KDS-transformation for data compression

IONUŢ POPA
Faculty of Computer Science,
"Al.I.Cuza" University of Iaşi,
700483 Iaşi,
ROMANIA

Abstract:- In this paper we present a new simple method to rewrite a given input string into another one which can be efficiently compressed (in some cases) by traditional compression methods.

Key-Words:- Data compression, Reversible transformation

1. Introduction

It is an well known observation in data compression theory that, given an input source, symbols with low probabilities are responsible for the high entropy of the input source. Lets call those symbols *rare* symbols and the others *dominant* symbols.

At least empirically, an input source with rare symbols would be more efficiently compressed if those symbols can be encoded somehow using dominant symbols. Obviously, this encoding should be completely reversible. We present here such a method, similar somehow with *difference encoding* and other transformation techniques used in image or sound compression.The fundamentals of this method can also be found in Huffman compression algorithm.

The paper is structured as follows: in the next section the basic idea of our transformation is presented. In Section 3, an analysis of this technique is presented. In the last sections some conclusions and remarks are also given.

2. KDS - transformation

KDS -transformation stands for *Keep the Dominant Symbols - transformation.* The advantage of this method is that those statistical properties of the input stream that can be efficiently used by any compression algorithm are not dramatically changed.

Consider the following procedure to build a Huffman code for an alphabet V, where p_a is the probability of appearance of symbol a in an input source:

- if $|V| = 2$ then $V = \{a, b\}$ and $code(a) = 0$ and $code(b) = 1$.
- if $|V| > 2$ then select from V two symbols with smallest probabilities (call them a and b). Then $code(a) = code(\#)0$, $code(b) = code(\#)1$, where $\#$ is a new symbol with $p_\# = p_a + p_b$, and $code(\#)$ is computed applying the same procedure with the alphabet $V \setminus \{a, b\} \cup \{\#\}$.

The same is the essence of our transformation, to replace symbols with low probabilities with a new symbol.

Further we present formally this transformation.

Consider A is an alphabet and w is a word over the alphabet A. Consider $A = R \cup D$ with $|R| = |D|$, where R is a subset of the alphabet A containing the "rare" symbols and D a subset containing the "dominant" symbols in the word w.

Is is easy to see that there is an unique decomposition of w:

$$w = r_1 d_1 r_2 d_2 \dots r_k d_k,$$

where:

- $r_i \in R^+$, $1 < i \leq k$;
- $r_1 \in R^*$;
- $d_i \in D^+$, $1 \leq i < k$;
- $d_k \in D^*$.

The first step of our transformation is rewriting w in w', where:

$$w' = \#^{|r_1|}d_1\#^{|r_2|}d_2 \ldots \#^{|r_k|}d_k,$$

where $\#$ is a new symbol, $\# \notin A$.

Consider now a bijective mapping $f : R \to D$. In the second step of the transformation the string w'' is obtained in the following way:

$$w'' = f(r_1)f(r_2)\ldots f(r_k)\#w'.$$

In the final step the new symbol $\#$ and an explicit representation of the mapping f are added at the beginning of string w''. This way the string w''' is obtained:

$$w''' = \#r_{i_1}f(r_{i_1})\ldots r_{i_{|R|}}f(r_{i_{|R|}})\#w'',$$

where $R = \{r_{i_1} \ldots r_{i_{|R|}}\}$.

The reverse transformation works in the following way: given a string w'''(obtained as above) consider the first symbol of w''', the separator $\#$. Consider now the following factorization of w''':

$$w''' = \#w_1\#w_2\#w_3,$$

where $w_1 \in A^+$, $w_2 \in R$ and $w_3 \in D \cup \{\#\}$. It is easy to observe that there is an unique such a factorization of w'''.

3. A Case Study

In this section we study the effect of this transformation combined with some traditional compression methods.

The length (in bits) of the Lampel-Ziv[2] code of a word $w \in V^*$ is:

$$LZ_w = \sum_{i=1}^{b} \lceil \log_2(ki) \rceil,$$

where b is the number of blocks resulting from LZ-parsing and k is the size of the alphabet of the string w.

Consider now the case of the alphabet A where $|A| = 2^p, p > 1$. Let u and v be two words, $u \in A^*$ and $v \in (A \cup \{\#\})^*$, v is obtained form u using the above transformation.

$$LZ_u = \sum_{i=1}^{b} \lceil \log_2(2^p i) \rceil$$

$$= bp + \sum_{i=1}^{b} \lceil \log_2(i) \rceil$$

which telescope[1] into

$$LZ_u = bp + b(\lfloor \log_2 b \rfloor + 1) + b2^{1-\log_2 b + \lfloor \log_2 b \rfloor} - 2,$$

where b is the number of blocks resulting from LZ parsing of the string u.

$$LZ_v = b'(p-1) + b'(\lfloor \log_2 b' \rfloor + 1) +$$

$$+ b'2^{1-\log_2 b' + \lfloor \log_2 b' \rfloor} - 2,$$

where b' is the number of blocks resulting from LZ parsing of the string v.

If $\frac{LZ_v}{LZ_u} > 1$ compression is improved by transformation presented above.

In Figure 1 the dark spot represent values for b and b' when compression can be improved

Moreover consider $A = R \cup D$, where $|R| = |D|$, and the probabilities of appearance of symbols form R in u is p_1, and the probabilities of appearance of symbols from D in u is p_2. In this case we can estimate the number b' using b, p_1 and p_2 and determinate the cases when $LZ_v \leq LZ_u$.

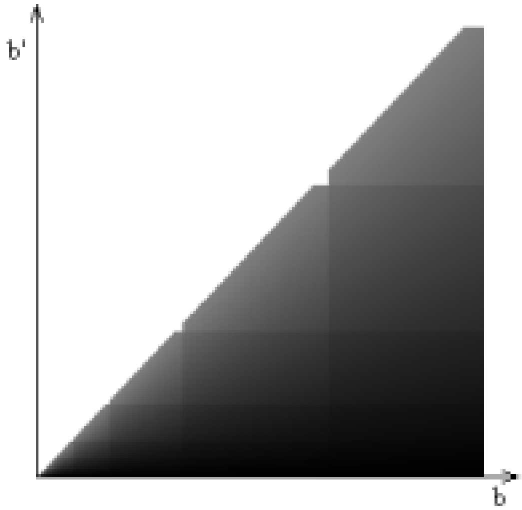

Fig. 1: Compression gain

4. Conclusion

The idea of this transformation can be also used to encode pointers/references in dictionary compression schemes.

An implementation of this algorithms improve compression from 5% up to 10% in combination with software products like *gzip* or *WinRAR 3.30*. In the following table some experimental results are presented:

Original size	WinRAR	KDS+WinRAR
104448	19926	**19756**
263680	**68084**	68527
285696	62008	**60657**
829440	211659	**202124**
913755	293157	**287293**
1016320	244126	**232476**
1150976	229790	**229350**
1691136	424372	**407384**
2156032	511538	**489263**
3259904	597233	**579094**
5751296	1321138	**1278913**
6125056	1168909	**1126271**
28844032	6409980	**6228254**

Table 1: Experimental results

The first column of Table 1 contains the size (in bytes) of original files. Test were performed using Java bytecode files. The second file contains the size of the files compressed using WinRAR 3.30. Finally, the last column contains files encoded first using KDS transformation and then WinRAR 3.30.

In most of the cases KDS transformation improved compression efficiency. It also works well on images, sounds or others files containing "rare" symbols.

References

[1] D. Knuth *The Art of Computer Programming: Fundamental Algorithms*, Vol. 1, Addison-Wesley, 1973

[2] J. Ziv, A. Lempel - *Compression of individual sequences via variable-rate coding.* IEEE Transactions on Information Theory 24, 1978. 530536.