# 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/M42591**
**April 2018, San Diego, USA**

**Source:** **UPC**
**Status:** **Proposal**
**Title:** **Variant Calling in MPEG-G. An example use case**
**Authors:** **Daniel Naro, Jaime Delgado, Alba Pagès-Zamora (Universitat Politècnica de Catalunya, Barcelona)**

**Table of Contents**

1 Use case ..... 2
2 Problem ..... 2
3 How to support researchers in the use case ..... 2
4 Possible solution in MPEG-G ..... 2
4.1 Variant calling ..... 2
4.2 Statistics for a given position ..... 3
5 References ..... 6

# 1 Use case

Researchers want to test a Bayesian-based method to determine SNP for variant calling, namely Blind Ensemble Learning [1] [2].

# 2 Problem

Blind ensemble learning requires to know the sequenced statistics for a given position (i.e. bases of the different reads for that particular position together with the corresponding quality scores, and operation). During the validation phase of the technique, the ground-truth (i.e. a reliable variant calling) would be needed as well.

# 3 How to support researchers in the use case

Researchers are interested in retrieving as quickly and easily as possible all statistics for a given position, and the variant calling for the same position for validation.

# 4 Possible solution in MPEG-G

## 4.1 Variant calling

MPEG-G already provides descriptors to represent mutations shared among many reads, namely the pair `rftp` and `rftt`. This mechanism could be improved by indicating whether the modifications should be considered as a variant call or not, and if yes, to which chromosome it applies to. By doing so, MPEG-G would provide a basic integration of VCF, although representing only a subset of the possibilities.

Such functionality could be implemented by adding a new descriptor, for which we propose the name `rftc`. If `rftc` is present, it means that that `rftp` and `rftt` should be interpreted as variant calls. Each symbol of `rftc` would be composed of three bits: the first bit indicates whether the sequencing is phased, the second and third to which chromosome the mutation applies to.

| `rftc` symbol | Chromosome 1 | Chromosome 2 |
|---|---|---|
| 000 | Mutation present in neither of the chromosomes | |
| 001 | Mutation present in one of the chromosomes | |
| 010 | Mutation present in one of the chromosomes | |
| 011 | Mutation present in both chromosomes | |
| 100 | Mutation not present | Mutation not present |
| 101 | Mutation not present | Mutation present |
| 110 | Mutation present | Mutation not present |
| 111 | Mutation present | Mutation present |

This approach has certain issues: the calling process can only call heterozygous cases where one chromosome has the major allele variant, and the second chromosome the variant called. A more complex solution could be devised such that the pair `rftp` and `rftt` could define multiple

mutations for a given position, and `rftc` could then call one of the alternatives for each chromosome. Such solutions could rely on a fourth descriptor. This descriptor would both give a name to the variants and indicate which mutation they include. For example, it would allow to group as variant 1 for one position, one SNP, and as variant 2 for the same position, three insertions.

## 4.2 Statistics for a given position

In order to apply blind ensemble learning algorithm, the information for each base sequencing needs to be retrieved. Allowing this operation to be done in constant time, instead of iterating over each record, would mean to duplicate information. As such, it does not seem practical to modify MPEG-G part 1 and 2 to support it.

However, as the pile-up method would be an important feature for MPEG-G, an informative annex should be considered to explain how to convert the output of a getData operation (as specified in Part 3), into a pile-up for a given range of base pairs. The informative annex could take the shape of a pseudocode which creates a structure similar to the result of the mpileup command of samtools [3].

For each position covered by the pileup the result would be a structure as follows:

```
struct{
    st(v)   sequenceName
    u(64)   position
    c(1)    referenceBase
    st(v)   operations
    st(v)   quality
} pileup_position
```

`sequenceName` The name of the sequence the pileup for that position is being generated for
`position` The 0-based of the position for which the pileup is being generated for.
`referenceBase` The base as stored in the reference for that sequence, and that position.
`Operations` A list of characters describing the operations involved in the position
`Quality` quality of each sequenced based mapped to the position

The operations are represented as follows. If there is a match between the sequenced base and the reference base, the operation is represented with a dot (‘.’) or a comma (‘,’) if the read was mapped not mapped on the reverse complement or was mapped respectively.

If the sequenced based does not match the reference, but the operation is neither an insertion nor a deletion, the base represented as character is appended to the operations. If the read is mapped to the reverse complement, the base is represented in lower case, otherwise in upper case.

If the next operation is an insertion, the involved bases are stored as follows: a plus sign (‘+’), is followed by the number of inserted bases, and finally the sequence of inserted bases, stored as lower case if the read is mapped on the reverse complement, otherwise in upper case.

If the next operation is a deletion, the involved bases are stored as follows: a minus sign (‘-‘) is followed by the number of deleted bases, and finally the sequence of deleted bases, stored as lower case if the read is mapped on the reverse complement, otherwise in upper case.

Finally, if the operation is the first operation of the read, the previously described representation is prepended with '^' followed by the quality of the reads mapping. If the operation is the last operation of the read, the previously described representation is appended with '$'.

The following pseudocode is proposed as a starting point to obtain the previously described representation:

```
pileup generatePileUp(uint64 from, uint64 to, reference, dataUnits){
  pileup = pileup_position[to - from]
  for(position = from..to){
    pileup[position - from].sequenceName  = reference.sequenceName
    pileup[position - from].position      = position
    pileup[position - from].referenceBase =
reference.baseAt(position)
    pileup[position - from].operations      = ""
    pileup[position - from].quality         = ""
  }

  for(read : dataUnits.filter(from, to)){
    for(alignment: read.alignments){
      for(segment: alignment.segments){
        Sequence sequence = segment.sequence
        int currentBase = 0
        if(segment.mapping_pos < from){
          sequence.skip(from - segment.mapping_pos)
          currentBase = from - segment.mapping_pos
        }
        while(currentBase < segment.numberBases){
          int shiftSequenceReference = 0
          if(currentBase == 0){
            pileup[segment.mapping_pos + currentBase +
shiftSequenceReference].operations += "^"+segment.mapping_quality
          }
          if(ecigar.operationAt[currentBase] == MATCH){
            if(segment.mappedToReverse){
              pileup[segment.mapping_pos + currentBase +
shiftSequenceReference].operations += ','
            }else{
              pileup[segment.mapping_pos + currentBase +
shiftSequenceReference].operations += '.'
            }
          }
          if(ecigar.operationAt[currentBase] == SNP){
            if(segment.mappedToReverse){
              pileup[segment.mapping_pos + currentBase +
shitfSequenceReference].operations +=
lowerCase(segment.sequence[currentBase])
            }else{
              pileup[segment.mapping_pos + currentBase +
shiftSequenceReference].operations +=
upperCase(segment.sequence[currentBase])
            }
          }
          pileup[segment.mapping_pos + currentBase].quality +=
segment.quality[currentBase]
          if(currentBase == segment.numberBases -1){
```

```
            pileup[segment.mapping_pos + currentBase].operations +=
'$'
          }else{
            if(ecigar.operationAt[currentBase+1] == INS){
              insertion = "";
              while(ecigar.operationAt[currentBase+1] == INS){
                currentBase++
                shiftSequenceReference--
                if(segment.mappedToReverse){
                  insertion +=
lowerCase(segment.sequence[currentBase])
                }else{
                  insertion +=
upperCase(segment.sequence[currentBase])
                }
                if(currentBase == segment.numberBases -1){
                  break;
                }
              }
              pileup[segment.mapping_pos + currentBase +
shiftSequenceReference].operations += "+"+insertion.size+insertion
            } else if(ecigar.operationAt[currentBase+1] == DEL) {
              deletion = "";
              initialBase = currentBase
              while(ecigar.operationAt[currentBase+1] == DEL){
                currentBase++
                shiftSequenceReference++
                if(segment.mappedToReverse){
                  deletion +=
lowerCase(segment.sequence[currentBase])
                }else{
                  deletion +=
upperCase(segment.sequence[currentBase])
                }
                if(currentBase == segment.numberBases -1){
                  break;
                }
              }
              pileup[segment.mapping_pos + initialBase +
shiftSequenceReference].operations += "-"+deletion.size+deletion
              for(base: initialBase..currentBase){
                pileup[segment.mapping_pos + initialBase +
shiftSequenceReference].operations += "*"
              }
            }
          }
        }
      }
    }
  }
}
```

# 5 References


[1] Panagiotis A. Traganitis, A. Pagès Zamora and Georgios B. Giannakis, "Learning from Unequally Reliable Blind Ensembles of Classifiers", IEEE Global Conference on Signal and Information Processing, Montreal Canada, November 2017, pp. 106 -110.

[2] Panagiotis A. Traganitis, A. Pagès Zamora and Georgios B. Giannakis, "Blind Multi-class Ensemble Learning with Unequally Reliable Classifiers", *Submitted to Trans. on Signal Processing*, December 2017, pp. 1 - 13. Available at arXiv:1712.02903.

[3] "Samtools", http://www.htslib.org/doc/samtools.html, April 2018.