Read Alignment

CS 4390/5390
Fall 2019

Adapted from Mäkinen, *et al.* Chapter 10

Computational Problem

Given

- a reference genome G, and
- a set of reads $R = (r_1, r_2, r_3, \ldots, r_k) \in (\Sigma^n)^k$ where each read r is a subsequence of G with a small number changes

Output

- the semi-global alignment of r_i and G for all $r_i \in R$

Computational Problem

Given

- a reference genome G, and
- a set of reads $R = (r_1, r_2, r_3, \ldots, r_k) \in (\Sigma^n)^k$ where each read r is a subsequence of G with a small number changes

Output

- the semi-global alignment of r_i and G for all $r_i \in R$ with $<k$ changes

call these k-error mappings

Read Filtering

If a read is long enough, it should not align well to a random region of *G*

This assumes that the sequence was read correctly

Sequencing machines output a *quality score* for each position of a read

- this can be interpreted as a probability *P(r[j])* that the character is correct
- in other words with probability *P(r[j]),* position *r[j]* is a random character

This means that a given sequence will match a random sequence with probability

$$\mathbb{P}(r) = \prod_{1 \leq i \leq n} \left(\mathbb{P}\left(r\left[j\right]\right) q_{r[j]} + \left(1 - \mathbb{P}\left(r\left[j\right]\right)\right) \right)$$

- where q_c is the probability of c in a random sequence

Read Filtering

We expect the number of random matches between *r* and a given string *T* such that $|T| = m$ to be $\mathbb{E}(r, T) = (m - n + 1)\mathbb{P}(r)$

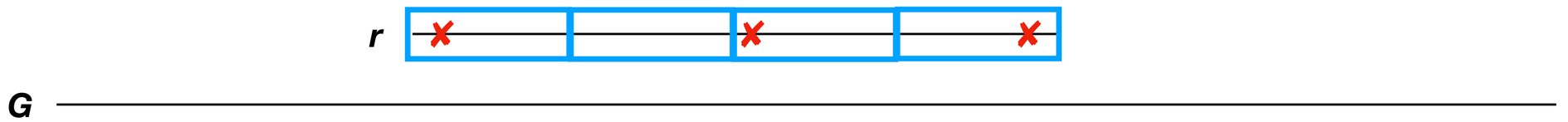
We can then threshold the reads that are too low quality by this expectation

Pigeonhole Principle

Assume for now we're not dealing with insertions or deletions

The **pigeonhole** principle in this case says that if the read is partitioned into *k+1* pieces, one must appear in the genome exactly if the read has a *k*-error mapping.

All *k*-error mappings will have at least one exact match, not all exact matches lead to *k*-error mappings



Initial ideas to read mapping

Construct an index of *G* (say a succinct suffix array)

Search for each of the pieces of the read in the index

Verify each occurrence's alignment against that region of *G*

$O(m \log \sigma + (\log^{(1+\varepsilon)} n + m^2/w)\, c)$ time

number of candidates

Aligning reads

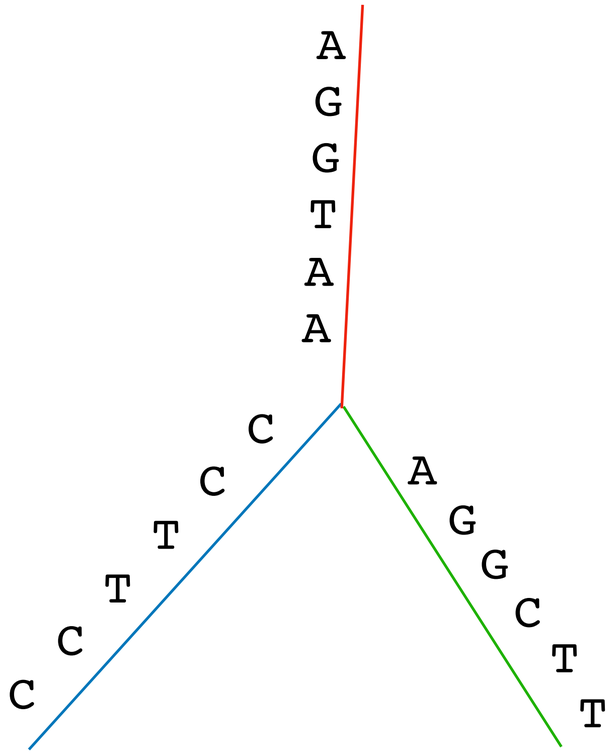
We mentioned that we want the semi-global alignment of each read to the genome, ignoring any deletions (from the genome) at the start or end of the alignment

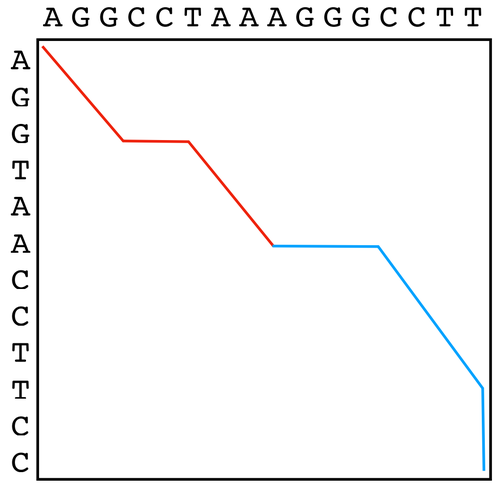
We can align the read along the suffix tree of the genome, where each row is a position in the genome

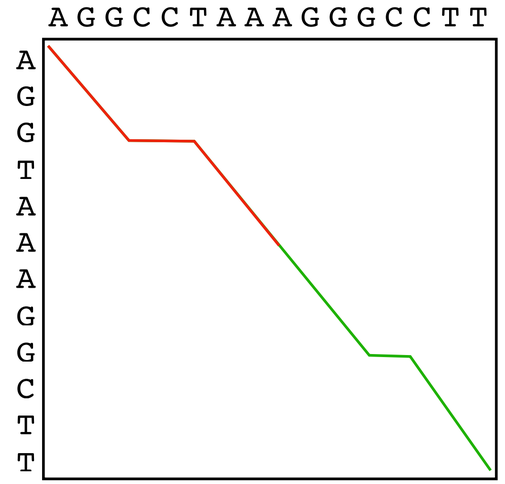
parts of the alignment matrix will be shared.

Aligning reads

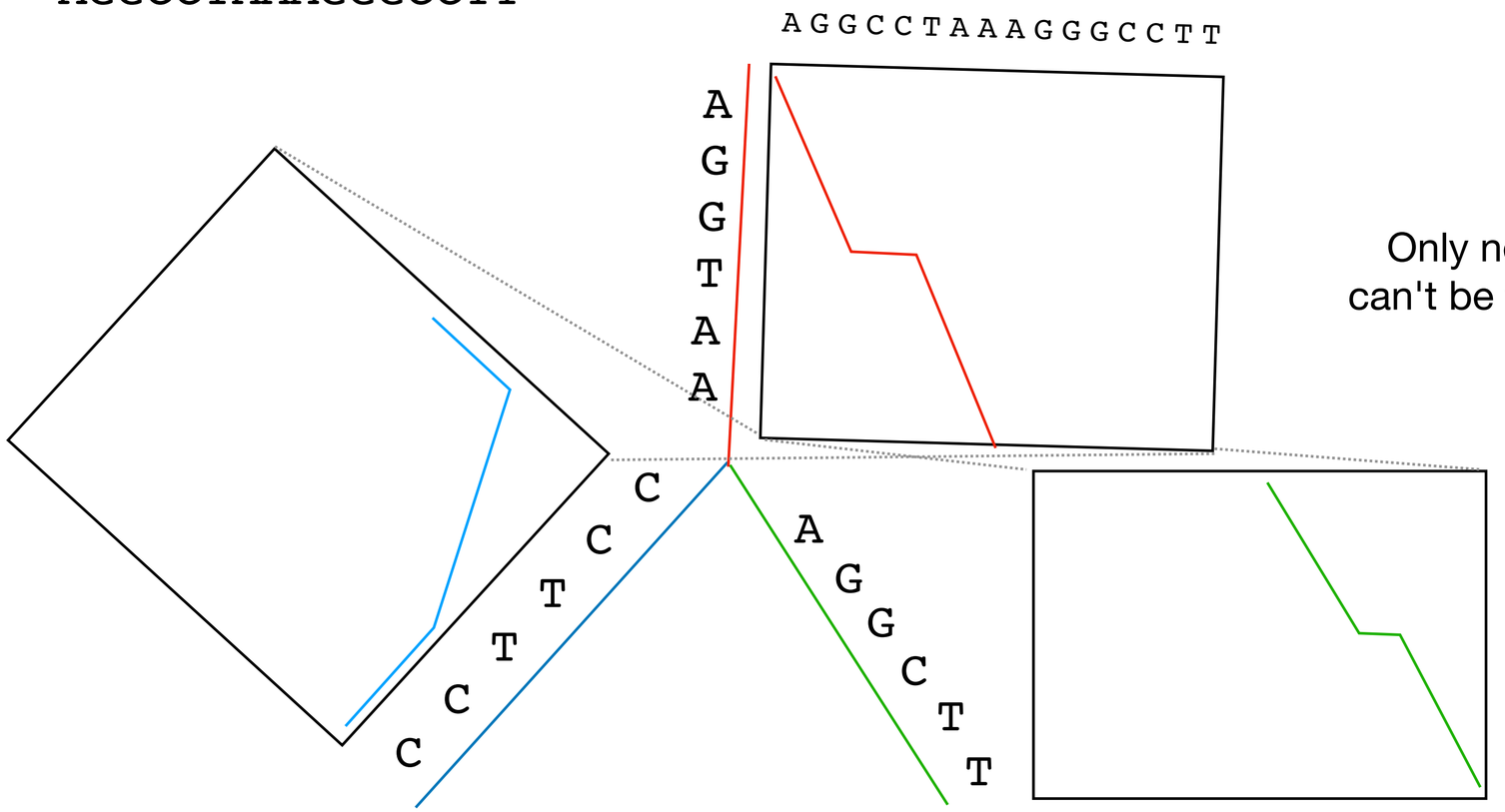
AGGCCTAAAGGGCCTT







Aligning reads



Only need to go to a depth of *2m* since the best alignment can't be worse than deleting one string and inserting the other.

We don't have the suffix tree!

Dynamic Programming using a BWT

Since we want to save the previous rows

- we can read the characters one-by-one from the sequence
- when you reach the max depth, backtack up the tree to the last branch
- overwrite the new rows with the characters read down the new branch

How do we backtrack on a BWT?

Dynamic Programming using a BWT

```
define Branch(d,[i...j]):
    for c ∈ idx.enumerateRight(i,j) do
        process (c,d)
        if d = 2m and score > threshold do
            output alignment
        if d < 2m do
            Branch(d+1,idx.extendRight(c, [i,j]))
```

compute the dynamic programming table row using character *c* in row *d*

$O(m\sigma)$-time

$O(m^2+m\sigma)$-space

Prefix Pruning

The full dynamic program is still slow

What if we go back to hamming distance, but still use the BWT

Prefix Pruning

```
define Branch(d,k,[i...j]):
    for c ∈ idx.enumerateRight(i,j) do
        if p[d] ≠ c do
            k = k-1
        if k ≥ 0 do
            if d = m do
                output locations in [i,j]
            else
                Branch(d+1, k, idx.extendRight(c,[i,j]))
```

O(mσ)-time

O(mσ)-space

Approximate Overlaps

If we're not given a reference genome, we are left to do *de novo* assembly.

The first step is known as overlap mapping.

Given a set of reads $\boldsymbol{R} = \{R^1, R^2, \ldots, R^d\}$ find the set of suffix-prefix overlaps (R^i, R^j, o^{ij}).

- Search is performed using the **reverse** BWT for $T = R^1\$^1R^2\$^2R^3\$^3 \ldots R^d\$^d\$$

Paired-end reads

We mentioned previously that many times reads come in pairs from the sequencer

These pairs can be used to determine exactly where a read maps (in the case of reads that can be placed in multiple locations)

Mapping can be done independently (useful for large scale change detection), but can also be performed together

- Search performed using the suffix array of the genome, which is large

Split alignment of reads

In RNA-Sequencing (RNA-Seq) the reads will have the introns removed, meaning there will be large gaps in the mapping position on the genome

If we have a complete *transcriptome* (all possible spliced transcripts), we could map reads to that, but we may not have it

There is no clean solution to this, possibilities include:

- find all error free regions of a read, piece them together if the distances are reasonable
- predict exon boundaries in the read, then align those contiguous regions