INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11 MPEG2018/Mm45056
October 2018, Macau, China

Source: **BSC, UB, UPC**
Status: **Proposal**
Title: **MPEG-G: New compression method for read names**
Authors: **Dmitry Repchevsky (Barcelona Supercomputing Center); Daniel Naro (Universitat de Barcelona); Jaime Delgado (Universitat Politècnica de Catalunya)**

Table of Contents

1 Problem

The current strategy on encoding read-names relies on breaking each read identifier into sub-units, which are represented and compressed individually. This approach ensures significant compression ratios, but is tightly bounded to the read identifier structure, thus requiring a high specialization of the encoder in order to ensure good compressions, i.e. the encoder needs to have a specialized strategy for each manufacturer's read identifiers strategy.
We here propose a new way to represent read-names, which would remove the specialization constraints, albeit losing some percentages of compression. It is based on the known BWT method, coupled to the CABAC arithmetic encoder.

2 Method

Usage of data transformation for better compression is a common method already introduced in MPEG-G. One of the well-known transformations is the Burrows–Wheeler transform, which has a long history in data compression and is used in many compressors, for instance BZip2. Our proposal is to include BWT transformation in the list of transformations that could be applied to the `transform_ID_subseq` enumeration table (Table 68). Proposed BWT transformation does not require special termination EOF character, but needs to store a first symbol position instead.

3 Proposed changes to the part 2 specification

Table 68. Values of transform_ID_subseq and transform_ID_subsym, must be updated to include new subsequence transformation identifier:

Sub-sequence level transformations		
transform_ID_subseq	**name**	**Remarks**
0	no_transform	no transform is applied
1	equality_ coding	As specified in 13.2.5.2.9.1
2	match_coding	As specified in 13.2.5.2.9.2
3	rle_coding	As specified in 13.2.5.2.9.3
4	bwt_transform	As specified in 13.2.5.2.9.5
5 .. 255	Reserved for future use	

13.2.5.2.9.5 BWT transformation

In the case that `transform_ID_subseq` is equal to `bwt_transform`, this process shall be used to reverse BWT transformed data into the original content.

```
reverse_bwt_transform(descriptor_ID, descriptor_subsequence_ID,
transform_ subseq_symbols[], first_char_index) {
      c[] = new int[256]
      occ[] = new int[Size(transform_subseq_symbols[])];
      for (i = 0; i < Size(transform_subseq_symbols[]); i++) {
            occ[i] = c[transform_subseq_symbols[i]]++
      }
      for (i = 0, s = 0; i < Size(c[]); i++) {
            t = c[i]
```

```
            c[i] = s
            s += t
      }
      sym = transform_subseq_symbols[0]
      for (i = Size(transform_subseq_symbols[]) - 1, j = 0; i >= 0; i-
-) {
      decoded_symbols[descriptor_ID][descriptor_subsequence_ID][i]  =
sym
            j = c[sym] + occ[j]
            if (j < first_char_index) j++
            sym = transform_subseq_symbols[j]
      }
}
```

In clause 13.2.2.4 Transform Subsequence Parameters needs to be updated to provide `first_char_index`:

Syntax	**Type**
transform_subseq_parameters(){	
transform_ID_subseq	u(8)
if(transform_ID_subseq == equality_coding){	
transform_subseq_counter += 1	
} else if(transform_ID_subseq == match_coding) {	
match_coding_buffer_size	u(16)
transform_subseq_counter += 2	
} else if(transform_ID_subseq == rle_coding) {	
rle_coding_guard	u(8)
} else if (transform_ID_subseq == bwt_transform {	
first_char_index	u(32)
}	
}	

We need to integrate to the table 10 the option to choose between the curent read identifiers compression method and the new method relying solely on transformations and CABAC. Therefore we introduce a new conditional statement in Table 10:

Table 1. Block Payload syntax.

Syntax	**Type**
block_payload() {	
if(descriptor_ID == 11 || descriptor_ID == 15){	
if(token_mode_ID == 0){	As specified in //13.2.2.5//
encoded_tokentype()	As specified in 11.5.18.1
}else{	

encoded_descriptor_sequences(descriptor_ID)	As specified in 13.2.5.2.1
}	
}	
else {	
encoded_descriptor_sequences(descriptor_ID)	As specified in 13.2.5.2.1
}	
while(!byte_aligned( ))	
nesting_zero_bit	f(1)
}	

We also propose to modify section 13.2.5.2.1, in order to store which of the two modes is being used.

Syntax	**Type**
decoder_configuration_tokentype(encoding_mode_ID){	
token_mode_ID = encoding_mode_ID	
if (encoding_mode_ID == 0){	
transform_subseq_counter = 1	
transform_subseq_parameters()	As specified in 13.2.2.4
for (j = 0; j < transform_subseq_counter; j++){	
support_values()	As specified in 13.2.2.2
cabac_binarizations()	As specified in 13.2.2.3
}	
}else if (encoding_mode_ID == 1){	
/* configuration for RLE specified in clause 11.5.18.3.3 */	
rle_guard_tokentype	u(8)
/* configuration for CABAC_ORDER_0 specified in clause 11.5.18.3.4 */	
decoder_configuration_ tokentype_cabac(0)	
/* configuration for CABAC_ORDER_1 specified in clause 11.5.18.3.5 */	
decoder_configuration_ tokentype_cabac(1)	
} else if(encode_mode_ID >= 2){	
/* reserved for future use */	
}	
}	

Finally, we need to introduce new default parameters for the CABAC strategy:

	pos, rftp	**rcomp**	**flags**	**mmpos**		**mmtype, rftt**					**clips**					**ureads**		**rlen**	**pair**				**mscore**	**mmap**	**read names**	**rtype**	**rgroup**
Descriptor_ID	**0, 1716**	**1**	**2**	**3**		**4, 1817**					**5**					**6**		**7**	**8**				**9**	**10**	**11**	**12**	**13**
$S_{alphabet\_ID}$	-	-	-	-	-	-	0, 2		1, 3				0, 2	1, 3		0, 2	1, 3								-		
descriptor_subsequence_ID	0	0	0 .. 2	0	1	0	1	2	1	2	0	1	2	2	3	0	0	0	0	1	X	Y	0	0	0	0	0
transform_ID_subseq	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**4**	0	0
output_symbol_size	32	2	1	1	16	2	3	3	4	4	32	4	3	5	32	3	4	32	3	16	32	16	8	16	8	3	8
coding_symbol_size	32	2	1	1	4	2	3	3	4	4	32	4	3	5	32	3	4	32	3	4	32	16	8	4	8	3	8
coding_order	0	1	2	2	1	1	1	2	1	2	0	1	2	2	0	2	2	0	1	1	0	0	1	1	1	2	0
binarization_ID	8,9*	1	1	1	1	1	1	1	1	1	6	1	1	1	6	1	1	6	1	1	6	6	1	1	1	1	6
bypass_flag	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BIFullCtxMode	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
cMaxDTU	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
splitUnitSize	4	-	-	-	-	-	-	-	-	-	4	-	-	-	4	-	-	4	-	-	4	4	-	-	-	-	4
adaptive_mode_flag	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
num_contexts	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
share_sub_symbol_data_flag	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	0	-	-	-
share_sub_symbol_ctx_flag	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	-	-	-	0	-	-	-
segment_ctx_flag	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
transform_ID_subsym	0	4	0	0	4	4	4	4	4	4	0	4	4	4	0	4	4	0	4	0	0	0	4	0	1	4	0

4 Results

The following table compares results for multiple compression strategies. All of the data has been generated from the files available for testing. In other words, this data is only representative of the compression performance of each tool for read identifiers generated according to the Illumnia naming schema.

file	original size	cmix 14	mpegg	bwt+cabac(o1)	lzma2	gzip
Seq_0_Au_0_Class_1	918888	134,643	142898	157,290	182,720	197002
Seq_0_Au_0_Class_2	1406	278	460	483	402	391
Seq_0_Au_0_Class_3	984018	144430	153604	168,353	195691	211056
Seq_0_Au_0_Class_4	225522	33521	35784	40,673	46037	48530
Seq_0_Au_1_Class_1	997589	146034	154973	170298	197962	213990
Seq_0_Au_1_Class_2	1549	287	464	512	407	417
Seq_0_Au_1_Class_3	886755	130245	138482	152271	176752	190493
Seq_0_Au_1_Class_4	244369	36294	38732	43917	50055	52407
Seq_0_Au_2_Class_1	959636	140579	149082	164034	190582	205980
Seq_0_Au_2_Class_2	1709	315	499	546	465	455
Seq_0_Au_2_Class_3	941052	138168	146861	161257	187197	202155
Seq_0_Au_2_Class_4	227771	33868	36141	41017	46719	48958
…						
Seq_0_Au_32_Class_4	39736	6105	6641	7811	8703	8745
TOTAL	68,653,399	10,082,084	10,719,153	11,806,548	13,685,336	14,732,900
	100%	14.69%	15.61%	17.20%	19.93%	21.46%