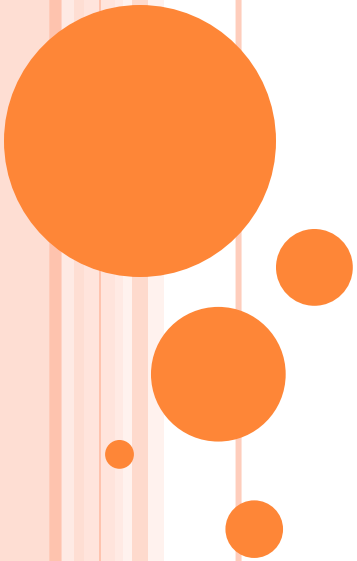


Project 7: Alu Detection (Easy)

Spencer Brett

CS CM 124

5/23/2014

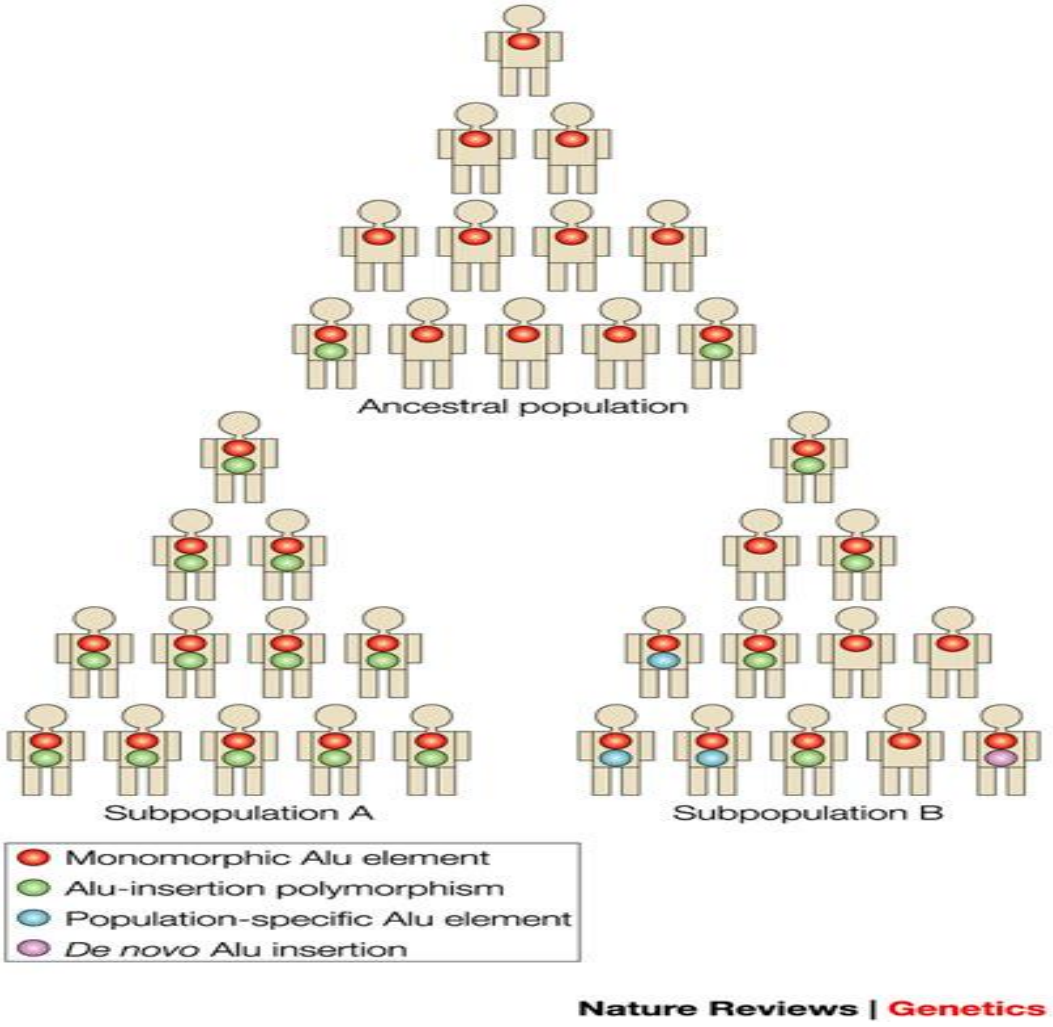
Background

- Alu elements are classified as SINEs, or Short INterspersed Elements. All Alus are approximately 300 bp in length.
- Human chromosomes contain about 1,000,000 *Alu* copies, which equal 10% of the total genome.
- *Alu* is an example of a so-called "jumping gene" – a transposable DNA sequence that "reproduces" by copying itself and inserting into new chromosome locations.

Motivation

- *Alu* elements are found only in primates.
- *Alu* elements can be sorted into distinct lineages, or families, according to inherited patterns of new mutations.
- Once an *Alu* is inserted, there is no evidence that it is ever excised or lost from a chromosome. So, each *Alu* insertion is stable through evolutionary time.
- Using this information we can infer common ancestors and learn how primates are related through time.

Motivation


Ancestral population
Subpopulation A
Subpopulation B
Monomorphic Alu element
Alu-insertion polymorphism
Population-specific Alu element
De novo Alu insertion
Nature Reviews | Genetics

Problem

- Given a reference genome, Alu sequence, and paired end reads corresponding to a donor genome, we want to find the locations of the Alu sequence in the reference genome as well as detecting any potential Alu insertions in the donor genome
- **Metrics**
  - Computation Time
  - Accuracy of result
    - How close the calculated position is
    - Incorrect detection

Tools

- Python for everything
  - Programmed everything from scratch
  - Easy to use and easy to read
  - Fantastic dynamic data structures built in
  - Great for rapid prototyping
  - Great for getting results quickly (even if you don't like them)
  - Not very memory efficient or fast

Data

- Inputs: Reference genome, paired end reads, Alu sequence
- Alu Sequence(Length 300): Generate a random sequence of length 300
- Reference Genome(Length N, x Alus): Generate a random sequence of length N, insert x Alu sequences in random positions in the genome
- Paired End Reads(Read Length L, Coverage C, Alteration Frequency f, Mean distance d, Standard Deviation std_dev):
  - Alter reference genome at frequency f, and insert 2 new Alu sequences
  - Generate (N * C)/L reads set apart a certain distance following a normal distribution centered at d with standard deviation std_dev through sampling of the altered genome with new Alu insertions

Baseline Method

- Stage 1: Basic Search
- Stage 2: Identify Candidates
- Stage 3: Identify Anchors
- Stage 4: Calculate Approximate Position
- Stage 5: Cluster Data Points
- Stage 6: Average Clusters

Add an Optimization!

- Stage 1: Basic Search

Bonus Stage: Filter Potential Candidates!

- Stage 2: Identify Candidates
- Stage 3: Identify Anchors
- Stage 4: Calculate Approximate Position
- Stage 5: Cluster Data Points
- Stage 6: Average Clusters

Stage 1: Basic Search

- Scan the reference genome for all occurrences of the Alu sequence
  - Assuming no errors is a bit quicker

Bonus Stage: Filter Potential Candidates

- Filter all the pair reads down to those that have at least one that maps to the Alu sequence
- Filter all the pair reads that don't map at all
  - This can be due to an odd concentration of alterations in the donor genome over our threshold or a read that crosses over a newly inserted Alu boundary point

Stage 2: Identify Candidates

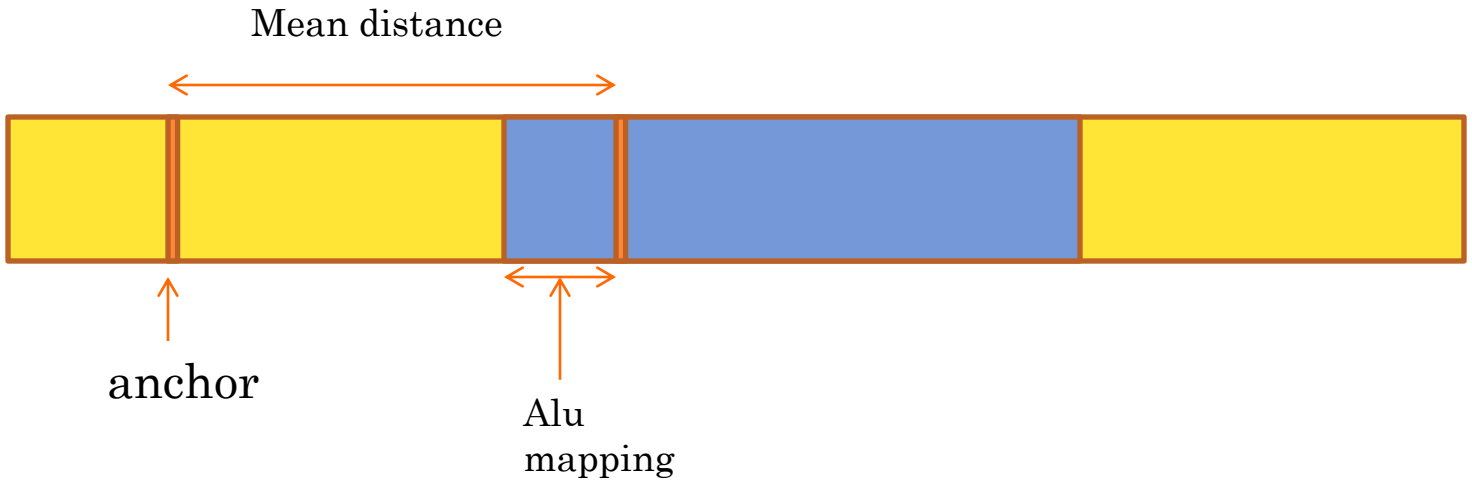
- Given an approximate expected distance between reads, we can identify candidates by mapping the pairs
- If the left and right reads of the pair do not have a mapping following the expected distance, we label this as a mismatched pair and add it to the return list
- This step is still necessary for the optimized version because the filter includes Alus from the reference sequence that remain

Stage 3: Identify Anchors

- Given our candidate pairs, we find which maps to the Alu sequence and where in that Alu sequence it maps
- We do this by simply remapping each read from each pair to the Alu sequence and getting a position
- The read that does not map to the Alu sequence is the anchor

Stage 4: Calculate Approximate Position

- Using the anchor, approximate pair distance, and Alu mapping position, we get an approximation of where the Alu was inserted
- The anchor plus the distance, closed by the Alu mapping position

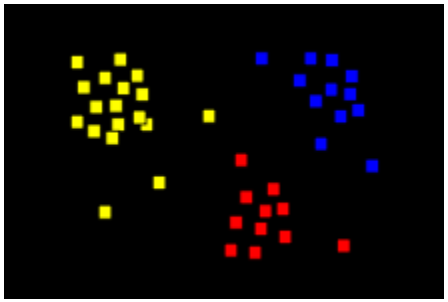


Stage 5: Cluster Data Points

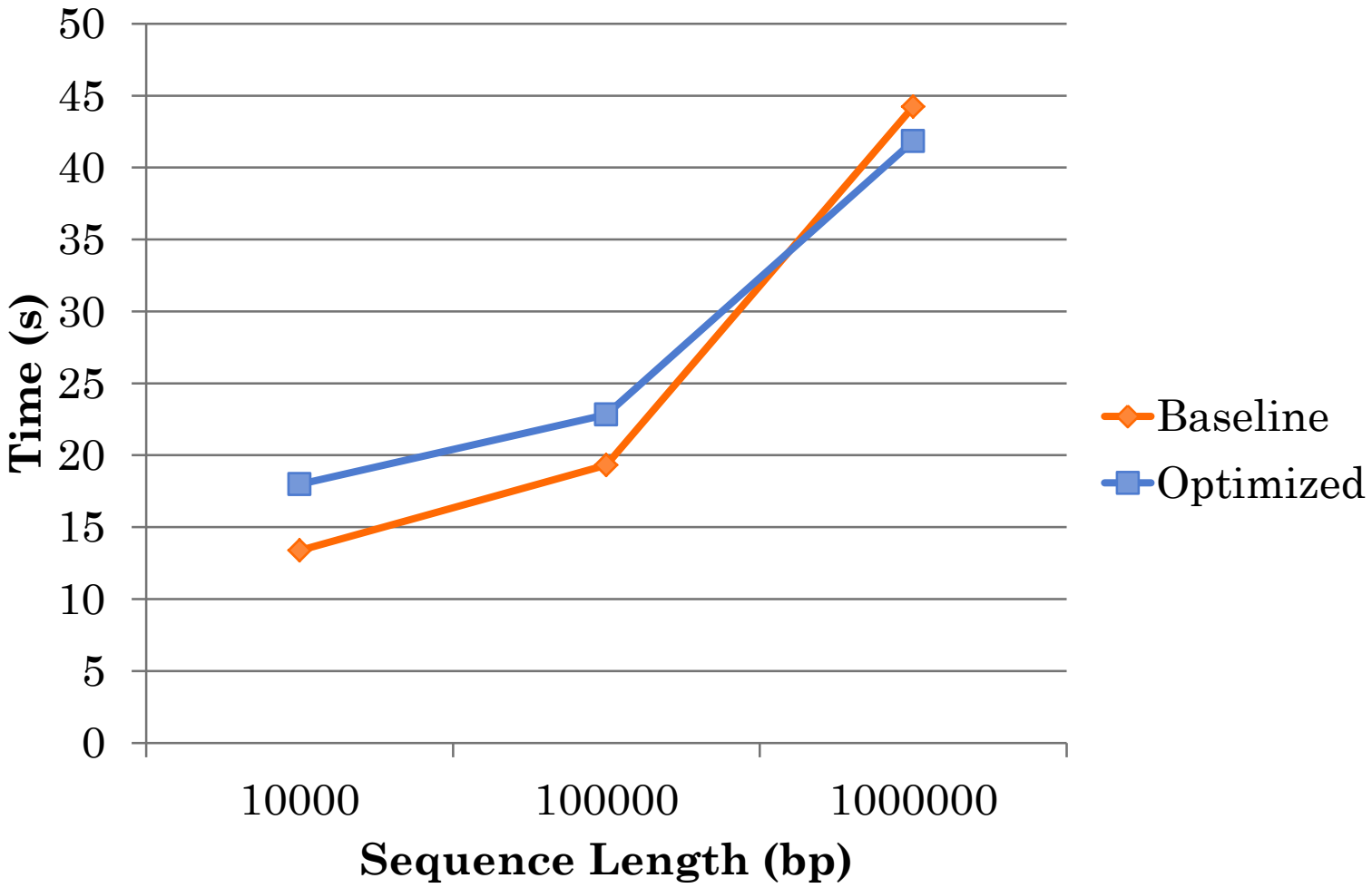
- Through previous steps, we have collected a bunch of approximate positions where we think Alu insertions may have occurred
- Because there is variance in the paired end read distance, these approximations have variance as well
- Using this variance as a threshold, we cluster the data points so that they are likely to describe the same Alu insertion

Stage 6: Average Clusters

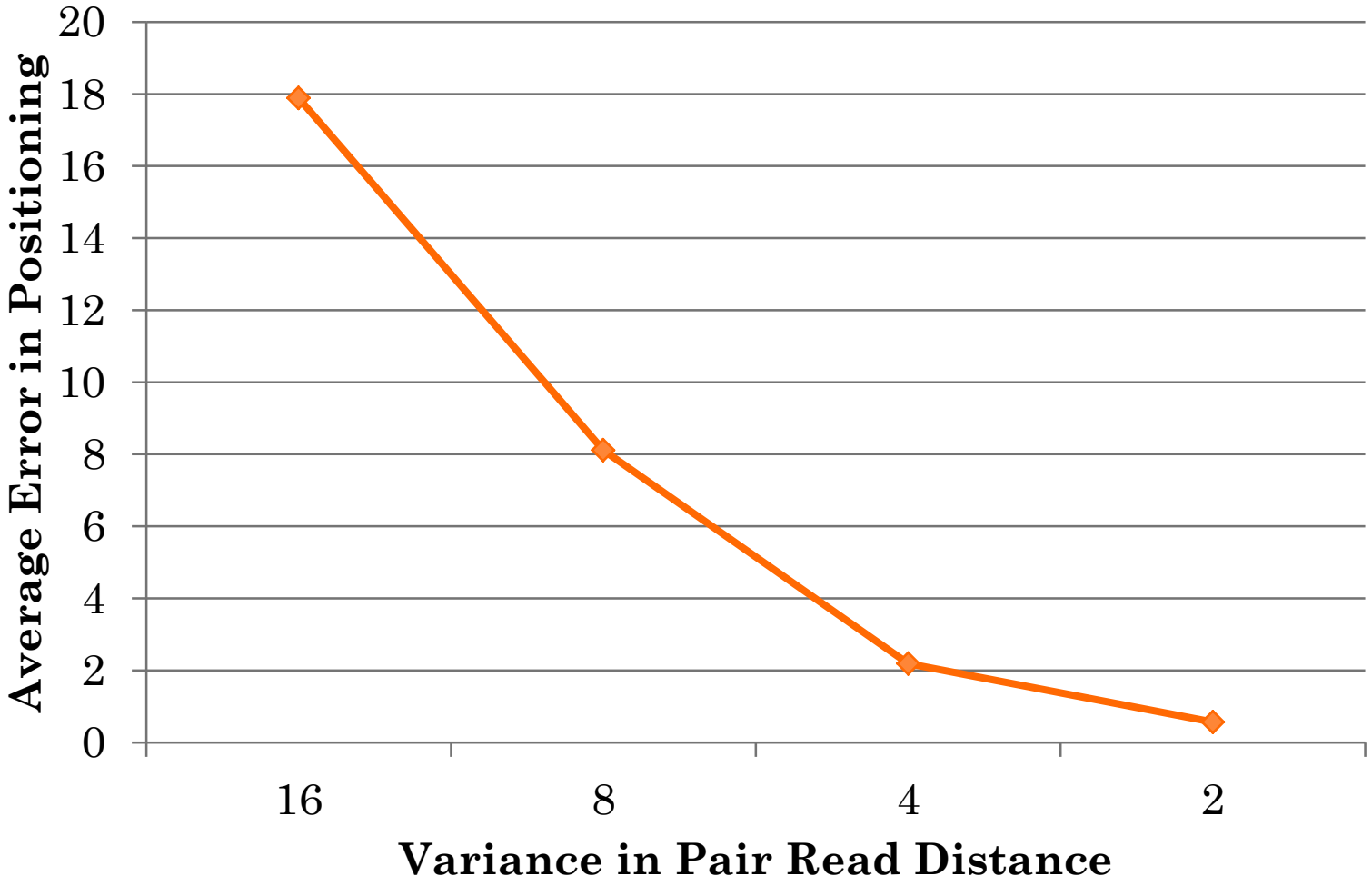
- Simply take the rounded average of each cluster of data
- If the paired end read distance varies little, the data should be closer to the correct position



Results: Computation Time



Results: Accuracy



Observations

- Data depended largely on accuracy of reads (obviously)
- Finding Alus in the reference genome is dependent on them not interlacing. Otherwise, accuracy was nearly 100% (not very interesting)
- Slower than I'd like, there are many repeat operations in favor of modularity
- My optimization did not really work out as I'd hoped

References

- http://www.geneticorigins.org/pv92/aluframeset.htm
- http://www.nature.com/nrg/journal/v3/n5/images/nrg798-f4.jpg