Minimap and other things

Alex Ledger

What we’re going to talk about

- Minimap
- What Minimap does
  - How minimap fits into the literature and pipeline
- Go over the Minimap algorithm
  - Walk through the pseudocode
- Reflect on the algorithm

Mapping an individual genome (Big Picture)

1. Input: a person
2. Person goes to lab, has some tests done
    a. short reads
    b. long reads
3. Plug reads + reference genome into an algorithm
    a. Map the genome to the reference genome (Minimap!)
    b. For each read, find where the location where it is best associated on the reference genome
4. Output: A person’s genome
    a. along with how the individual is the same/different from the reference genome
    b. i.e., a list of structural of variants

What is Minimap?

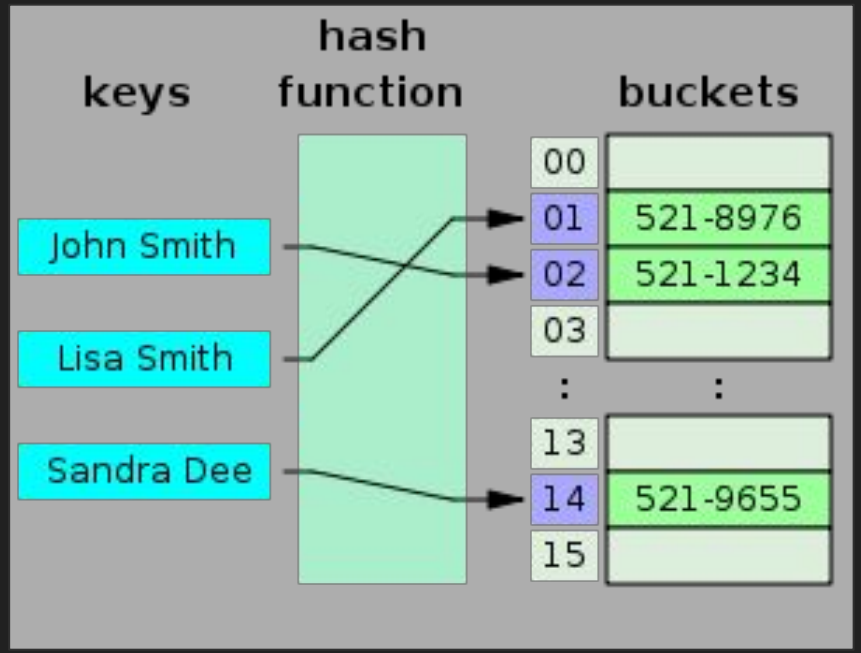
- Minimap is a new, popular mapping algorithm
  - Created by Heng Li
  - Input: list of reads and reference genome
  - Output: An assignment to every read to a location on the reference genome
- What's new about it?
  - designed for SMRT sequencing technologies
  - Minimap uses the following:
    - Hash table
      - BLAST (1997) and BLAT (2002)
    - Merging and sorting lists (highly cache efficient operations)
      - DALINGER (2014)
    - Minimizing hashes
      - MHAP (2015)
  - Best feature: it's fast
- Not actually published - preprint on Arxiv (http://arxiv.org/abs/1512.01801)

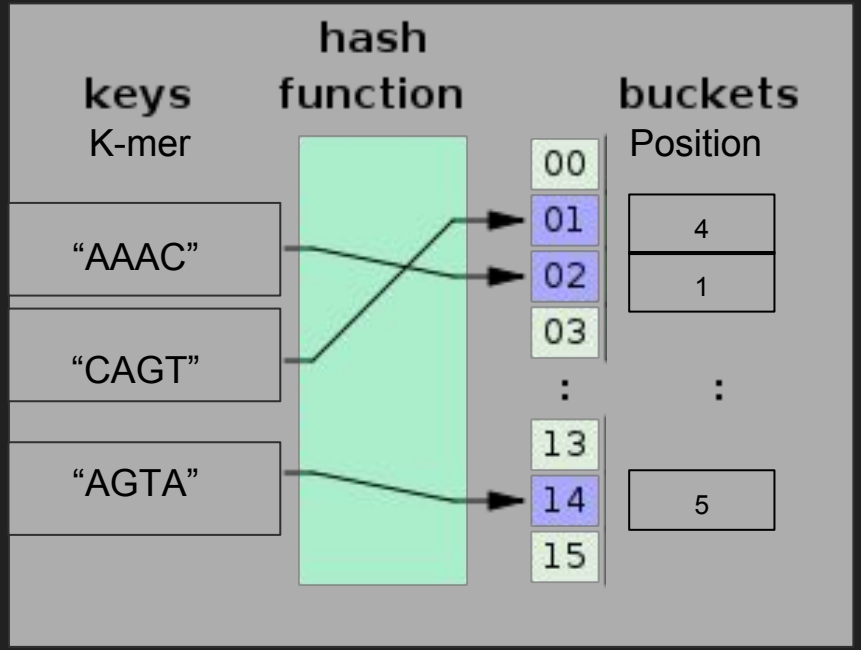
Terminology

- Read
  - A string of DNA-bases that are unmapped
  - Looks like: ACGGTACCACTTG
- K-mer
  - A string of k DNA-bases (aka a substring)
  - K = 3 gives ACG
- Reference Genome
  - A huge sequence of DNA-bases whose order (we are confident) is correct
- Mapping
  - The most likely position that a k-mer corresponds to on the reference genome

On Hash Tables

- Hash table = your python dictionary
- Key-value pair <k,v>
- Dict[”John Smith”] = 521-8976
- H[“AAAC”] = 4
- <”CAGT”, 1>





Hash Table for read: “AAACAGTA”

Minimap in clipart: (Parameters: k = 2, w = 1)

Compute Minimizers

Aka, Make-Hash-Table

- Inputs:
  - Read s
  - integer w: window size
  - integer k: substring size
- Loop over read s
  - Loop over window
    - find min hash of k-mers in window
  - Save min hash of window

Algorithm 1: Compute minimizers

Input: Parameter w and k and sequence s with $|s| \geq w + k - 1$

Output: (w,k)-minimizers, their positions and strands

Function MINIMIZERSKETCH(s, w, k) **begin**

  $\mathcal{M} \leftarrow \emptyset$ ▷ *NB: $\mathcal{M}$ is a set; no duplicates*

  for $i \leftarrow 1$ **to** $|s| - w - k + 1$ **do**

    $m \leftarrow \infty$

1   **for** $j \leftarrow 0$ **to** $w - 1$ **do** ▷ *Find the min value*

      $(u, v) \leftarrow (\phi(s^k_{i+j}), \phi(\overline{s}^k_{i+j}))$

      if $u \neq v$ **then** ▷ *Skip if strand ambiguous*

        $m \leftarrow \min(m, \min(u, v))$

2   **for** $j \leftarrow 0$ **to** $w - 1$ **do** ▷ *Collect minimizers*

      $(u, v) \leftarrow (\phi(s^k_{i+j}), \phi(\overline{s}^k_{i+j}))$

      if $u < v$ **and** $u = m$ **then**

        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(m, i + j, 0)\}$

      else if $v < u$ **and** $v = m$ **then**

        $\mathcal{M} \leftarrow \mathcal{M} \cup \{(m, i + j, 1)\}$

  return M

Map it

- Inputs:
  - Hashtable H
    - reference genome hashes saved here
  - Query sequence q
    - string being mapped to genome
  - integer w: window size
  - integer k: substring size
  - integer g: cluster variance (they set to 4)
    - How different
- For each tuple in HashTable[q]
  - For each corresponding hash in H
    - save to array A
- b = 1
- # inspired by Hough Transformation
- For each element A[e] in A
  - if A[e] is different in any from A[e+1]
    - Add A[b..e] to C
    - Print
    - b = e + 1 # reset b to this location

Algorithm 4: Map a query sequence

Input: Hash table $\mathcal{H}$ and query sequence q
Output: Print matching query and target intervals

Function MAP$(\mathcal{H}, q, w, k, g)$ **begin**
  $\mathcal{A} \leftarrow$ empty array
  $\mathcal{M} \leftarrow$MINIMIZERSKETCH(q, w, k)
1 **foreach** $(h, i, r) \in \mathcal{M}$ **do** ▷ *Collect minimizer hits*
    foreach $(t, i', r') \in \mathcal{H}[h]$ **do**
      if $r = r'$ **then** ▷ *Minimizers on the same strand*
        Append $(t, 0, i - i', i')$ to $\mathcal{A}$
      else ▷ *On different strands*
        Append $(t, 1, i + i', i')$ to $\mathcal{A}$
  Sort $\mathcal{A} = [(t, r, c, i')]$ in the order of the four values in tuples
  $b \leftarrow 1$
2 **for** $e = 1$ **to** $|\mathcal{A}|$ **do** ▷ *Cluster minimizer hits*
    if $e = |\mathcal{A}|$ **or** $\mathcal{A}[e+1].t \neq \mathcal{A}[e].t$ **or** $\mathcal{A}[e+1].r \neq \mathcal{A}[e].r$
    or $\mathcal{A}[e+1].c - \mathcal{A}[e].c > g$ **then**
3 $\mathcal{C} \leftarrow$ the maximal colinear subset of $\mathcal{A}[b..e]$
      Print the left- and right-most query/target positions in $\mathcal{C}$
      $b \leftarrow e + 1$

My Hypotheses

1. Statistical bias in read mappings because comparing hash function
    - certain k-mers are more likely to be mapped than others
    - Minimap addresses this by:
        - Picking good parameters (specifically, the ratio w:k)
    - Also sorting A based on hash function
2. Hash functions aren't great for highly erroneous reads
    - Good when error rates was 99%, but SMRT reads have 15% error rates
        - if two k-mers are different by one base, their hashes are totally different
        - because hash is pseudorandom (if it's good)
    - Weird because Minimap is specifically designed for SMRT sequencing
    - Minimap addresses this by:
        - Using windows
        - "Maximal collinear subset" of mapped k-mers

My current thinking

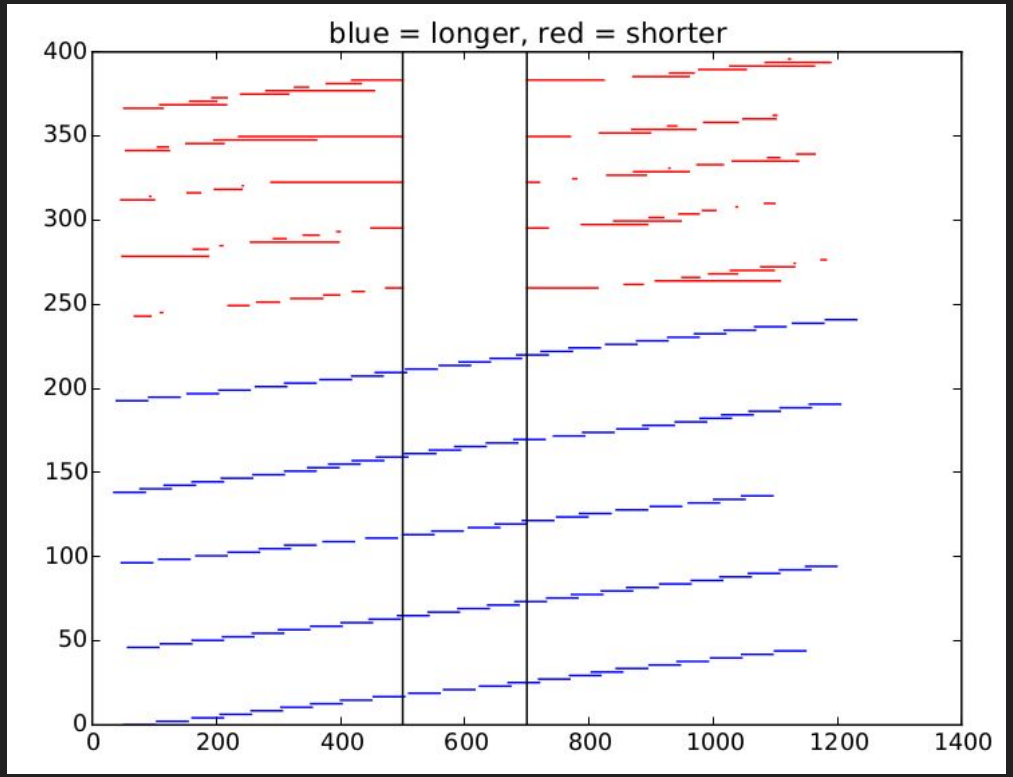
- Use a distance-preserving hash function
  - A hash function that preserves distances between strings
  - Then finding similar subsequences is achievable with hashing
- Is comparing hash values critical?
  - It seems unintuitive to me.
  - If there is a distance preserving hash value, maybe you can take the mean of a window?
- Understand the usage of a window
  - What is the point of a window?
  - I think it relates to the high error rate
- Some questions are answerable in two ways:
  - Run experiments on Minimap
  - Look at MHAP

Assembly

- Miniasm
  - Start with two reads u and v.
    - Trying mapping u to v and v to u
    - If they map, then we say they overlap
  - Make a graph
    - Treat all reads as nodes
    - Two reads/nodes share an edge if they overlap
  - Run some graph algorithm on this graph to generate genome

End

Extra slides beyond here



Auxiliary Algorithm (don’t even talk about these)

Algorithm 3: Index target sequences

Input: Set of target sequences $\mathcal{T} = \{s_1, \ldots, s_T\}$
Output: Minimizer hash table $\mathcal{H}$

Function INDEX$(\mathcal{T}, w, k)$ **begin**
    $\mathcal{H} \leftarrow$ empty hash table
    for $t \leftarrow 1$ **to** T **do**
        $\mathcal{M} \leftarrow$MINIMIZERSKETCH(s_t, w, k)
        foreach $(h, i, r) \in \mathcal{M}$ **do**
            $\mathcal{H}[h] \leftarrow \mathcal{H}[h] \cup \{(t, i, r)\}$
    return $\mathcal{H}$