Inferring variants from assembled haplotypes in HaplotypeCaller and Mutect

David Benjamin[*†]
Broad Institute, 75 Ames Street, Cambridge, MA 02142
(Dated: September 27, 2017)

Despite its name, `HaplotypeCaller`does not actually call haplotypes. Rather, it generates haplotypes as an intermediate step to discover variants at individual loci. Here we describe how the GATK engine determines which alt alleles exist in locally assembled haplotypes.

I. FORWARD PASS

The first step is to align each assembled haplotype to the reference haplotype using the Smith-Waterman algorithm. Although the GATK's implementation is not complicated, it is also not a completely direct translation of the method into code. As it incurs a non-trivial computational cost, we describe it in detail here.

Our implementation has four score parameters, $w_{\rm match}$ and $w_{\rm mismatch}$ for equal and unequal reference and alternate bases, $w_{\rm open}$ for opening a gap (that is, starting an indel), and $w_{\rm extend}$ for extending a gap. Note the absence of a scoring matrix treating each possible type of match and mismatch differently. Although the idea of a score seems like a heuristic, the algorithm is equivalent to finding the maximum likelihood path in a hidden Markov model in which the scores are log transition and emission probabilities.

As in our pair-HMM probabilistic alignment, we fill a matrix M, the rows and columns of which correspond to bases of the reference and alternate haplotypes, from top to bottom and left to right. M_{ij} is the best score of alignments ending at the ith reference base and jth alternate base that do not end in a gap[1]. We also keep track of two arrays pertaining to the last row of our traversal. D_j is the best score of alignments ending at the previous reference base (ie the $(i-1)$th base when we are at the ith base in traversal) and the jth alternate base that end in a "downward" gap, i.e. a deletion with respect to the reference. S^d_j is the size of the gap in this best-scoring alignment. We also fill a backtrack matrix B, where B_{ij} is an instruction (see below) for reconstructing the best path after we fill M.

First we initialize the zeroth row and column as $M_{0,0} = 0$, $M_{0,1} = M_{1,0} = w_{\rm open}$, $M_{0,2} = M_{2,0} = w_{\rm open} + w_{\rm extend}$ etc. The zeroth row and column correspond to one base before the reference and alternate starts and this initialization penalizes leading indels. That is, this is a global alignment[2].

Then, for each row $1 \leq i \leq$ length(reference) we loop over all columns $1 \leq j \leq$ length(alternate) and do the following:

- Update deletion scores: The score for opening a downward gap is $M_{i-1,j} + w_{\rm open}$. The score for extending an existing deletion is $D_j + w_{\rm extend}$, where D_j at this point still pertains to the *previous* row $i-1$. We set D_j (modifying it in-place) to the greater of these values. If the gap-opening score is greater than the gap-extending score, we set $S^d_j = 1$, otherwise we increment S^d_j by 1.

- Update insertion scores: When we begin traversing row i we initialize the current best score for alignments ending in a "rightward" gap (i.e. an insertion) as $R = -\infty$ and we initialize the length of the terminal gap in this best-scoring alignment as $S^r = 0$. Note that these values are local to the inner loop over j and thus do not need to be arrays[3]. At each stage in the loop over j the score for opening a rightward gap is $M_{i,j-1} + w_{\rm open}$ and score for extending one is $R + w_{\rm extend}$, where R still pertains to the previous column $j-1$. We set R (modifying it in-place) to the greater of these values. If the gap-opening score is greater than the gap-extending score, we set $S^r_j = 1$, otherwise we increment S^r_j by 1.

- Record backtrack: The score for no gap, i.e. a match *alignment* (as opposed to matching bases), is $M_{i-1,j-1}$ plus $w_{\rm match}$ if the ith reference base and jth alternate base agree or $w_{\rm mismatch}$ if they do not. We now compare

[*] The author took no part in development of the methods described below – credit belongs to several others on the GATK team.
[†]Electronic address: `davidben@broadinstitute.org`
[1] That is, it includes alignments that have gaps somewhere earlier, just not at this position.
[2] The GATK assembly graph merges all alternate paths into the reference, hence alternate haplotypes start and end coincident with the reference and global alignment is appropriate.
[3] In the code, they *are* arrays, but in the ith iteration of the loop over rows, only their ith elements are used. Thus the elements are effectively scalars whose scope is the loop over j.

this score to the indel scores D_j and R. We set B_{ij} to 0 if the match score is greatest, S_j^d if the downward gap score is greatest, and $-S^r$ if the rightward gap score is greatest. This convention essentially uses the sign as an enum in order to encode whether the optimal path has an insertion, deletion or match.

II. BACKWARD PASS

After the forward pass the backtrack matrix B is full. We begin backtracking from the (i,j) that is the maximum among the bottom row and rightmost column of M. If this maximum is in the rightmost column that means all alternate haplotype bases are used and nothing special must be done. If, however the maximum is on the bottom row it means that the more alternate bases remain. In this case, we record a soft clip (S) CIGAR element with length equal to the number of remaining bases at the end of the alignment[4]. Then, from this (i,j) we iterate the following procedure until reaching $i=0$ or $j=0$ (recall that these correspond to immediately *before* the start of the corresponding haplotypes): If $B_{ij}=0$ add a match (M) CIGAR element to the left end of the alignment and move to $i,j \to i-1, j-1$. If $B_{ij}=k>0$ add a length-k deletion (D) CIGAR element to the left of the alignment and move to $i,j \to i-k, j$. If $B_{ij}=-k<0$ add a length-k insertion (I) CIGAR element to the left of the alignment and move to $i,j \to i, j-k$.

Similar to the initial step, if we end at $j=0$ nothing more needs to be done because all alternate bases are accounted for. Otherwise, add a length-j leading soft clip.

III. MAKING VARIANTS

For each haplotype from assembly it is easy to create variant alleles from the alignments found above. Starting from the beginning of the haplotype and its starting reference position, traverse every element in the CIGAR string, advance k bases in the reference for every length-k element. When we encounter a deletion element or insertion element we record a corresponding allele from the reference and alternate bases. When we encounter a match element we compare the matched reference and alternate sub-haplotypes base-by-base and record a SNV allele whenever they disagree.

Taking all the unique start positions and variant alleles from all haplotypes give an initial set of variants to genotype but we are not quite done. If multiple haplotypes have different variant alleles at the same position we may need to reconcile the representations. For example, we may have a single deletion $AA \to A$ and a double deletion $AAA \to A$, which need to be merged as $AAA \to A, AA$. Fortunately, this is essentially the canonical example and there are no edge cases to deal with. That is, all we need to do is find the allele with the longest reference, and pad the other alleles with whatever of these extra reference bases they were missing.

[4] The code behaves differently for different values of the `OverhangStrategy` enum, but this strategy is hard-coded to `SOFTCLIP`, which results in the behavior we describe here.