Sequence analysis

Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm

Can Firtina [1], Jeremie S. Kim[1,2], Mohammed Alser[1], Damla Senol Cali[2], A. Ercument Cicek [3], Can Alkan[3,*] and Onur Mutlu[1,2,3,*]

[1]Department of Computer Science, ETH Zurich, Zurich 8092, Switzerland, [2]Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA and [3]Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey

ExaBiome/PASSION/SAFARI Workshop on Architectures and HPC for Genomics

Can Firtina

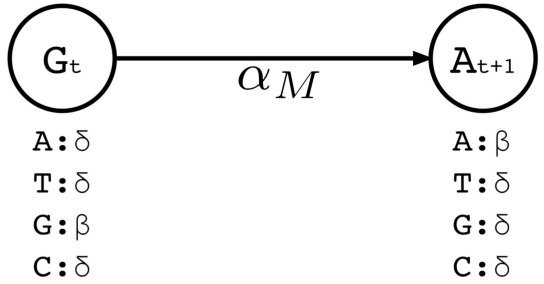
May 26th, 2021

Executive Summary

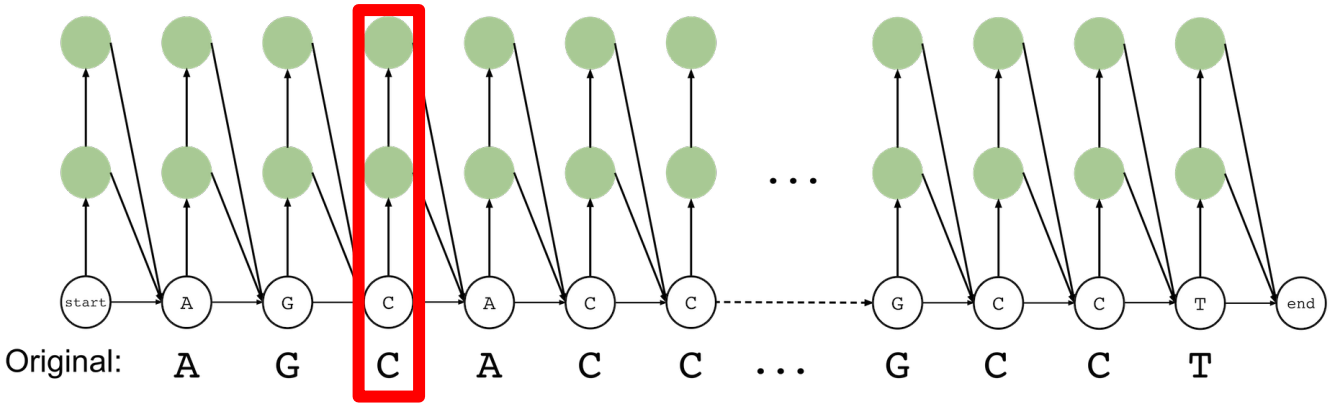
- **Problem:**
  - Long read de-novo assembly is inherently **erroneous**
  - Existing assembly polishing techniques **cannot adapt** to varying sequencing technologies and **do not scale** well for large genomes
- **Goal:** Propose a technology-independent and scalable assembly polishing algorithm -- Apollo
- **Key Ideas:**
  - Align reads to the erroneous contigs from the same sample
  - Construct a profile hidden Markov model (pHMM) for each contig
  - Use the read-to-contig alignments to update the parameters of pHMMs
  - Decode the consensus string from the updated pHMM to generate the corrected contig
- **Results/Observations**
  - Apollo is the **only** assembly polishing that is scalable to polish large genomes given the limited memory constraints (e.g., 192GB)
  - Apollo constructs the **most reliable assemblies** when hybrid set of reads (e.g., Illumina and PacBio) are used in a single run compared to other polishing tools
  - Apollo is ~25x slower on average (up to ~600x) than other polishers

Profile Hidden Markov Models (pHMMs)

- Three components:
  - States
  - Transitions (directed edges)
  - Emissions

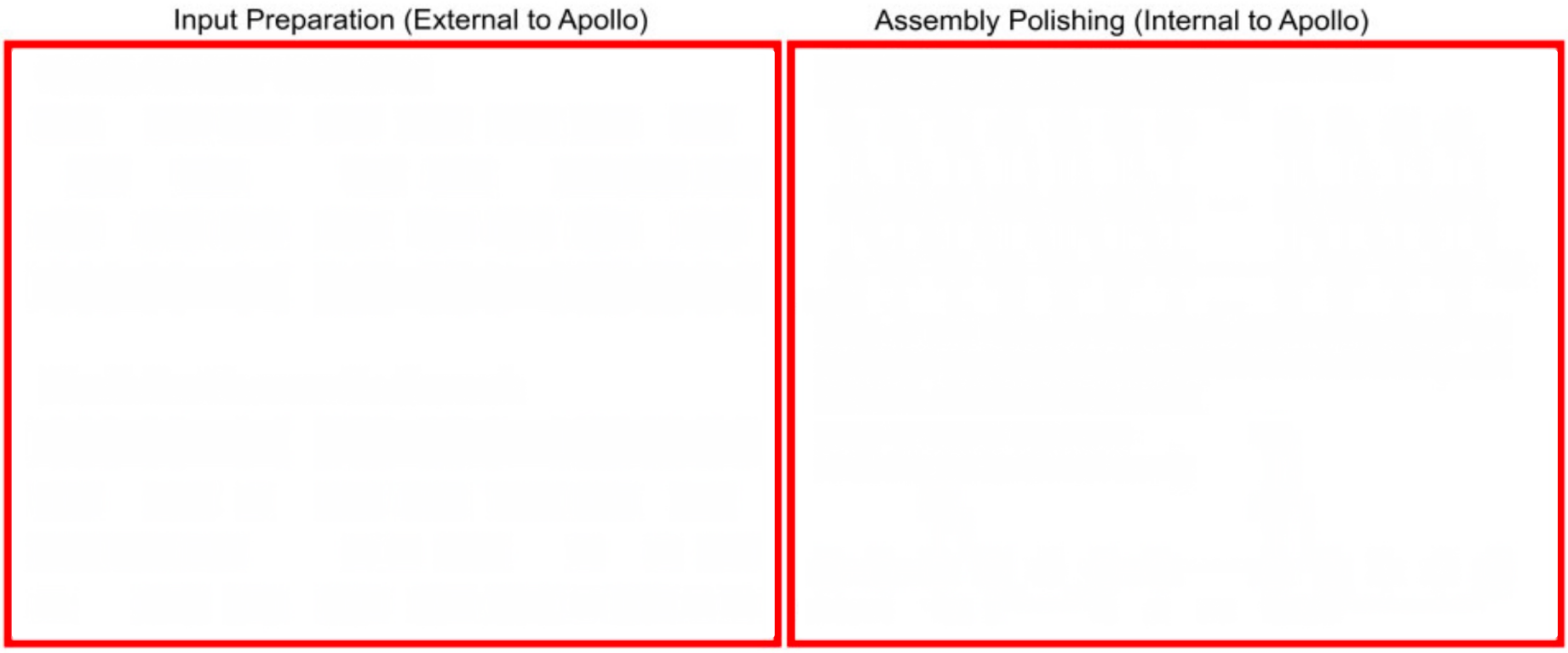


- Modification roles and probabilities are assigned to states
  - Substitution, insertion, deletion, or match (no modification)
- *A group of states* to perform all **probable** modifications on/after **each character** of a contig



Apollo Workflow

- Step1: An assembler uses erroneous long reads to construct contigs
  - Step2: We re-align the same reads (and additional reads) to contigs
- Steps 3-5: Apollo uses pHMMs to decode the consensus of alignments for a contig, which potentially eliminates majority of errors



Key Results

- State-of-the-art polishing tools: Racon, Pilon, Quiver, Nanopolish
- Scalability of polishing algorithms for a human genome
  - PacBio (35x and 8.9x) and Illumina (22x)
  - Racon, Pilon and Quiver exceeds memory requirements (192GB) when using high/medium coverage PacBio/Illumina reads
  - *Apollo is the only algorithm* that is scalable to polish large contigs given the memory constraints
- Pipeline to construct the most reliable contigs
  - Canu assembler rather than Miniasm
  - Polish using both long and Illumina reads (i.e., hybrid reads)
  - Apollo to use hybrid reads
    - It can use multiple read sets in a single run
- Apollo performs better than Nanopolish (~2-5x) but worse than Racon, Pilon, and Quiver (up to 600x, on average ~20-25x)

Future Work

- Apollo performs worse due to its computationally expensive parameter update (training) and decoding (inference) steps
  - Both training and inference steps are based on **embarrassingly parallel algorithms**
  - CPU *cannot* utilize all available parallelism
  - We implemented the training step in **GPU** and observe that we can achieve around 45x performance improvement compared to the CPU
    - Can we do better? **Hardware acceleration** for training?
  - **Combining training and inference steps** in an accelerator would potentially provide even better performance improvements
    - A generic pHMM accelerator rather than focusing only on Apollo
- **Parameter optimizations** for different sequencing technologies to improve sensitivity

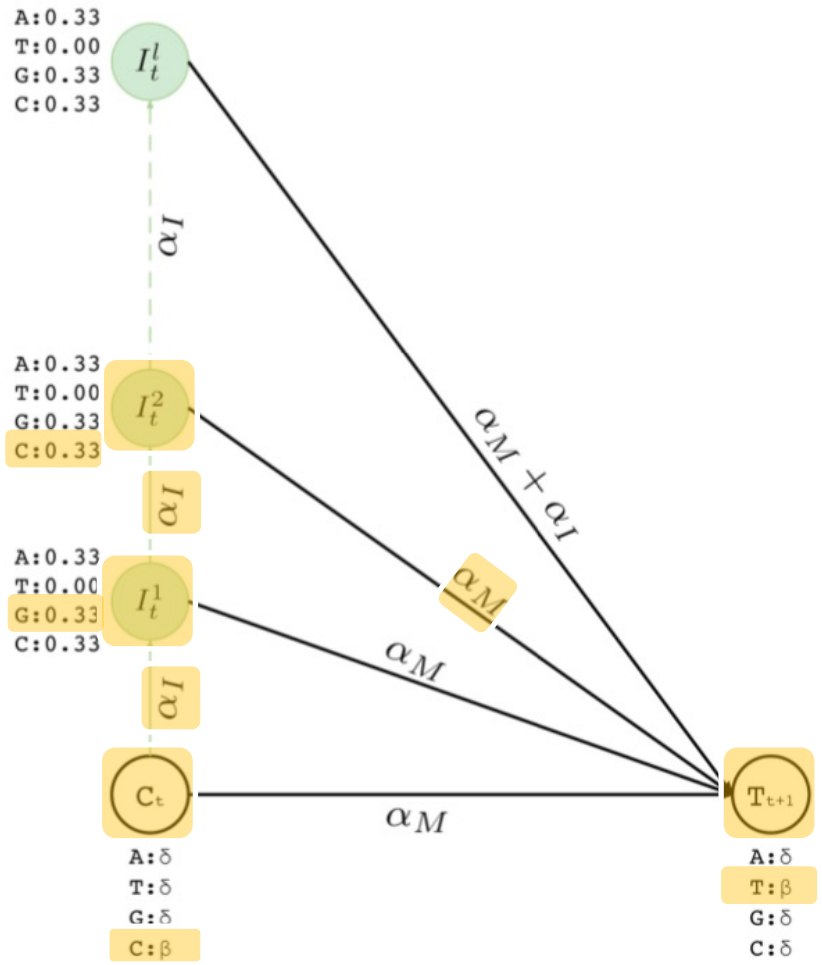
Executive Summary

- **Problem:**
  - Long read de-novo assembly is inherently **erroneous**
  - Existing assembly polishing techniques **cannot adapt** to varying sequencing technologies and **do not scale** well for large genomes
- **Goal:** Propose a technology-independent and scalable assembly polishing algorithm -- Apollo
- **Key Ideas:**
  - Align reads to the erroneous contigs from the same sample
  - Construct a profile hidden Markov model (pHMM) for each contig
  - Use the read-to-contig alignments to update the parameters of pHMMs
  - Decode the consensus string from the updated pHMM to generate the corrected contig
- **Results/Observations**
  - Apollo is the **only** assembly polishing that is scalable to polish large genomes given the limited memory constraints (e.g., 192GB)
  - Apollo constructs the **most reliable assemblies** when hybrid set of reads (e.g., Illumina and PacBio) are used in a single run compared to other polishing tools
  - Apollo is ~25x slower on average (up to ~600x) than other polishers

Backup Slides

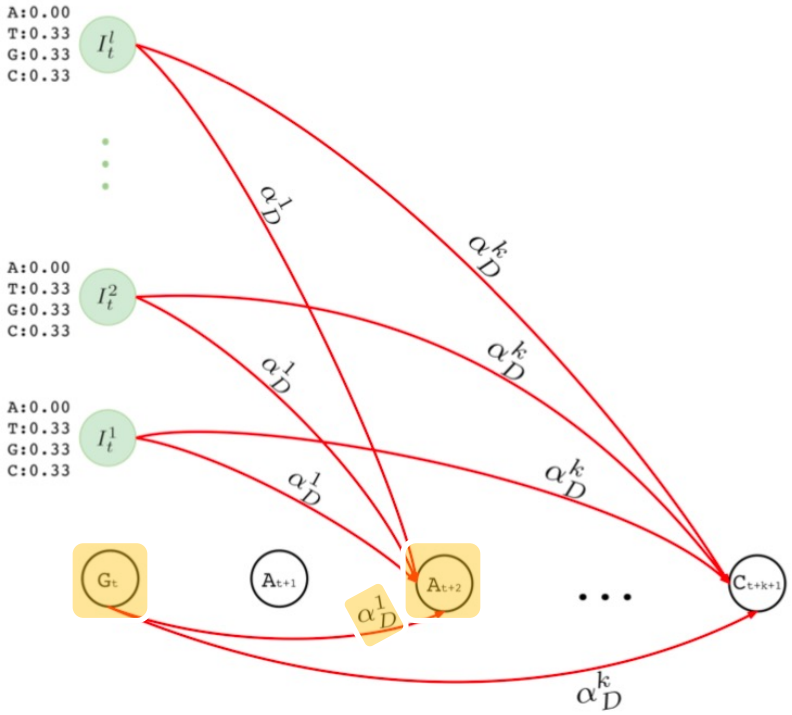
Resolving deletion errors

- Insertion states to insert *at most* / many bases *between two bases in a contig*
- To insert "GC" between "CT"
  - Visit match state at position t and *emit C*
  - Visit first *insertion state* after position t and *emit G with* ***deletion error*** *probability*
  - Visit second insertion state and *emit C with deletion error probability*
  - From second insertion state visit match state at position t+1 and *emit T*
  - Resulting sequence "CGCT"
- Maximum number of insertions is a parameter to Apollo



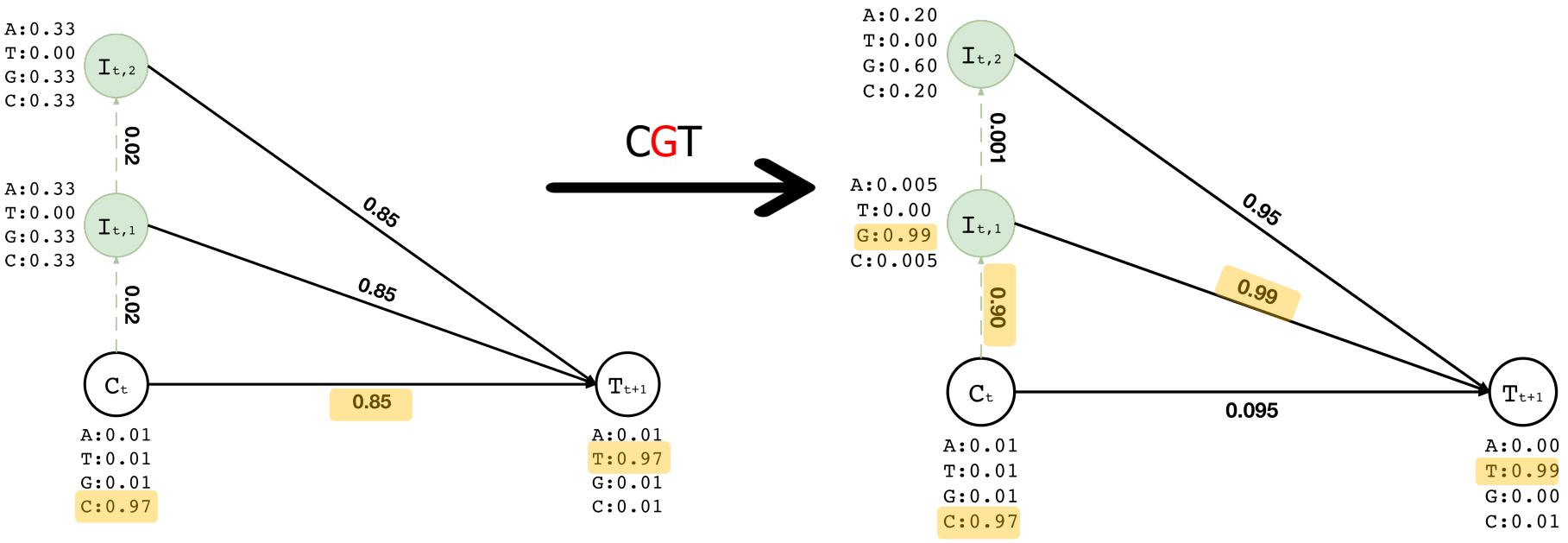
Resolving insertion errors

- Deletion transitions to delete one or many bases in a row
- To delete the first A in "GAA"
  - Visit match state at position t and *emit G*
  - Visit match state at position t+2 and emit A with *single* **insertion error** probability
  - Resulting sequence: "GA"
- Having single or more deletions in a row may not be necessarily equally likely
- Maximum number of deletions in a row is a parameter to Apollo



Training

- Training data:
  - Read aligned to the location t of a contig
- Assume we have the read "CGT" aligned to location t
- After training the corresponding region of the graph we would expect change in the probabilities so that it will be likely to emit "CGT" **somehow**



The Forward-Backward algorithm

- Calculating the likelihood of visiting a state to emit a certain character of a given sequence (i.e., aligned read)
  - Forward calculation (F)

$$F_1(j) = \alpha_{0j} e_j(r[1]) \quad s.t. \quad j \in V_s, \quad E_{0j} \in E_s$$

$$F_t(j) = \sum_{i \in V_s} F_{t-1}(i) \alpha_{ij} e_j(r[t]) \quad j \in V_s, \quad 1 < t \leq m$$

  - Backward calculation (B)

$$B_m(i) = \alpha_{i(m+1)} \quad i \in V_s, \quad E_{i(m+1)} \in E_s$$

$$B_t(i) = \sum_{j \in V_s} \alpha_{ij} e_j(r[t+1]) B_{t+1}(j) \quad j \in V_s, \ 1 \leq t < m$$

- Backward calculation needs a starting point

Training: The Baum-Welch algorithm

- Expectation maximization step using the Baum-Welch algorithm

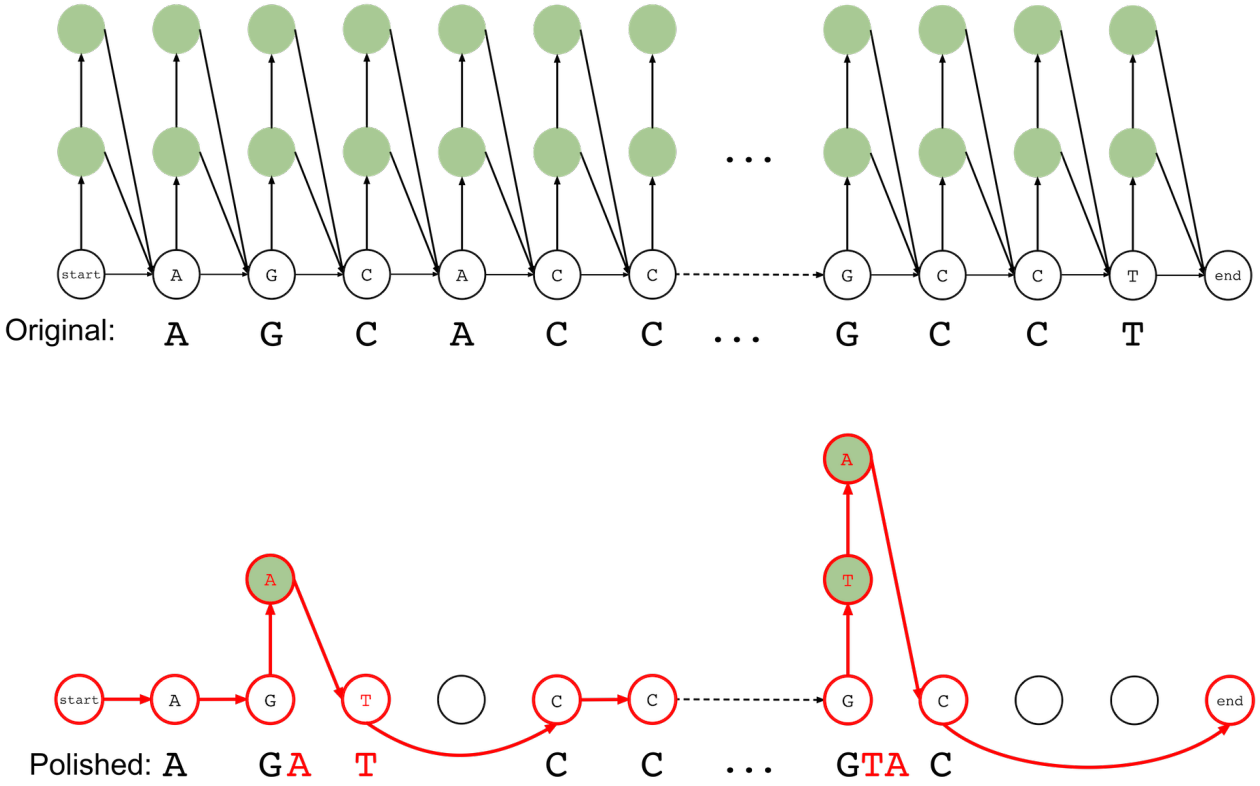
$$e_i^*(X) = \frac{\sum_{t=1}^{m} F_t(i) B_t(i) (r[t] == X)}{\sum_{t=1}^{m} F_t(i) B_t(i)} \quad \forall X \in \Sigma, \forall i \in V_s$$

$$\alpha_{ij}^* = \frac{\sum_{t=1}^{m-1} \alpha_{ij} e_j(r[t+1]) F_t(i) B_{t+1}(j)}{\sum_{t=1}^{m-1} \sum_{x \in V_s} \alpha_{ix} e_x(r[t+1]) F_t(i) B_{t+1}(x)} \quad \forall E_{ij} \in E_s$$

- If there are multiple reads aligning to same region, we have multiple F(i) for a position t
  - Take the average and use it as F(i) for position t

Inference: The Viterbi algorithm

- Our original contig before polishing was: "AGCACC…GCCT"
- After updating the probabilities, the most likely path from start to end reveals the corrected contig: "AGATCC…GTAC"



Data Sets

Data Set	Accession Number	Details
E.coli K-12 - ONT	Loman Lab*	164,472 reads (avg. 9,010bps, 319X coverage) via Metrichor
E.coli K-12 - Ground Truth	GenBank NC_000913	Strain MG1655 (4,641Kbps)
E.coli O157 - PacBio	SRA SRR5413248	177,458 reads (avg. 4,724bps, 151X coverage)
E.coli O157 - Illumina	SRA SRR5413247	11,856,506 paired-end reads (150bps each, 643X coverage)
E.coli O157 - Ground Truth	GenBank NJEX02000001	Strain FDAARGOS_292 (5,566Kbps)
E.coli O157:H7 - PacBio	SRA SRR1509640	76,279 reads (avg. 8,270bps, 112X coverage)
E.coli O157:H7 - Illumina	SRA SRR1509643	2,978,835 paired-end reads (250bps each, 265X coverage)
E.coli O157:H7 - Ground Truth	GCA_000732965	Strain EDL933 (5,639Kbps)
Yeast S288C - PacBio	SRA ERR165511(8-9), ERR1655125	296,485 reads (avg. 5,735bps, 140X coverage)
Yeast S288C - Illumina	SRA ERR1938683	3,318,467 paired-end reads (150bps each, 82X coverage)
Yeast S288C - Ground Truth	GCA_000146055.2	Strain S288C (12,157Kbps)
Human CHM1 - PacBio	SRA SRR130433(1-5)	912,421 reads (avg. 8,646bps, 2.6X coverage)
Human CHM1 - Ground Truth	GCA_000306695.2	3.04Gbps
Human HG002 - PacBio	SRA SRR2036(394-471), SRR203665(4-9)	15,892,517 reads (avg. 6,550bps, 35X coverage)
Human HG002 - Illumina	SRA SRR17664(42-59)	222,925,733 paired-end reads (148bps each, 22X coverage)
Human HG002 - Ground Truth	GCA_001542345.1	Ashkenazim trio - Son (2.99Gbps)

Experimental Setup

- CPU: Intel®Xeon®Gold 5118 CPU @ 2.30GHz
  - 24 cores (2 threads per core)
- Max memory: 192GB
- Assigned 45 threads to all tools
- Apollo was compared with the state-of-the-art polishing tools
  - Racon, Pilon, Quiver, Nanopolish