Quiver technical summary

David Alexander Patrick Marks

October 23, 2012

What is Quiver?

- A multiple-read consensus calling algorithm for PacBio® *RS* reads
- Takes multiple reads of a given DNA template, outputs best guess of template's identity
- QV-aware conditional random field model to model our sequencing errors; a greedy algorithm to find the maximum likelihood template.
- **Can achieve accuracy >Q50 (i.e. >99.999%) in applications to de novo assembly and resequencing using pure PacBio long reads.**

How Quiver works

Quiver uses a greedy algorithm to maximize the likelihood $\Pr(\mathbf{R} \mid T)$ in the unknown template T.

- $\Pr(\mathbf{R} \mid T)$ encodes our sequencing error model and is specific to a chemistry and enzyme—currently requires a training step, which is performed in-house at PacBio.

QuiverConsensus for reference window W: (*Rough sketch*)

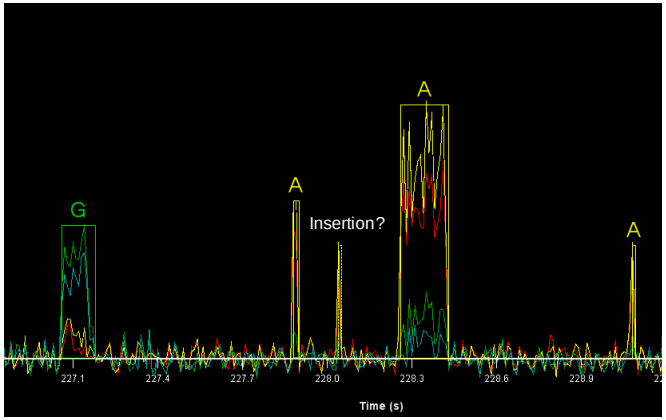
- Use reference alignment to identify reads $\mathbf{R} = \{R_1, R_2, \ldots R_K\}$ corresponding to W
- *Throw away reference—not used in computing consensus*
- $\hat{T}_1 \leftarrow \text{PoaConsensus}(\mathbf{R})$
- Repeat until convergence:

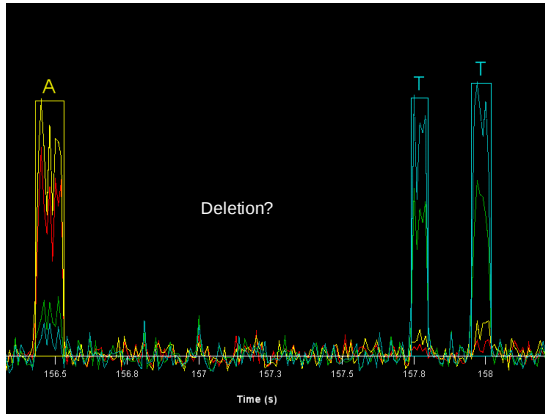
$$\hat{T}_{s+1} \leftarrow \hat{T}_s + \{\text{single base mutations } \mu \mid \Pr(\mathbf{R} \mid \hat{T}_s + \mu) \geq \Pr(\mathbf{R} \mid \hat{T}_s)\}$$

Where to get Quiver

- Quiver will be integrated in the 1.4 SMRT® analysis release
- Until then, you can install it from GitHub, using instructions here: http://git.io/AERlEA
- Quiver is open source, under the BSD license, so feel free to integrate it in your programs and workflows.

Overview of PacBio® *RS* data





- Very long reads
- Errors are dominated by indels, not substitutions
  - Mostly cognate extras (homopolymer expansion)
  - Some pulse merging (homopolymer contraction)
  - Some noncognate extras
  - Essentially no substitutions

Pulse metrics

In addition to basecalls, the basecaller software includes metrics reflecting its confidence against the various types of errors.

Base	Insertion QV	Substitution QV	Deletion QV	Deletion Tag	Merge QV
A	8	12	16	N	14
T	2	12	5	T	100
T	11	30	4	G	25
G	12	30	11	A	11
G	3	30	16	N	27
C	6	30	16	N	19
C	3	19	3	C	21
G	2	21	4	G	22

$$QV = -10 \log_{10} p_{error}$$

Definition of pulse metrics

- **InsertionQV**, **SubstitutionQV**: Probability that this base call is actually an insertion (substitution) relative to the true template.
- **DeletionQV**: Probability that the basecaller omitted a base relative to the true template, *prior* to this basecall. Maximum likelihood missed base is encoded in **DeletionTag**.
- **MergeQV**: Probability that the basecaller merged together two identical adjacent template bases into this basecall.

All probabilites are phred-encoded.

How to compute $\Pr(\mathbf{R} \mid T)$?

1. Reads are assumed independent, so

$$\Pr(\mathbf{R} \mid T) = \prod_{k=1}^{K} \Pr(R_k \mid T)$$

2. For PacBio, indels are the rule, not the exception, so the model considers the possible *alignments*—the ways T can be construed to have generated R_k:

$$\Pr(R_k \mid T) = \sum_{\mathscr{A}} \Pr(R_k \mid T, \mathscr{A})\, \pi(\mathscr{A} \mid T)$$

This summation can be computed efficiently using a standard Sum-Product dynamic programming approach.

Sketch of dynamic programming

- Sum-Product definition:

  $A_{ij} \doteq$ marginal prob. of an alignment of R[0:i+1] to T[0:j+1]

  $B_{ij} \doteq$ marginal prob. of an alignment of R[i:I] to T[j:J]

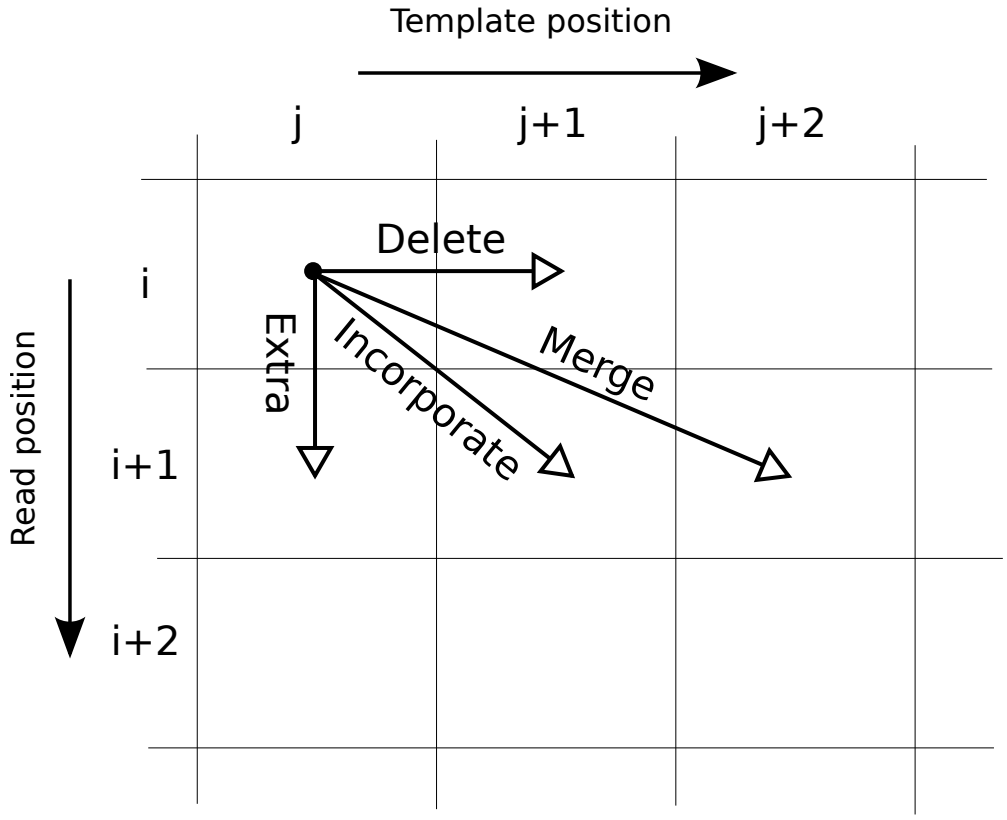
- Sum-Product recursion:

$$A_{ij} = \sum_{m:(i',j')\rightarrow(i,j)} (A_{i'j'} \times \text{moveScore}(m))$$

$$B_{ij} = \sum_{m:(i,j)\rightarrow(i',j')} (\text{moveScore}(m) \times B_{i'j'})$$

- For Viterbi approximation, replace *marginal* by *maximum*, replace *sum* by *max*.

Alignment moves



- Additional “merge” move helps better account for pulse merging

Alignment move scores

- Modulated by observed pulse metrics (supply more detail here)

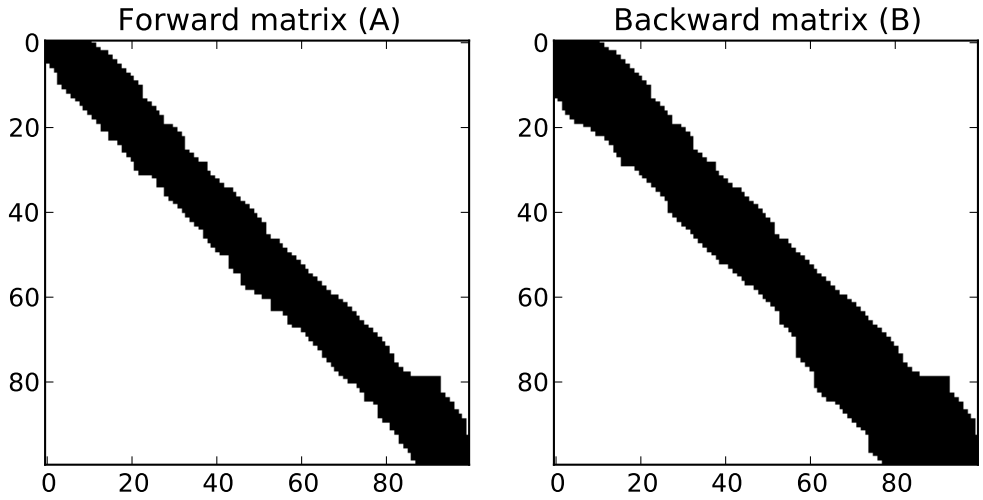
Efficiently computing $\Pr(R_k \mid T + \mu)$

- Need to compute score of mutation μ quickly as this is the *rate-limiting operation* in computing the consensus.
- Do not refill entire A, B matrices–we just recalculate two columns of A and join with one column of B.
- Exploit identity

$$\begin{aligned}\text{Score}(T) =& A_{IJ} = B_{00} \\ =& \max_{m:(i',j')\to(i,j)} A_{i'j'} \times B_{ij}, \text{ for } \textbf{any } j\end{aligned}$$

- Requires $O(L)$ time and space, naively.

Banding for memory and CPU efficiency



- Optimization 1: *banded dynamic programming*: only compute a narrow band of high-scoring rows within each column.
- Optimization 2: Only *store* the bands.

	Naive	Banded
Initial computation of A, B	$O(L^3)$	$O(L^2)$
Computation of mutation score	$O(L)$	$O(1)$
Storage space for A, B	$O(L^2)$	$O(L)$

A good starting point

- Prime the “hill-climbing” loop with a good starting point
- We use a heuristic based on Partial-Order Alignment (POA) to for a fast approximate consensus. With 11x coverage it is typically >99.5% accurate.
- $O(KL^2)$ time; in practice fast enough, but could make faster by using a banded approach.