functions pursue the same objective, minimization of the entropic size of the message, they achieve that by a different evolution of zero order entropy. Transformation-based strategies minimize zero order entropy; meanwhile, both compression strategies introduce new symbols, and as a result, zero order entropy grows. The initial values of the quantities of the examined test file are summarized in Table 1. The example of the comparison of the zero order entropy evolution of different strategies is provided in Figure 1a.

Table 1. Characteristics of the paper5 file from the Calgary corpus: the initial size of alphabet $|\Sigma|$, the initial file size $|m_0|$ measured in bytes, the initial entropy H_0 measured in bits and the initial entropic size $|m_0|^H$ measured in bytes.

File Name	$\|\Sigma\|$	$\|m_0\|$	H_0	$\|m_0\|^H$
paper5	91	11 954	4.936	7 376



(**a**) (**b**)

Figure 1. Comparison of zero order entropy evolution over the paper5 file from the Calgary corpus. (**a**) Evolution of zero order entropy for different strategies; (**b**) evolution of zero order entropy for different values of the limit (LIM) in the *HOCT* strategy.

Both compression functions achieve a very similar resulting value of zero order entropy. The Re-Pair strategy begins with the highest growth of entropy, but the increase slows down with the number of iterations as the frequency of each consecutive digram drops. As will be discussed in Section 4.2.2, digrams consisting of symbols with a lower frequency will be preferred by MinEnt,

because they will be able to achieve a larger decrease of entropic size, and their replacement brings less costs in the zero order entropy increase. This behavior can be observed especially in later iterations of the Re-Pair and MinEnt algorithms.

Both transformations reduce the value of zero order entropy. *GCT* initially drops faster, but in the end, it significantly slows down. The application of the *HOCT* strategy achieves the lowest resulting value of entropy, and the interesting fact is that it decreases at an almost constant rate. The behavior of entropy evolution for different values of the limit in *HOCT* is presented in Figure 1b. The unrestricted case ($Lim = 0$) shows us the bottom limit of zero order entropy reduction using the *HOCT* strategy.

3.3.3. Comparison of Entropic Size Evolution

The selection of the most frequent digram will produce the largest decrease of the number of symbols in each iteration. Surprisingly, the Re-Pair strategy does not necessarily have to converge to its minimum in the lower number of iterations than MinEnt. Figure 2 presents this behavior for the paper5 file of the Calgary corpus. Both approaches end with a similar number of symbols in the resulting message.

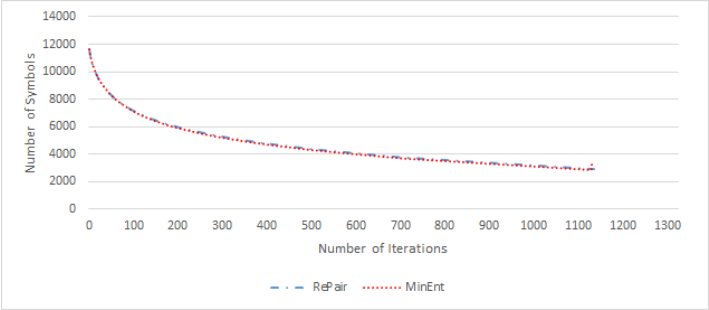


Figure 2. Comparison of Re-Pair and MinEnt algorithms: evolution of the message size measured in the number of symbols over the paper5 file from the Calgary corpus.

The MinEnt strategy achieves the lowest entropic size of the message, and at each iteration, the entropic size of the message is lower than in the case of the Re-Pair strategy, see Figure 3. The overall efficiency depends on our ability to compress the resulting dictionary.

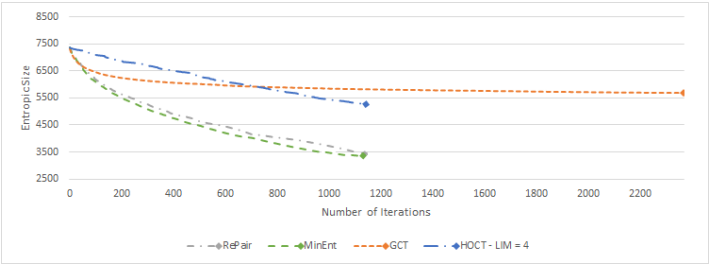


Figure 3. Comparison of Re-Pair, MinEnt, *GCT* and *HOCT* algorithms: evolution of the entropic message size measured in bits per byte over the paper5 file from the Calgary corpus.

A summary of different transformation strategies is provided in Table 2. A summary of compression functions is then given in Table 3. The least number of iterations was achieved by *HOCT* with $LIM = |D|$; this strategy also leads to the least final size $|m_f|$, but it should be emphasized that the resulting entropic size of the message $|m_f|$ is a very pessimistic estimate, due to the construction of the size of dictionary entries.

Table 2. The comparison of transformation strategies using different criteria: LIM is the limiting size of the dictionary entry in bytes, i is the number of iterations; H_i is the final entropy measured in bits; $|m_i|^H$ is the final entropic size measured in bytes; $\log|\Sigma_i|$ is the upper boundary on the amount of information needed to store one symbol to dictionary; and $|m_f|$ is the final size of the file given as the sum of the entropic size of the message and the size of the dictionary measured in bytes.

Strategy	LIM	i	H_i	$\lvert m_i\rvert^H$	$\log\lvert\Sigma_i\rvert$	$\lvert m_f\rvert$
GCT	0	2367	3.796	5674	6.508	11,451
GCT	$\lvert D\rvert$	127	4.260	6366	6.508	6676
HOCT	0	3 821	2.786	4163	6.508	13,488
HOCT	4	1 143	3.528	5272	6.508	8061
HOCT	8	525	3.830	5713	6.508	6994
HOCT	$\lvert D\rvert$	222	4.067	6078	6.508	6439

Table 3. The comparison of compression strategies using different criteria: i is the number of iterations; $|\Sigma_i|$ is the size of the final alphabet; $|m_i|$ is the resulting size of the file measured as the number of symbols; H_i is the final entropy measured in bits; $|m_i|^H$ is the final entropic size measured in bytes; $|D_i|$ is the average number of bits needed to store one phrase in the dictionary; and $|m_f|$ is the final size of the file given as the sum of the entropic size of the message and the size of the dictionary measured in bytes.

Strategy	i	$\lvert\Sigma_i\rvert$	$\lvert m_i\rvert$	H_i	$\lvert m_i\rvert^H$	$\lvert D_i\rvert$	$\lvert m_f\rvert$
Re-Pair	1146	965	2832	9.283	3286	10.240	4753
MinEnt	1129	944	2798	9.395	3286	10.281	4737

Even though the achieved results of both approaches are similar, we see that the resulting message size $|m_f|$ and alphabet size are lower in the case of MinEnt. The message size $|m_f|$ is given by the sum of the entropic size of the message and the size of the dictionary stored by interpolative encoding. Using values in the columns of Table 3, we express $|m_f| = |m_i|H_i + i|D_i|$; the term $i|D_i|$ represents the size of the dictionary given as a product of the total number of iterations and the average number of bits needed to encode one iteration. See Tables 4 and 5 for more results on files from the Calgary and Canterbury corpora.

Table 4. The comparison of strategies using different criterions, i is the number of iterations, $|\Sigma_i|$ is the size of the final alphabet, $|m_i|$ is the resulting size of file measured as the number of symbols, H_i is the final entropy measured in bits, $|m_i|^H$ is the final entropic size measured in bytes, $|D|/|\Sigma_i|$ is the average number of bytes needed to store one phrase into the dictionary and $|m_f|$ is the final size of the file given as the sum of entropic size of the message and the size of the dictionary measured in bytes.

File Name	$\lvert m_0\rvert$	Σ_0	H_0	i	$\lvert\Sigma_i\rvert$	$\lvert m_i\rvert$	H_i	$\lvert m_i\rvert^H$	$\frac{\lvert D\rvert}{\lvert\Sigma_i\rvert}$	$\lvert m_f\rvert$
Calgary corpus										
bib	111,261	81	5.257	5469	4216	15,159	11.410	21,621	11.589	29,544
book1	768,771	82	4.528	23,587	22,649	128,059	13.422	214,859	11.904	249,957
book2	610,856	96	4.681	21,147	18,501	82,446	13.213	136,165	12.829	170,079
news	377,109	98	5.226	20,079	13,602	55,500	12.809	88,863	12.761	120,892
obj1	21,504	256	5.929	1650	1475	6464	9.888	7990	10.918	10,242
obj2	246,814	256	6.280	14,635	9569	35,540	12.323	54,743	13.044	78,607
paper1	53,161	95	4.967	3559	2678	8800	10.890	11,979	11.360	17,033
paper2	82,199	91	4.506	4297	3753	14,102	11.235	19,805	11.181	25,811
paper3	46,526	84	4.588	2989	2575	9061	10.767	12,195	10.791	16,227
paper4	13,286	80	4.602	1194	997	3136	9.622	3772	10.130	5284
paper6	38,105	93	5.000	2834	2108	6670	10.585	8826	11.220	12,801
progc	39,611	92	5.282	2854	2066	6526	10.641	8681	11.254	12,696
progl	71,646	87	4.830	4162	2577	7216	10.851	9788	12.003	16,033
progp	49,379	89	4.823	3147	1684	4528	10.272	5814	11.952	10,516
trans	93,695	99	5.545	5918	2505	6513	10.968	8929	12.419	18,116

Table 4. *Cont.*

File Name	$\|m_0\|$	Σ_0	H_0	i	$\|\Sigma_i\|$	$\|m_i\|$	H_i	$\|m_i\|^H$	$\frac{\|D\|}{\|\Sigma_i\|}$	$\|m_f\|$
Canterbury corpus										
alice29.txt	152,089	74	4.435	6733	6068	25,077	11.985	37,568	11.482	47,232
asyoulik.txt	125,179	68	4.889	5799	5293	23,532	11.774	34,634	10.932	42,559
bible.txt	4,047,392	63	4.260	81,229	71,256	386,094	15.017	724,728	14.525	872 215
cp.html	24,603	86	5.107	1785	1271	4242	9.590	5085	10.689	7470
E.coli	4,638,690	4	2.000	67,368	62,924	652,664	13.725	1,119,687	7.462	1182 530
fields.c	11,150	90	4.924	927	658	1503	9.304	1748	10.822	3002
kennedy.xls	1,029,744	256	3.584	2446	2545	160,177	9.788	195,978	8.274	198 508
lcet10.txt	426,754	84	4.627	14,515	12,395	55,691	12.759	88,823	12.426	111,369
ptt5	513,216	159	1.049	5995	5697	30,463	11.424	43,503	11.178	51,880
random.txt	100,000	64	6.000	5065	5126	54,182	11.182	75,731	3.983	78,253
sum	38,240	255	5.447	3116	1749	6251	10.290	8041	11.912	12,681
world192.txt	2,473,400	94	5.024	55,473	47,150	212,647	14.552	386,808	13.973	483 705
xargs.1	4227	74	4.863	468	384	990	8.255	1022	9.811	1596

Table 5. The comparison of the strategies using different criteria: i is the number of iterations; $|\Sigma_i|$ is the size of final alphabet; $|m_i|$ is the resulting size of the file measured as the number of symbols; H_i is the final entropy measured in bits; $|m_i|^H$ is the final entropic size measured in bytes; $|D|/|\Sigma_i|$ is the average number of bytes needed to store one phrase in the dictionary; and $|m_f|$ is the final size of the file given as the sum of entropic size of the message and the size of the dictionary measured in bytes.

File Name	$\|m_0\|$	$\|\Sigma_0\|$	H_0	i	$\|\Sigma_i\|$	$\|m_i\|$	H_i	$\|m_i\|^H$	$\frac{\|D\|}{\|\Sigma_i\|}$	$\|m_f\|$
Calgary corpus										
bib	111,261	81	5.201	5513	4150	15,103	11.307	21,346	11.717	29,421
book1	768,771	82	4.527	23,843	22,616	127,777	13.377	213,656	12.134	249,822
book2	610,856	96	4.793	20,852	17,997	80,814	13.170	133,045	12.847	166,533
news	377,109	98	5.190	20,118	13,388	55,347	12.697	87,845	12.918	120,333
obj1	21,504	256	5.948	1638	1418	6459	9.727	7853	11.135	10,133
obj2	246,814	256	6.260	14,673	9337	35,510	12.173	54,031	13.218	78,275
paper1	53 161	95	4.983	3579	2633	8726	10.741	11,716	11.413	16,822
paper2	82,199	91	4.601	4247	3612	13,797	11.088	19,123	11.221	25,080
paper3	46,526	84	4.665	3004	2529	8993	10.676	12,002	10.892	16,092
paper4	13,286	80	4.700	1136	930	3133	9.232	3615	10.288	5076
paper6	38,105	93	5.010	2841	2080	6662	10.398	8659	11.297	12,671
progc	39,611	92	5.199	2871	2034	6530	10.444	8525	11.307	12,583
progl	71,646	87	4.770	4175	2495	7134	10.742	9579	12.100	15,894
progp	49,379	89	4.869	3145	1631	4509	10.166	5730	12.194	10,524
trans	93,695	99	5.533	5916	2425	6515	10.736	8743	12.713	18,145
Canterbury corpus										
alice29.txt	152,089	74	4.568	6649	5903	24,825	11.767	36,516	11.573	46,135
asyoulik.txt	125,179	68	4.808	5804	5220	23,359	11.563	33,764	11.018	41,758
bible.txt	4,047,392	63	4.343	77,117	66,593	386,092	14.649	706,991	14.543	847,187
cp.html	24,603	86	5.229	1748	1212	4313	9.462	5101	10.897	7482
E.coli	4,638,690	4	2.000	66,995	62,463	652,663	13.717	1,119,067	7.667	1,183,281
fields.c	11,150	90	5.008	868	587	1606	8.724	1751	11.013	2946
kennedy.xls	1,029,744	256	3.573	2612	2511	159,999	10.012	200,240	8.575	203,040
lcet10.txt	426,754	84	4.669	14,506	12,178	54,937	12.661	86,941	12.462	109,539
ptt5	513,216	159	1.210	23,203	6314	94,463	4.566	53,918	12.308	89,618
random.txt	100,000	64	5.999	5145	5209	54,011	11.235	75,854	4.075	78,475
sum	38,240	255	5.329	3130	1683	6245	10.034	7833	12.184	12,600
world192.txt	2,473,400	94	4.998	54,946	45,920	212,499	14.340	380,896	14.078	477,588
xargs.1	4227	74	4.898	342	326	1235	7.755	1197	9.988	1624

4. Zero Order Entropy and Entropic Message Size Reduction

The primary purpose of context transformation and other derived transformations is to reduce the zero order entropy measured by Shannon's entropy [2] defined in Equation (1). In this section, we shall show under what conditions the transformation and compression function reduces zero order entropy resp. the entropic size of the message. Suppose that H_0 is a zero order entropy of message m, and H_1 is a zero order entropy after a transformation T is applied. The conditions under which the following inequalities hold are the major subject of interest.

$$\Delta H = H_0 - H_1 = \sum_{x \in \Sigma} p_1(x) \log p_1(x) - \sum_{x \in \Sigma} p_0(x) \log p_0(x) > 0 \tag{5}$$

Let $\Sigma^T \subset \Sigma$ be a set of symbols whose frequencies before and after transformation differ, and $\Sigma^I \subset \Sigma$ is a set of symbols whose frequencies are intact. For transformations, the inequality (5) can be further restricted only to the set of symbols Σ^T, since the terms containing symbols from Σ^I subtract:

$$\Delta H = \sum_{x \in \Sigma^T} p_1(x) \log p_1(x) - \sum_{x \in \Sigma^T} p_0(x) \log p_0(x) \tag{6}$$

In the following paragraph, we will specify the forms of the set Σ^T and the relations for the probabilities of its symbols after transformations, so that the change of entropy given by Equation (6) can be computed before any transformation actually occurs.

4.1. Transformation of Probabilities

We begin with the simplest case: suppose the context transformation $CT(\alpha\beta \rightarrow \alpha\gamma, w)$. Since only the probabilities of symbols β and γ will change, then $\Sigma^{CT} = \{\beta, \gamma\}$, and it is sufficient to express probabilities only for β and γ:

$$p_1^{CT}(\beta) = p_0(\beta) - p_0(\alpha, \beta) \tag{7}$$

and:

$$p_1^{CT}(\gamma) = p_0(\gamma) + p_0(\alpha, \beta) \tag{8}$$

In the case of the generalized context transformation $GCT(\alpha\beta \leftrightarrow \alpha\gamma, w)$, the set $\Sigma^{CT} = \Sigma^{GCT}$ is identical, and the probabilities transform according to:

$$p_1^{GCT}(\beta) = p_0(\beta) - p_0(\alpha, \beta) + p_0(\alpha, \gamma) \tag{9}$$

and:

$$p_1^{GCT}(\gamma) = p_0(\gamma) + p_0(\alpha, \beta) - p_0(\alpha, \gamma) \tag{10}$$

In the last case of higher order transformation, the probabilities transform according to:

$$p_1^{HOCT}(\beta) = p_0(\beta) - p_0(w, \beta) + p_0(w, \gamma) \tag{11}$$

and:

$$p_1^{HOCT}(\gamma) = p_0(\gamma) + p_0(w, \beta) - p_0(w, \gamma) \tag{12}$$

In all cases, the set Σ^T forms a binary alphabet. The following theorem then describes the condition for zero order entropy reduction:

Theorem 1. *Suppose the generalized context transformation $GCT(\alpha\beta \leftrightarrow \alpha\gamma)$. Let $p_0(\beta)$ and $p_0(\gamma)$ be the probabilities of symbols before the transformation is applied, and let $p_{0,max} = max\{p_0(\beta), p_0(\gamma)\}$. After the transformation, the associated probabilities are $p_1(\beta)$, $p_1(\gamma)$ and $p_{1,max} = max\{p_1(\beta), p_1(\gamma)\}$. If $p_{1,max} > p_{0,max}$, then the generalized context transformation T reduces entropy.*

The proof of Theorem 1 is based on the properties of entropy when only two letters from alphabet Σ are considered. Let $p(\beta) + p(\gamma) = c$, where $c \leq 1$, c is invariant, it does not change during the transformation. We can express one of these probabilities using the other one; for example, let $p(\gamma) = c - p(\beta)$; this allows us to express the entropy function as a function of only one variable.

A few examples of such functions are shown in Figure 4. The maximum value of the function is located in the value $c/2$, and it has two minimums at zero and at c.

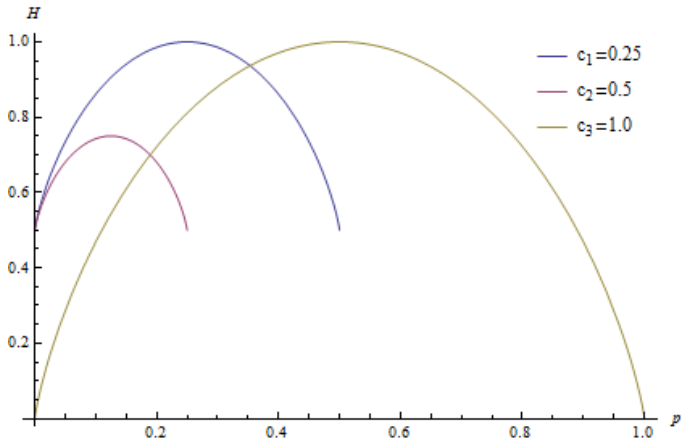


Figure 4. The entropy of two letters with different $c_i \in \{0.25, 0.5, 1.0\}$.

Proof. Since the entropy function for two different letters is defined on the interval $\langle 0; c \rangle$ and it is concave with a maximum at $c/2$ and minimums at zero and c, then $p_{0,max}$ has to be located on the interval $p_{0,max} \in \langle c/2; c \rangle$; but on that particular interval, the higher the maximum is, the lower the entropy is, so if we increase the maximum (or we can say increase the absolute value of the difference $|p_\beta - p_\alpha|$), then the entropy will decrease. □

4.2. General Entropy Change Relations

In this section, we generalize the notion of zero order entropy change on the exchange of any two words. The solution is divided into three parts. The first part deals with the set of symbols Σ^I whose frequency does not change before and after the substitution function is applied; the second part establishes relations for the set of symbols Σ^T whose probability is changed, but their initial and final frequencies are non-zero; the third part discusses symbols introduced to and removed from the alphabet. Let Σ^R be a set of removed symbols, and Σ^N is a set of introduced symbols; then, we can split the sum in Equation (1), yielding:

$$H(X) = -\sum_{\alpha \in \{I,T,R,N\}} \sum_{x \in \Sigma^\alpha} p(x) \log p(x) \tag{13}$$

The four sets of symbols in Equation (13) exhibit different behaviors under the substitution function, and they will be discussed in separate sections. The entropic size of the message $|m|^H$ can also be handled separately; let $H(p(\Sigma^\alpha))$ be a portion of entropy conveyed by symbols from alphabet Σ^α, and let $|m|^{H(\Sigma^\alpha)}$ be particular portions of the entropic size of the message; then, we can split the resulting entropic size as we did before:

$$\begin{aligned} |m|^H &= |m|[H(p(\Sigma^I)) + H(p(\Sigma^T)) + H(p(\Sigma^R)) + H(p(\Sigma^N))] \\ &= |m|H(p(\Sigma^I)) + |m|H(p(\Sigma^T)) + |m|H(p(\Sigma^R)) + |m|H(p(\Sigma^N)) \\ &= |m|^{H(\Sigma^I)} + |m|^{H(\Sigma^T)} + |m|^{H(\Sigma^R)} + |m|^{H(\Sigma^N)} \end{aligned} \tag{14}$$

4.2.1. The First Part: Symbols Remaining Intact by the Substitution Function

We begin with symbols that are not part of either of the substituting words s_1 or s_2. Suppose that the length $|m_0|$ of the message m_0 turns into some message m_1 of the size $|m_1|$. Generally, $|m_0| \neq |m_1|$, but in a special case of context transformations, these two quantities are equal. However, when the

compression or expansion of the message occurs, the part of the Shannon equation will also change due to the change in the total number of symbols.

Suppose that the symbol x is initially in the message m_0 with the probability $p_0(x)$. This probability can be expressed using the frequency $f_0(x)$ and the size of the message as:

$$p_0(x) = \frac{f_0(x)}{|m_0|} \tag{15}$$

Later, after the transformation was applied, the probability changes to:

$$p_1(x) = \frac{f_0(x)}{|m_0| + \Delta m} \tag{16}$$

where Δm is a change of the message size. In the case of context transformations where the message size remains the same size, the probability remains the same, as well as the part of entropy formed by non-transformed symbols.

When the two probabilities are placed in relation by some stretching factor c_1 we arrange them into the form:

$$p_1(x) = c_1 p_0(x) \tag{17}$$

The factor c_1 (the introduction of c_1 is motivated by the properties of logarithms if we would actually stay with $p_1(x)$ given by Equation (16), we would get logarithm $\log \frac{f_0(x)}{|m_0|+\Delta m} = \log f_0(x) - \log(|m_0| + \Delta m)$. If instead, we express p_1 using (17), then the logarithm is in product form, and its arguments are single numbers $\log cp_0 = \log c + \log p_0$) can be expressed by substitution of $p_i(x)$ in Equation (17) by the terms in Equations (15) and (16), leading to:

$$\begin{aligned} \frac{f_0(x)}{|m_0| + \Delta m} &= c_1 \frac{f_0(x)}{|m_0|} \\ c_1 &= \frac{|m_0|}{|m_0| + \Delta m} \end{aligned} \tag{18}$$

Then, the relation for zero order entropy after transformation will have the form:

$$\begin{aligned} H(p_1(\Sigma^I)) &= -\sum_{x \in \Sigma^I} c_1.p_0(x) \log [c_1.p_0(x)] \\ &= -c_1 \sum_{x \in \Sigma^I} p_0(x)[\log c_1 + \log p_0(x)] \\ &= -c_1 \log c_1 \sum_{x \in \Sigma^I} p_0(x) - c_1 \sum_{x \in \Sigma^I} p_0(x) \log p_0(x) \\ &= -c_1 \log c_1 \sum_{x \in \Sigma^I} p_0(x) + c_1 H(p_0(\Sigma^I)) \\ &= c_1 [H(p_0(\Sigma^I)) - \log c_1 \sum_{x \in \Sigma^I} p_0(x)] \end{aligned} \tag{19}$$

The example of the behavior of $H(p_1(\Sigma^I))$ of the intact part is visualized in Figure 5. When the compression of the message occurs, i.e., $\log c_1 > 0$, then the zero order entropy of intact symbols increases. The less the probability is conveyed by symbols from Σ^I, the more their zero order entropy is sensitive to the change of c_1.

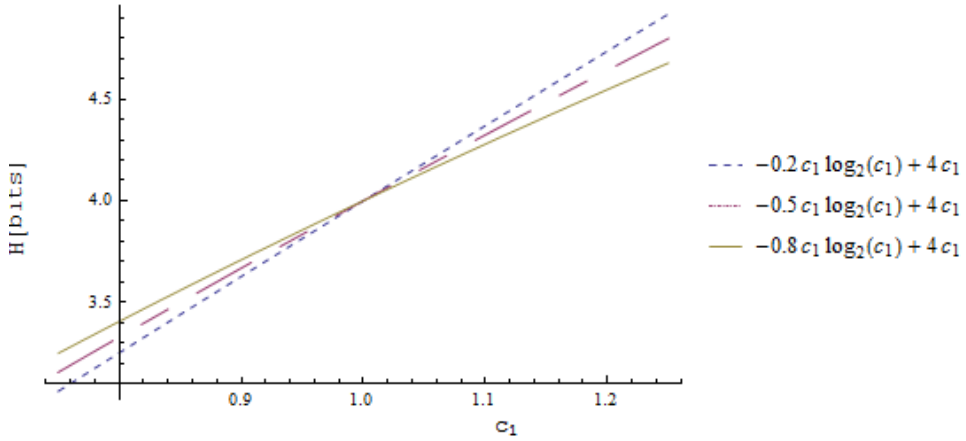


Figure 5. The portion of entropy $H(p_1(\Sigma^I))$ given by symbols from Σ^I as a function of c_1 for the constant $H(p_0(\Sigma^I)) = 4$ and $\sum p_0(x) \in \{0.2, 0.5, 0.8\}$.

The final entropic size is given as follows:

$$\begin{aligned} |m_1|^{H,\Sigma_I} &= |m_1|H(p_1) = |m_1|c_1[H(p_0(\Sigma^I)) - \log c_1 \sum_{x\in\Sigma^I} p_0(x)] \\ &= |m_0|H(p_0(\Sigma^I)) - |m_0| \log c_1 \sum_{x\in\Sigma^I} p_0(x)) \\ &= |m_0|^{H,\Sigma_I} - |m_0| \sum_{x\in\Sigma^I} p_0(x) \log c_1 \\ &= |m_0|^{H,\Sigma_I} - \log c_1 \sum_{x\in\Sigma^I} f_0(x) \end{aligned} \tag{20}$$

If we apply one of transformations, then $\Delta m = 0$, and as a consequence, $c_1 = 1$; the last term on the right will be zero due to $\log c_1 = 0$, so Equation (20) tells us that the entropic size of the message carried by these symbols does not change during transformation. When $|\Sigma^T|$ is much smaller than $|\Sigma^I|$, it is convenient to rewrite Equation (20) in terms of Σ^T:

$$|m_1|^{H,\Sigma_I} = |m_0|^{H,\Sigma_I} - \log c_1[|m_0| - \sum_{x\in\Sigma^T} f_0(x)] \tag{21}$$

Corollary 1. *No compression function ever increases the entropic size of the part of the message consisting of intact symbols.*

Proof. The compression function has the value of c_1 larger than one, as a consequence $\log c_1 > 0$, and so, $|m_1|^{H(\Sigma_I)} \leq |m_0|^{H(\Sigma_I)}$. □

The equality in $|m_1|^{H,\Sigma_I} \leq |m_0|^{H,\Sigma_I}$ occurs when $\sum_{x\in\Sigma^I} f_0(x) = 0$, i.e., when there are no intact symbols. When the expansion of the message occurs, then $\log c_1 < 0$ and the second term of Equation (20) on the right will change to a positive number. Expansion of the message leads to the increase of the entropic size; meanwhile, compression leads to the decrease of the entropic size of intact symbols.

In each iteration of the Re-Pair algorithm, the most frequent digram is selected. This corresponds to the selection of a digram with maximal value of $\log c_1$, but it does not have to be the digram minimizing the entropic size of this part of the resulting message the most. Consider two digrams d_1 and d_2, so that their frequencies are equal: $f(d_1) = f(d_2)$; replacing them with a new symbol yields the same stretching factor c_1, but not necessarily $\sum_{x\in\Sigma^I} f_0(x)$. The larger reduction of the entropic size of a message will be achieved when compressed digrams or words consist of less frequent symbols.

4.2.2. The Second Part: Symbols Participating in the Substitution Function

In the second case, the frequencies of symbols and their total number will change. The equation for stretching factor c_2 will be derived in the following way:

$$p_1(x) = \frac{f_1(x)}{|m_1|} = \frac{f_0(x) + \Delta f(x)}{|m_0| + \Delta m}$$

$$\frac{f_0(x) + \Delta f(x)}{|m_0| + \Delta m} = c_2 \frac{f_0(x)}{|m_0|}$$

The main difference in both cases is that c_1 is a constant; meanwhile, c_2 is a function of the particular symbol x.

$$c_2(x) = \frac{(f_0(x) + \Delta f(x))|m_0|}{f_0(x)(|m_0| + \Delta m)} = \frac{f_0(x) + \Delta f(x)}{f_0(x)} c_1 = F(x)c_1 \tag{22}$$

where in the last step, we made the substitution: $F(x) = (f_0(x) + \Delta f(x))/f_0(x)$ The rest of the derivation follows the derivation of Equation (19).

$$H(p_1(\Sigma^T)) = -\sum_{x \in \Sigma^T} p_0(x)c_2(x) \log c_2(x) - \sum_{x \in \Sigma^T} c_2(x)p_0(x) \log p_0(x) \tag{23}$$

The behavior of Equation (23) for different values of $p_0(x)$ is visualized in Figure 6. The substitution of less frequent symbols leads to a lower increase of zero order entropy.

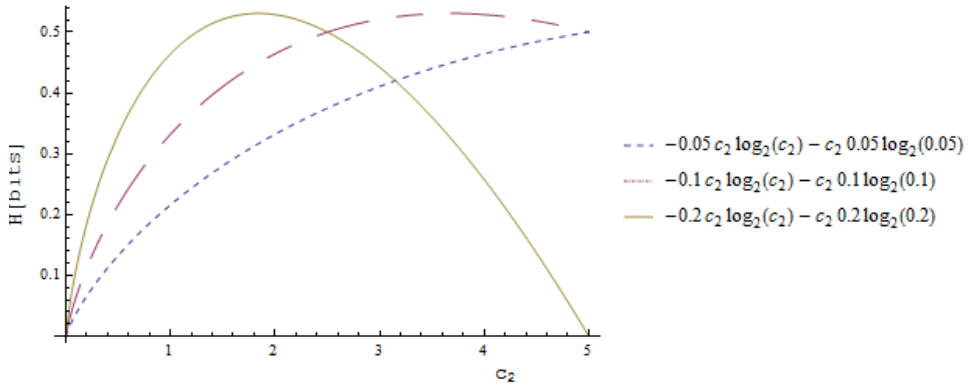


Figure 6. Dependency of $H(p_1(\Sigma^T))$ on different values of c_2 for three cases of $p_0(x) \in \{0.05, 0.1, 0.2\}$.

The resulting entropic size simplifies given that:

$$|m_1|c_2(x)p_0(x) = f_0(x) + \Delta f(x) = f_1(x) \tag{24}$$

yields:

$$\begin{aligned} |m_1|^H &= |m_1|H^T(p_1) \\ &= -\sum_{x \in \Sigma^T} [f_0(x) + \Delta f(x)](\log c_2(x) + \log p_0(x)) \end{aligned} \tag{25}$$

We now analyze both terms in (25) from the perspective of different values of $c_2(x)$. We will be particularly interested in compression functions. We know that for compression function $c_1 > 1$, symbols with $\Delta f(x) < 0$, i.e., symbols whose frequency decreases, will have $F(x) < 1$. The positivity or negativity of $\log c_2$ then depends on the value of product $F(x)c_1$.

The case when $F(x)c_1 = 1$ has a solution $F(x) = 1/c_1$, then $\log c_2(x) = 0$. The term $\log p_0(x)$ is always negative. The value of $F(x)$ must be larger than $1/c_1$ to decrease the zero order entropy conveyed by symbol x, since then, $c_2(x) > 1$ and, as a consequence, $\log c_2(x) > 0$.

$$\begin{aligned} F(x) &> \frac{1}{c_1} \\ \frac{f_0(x) + \Delta f(x)}{f_0(x)} &> \frac{|m_0| + \Delta m}{|m_0|} \\ 1 + \frac{\Delta f(x)}{f_0(x)} &> 1 + \frac{\Delta m}{|m_0|} \\ \frac{\Delta f(x)}{f_0(x)} &> \frac{\Delta m}{|m_0|} \\ \frac{|\Delta f(x)|}{f_0(x)} &< \frac{|\Delta m|}{|m_0|} \\ \frac{|\Delta f(x)|}{|\Delta m|} &< \frac{f_0(x)}{|m_0|} = p_0(x) \end{aligned} \tag{26}$$

The introduction of the absolute value in the middle step of the derivation of Inequality (26) is allowed since using compression functions values of $\Delta f(x)$ and Δm can only be negative. Suppose now that we have a digram $d = \alpha\beta$, given that $\alpha \neq \beta$, and we replace it by the newly-introduced γ, then $\Delta m = \Delta f(\alpha) = \Delta f(\beta)$. The left part of Inequality (26) becomes equal to one, so Inequality (26) cannot be satisfied, and $\log c_2(x)$ in this case will be negative and will always increase the amount of information carried by the symbols α and β.

Finally, we state the condition for the entropic size decrease:

Corollary 2. *The entropic size of the part of the message formed by symbol x decreases when:*

$$\frac{\Delta f(x)}{f_0(x)} < -\frac{\log c_2(x)}{\log c_2(x) + \log p_0(x)} \tag{27}$$

Proof.

$$\begin{aligned} |m_1|^H &< |m_0|^H \\ [f_0(x) + \Delta f(x)][\log c_2(x) + \log p_0(x)] &< f_0(x) \log p_0(x) \\ f_0(x) \log c_2(x) + \Delta f(x) \log p_0(x) + \Delta f(x) \log c_2(x) &< 0 \\ \Delta f(x)(\log p_0(x) + \log c_2(x)) &< -f_0(x) \log c_2(x) \\ \frac{\Delta f(x)}{f_0(x)} &< -\frac{\log c_2(x)}{\log c_2(x) + \log p_0(x)} \end{aligned}$$

□

4.2.3. Third Part: Introduced and Removed Symbols

We begin with symbol x, which is completely removed from the message, so that initially, $p_0(x) \neq 0$, but $p_1(x) = 0$. This case is trivial, and it has zero participation in the final value of the entropy and the entropic size of message. The remaining case we have to deal with is a case when initially symbol x has zero probability $p_0(x) = 0$, but after substitution, its probability will increase to some $p_1(x) \neq 0$. The final probability is given as:

$$p_1(x) = \frac{\Delta f(x)}{|m_0| + \Delta m} \tag{28}$$

Since the symbol x initially has zero participation in entropy and entropic size, it will always lead to the increase of both quantities. For the set Σ^N of all such symbols, its portion on total entropy is then given by:

$$H(p_1(\Sigma^N)) = -\sum_{x\in\Sigma^N} \frac{\Delta f(x)}{|m_0| + \Delta m} \log \frac{\Delta f(x)}{|m_0| + \Delta m} \tag{29}$$

and the corresponding final entropic size will be given by:

$$\begin{aligned} |m_1|^H &= |m_1| H^N(p_1) \\ &= -\sum_{x\in\Sigma^N} \Delta f(x)[\log \Delta f(x) - \log(|m_0| + \Delta m)] \end{aligned} \tag{30}$$

It is important to remark that it does not make much sense to introduce more than one symbol in one substitution function, because both quantities would then add themselves twice.

4.3. Calculation of $\Delta|m|^H$

At first glance, it seems that we need to evaluate all symbols to predict zero order entropy, but instead, it is possible to predict the exact change of the entropic size of the message after the application of the compression function by the evaluation of entropic sizes given by Equations (21), (25) and (30) dealing only with symbols $x \in \Sigma \setminus \Sigma_I$. In the particular case of the Re-Pair algorithm, there are only two symbols whose frequency knowledge is sufficient to evaluate the change of the entropic size of the message; suppose a compression function $CF(\alpha\beta \to \gamma)$ so that $p_1(\alpha) \neq 0$, $p_1(\beta) \neq 0$ and $p_0(\gamma) = 0$, then the resulting entropic size is given as:

$$\begin{aligned} \Delta|m|^H = {} & |m_0| \log c_1 - \log c_1 \sum_{x\in\{\alpha,\beta\}} f_0(x) \\ & + \sum_{x\in\{\alpha,\beta\}} f_0(x) \log c_2(x) + \Delta f_0(x) \log p_0(x) + \Delta f_0(x) \log c_2(x) \\ & - \Delta f(\gamma)[\log \Delta f(\gamma) - \log(|m_0| + \Delta m)] \end{aligned} \tag{31}$$

finally, for the Re-Pair, it holds that if $\alpha \neq \beta$, then $\Delta m = \Delta f(\alpha) = \Delta f(\beta) = \Delta f(\gamma) = f(\alpha, \beta)$, and all Δ's in (31) turn into $f(\alpha\beta)$. If $\alpha = \beta$, then $\Delta f(\alpha)/2 = \Delta f(\gamma)$.

5. Conclusions

We described three types of transformations for the preprocessing of messages so that the zero order entropy of messages drops so the resulting message can be more efficiently encoded using zero order entropy compression algorithms like Huffman or arithmetic coding.

We presented relations that govern the change of the message size for transformations and compression functions. Transformations have the advantage that they do not modify the size of the alphabet, especially in the case of digram substitution used by Re-Pair and our proposal of the MinEnt strategy; the resulting size of the alphabet significantly grows, and it brings additional complexity in the storage of the entropy coding model, i.e., the storage of the output alphabet.

The MinEnt strategy selects digrams to be replaced by the minimal entropic size of the resulting message, and it is shown that in most cases, the resulting message size is smaller than the one achieved by Re-Pair. We also showed that the two algorithms follow slightly different execution paths, as MinEnt prefers digrams that consist of less frequent symbols; meanwhile, Re-Pair does not take this into consideration.

The compression functions take advantage of transformations as they achieve a better resulting compression ratio. In future work, we will focus on the storage of the dictionary that will be used in transformation algorithms, because this area can significantly improve the resulting compression ratio.

Further, we will focus on the description of the relation between the entropy coding model of the final message and the size of the final alphabet.

Acknowledgments: This work was supported by the project SP2017/100 Parallel processing of Big Data IV, of the Student Grant System, VSB-Technical University of Ostrava. The costs for open access were covered.

Author Contributions: Michal Vasinek realized this work, proposed and developed the implementation of the *CT*, *GCT*, *HOCT* and MinEnt algorithms. Jan Platos provided the guidance during the writing process and revised the paper.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The following sections present properties of transformations. Specifically for each type of transformation we will provide a proof of the inverse transformation existence. Further we will describe how the frequencies of symbols will be altered if the particular transformation is going to be applied.

Appendix A.1. CT—Proof of the Correctness

This theorem defines the inverse transformation of the context transformation:

Theorem A1. *The context transformation* $CT^{-1} \equiv CT_{\rightarrow}(\alpha\gamma \rightarrow \alpha\beta)$ *is inverse transformation of the context transformation* $CT_{\leftarrow}(\alpha\beta \rightarrow \alpha\gamma)$.

Proof. Let $CT^{-1} \equiv CT_{\rightarrow}$, if CT^{-1} is inverse then the following must be true for any message m: $CT^{-1}(CT(m)) = m$. Suppose that we are passing message m from the end to the beginning and suppose that in positions i and $i+1$ digram $\alpha\beta$ is located, this digram is replaced by $\alpha\gamma$, the next pair of positions explored are $i-1$ and i, but their value is independent of the preceding replacement, because the replacement has taken place in position $i+1$, so when CT^{-1} is applied in position i it will find there the digram $\alpha\gamma$ and reverts it back to $\alpha\beta$. □

Other combinations of directions do not form a pair of transformation and its inverse. We give an example for each case showing this property: $CT_{\rightarrow}(\alpha\beta \rightarrow \alpha\alpha)$ and $CT_{\leftarrow}^{-1}$ over the message $m = \alpha\beta\alpha$: $CT_{\rightarrow}(\alpha\beta\alpha) = \alpha\alpha\alpha$ but $CT_{\leftarrow}^{-1}(\alpha\alpha\alpha) = \alpha\beta\beta \neq \alpha\beta\alpha$. Next consider $CT_{\rightarrow}(\alpha\beta \rightarrow \alpha\alpha)$ and $CT_{\rightarrow}^{-1}$ over the message $m = \alpha\alpha\beta$: $CT_{\rightarrow}(\alpha\alpha\beta) = \alpha\alpha\alpha$ but $CT_{\rightarrow}^{-1}(\alpha\alpha\alpha) = \alpha\beta\alpha \neq \alpha\alpha\beta$. And in the last case let's consider $CT_{\leftarrow}(\alpha\beta \rightarrow \alpha\alpha)$ and $CT_{\leftarrow}^{-1}$ over the message $m = \alpha\beta\alpha$: $CT_{\rightarrow}(\alpha\alpha\beta) = \alpha\alpha\alpha$ but $CT_{\rightarrow}^{-1}(\alpha\alpha\alpha) = \alpha\beta\beta \neq \alpha\alpha\beta$.

Let $f_0(\alpha\gamma, m) = 0$ is a number of occurrences of particular digram $\alpha\gamma$ in a message m, then the following corollary tells us how many digrams $\alpha\gamma$ is introduced by context transformation $CT_{\rightarrow}(\alpha\beta \rightarrow \alpha\gamma)$:

Corollary A1. *Under assumption that* $\alpha \neq \gamma$ *the number of occurrences of digrams* $\alpha\gamma$ *and* $\alpha\beta$ *after application of transformation* $CT_{\rightarrow}(\alpha\beta \rightarrow \alpha\gamma)$ *is* $f_1(\alpha\gamma, CT(m)) = f_0(\alpha\beta, m)$ *and* $f_1(\alpha\beta, CT(m)) = 0$.

Proof. A proof is a consequence of Theorem A1, since each replacement is independent of each other and so each digram $\alpha\beta$ is replaced by $\alpha\gamma$ leaving $f_1(\alpha\beta) = 0$ and $f_1(\alpha\gamma) = f_0(\alpha\beta)$. □

The corollary allows us to precisely predict not only the frequencies of the interchanged digrams $\alpha\beta$ and $\alpha\gamma$ but also as a consequence the frequencies of individual symbols after transformation. The special case of transformations on a diagonal (see Definition A1) will be discussed in the next paragraph.

Appendix A.2. Diagonal Context Transformation

Diagonal transformation is a transformation where one of the digrams participating in the transformation is of the form $\alpha\alpha$. The resulting frequency of such a digram is unpredictable without

knowledge of the distribution of all n-grams of the form α^n, where $n \geq 2$, but we show that for any diagonal CT, it is possible to predict frequencies of symbols α and β. The problems with predictability arise from the repetition of symbols.

Definition A1. *Diagonal context transformation is a context transformation of the form* $CT_{\leftarrow}(\alpha\alpha \rightarrow \alpha\beta)$.

Consider two transformations, $CT_1 \equiv CT_{\leftarrow}(\alpha\alpha \rightarrow \alpha\beta, \alpha\alpha\alpha) = \alpha\beta\beta$ and $CT_2 \equiv CT_{\leftarrow}(\alpha\beta \rightarrow \alpha\alpha, \alpha\beta\alpha) = \alpha\alpha\alpha$, if Corollary A1 would also be valid for diagonal transformations, then for instance in the case of CT_1, the frequency $f_1(\alpha\beta) = f_0(\alpha\alpha)$ but this obviously is not true, instead we see that the new frequency $f(\beta)$ of symbol β is $f_1(\beta) = f_0(\alpha\alpha)$.

Suppose we have a message $s = \alpha^n$, then $CT_1(s) = \alpha\beta^{n-1}$, we clearly see that the frequency $f(\alpha\beta, CT_1(s)) = 1$ and $f(\beta\beta, CT_1(s)) = n - 2$, because the number of digrams in a message is given by the length of the message minus one. We can now express the frequency $f(\alpha\beta, CT_1(m))$ of the newly introduced occurrences of digram $\alpha\beta$ as a sum of all sub-messages enclosed in m in the form xsx, where $x \neq \alpha$ for all $n \geq 2$. So we see that it is possible to precisely predict the change of frequency of $\alpha\beta$, but it demands knowledge of the distribution of all enclosed sub-messages s.

From the other perspective, since each occurrence of digram $\alpha\alpha$ in the former message is transformed into $\alpha\beta$ we can see that the frequency $f_1(\beta, CT_1(m)) = f_0(\beta) + f_0(\alpha\alpha)$ and $f_1(\alpha, CT_1(m)) = f_0(\alpha, m) - f_0(\alpha\alpha)$.

Very similar behavior is observed in the second case of CT_2. The problem is in the repetition of the pattern $t = (\alpha\beta)^n$, then $CT_2(t) = \alpha^{2n}$ and $f_1(\alpha\alpha) = 2n - 1$. Again without knowledge of all sub-messages t enclosed in m we cannot predict the exact change of frequency of neither digram $\alpha\alpha$ nor $\alpha\beta$, but since we know that each pair $\alpha\beta$ in the former message will be transformed to $\alpha\alpha$, we can again precisely predict frequencies of individual symbols $f_1(\alpha) = f_0(\alpha) + f_0(\alpha, \beta)$ and $f_1(\beta) = f_0(\beta) - f_0(\alpha\beta)$.

With the knowledge of the preceding discussion and of Corollary A1 we conclude that for any context transformation CT we are able to compute the frequency and corresponding probability of arbitrary symbol after application of any CT from the knowledge of initial distribution of symbols and digrams. In [26] we showed that under certain conditions it is possible to process several context transformations simultaneously.

Appendix A.3. GCT—Frequencies Alteration

Corollary A2. *Under assumption that* $\alpha \neq \gamma$, $\alpha \neq \beta$ *and* $\beta \neq \gamma$ *the number of occurrences of digrams* $\alpha\gamma$ *and* $\alpha\beta$ *after application of transformation* $GCT_{\leftarrow}(\alpha\beta \leftrightarrow \alpha\gamma)$ *is* $f_1(\alpha\gamma, GCT(m)) = f_0(\alpha\beta, m)$ *and* $f_1(\alpha\beta, GCT(m)) = f_0(\alpha\gamma, m)$.

Proof. Since each digram $\alpha\beta$ resp. $\alpha\gamma$ is replaced by $\alpha\gamma$ resp. $\alpha\beta$, and neither of the digrams influence the transformation of the other, their frequencies must interchange. □

Appendix A.4. Generic Transformation—Proof of Correctness

Generic transformation GT exchanges any two digrams. In the design of algorithms, we prefer GCT over GT since the space from which generic transformations are selected is in this case of order the $|\Sigma|^4$ and when alphabets of the large size are dealt with, the search in such a space would be computationally very expensive.

Definition A2. *Generic transformation (GT) is the mapping* $GT(\alpha\beta \leftrightarrow \gamma\rho, w) : \Sigma^n \rightarrow \Sigma^n$, Σ *is the alphabet of the input message* w *and* n *is the length of the input message, that exchanges all digrams* $\alpha\beta$ *for digram* $\gamma\rho$ *and vice-versa.*

The inverse transformation of GCT and GT is defined by the following theorem:

Theorem A2. *Generic transformation* $GT^{-1} \equiv GT_{\leftarrow}(\alpha\beta \leftrightarrow \gamma\rho)$ *resp.* $GT^{-1} \equiv GT_{\rightarrow}(\alpha\beta \leftrightarrow \gamma\rho)$ *is the inverse of generic transformation* $GT_{\rightarrow}(\alpha\beta \leftrightarrow \gamma\rho)$ *resp.* $GT_{\leftarrow}(\alpha\beta \leftrightarrow \gamma\rho)$

Proof. First, we show that it is sufficient to prove that for any string $s = xwx$, it holds that $GT^{-1}(GT(s)) = s$, where $x \notin \Sigma^{GT} = \{\alpha, \beta, \gamma, \rho\}$ and $w[i] \in \Sigma^{GT}$. Suppose that x is located in position p then for digrams d in positions $(p-1, p)$ and $(p, p+1)$ it holds that $GT(d) = d$. So the first possible application of GT can occur in positions $(p-2, p-1)$ and $(p+1, p+2)$ and these are independent, i.e., non-overlapping.

Next, we show that each replacement made in the forward transformation will be reverted back by inverse transformation. Take for example transformation $GT_{\leftarrow}(\alpha\beta \leftrightarrow \gamma\rho)$ the transformation is applied in the right to left direction. The last applied forward transformation in positions $(r, r+1)$ replaces for instance digram $\alpha\beta$ for $\gamma\rho$ leaving $w[r, r+1] = \gamma\rho$, the inverse transformation, by definition the same transformation applied in the opposite direction, reverts digram $\gamma\rho$ back to $\alpha\beta$. Now consider any triplet of positions $(r-1, r, r+1)$ in a transformed message, the input of the inverse transformation in $(r, r+1)$ is dependent on the result of the inverse transformation in the preceding pair of positions, but as we saw the first applied inverse reverted digram back correctly so the state in positions $(r+1, r+2)$ is exactly like the one of the state left by forward transformation in these positions, so any other digram will be reverted back correctly, because every preceding application of the inverse leaves the state of the digram in the state that was left by the forward transformation and this digram is trivially reverted back to initial state. The same rules are valid for GT in the opposite direction, since the transformation $GT_{\leftarrow}(m) = GT_{\rightarrow}(m^T)$, where m^T is a mirror message of m. □

Appendix A.5. HOCT—Proof of the Correctness

The following trivial Lemma will help us to formulate a theorem about inverse transformation to *HOCT*:

Lemma A1. *Let* $T = HOCT(w\beta \leftrightarrow w\gamma, m, P(w, m))$ *is a higher order context transformation over the input message* m*, given that we possess the knowledge of* w *and positions* $P(w, m)$*, then* $T^{-1} = T$.

Proof. Because we don't have to pass through the whole message either in the forward or inverse transformation case, but only through the set of positions $P(w, m)$, then the symbol in position $i \in P(w, m)$, for instance $m[i] = \beta$ will switch by *HOCT* to $m[i] = \gamma$ and by repeated application of *HOCT* it reverts back to $m[i] = \beta$. □

The Lemma A1 is trivial but comes into play when $P(w, m)$ is a product of some other higher order context transformation, i.e., the one with an order lower by one.

Theorem A3. *Let* $m_1 = HOCT_1(w\alpha \leftrightarrow w\beta, m, P(w, m))$ *and* $m_2 = HOCT_2(w\alpha\gamma \leftrightarrow w\alpha\rho, m_1, P(w\alpha, m_1))$ *are two higher order context transformations. Let* $T(m) = HOCT_2(HOCT_1(m))$ *be a transformation composition of two higher order context transformations over input message* m*. Then* $HOCT_2^{-1}(w\alpha\gamma \leftrightarrow w\alpha\rho, m_3, P(w\beta, m_3))$*, such that* $m_3 = HOCT_1(m_2))$ *then the transformation composition* $T^{-1} \equiv HOCT_2^{-1}(HOCT_1(m_2)) = m$ *is the inverse transformation of* T.

Several remarks to the formulation of Theorem A3: Transformations $HOCT_1$ and $HOCT_2$ are applied over two consecutive states of the message. The positions $P(w\alpha, m_1)$ correspond to the positions $P(w\beta, m)$, since sub-messages $w\alpha$ have been replaced by $w\beta$ in the application of $HOCT_1$. The inverse transformation by $HOCT_2^{-1}$ is applied instead over positions $P(w\beta, m_3)$, since these positions have already been reverted back by $HOCT_1$.

The proof is based on the restriction that $w[0] \neq w[i]$, $i > 0$, it can be viewed as we would split the input message m to sub-messages s_i separated by $w[0]$. For instance, suppose that $w[0]$ is a space character in ordinary text, since, by Definition 3, no other character in w can be a space character,

it follows that the possible transformations are being applied on words following the space character. Now using the fact that s_i is enclosed by $w[0]$, i.e., they do not overlap, allows us to handle each sub-message s_i independently.

Proof. For the two sets of positions, it holds that $P(w,m) \cap P(w\alpha, m_1) = \emptyset$, because elements of the former are predecessors of the latter and s does not overlap. The locations of w in m and m_2 are identical as they were not modified during $HOCT$, i.e., $P(w,m) = P(w,m_2)$. When we apply $HOCT_1$ again it will simply revert the symbols in positions given by $P(w,m)$ back according to Lemma A1 yielding the message state m_3. In the forward transformation $HOCT_2$ was applied over positions of $P(w\alpha, m_1)$, but these are the former positions of $P(w\beta, m)$, that are already transformed back by the application of $HOCT_1$, so $P(w\alpha, m_1)$ is equal to $P(w\beta, m_3)$ and when $HOCT_2$ is applied over positions $P(w\beta, m_3)$ it exchanges symbols γ and ρ and eventually yields m. □

The recursive application of Theorem A3 leads to the conclusion that this process can be repeated until there is no other pair of symbols then these containing $w[0]$ as one of the symbols α or β or we simply reach the end of the message.

Corollary A2 about the prediction of frequencies in the case of GCT is also applicable in the case of $HOCT$, because the principle that the exact number of replacements is known is also valid and we are able to precisely compute the future probabilities of symbols before the arbitrary $HOCT$ is applied.

If we implement the inverse algorithm as a sequential algorithm operating in the left-right manner, it is possible to have one of the transformation symbols if β, γ is equal to $w[0]$. Suppose the following example: $m = abcabc$, $P(a,m) = \{0,3\}$, $HOCT_1(ab \leftrightarrow aa)$ and $HOCT_2(aac \leftrightarrow aaa)$ yielding the output message $m_2 = aaaaaa$. Now applying inverse transformation sequentially from left to right, we first replace aa by ab yielding $m_i = abaaaa$, then applying replacement aba for abc yielding $m_{i+1} = abcaaa$, now because there is no other transformation that is induced from abc we know that the next a symbol is $w[0]$ and we can repeat the preceding process again starting from this a. The sufficient condition for the introduction of $w[0]$ as the transformation symbol β or γ is that w contains no other $w[0]$ in $w[i]$, $i > 0$, because the inverse process removes all introduced $w[0]$ symbols from the transformed message during left to right sequential inverse transformation.

References

1. Cover, T.M.; Thomas, J.A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*; Wiley-Interscience: New York, NY, USA, 2006.
2. Shannon, C.E. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423.
3. Huffman, D.A. A Method for the Construction of Minimum-Redundancy Codes. *Proc. Inst. Radio Eng.* **1952**, *40*, 1098–1101.
4. Witten, I.H.; Neal, R.M.; Cleary, J.G. Arithmetic Coding for Data Compression. *Commun. ACM* **1987**, *30*, 520–540.
5. Charikar, M.; Lehman, E.; Lehman, A.; Liu, D.; Panigrahy, R.; Prabhakaran, M.; Sahai, A.; Shelat, A. The Smallest Grammar Problem. *IEEE Trans. Inf. Theory* **2005**, *51*, 2554–2576.
6. Nevill-Manning, C.G. Inferring Sequential Structure. Ph.D. Thesis, University of Waikato, Hamilton, New Zealand, May 1996.
7. Nevill-Manning, C.G.; Witten, I.H. Identifying Hierarchical Structure in Sequences: A Linear-time Algorithm. *J. Artif. Int. Res.* **1997**, *7*, 67–82.
8. Kieffer, J.C.; Yang, E.-H. Grammar Based Codes: A New Class of Universal Lossless Source Codes. *IEEE Trans. Inf. Theory* **2000**, *46*, 737–754.
9. Ziv, J.; Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory* **1978**, *24*, 530–536.
10. Yang, E.; He, D. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform 2. With context models. *IEEE Trans. Inf. Theory* **2003**, *49*, 2874–2894.
11. Gage, P. A New Algorithm for Data Compression. *C Users J.* **1994**, *12*, 23–38.

12. Nakamura, H.; Marushima, S. Data Compression by Concatenation of Symbol Pairs. In Proceedings of the IEEE International Symposium on Information Theory and Its Applications, Paris, France, 13–17 September 1996; pp. 496–499.
13. Larsson, N.J.; Moffat, A. Off-line dictionary-based compression. *Proc. IEEE* **2000**, *88*, 1722–1732.
14. Claude, F.; Farina, A.; Navarro, G. Re-Pair Compression of Inverted Lists. *arXiv* **2009**, arXiv:cs.IR/0911.3318.
15. Masaki, T.; Kida, T. Online Grammar Transformation Based on Re-Pair Algorithm. In Proceedings of the Data Compression Conference (DCC), Snowbird, UT, USA, 29 March–1 April 2016; pp. 349–358.
16. Grassberger, P. Data Compression and Entropy Estimates by Non-sequential Recursive Pair Substitution. *Physics* **2002**, arXiv:physics/0207023.
17. Calcagnile, L.M.; Galatolo, S.; Menconi, G. Non-sequential recursive pair substitutions and numerical entropy estimates in symbolic dynamical systems. *arXiv* **2008**, arXiv:cond-mat.stat-mech/0809.1342.
18. Navarro, G.; Russo, L. Re-pair Achieves High-Order Entropy. In Proceedings of the Data Compression Conference, DCC 2008, Snowbird, UT, USA, 25–27 March 2008; p. 537.
19. Vasinek, M.; Platos, J. Entropy Reduction Using Context Transformations. In Proceedings of the Data Compression Conference (DCC), Snowbird, UT, USA, 26–28 March 2014; p. 431.
20. Vasinek, M.; Platos, J. Generalized Context Transformations—Enhanced Entropy Reduction. In Proceedings of the Data Compression Conference (DCC), Snowbird, UT, USA, 7–9 April 2015; p. 474.
21. Vasinek, M.; Platos, J. Higher Order Context Transformations. *arXiv* **2017**, arXiv:cs.IT/1701.01326.
22. Kida, T.; Matsumoto, T.; Shibata, Y.; Takeda, M.; Shinohara, A.; Arikawa, S. Collage System: A Unifying Framework for Compressed Pattern Matching. *Theor. Comput. Sci.* **2003**, *298*, 253–272.
23. González, R.; Navarro, G. Compressed Text Indexes with Fast Locate. In Proceedings of the 18th Annual Conference on Combinatorial Pattern Matching, CPM'07, London, ON, Canada, 9–11 July; Springer: Berlin/Heidelberg, Germany, 2007; pp. 216–227.
24. Claude, F.; Farina, A.; Navarro, G. Re-Pair compression of inverted lists. *arXiv* **2009**, arXiv:0911.3318.
25. Vasinek, M. Kontextove Mapy a Jejich Aplikace. Master's Thesis, Vysoka Skola Banska—Technicka Univerzita Ostrava, Ostrava, Czech Republic, 2013.
26. Vasinek, M.; Platos, J. Parallel Approach to Context Transformations. Available online: http://ceur-ws.org/Vol-1343/paper4.pdf (accessed on 11 May 2017).