ON THE RECURSIVE PAIRING COMPRESSION ALGORITHM

RADU RĂDESCU, RADU MITOI*

Key words: Lossless compression, Dictionary-based algorithms, Re-Pair.

This paper is a short overview of the Re-Pair algorithm, used in lossless data compression of text files, with contributions in improving the data structures and the space-time analysis. Re-Pair is combined with transforms and compared in terms of performance to other popular algorithms.

1. **INTRODUCTION**

Dictionary based models are used in many compression methods [1]. One of the best examples is Lempel-Ziv family of algorithms that build, based on the input message, a dictionary consisting of the initial input phrases. Compression is achieved because the references to these phrases take less space than the original ones. In most implementations, the encoder operates *online*, updating the available phrase dictionary based on the input stream. The changes on the dictionary must be sent together with the phrases so that the decoder can also update his dictionary [2].

An alternative approach (implemented by Re-Pair algorithm [3]) is to create a dictionary in the beginning and then send it with the encoded file. Because the entire message is available, optimizations can be performed right from the start and certain phrases can be selected so that the compression will be improved. For the Re-Pair algorithm, only the compression is executed *offline*. The decompression can be done while sequences are received, as long as there is a dictionary available. That is why the Re-Pair algorithm requires a reduced amount of memory for decompression when compared to other dictionary-based methods [4].

To simplify the compact phrase encoding a hierarchical schema is used where longer phrases are coded by references to shorter ones. The process is similar to Lempel-Ziv algorithms, but has the advantage of using at least twice any phrase in the dictionary. The Re-Pair algorithm was developed by Moffat and Larsson [4]. It is an *offline* algorithm that begins compression only after the entire message was received. This is no longer a problem since computer memory has dramatically increased lately and can store a larger amount of information for processing. The decoding is also far more common that the

* "Politehnica" University of Bucharest; E-mail: radu.radescu@upb.ro, mitoi.radu@gmail.com

Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., **57**, *4*, p. 401–411, Bucarest, 2012

compression so using an algorithm that is slow for coding, but offers a fast decoding is preferred to the algorithms that offer a fast encoding but a slower decoding. It can be considered that the time passed for compression is regained during decompression.

Re-Pair transmits the dictionary explicitly as a distinct sequence, compared to Lempel-Ziv77 that sends incremental results as the compression string is built. This results in two outputs: a dictionary and a reference sequence to dictionary entries. These two parts are independently coded and transmitted to the decompressor (Des-Pair) as two distinct streams of data. With these streams, the original message can be recovered [2].

1.1. THE RE-PAIR MODEL

Re-Pair reduces message length by replacing repeating symbol pairs with a new symbol that is not present in the original message. Initially, the most frequent pair is localized and all its appearances are replaced with a new symbol. Every replacement reduces the message length by 1 symbol. The new symbol is added to the dictionary together with the pair that it replaces. The process then repeats and the next most frequent pair goes through the same algorithm (including the new symbols). The process stops when every pair appears only once in the dictionary.

Let (Σ) be the initial alphabet composed of *primitives*. Phrases created and added by Re-Pair in the dictionary will be referenced by ρ. Phrases and primitives can be generically be called symbols. Every symbol α has a generation $g(\alpha)$ in phrase hierarchy. $g(\alpha)$ indicates the depth of a symbol in phrase hierarchy:

$$g(\alpha)=\begin{cases} 0, \text{ if } \alpha \text{ is a primitive} \\ 1+g\left(\max\{g(\beta),g(\gamma)\}\right), \text{ if } \alpha \text{ is not a primitive} \end{cases}, \quad (1)$$

where β, γ are symbols of $\alpha = \beta\gamma$ phrase.

The output of Re-Pair algorithm consists of a dictionary and a sequence of references to its inputs. The dictionary contains all primitives and phrases created during the recursive pairing. Because every phrase has two symbols from a previous generation, the dictionary resembles a phrase hierarchy. The generated dictionary will be referenced as *P* (*P*hrase Hierarchy) and the list of pointers to P's symbols is called a reduced sequence (S).

1.2. THE RE-PAIR COMPRESSION ALGORITHM

The steps of the Re-Pair algorithm are:

Step 1. The input message is read and a list of all pairs of symbols is generated.

Step 2. The most frequent pairs are replaced with a symbol that is not in the original stream. If there is no pair that repeats then the algorithm stops.

Step 3. The list of symbols is rebuilt and the process restarts from *Step* 1 until there will be no pair that repeats.

Example. Given the sequence: S = ABABABABCBCCCCDDCABABABAAA, the list of all pairs in the sequence is in Table 1.

Table 1

Number of repetitions for 0-generation pairs of S sequence

Pair	Number of repetitions	Generation
AB	7	0
BA	6	0
BC	2	0
CC	2	0
CD	1	0
DD	1	0
DC	1	0
AA	1	0

The most frequent pair is AB so it will be replaced with a symbol that is not part of the original message: **a**.

The new sequence will be: S = **aaaa**CBCCCCDD**aaa**AAA. The next step is to rebuild the list of pairs (including the new symbol **a**). The most frequent pair is: **aa**. It will be replaced by **b**.

The sequence will become: **bb**CBCCCCDDC**ba**AAA. The list of pairs is rebuilt (including the new symbol **b**). The process continues until there will be no repeating pairs. The reduced sequence is obtained: S = **bb**CB**cc**DDC**ba**AAA.

The resulting dictionary is in Table 2.

Table 2

The dictionary obtained after applying Re-Pair algorithm to S sequence

Phrase (Pair)	Reference (Code word)
AB	a
aa	b
CC	c

1.3. THE DECOMPRESSION ALGORITHM (DESPAIR)

Given the reduced sequence S and the dictionary, the decompression algorithm is:

Step 1. The dictionary is read in referred order (the phrase hierarchy is processed starting from the last generating to 0-generation phrases) and one reference is selected

Step 2. In the reduced sequence, all symbols that match the reference are replaced with the phrase in the dictionary corresponding to the reference.

Step 3. The process is repeated until all references in the dictionary are replaced in the reduced sequence.

Example. Given the sequence S = **bb**CB**cc**DDC**ba**AAA and the dictionary previously obtained. The last reference in the dictionary is **c** → CC. This will be replaced in the reduced sequence: S = **bb**CBCCCCDDC**ba**AAA. The next reference is **b** → **aa**. The reduced sequence becomes: S = **aaaa**CBCCCCDDC**aaa**AAA. The next reference is **a** → AB. The reduced sequence becomes: S = ABABABABCBCCCCDDCABABABAAA. All references in the dictionary were processed and the decompression algorithm is complete.

2. IMPLEMENTATION

Like in all compression systems, there are various implementation methods for the Re-Pair algorithm, each of them offering a compromise between compression speed and compression quality. In this particular case the software was implemented using C# language.

2.1. THE DATA STRUCTURES

2.1.1. The reduced sequence

The reduced sequence is implemented with a one-dimensional array list Fig. 1. Each node of the list contains a symbol. The reason behind using a list is the possibility of deleting and inserting nodes without the need of copying the content of the sequence to a temporary memory location as in the case of using a simple array. Re-Pair can perform delete and insert node operations over this data structure.

Another advantage of the array list is the way the memory is allocated. For arrays, a continuous memory zone is required to be statically allocated. For the array list, the memory is dynamically allocated as new elements are added to the list. The disadvantage of the array list over the static arrays is the increased access time to the elements. Because a non-continuous memory space is used, for finding an element in the list a full element scan is required. For the arrays a memory location and an *offset* is sufficient to point to identify any element.

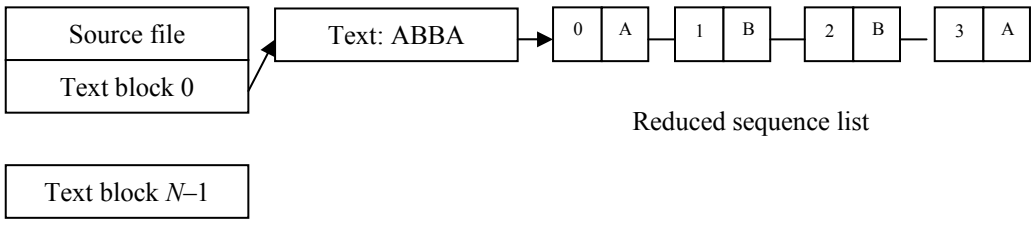

Fig. 1 – Building the reduced sequence list from the source file.

The paper proposes to build the initial reduced sequence upon reading a block of text from a file (Fig. 1). According to this new procedure, the corresponding block is inserted into a string and each character is inserted into a list node.

2.1.2. The phrase array

The phrase array is a bi-dimensional array consisting of the frequencies of the pairs for each text block inserted in the sequence list. The phrase array is built using the encoding list previously described:

Step 1. A $|\Sigma| \times |\Sigma|$ m array is initialized, where $|\Sigma|$ is the alphabet size.

Step 2. The encoding list is processed and for each $a_i a_j$ pair:

- If $a_i = a_j$, $m[(int)a_i,(int)a_j]$ is incremented, where $int(x)$ is the integer value of (x) character. (Ex.: ASCII code for A is 65, so $int(A) = 65$). Then the pair $a_{i+2}a_{i+3}$ is verified.
- If $a_i \neq a_j$, $m[(int)a_i,(int)a_j]$ is incremented and $a_{i+1}a_{i+2}$ is verified.

Step 3. Step 2 is repeated until the entire list is processed.

Building this way the phrase array prevents the double counting of consecutive character pairs. This particular case is not part of the original paper of Moffat and Larsson because it does not have a great effect on compression in most cases [2]. Because the purpose of this article is to study compression methods combined with text transforms for specific data structures, neglecting this aspect might have a significant effect on compression quality. Finding the most frequent pair consists in finding m's maximum. In the current example, the maximum equals 7, which correspond to **AB** pair. The present paper states that the operation that is performed on the phrase array consists in updating pair's symbol frequency. This operation depends on the sequence context, as proved in the present section. The context only depends on the previous and next symbol related to the current pair.

2.1.3. The encoding alphabet

Recursive pairing algorithm replaces pairs of characters with a new symbol that does not exist in the original message. To accomplish this it is required that the original alphabet is known in advance so that the space of the symbols that can replace phrases can be computed. As an initial alphabet, Re-Pair will be implemented using extended ASCII (256 characters or 8 bits each). The text is processed and symbols that were used at least once are eliminated from the ASCII encoding alphabet. The remaining characters form the encoding alphabet.

Example.

Given the sequence: S = ABABABABCBCCCCDDCABABABAAA Its alphabet is: Σ_S= {A, B, C, D}. The encoding alphabet will be: Σ_C = ASCII – Σ_S.

The problem of this alphabet is what happens if all the symbols of the ASCII code appear in the original message. In this case, the pairs cannot be replaced with any symbol so no compression will result. Using symbols outside the alphabet scope

(that require more bits to represent) will also result either into an uncompressed text or an expanded text. The newly proposed solution to this problem is to reduce the length of the text blocks so that that they will not contain all ASCII characters. Statistically, any combination with lesser characters will help to improve the process.

Just like the encoding list, the encoding alphabet is implemented using an array list. Each node of the list contains one alphabet symbol. Over this list, two operations are performed: delete and insert of a node.

2.1.4. The dictionary

The dictionary is built during the encoding process to be used in decoding of files. For implementing it two array lists are required. The first list contains the symbols that replace pairs in the reduced sequence. These symbols are named *tokens*, and the list containing them: *token list*. The second list contains pairs of characters referred by the tokens and is called *phrase list*. Each time a pair is replaced in the reduced sequence; inserting operations are performed on the lists of tokens and phrases. Inserts are performed so that the last one corresponds to the maximum generation tokens. Lists are linked together using their position indexes.

2.2. THE SPACE ANALYSIS

Space analysis is a theoretical estimation of the required memory amount for the Re-Pair algorithm. The analysis computes the used space by each data structure, based on the input data. It is assumed that the input sequence has a length of n symbols. Each symbol represents a memory word. The encoding list that memorizes this sequence will also have n words. The phrase array will have a fixed size, equal to the square of the alphabet size. If extended ASCII is used then the size will be equal to 256×256 bytes = 64 Kbytes. The encoding alphabet will at most contain 256 entries of 1 byte each and the dictionary at most $256 \times 3 = 0.75$ Kbytes. Therefore, the dominant factor that determines the amount of memory is the reduced sequence size. In addition, an increase in alphabet size determines a square increase in phrase array size.

2.3. THE TIME ANALYSIS

Time analysis is realized based on the encoding algorithm. At first, an initial pass over the sequence is made and the repeating symbols are identified. The most time-consuming step of the algorithm consists in replacing the previously identified pairs with new symbols. This step does two deletions and one insert into the encoding list. Because array lists are used, this operation has $O(1)$ complexity. The file is passed multiple times so the encoding time will increase based on the number of steps performed by the algorithm and is strongly dependent on the source file structure. The ideal case consists of a file with identical symbols. For such a file, the

number of passes has the minimum value. Therefore, the entire Re-Pair algorithm is executed into an amount of time that increases linearly with the input sequence.

2.4. THE RE-PAIR ARCHIVE STRUCTURE

In order to be decoded, compressed files (archives) must be recognized by the decoded. This involves a certain structure of the archive. The Re-Pair archive has the structure shown in Table 3. Because the encoding is made upon blocks of text, the resulting archive will also contain such blocks. Each block contains the number of entries of the dictionary, the dictionary, the number of characters of the encoded block and the encoded block. The first block will also contain the number of blocks of the initial file. To decode, the number of blocks in the archive is first read. This is an integer number between 0 and 10^{10}–1, memorized like a string: 00000XXXXX, where XXXXX represents the number of blocks.

Table 3

Re-Pair archive structure

Block 0	Block 1	…	Block *N*–1
Number of blocks (integer between 0 and 10^{10}–1)	…	…	Number of blocks (integer between 0 and 10^{10}–1)
Number of dictionary entries (integer between 0 and 10^{10}–1)	…	…	Number of dictionary entries (integer between 0 and 10^{10}–1)
Dictionary	…	…	Dictionary
Number of characters in the encoded text (integer between 0 and 10^{10}–1)	…	…	Number of characters in the encoded text (integer between 0 and 10^{10}–1)
Encoded text	…	…	Encoded text

3. **EXPERIMENTAL RESULTS**

In this chapter the Re-Pair algorithm is tested and the results are analyzed and compared with other compression algorithms to establish its performance and the optimum conditions in which the algorithm should be used.

3.1. THE TESTING ENVIRONMENT

The compression/decompression experiments were executed on an Intel Core2 processor at 2133 MHz with 2 MB cache memory and 2GB of RAM memory operating at 800 MHz. The standard testing corpora is *Calgary* and *Canterbury Corpora.* The corpora consist of a file collection specially designed to testing lossless compression algorithms. The *Calgary* corpus is made out of 18 text files that have more than 3.2 million bytes of data. *Calgary* corpus was built in

1987 by researches to develop, test and compare different compression methods. Because a long time it was the only testing corpus used, it was assumed that some algorithms were built so that they are optimum only for it. The standard files also changed so a new corpus was necessary: *Canterbury*. A third corpus was used to test the effects of compression over other files. These were grouped in an *artificial* corpus.

3.2. THE COMPRESSION RATIO

Different measures of performance can be chosen to analyze a compression algorithm. One of the most frequently used is the compression ratio, which represents the size of the input file divided by the size of the output file.

After encoding different files the following results were obtained:

Table 4

Compression ratio for different test files

Text	Compression ratio	Text	Compression ratio
bib	1.520582206	plrabn12	1.541777587
book1	1.457866671	XML	2.76364362
book2	1.438089879	bible	1.767138964
news	1.266409876	E.coli	2.656328113
paper1	1.396217991	world192	1.471057914
progc	1.431550416	random	0.998422492
progl	1.729201361	biblioteca	2.6695758
progp	1.707493343	codul penal	1.834030827
trans	1.559971363	luceafărul	1.48630137
alice29	1.57968591	programare	1.508632881
asyoulik	1.511057193	site atm	5.676609573
cp	1.587904995	plrabn12	1.541777587
fields	1.667165072	XML	2.76364362
grammar	1.583404255	bible	1.767138964

Good results were obtained for the website source, XML and E.Coli. For most of the other files, the compression ratio is close to 1.5 and is a result of a relatively low number of repeating pairs. Literary and scientific texts do not usually have a repeating pattern therefore the repeating pairs have a low frequency.

The only file that does not achieve compression is „random.txt" which contains a pseudorandom sequence with a sufficient non-repeating period so that a pattern is not achieved. These types of files are usually incompressible and because the encoded information is stored with decoding information (the dictionary for example), the output files are larger than the input files (they are expanded).

The following results can be concluded:

1. If the initial alphabet is reduced, the compression ratio will increase.

2. If in the original sequence there are some primitives with a significantly higher frequency that the others, then is possible to obtain a high compression ratio.

To demonstrate these affirmations two tests were performed over files in the *Artificial Canterbury* corpus. These files have special properties over "natural" files. The "aaa.txt" file contains the letter "a" repeated 100 000 times and "alphabet.txt" contains the alphabet letters repeated until 100 000 characters are achieved. The following compression ratios were obtained: 298.5074627 and 158.7301587, respectively.

A lossless compression ratio of over 100 is considered huge. The phenomenon appears because of the particular file structure. In "aaa", with a single pass 50000 pairs are replaced with a character from the encoding alphabet. The process is repeated until there are no repeating pairs. At each pass the file size is halved because the new introduced symbols will also form repeating pairs. The same thing happens with "alphabet.txt" because of the repeating pattern of characters.

3.3. COMPARISON BETWEEN DIFFERENT COMPRESSION ALGORITHMS

The algorithm performance is compared with other encoding algorithms. The testing algorithms were: arithmetic encoding [5], Huffman combined with Lempel-Ziv and a commercial algorithm that uses different methods to optimize performance (WinRAR). The following results were obtained:

Table 5

Average compression ratio

Text	**Re-Pair**	**WinRAR**	**Lempel-Ziv**	**Arithmetic**
Bib	1.520582206	3.303030303	2.595238095	1.79
book1	1.457866671	2.771217712	2.091922006	2.02
book2	1.438089879	3.372881356	2.4875	1.87
alice29	1.57968591	2.923076923	2.375	1.99
Asyoulik	1.511057193	3	2.4375	1.89
E.coli	2.656328113	2	2	4
world192	1.471057914	3.336	2.452941176	1.85
biblioteca	2.6695758	6.25	3.85	1.9
codul penal	1.834030827	4.82	3.36	2.1
luceafărul	1.48630137	2.1	2	1.57
programare	1.508632881	3.333	2.8	1.87
Average	1.836419417	7.189000576	2.841045205	1.908076923

The Re-Pair compression algorithm is outperformed by other algorithms in most cases (with the exception of E.coli and XML). The algorithm with the highest compression ratio is WinRAR followed by Lempel-Ziv and arithmetic encoding. This result combined with the high encoding time makes Re-Pair an inefficient

algorithm when used without transforms. When applying the text transforms, the medium ratio is close to 2.6, which is superior to arithmetic and Lempel-Ziv encoding. Re-Pair algorithm is inferior to other compression algorithms but can be improved by using pre-compression transforms. For certain file types (web pages, XML), this encoding method produces great results.

4. **CONCLUSION**

Certain advantages and disadvantages can be outlined after analyzing the algorithm behavior in all experimental cases. Advantages:

- The decompression algorithm is simple, easy to implement and very fast.
- A good compression quality is obtained for most file types if a preprocessing transform (Burrows Wheeler [6] or Burrows-Wheeler followed by Move to Front) is applied to the original file.
- For certain files, (XML, HTML, source files, files with repetitive structure) the compression quality is impressive when used with BWT and MTF [7]. The compression ratio is sometimes far superior to the classical compression methods [8].
- Best results for periodic sequences.
- Memory amount and execution time can be controlled.
- The algorithm can be easily be parallelized so the execution time can be decreased.

Disadvantages:

- Compression time is high. This is the primary disadvantage of the algorithm. However, the encoding is *offline* and is usually performed only once so it is necessary that only the decompression must be fast (because it is executed numerous times). For the Re-Pair algorithm, the decompression time is close to other compression methods [9] therefore this disadvantage is minimized.
- The algorithm is context sensitive. Therefore, prior information is required about the input file [10] if a good compression quality is needed [11].

The present paper proposes as original contribution the new data structure with a reduced sequence list and a new procedure to build the phrase array accordingly.

Received on March 15, 2011

REFERENCES

1. D. Salomon, *Data Compression – The Complete Reference*, 3rd Edition, Springer, 2003.
2. R. Wan, *Browsing and Searching Compressed Documents*, Department of Computer Science and Software Engineering, The University of Melbourne, 2003.
3. G. Navarro, L. Russo, *Re-Pair Achieves High-Order Entropy*, Proc. DCC'08, p. 537 (poster).
4. N. Jesper Larsson, Alistair Moffat, *Offline Dictionary-Based Compression*, Proc. DCC'99, pp. 296.

5. E. Bodden, M. Clasen, J. Kneis, *Arithmetic Coding revealed – A guided tour from theory to praxis*, 2005; http://www-users.rwth-aachen.de/ Eric.Bodden/files/ac/ac_en.pdf
6. M. Dipperstein, *Burrows-Wheeler Transform – Discussion and Implementation*, 2004; http://michael.dippenstein.com/bwt/
7. R. Rădescu, *Lossless Compression – Methods and Applications*, Matrix Rom, Bucharest, 2003
8. M. Nelson, *The Data Compression Book*, 2nd Edition, M&T Books, 1995.
9. G.R. Tamayo, *Run-length Transform*, 2008; http://compgt.googlepages.com/rlt
10. R. Rădescu, C. Bontaş, *Design and Implementation of a Dictionary-Based Archiver*, Scientific Bulletin, Electrical Eng. Series C, "Politehnica" University of Bucharest, **70**, *3*, pp. 21-28, 2008.
11. R. Rădescu, *Transform Methods Used in Lossless Compression of Text Files*, Romanian Journal of Information Science and Technology, **12**, *1*, pp. 101-115, 2009.