`ConsensusCore`: a library for fast multiple sequence consensus

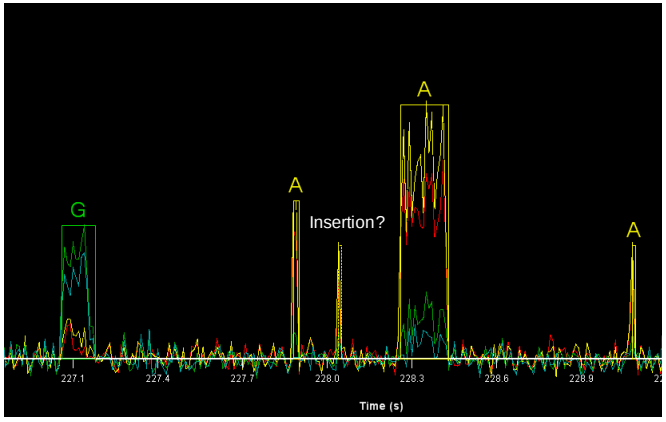
David Alexander

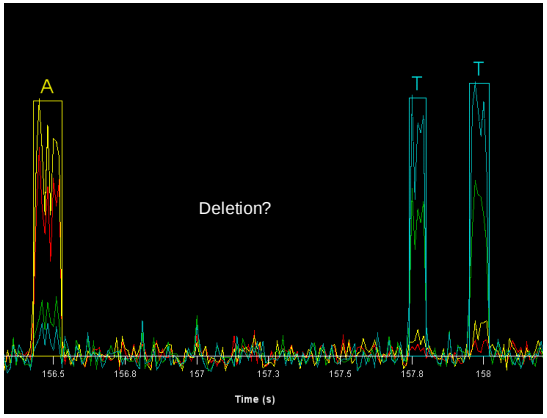
16 July 2012

Overview

- In the beginning PacBio had long(ish) reads...
- ...but they were highly inaccurate
- There was interest in trading off some readlength for accuracy
- CCS was born
- CCS algorithm ("Quiver") is now liberated from Primary—packaged in `ConsensusCore` library and available for multimolecule consensus

Background: PacBio error model





- Different from most other technologies
- Our errors are dominated by indels
  - Mostly cognate extras (homopolymer expansion)
  - Some pulse merging (homopolymer contraction)
  - Some noncognate extras
  - Essentially no substitutions

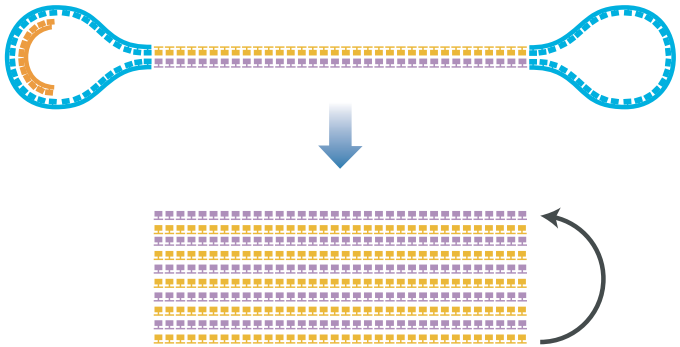
Background: PacBio data

`bas.h5` file contains a lot more than just sequence... basecaller is telling us where it wasn't so sure.

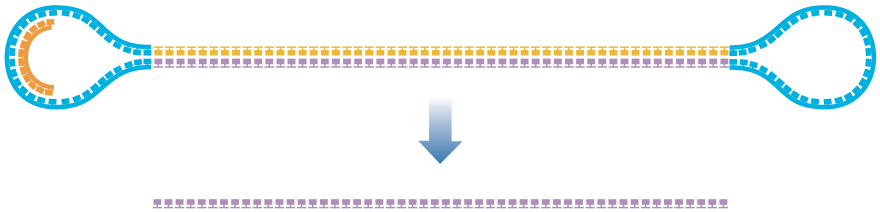
Base	Insertion QV	Substitution QV	Deletion QV	Deletion Tag	Merge QV
A	8	12	16	N	14
T	2	12	5	T	100
T	11	30	4	G	25
G	12	30	11	A	11
G	3	30	16	N	27
C	6	30	16	N	19
C	3	19	3	C	21
G	2	21	4	G	22

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

Circular consensus sequencing (CCS) worfklow

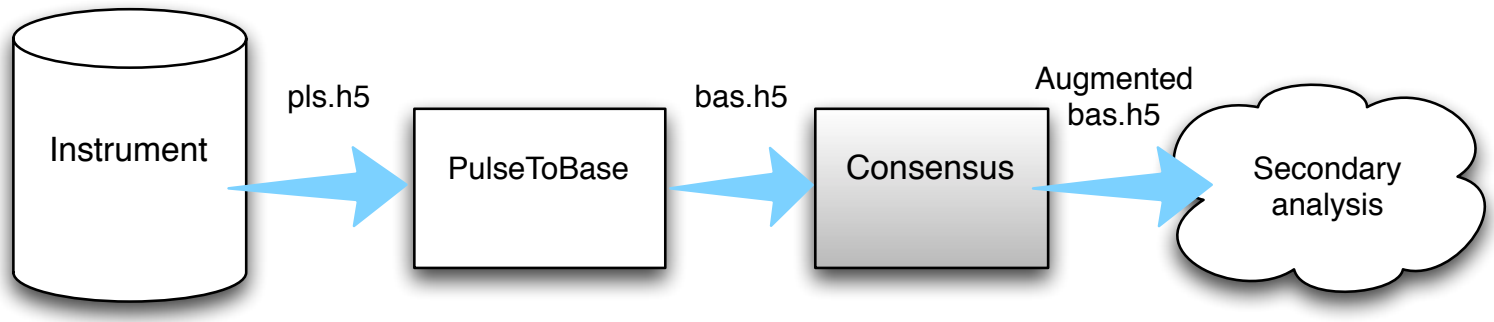


Circular consensus provides multiple subreads on shorter insert sizes.



Standard sequencing provides a single pass read on longer insert sizes.

CCS analysis flowchart



The consensus problem

- *Given*: A sequence of reads $\mathbf{R} = \{R_1, R_2, \ldots R_K\}$
- *Desired*: A consensus sequence $\widehat{T}$ that is, in some sense, a "best" estimate of the underlying true template sequence T that was present in the ZMW.

Example

Template	GATTACA
Read 1	GATTCA
Read 2	GATTTACA
Read 3	GATACA

Algorithmic approaches: MSA

- Build a multiple sequence alignment and call the consensus using a simple column-wise plurality rule
- Example:

Read 1	`GA-TT-CA`
Read 2	`GATTTACA`
Read 3	`GA--TACA`
Plurality	`GA-TTACA`

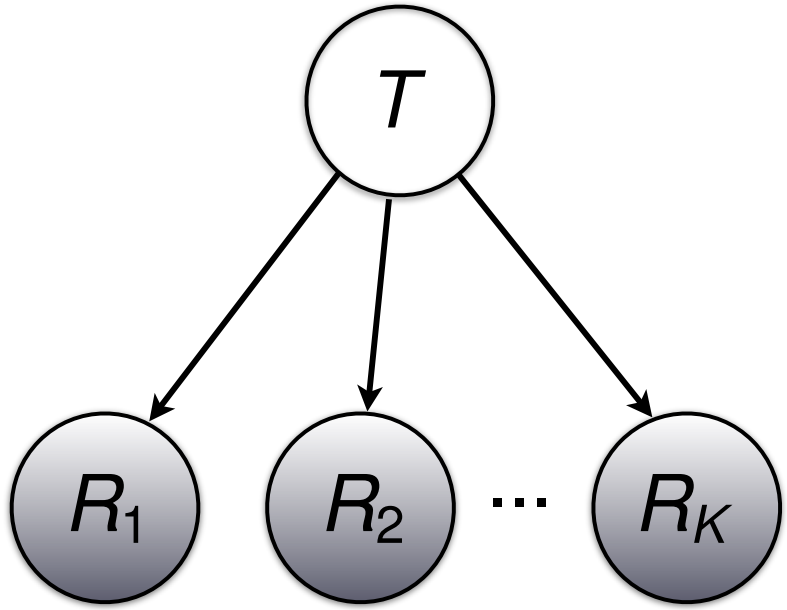
Shortcomings of MSA approach

- MSA approaches typically have no notion of template vs observations–no clean way to represent our error model.
- No way to take advantage of QV information

CCS approach

- Build an HMM to compute $\Pr(\mathbf{R} \mid T)$
  - HMM accounts for PacBio-specific error model
  - Model parameters can be learned by training
- Optimize the the probability in T using an efficient greedy algorithm

CCS details: handling multiple reads



- Reads are considered independent given the template:

$$\Pr(\mathbf{R} \mid T) = \prod_k \Pr(R_k \mid T)$$

- Bookkeeping code needs to take care of which strand each read is from, so for reverse strand reads we do $\Pr(R_k \mid T')$

CCS details: the HMM model

- We compute forward (A) and backward (B) matrices under a modified Needleman-Wunsch model.
- Sum-Product and Viterbi algorithms available; we use Viterbi by default (faster)
- Computations done in log-domain to prevent underflow.

CCS details: recursion

- Viterbi definition:

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

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

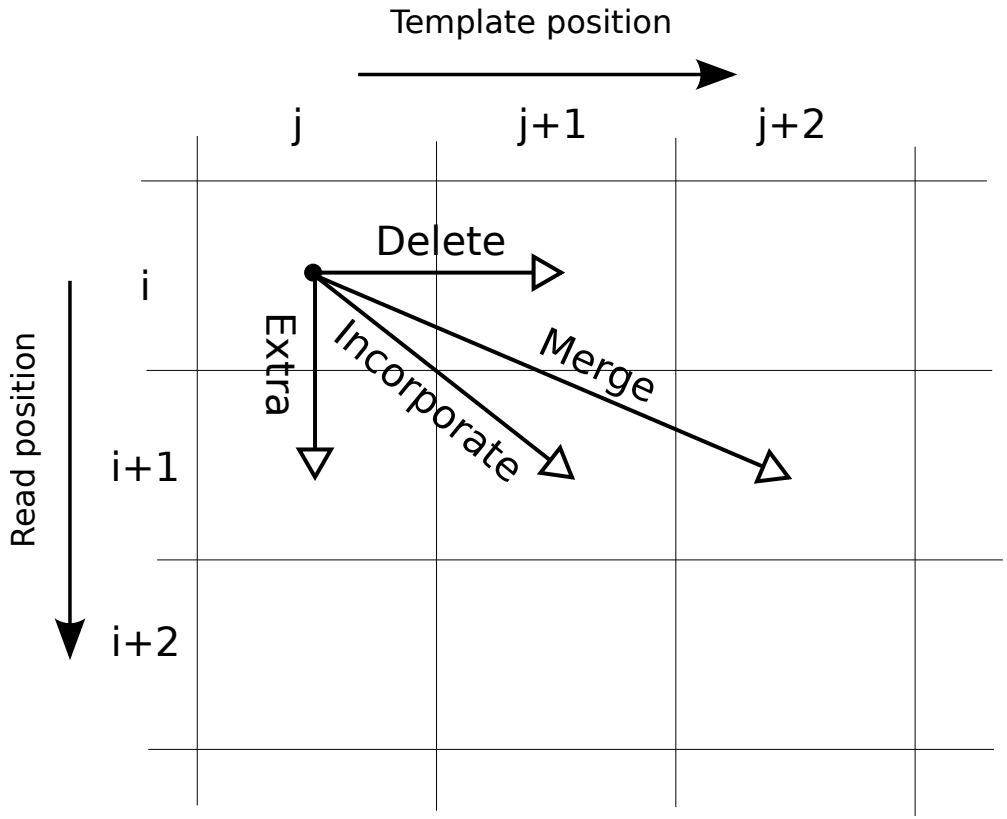
- Viterbi recursion:

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

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

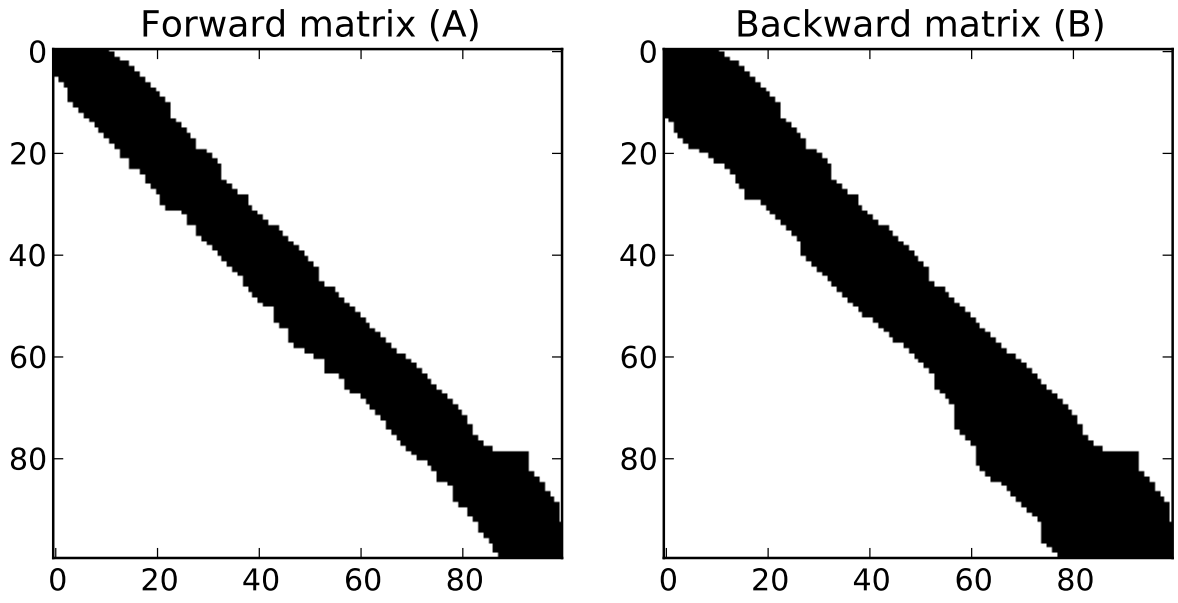
- For Sum-Product, replace *maximum* by *marginal*, replace *max* by *sum*.

CCS details: moves



- Additional “merge” move helps better account for pulse merging
- Move scores are modulated by the QV values in the read.

CCS details: sparsity



- Dynamic programming approaches like this are $O(L^2)$ for scoring a read against a template with lengths $\sim L$.
- We do *sparse dynamic programming*, where we essentially only compute a narrow band of high-scoring rows within each column of the DP matrix; reduces computation to $O(L)$.
- We also *store* the matrix sparsely—essential when scoring 100+ reads of length 2000+ (not needed for CCS...)

CCS details: greedy template mutation strategy

- Testing all 4^L possible templates is out of the question.
- Instead, starting from some template T enumerate all single base mutations μ and calculate the scores of the mutated templates $\mu(T) = T'$;
- Apply the highest scoring mutations to the template; repeat the mutation scoring procedure on this new T;
- If no favorable mutations found, we are done—T should be a good estimate.

CCS details: mutation scoring

- Need to compute score of mutation μ quickly—do not refill entire A, B matrices–we just recalculate two columns of A and join with one column of B
- Exploit identity

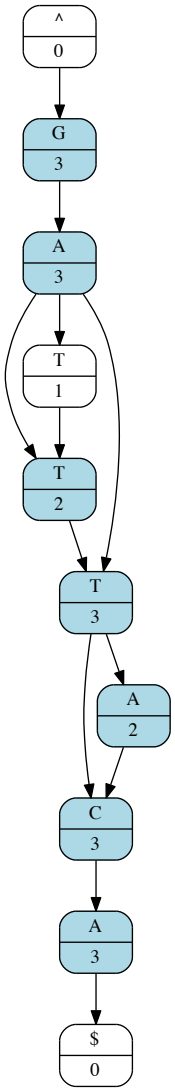
$$\begin{aligned}\text{Score}(T) &= A_{IJ} = B_{00} \\ &= \max_{m:(i',j')\to(i,j)} A_{i'j'} \times B_{ij}, \text{ for } \textbf{any } j\end{aligned}$$

- Requires $O(1)$ time and space (assuming sparsely stored matrix)

CCS details: a good starting point is essential

- We use a heuristic based on Partial-Order Alignment (POA) to come up with a fast approximate consensus. With 5x CCS coverage this is usually ~ 95% accurate; with 11x coverage in `GenomicConsensus` it is typically ~ 99.5% accurate.
- $O(KL^2)$ time; in practice fast enough, but could make faster by "sparseifying".

POA example

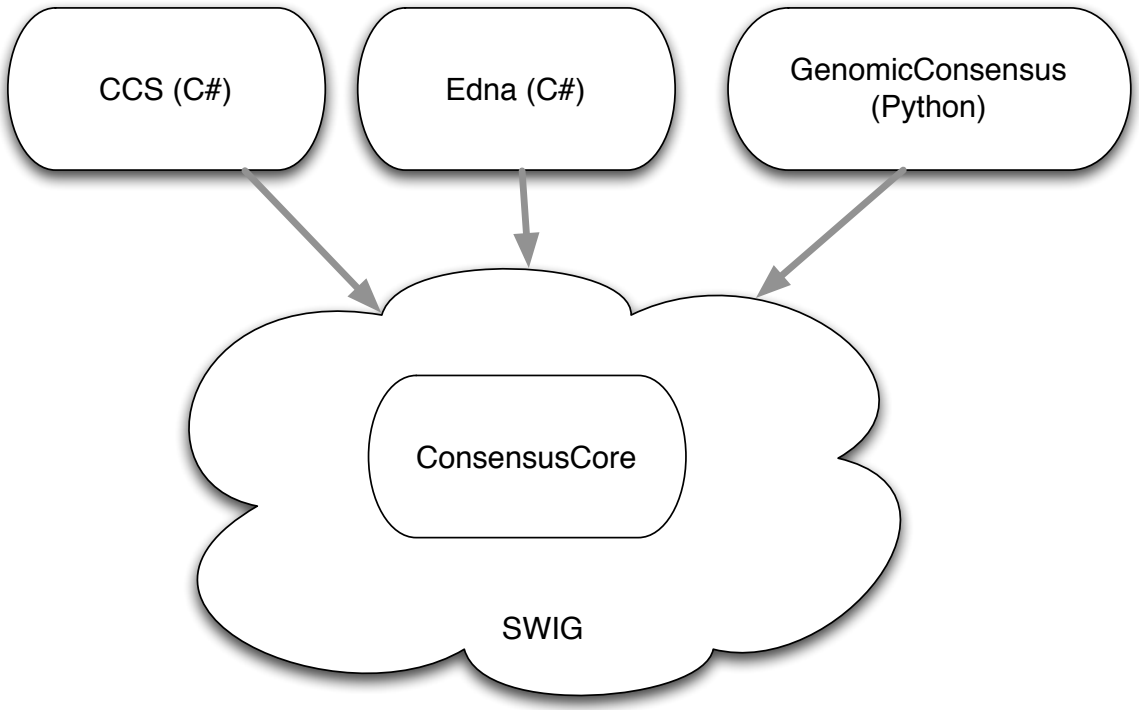


Reusing CCS

- Algorithm is pretty darn good
- Reuse it elsewhere? i.e., multi-molecule consensus calling in secondary?
- Yes!

ConsensusCore

A C++ library housing our consensus calling algorithms and making them available to arbitrary programming languages via SWIG bindings.



The core QV-aware algorithm is now rebranded **Quiver** since it is no longer limited to Circular Consensus calling.

About ConsensusCore

- 5000+ LOC, plus 1500+ LOC in over 100 test cases
- Passes cpplint.py
- SWIG bindings available today:
  - C#
  - Python
- SWIG bindings available tomorrow:
  - your language here

`ConsensusCore` / Quiver for multi-molecule consensus

- `GenomicConsensus` presently uses a crude plurality calling scheme and relies on alignments as provided by BLASR—no local realignment.
  - Fast and simple, but
  - Susceptible to a variety of "reference bias" issues
- Quiver serves as a form of local realignment, and leverages the extra information in the QVs
- Presently QV40+ with coverage $\leq$ 30x

What I am working on now

- Providing Quiver as a turn-key variant caller mode for 1.4
- Retraining the HMM parameters to achieve Q50+ by 30x, Q40 by much less
- Adaptive coverage depth

Plans for the near-term

- Diploid calling
- Sparsifying POA

How you can use `ConsensusCore` and Quiver

Check out (mainline) and install in your virtualenv:

- `pbcore`
- `ConsensusCore`
- `GenomicConsensus`

Run `GenomicConsensus/quiver/demo.py`

Demo?