Introduction

In my attempts to implement the C version of CRAM into Htslib I have met a variety of problems with the SAM specification, both incompatibilities with sensible algorithmic coding and also ambiguous phrasing.

This is an attempt to describe these issues with worked examples. I propose changes to the SAM specification in order to resolve these issues, and so are asking for comments.

Collation algorithms

Many algorithms need to know how many alignments there are for the same read identifier. One example of this would be read collation as part of a "fixmates" process. The obvious way of handling such collation methods with paired end data is to use a hash table keyed on read identifiers, and purge the line from the hash table once the second read has been processed in order to keep memory usage low. However what if we have more than two reads? We need to know how many to expect, hence the addition of a TC auxiliary field.

Similarly for CRAM, if all the reads for a template are within the same slice then we could optionally do lossy compression, throwing away the read-names and regenerating new ones. If the reads span multiple slices then rather than build up a potentially large hash table we simply store the read name verbatim and in doing so choose to trade efficiency of algorithm for a slight loss in potential lossy compression. Again, for this to work we have to know with certainty how many alignments are present for any template.

The TC field however is not entirely sufficient for either situation, or at least is weakly defined. Firstly it is not mandatory. It is in the recommended section, but I believe this requires expansion to clarify the implications of not using it so users have strong guidelines as to when and where it is critical.

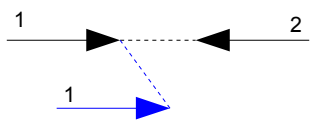
Secondly it does not adequately cover secondary alignments. FI (fragment index) and TC (total count of fragments) jointly indicate your position within the primary alignment. (Assumption: it doesn't actually state that, but I cannot see how it works otherwise.) We need to know something like **AC – the total number of alignment records with this sequence identifier**.

Template positioning fields

SAM has the notion of storing relationships between individual sequence reads, or segments of reads. The fields involved with these relationships are the 0x20 FLAG, the RNEXT, PNEXT and TLEN fields and some optional auxiliary fields.

To demonstrate the potential use cases and also the changes suggested to the specification, I will go over a variety of examples. In these black lines represent primary reads, blue lines represent secondary reads and small numbers and/or letters represent the nth segment in the template. (Typically 1 and 2 for a pair, or 1a, 1b, 1c for supplementary portions of the 1st end.)

A) A single read pair with one end having a secondary alignment, e.g. due to a repeat.

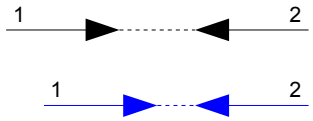


The specification states that PNEXT must always point to the next primary alignment. It also explicitly states that this applies for secondary reads too (recommended practices).

The above conforms to this by having the secondary alignment for segment /1 having RNEXT/PNEXT linking to segment /2. TLEN for the secondary read is not specified as far as I can see.

The first problem with the specification comes from the description of the 0x20 FLAG: "*SEQ of the next segment in the template being reversed*". This is at odds with PNEXT: "*Position of the **primary** alignment of the NEXT read in the template*". Note that PNEXT uses primary alignments while FLAG does not. I assume this is an accident given we would typically desire the primary alignment for /1 to indicate if the primary alignment for /2 is reversed, irrespective of the presence of the secondary alignment of /1 being between them.

B) As above, but we produce a secondary alignment for the 2nd segment too, so that the secondary read-pair forms a valid pair in its own right.

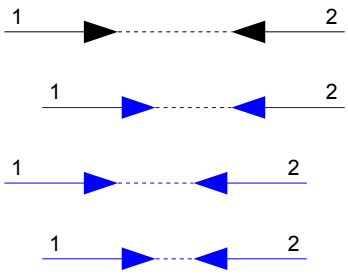


Technically /1 secondary hasn't changed at all. PNEXT is a position and is not explicitly linking to a specific segment, so it will still have the same RNEXT/PNEXT fields as case A. The new secondary copy of /2 though could link back to the secondary copy of /1, and both secondary alignments having a TLEN field consistent with themselves (i.e. shorter in this example).

C) A pair with an alternative secondary mapping for the entire template.

Here it clearly makes no sense for the secondary reads to link back to their primary templates. If the aligner can detect a self-consistent alternative position, then it should be permitted to set up PNEXT/RNEXT/TLEN accordingly. This is at odds with the specification.

D) Multiple mappings of the same pair, sharing some coordinates.



In this scenario each end has two possible alignment locations and we have produced all permutations for the template; one primary and three secondary. In order to disambiguate which secondary end belongs each other, given that there are two possibilities for some cases, we would

need an additional auxiliary field to label each template alignment with an index. Eg SI:i:1 to SI:i:3. This would be expected to be optional when there is only a single secondary template alignment and mandatory if the aligner produces more than two in a potentially ambiguous manner.

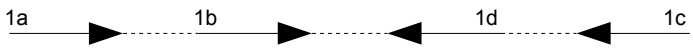
The downside to this system though is a combinatorial proliferation of alignments.

We would expect the above alignments to all have AC:i:8 as a mandatory field.

E) Multiple supplementary alignments for a single end.

I label these 1a to 1d to distinguish between primary alignments of *reads* /1 and /2 (or /1, /2, /3, ... in some technologies), but essentially it is the same thing.

This could be from RNASeq type experiments. We may wish to use the "N" cigar operator, but this cannot cope with structural variation so I'll take the more general case of using supplementary reads. Here the RNEXT/PNEXT chain is obvious; a to b to c to d, mapped in the logical order.



The above case though has a rearrangement, with the c and d portion being reverse complemented. Assuming that "*next read in the template*" refers to the next read in template order rather than genomic order, we still have RNEXT/PNEXT linking in a, b, c, d order. This isn't entirely obvious from the wording, but it makes the most sense.

I think this just works as-is. For secondary alignments, again it makes sense to replicate the entire set of segment alignments for clarity.

Another problem case E is knowing how many fragments there are. For a variety of algorithms we want to know when we have processed all fragments for a template. Typically such algorithms are written assuming two only, as it is inefficient to search indefinitely. The FI and TC auxiliary tags exist, but are not mandatory. We should make these mandatory for any read consisting of more than two segments.

I assume FI/TC are per whole template alignment, so a secondary mapping of all 4 segments above would still give TC of 4, not of 8 (primary + secondary combined). The proposed AC field would be 8.

F) Knowing the number of secondary fragments.

In CRAM we need to know how many total fragments.

The SC:i: auxiliary tag would be used as the total count of secondary alignments. It should be set to be the same value on all reads, and mandatory whenever there is at least one secondary alignment.

Questions

1) Should we always require secondary alignments for all members of a template if at least one aligns to a secondary position (B) or is it sufficient to link back to the primary if the next

primary segment only has one mapping (A)? Making this optional may also avoid the combinatorial issues raised by D.

2) Should the FI/TC auxiliary tags be per whole template alignment (so TC:i:2 for both primary and secondary segments in case C) or the sum of all alignments including secondary (TC:i:4 for case C).

Conclusion

We need ways to track how many total alignment records there are for a template, so algorithms can be coded efficiently without making assumptions of paired-end data with no secondary or supplementary reads.

Where we talk about the "next primary alignment" (PNEXT/RNEXT) or "all segments" (TLEN) we should expand the specification to state the next alignment in the current chain; i.e. TLEN is computed if and only if all reads linked together by PNEXT/RNEXT are mapped to the same reference. Technically it could even say the first and last segments are mapped to the same reference, as an unmapped middle segment does not change the observation of TLEN.

Proposals

1) FI auxiliary field should be clarified to indicate 1-centric numbering, such that all FI fields are >= 1 and <= TC value. (Unknown: does anyone use 0 counting for this in the wild? Or 1?)

2) Add an AC auxiliary field to count the total number of alignment entries (primary, secondary, supplementary, QC fail, unmapped, etc) in the file.

3) TLEN should still be defined if the internal (not first or last) fragments are unmapped.

4) FI/TC to be mandatory when we have more than 2 fragments to an alignment.

5) Define "template alignment" as "a set of read alignments for all reads in the template. In the case of multiple mappings for a template, multiple template alignments may exist. All read alignments (whether chimeric or linear) within a template alignment should share the same value of the 0x100 flag (secondary alignment)".

6) TLEN, PNEXT, RNEXT to refer to the current "template alignment", rather than primary only.

7) Give up on our TLEN definition and just revert to the current in-use practice of 5' to 3', albeit for the current "template alignment" rather than primary alignments.

8) The addition of a SI auxiliary field to indicate which secondary template alignment this fragment belongs in and SC to be the total count of secondary template alignments. For templates with more than one secondary mapping with potentially ambiguous linkage of fragments, SI becomes mandatory.

Other things still to consider

The specification introduces *chimeric alignment* stating that "*all the SAM records in a chimeric alignment have the same QNAME and the same values for 0x40 and 0x80 flags*". It also defines *segment* as "*A contiguous sequence or subsequence*". This a chimeric read consists of multiple segments.

However 0x40 and 0x80 flags elsewhere are defined to be the first and last segment, not the first and last read. This is at odds with chimeric reads having multiple segments with 0x40 and 0x80 set.