Cost per Raw Megabase of DNA Sequence

Moore's Law

NIH National Human Genome Research Institute

genome.gov/sequencingcosts
Two different strategies for parallel amplification

- BRIDGE PCR
- EMULSION PCR
Two different strategies for parallel amplification

BRIDGE PCR

EMULSION PCR
Two different strategies for parallel amplification

BRIDGE PCR

EMULSION PCR
Many of the problems of HTS inhere to:

1) Massively parallel DNA amplification (errors, etc)

2) Very short reads (mapping problems)

What if we could sequence long, unamplified individual DNA molecules?

How could you do this?
Single Molecule Sequencing: Fluorescent

Uses zero-mode waveguide enables detection of fluorescence only very near the bottom of the nanowell
Each nanowell contains one DNA polymerase and one template DNA molecule.
Sequencing is detected in real time by looking for pulses of fluorescence associated with base addition.
Fluorophores are attached to the triphosphate group so they diffuse away from the detection volume rapidly!
Protein nanopores enable water and ions to cross membranes…
The result is a current flow if there is a difference in potential across the membrane.
If a large molecule passes through the aperture of the pore, current flow is temporarily reduced.
Single Molecule Sequencing: Nanopores

Objects of different sizes cause different (and characteristic) current flows.
Single Molecule Sequencing: Nanopores

Sensing the different current flow induced by each different base is the key concept of nanopore sequencing.
Genome 373: Mapping Short Sequence Reads I

Doug Fowler
What can a sequencer do today?

In 3 days...

“Paired end” 150 bp reads

>6 billion read-pairs

>1.8 terabases (Gb) of total output

1 in 100 error rate for most base-calls

Initial draft of human genome based on 23 Gb
Outline

• The short read mapping challenge

• Spaced seed alignment
Short read mapping

• Input:
  – A reference genome
  – Millions of 25-100 bp reads (single or paired)
  – User-specified parameters
Short read mapping

• **Input:**
  – A reference genome
  – Millions of 25-100 bp reads (single or paired)
  – User-specified parameters

• **Output:**
  – Genomic coordinates for each read
The first big problem we confront is scale – hundreds of millions of reads mapping to 3 billion possible bases in the human genome.
Short read mapping

• **Input:**
  – A reference genome
  – Millions of 25-100 bp reads (single or paired)
  – User-specified parameters

• **Output:**
  – Genomic coordinates for each read

• **Question:**
  – Do you expect most reads to map to one location in the genome? Why or why not?
The Multiple Mapping Problem

<table>
<thead>
<tr>
<th>Repeat class</th>
<th>Repeat type</th>
<th>Number (hg19)</th>
<th>Cvg</th>
<th>Length (bp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minisatellite, microsatellite or satellite</td>
<td>Tandem</td>
<td>426,918</td>
<td>3%</td>
<td>2–100</td>
</tr>
<tr>
<td>SINE</td>
<td>Interspersed</td>
<td>1,797,575</td>
<td>15%</td>
<td>100–300</td>
</tr>
<tr>
<td>DNA transposon</td>
<td>Interspersed</td>
<td>463,776</td>
<td>3%</td>
<td>200–2,000</td>
</tr>
<tr>
<td>LTR retrotransposon</td>
<td>Interspersed</td>
<td>718,125</td>
<td>9%</td>
<td>200–5,000</td>
</tr>
<tr>
<td>LINE</td>
<td>Interspersed</td>
<td>1,506,845</td>
<td>21%</td>
<td>500–8,000</td>
</tr>
<tr>
<td>rDNA (16S, 18S, 5.8S and 28S)</td>
<td>Tandem</td>
<td>698</td>
<td>0.01%</td>
<td>2,000–43,000</td>
</tr>
<tr>
<td>Segmental duplications and other classes</td>
<td>Tandem or interspersed</td>
<td>2,270</td>
<td>0.20%</td>
<td>1,000–100,000</td>
</tr>
</tbody>
</table>

~50% of the human genome is repetitive
The Multiple Mapping Problem

Most repeats are longer than short reads meaning many reads covering repeated regions cannot be mapped uniquely.
The Multiple Mapping Problem

- A single read is likely map to more than once in the reference genome.
- The user may choose to ignore reads that map more than $n$ times.
- As you lower $n$, you get more data, but also more noise in the data.
Short read mapping

• Input:
  – A reference genome
  – Millions of 25-100 bp reads (single or paired)
  – User-specified parameters

• Output:
  – 1+ genomic coordinates for each read

• Question:
  – Do you expect most reads to map exactly? Why?
The Inexact Matching Problem

- A read may not **exactly match** any position in the reference genome.
- Sometimes, the read *almost* matches one or more positions.
- Such mismatches may represent either a SNP relative to the reference genome or a sequencer error.
The Inexact Matching Problem

- A read may not **exactly match** any position in the reference genome.
- Sometimes, the read *almost* matches one or more positions.
- Such mismatches may represent a SNP or an error.
- The user can specify the maximum number of mismatches, or a quality score threshold.
- As the number of allowed mismatches goes up, the number of mapped reads increases, but so does the number of incorrectly mapped reads.
The Inexact Matching Problem

This is the number of reads mapping with a range of mismatches – note that almost no reads map exactly! HTS is error prone…
Outline

• The short read mapping challenge

• Spaced seed alignment
Solving the Scale Problem

• 100 million reads (or more) vs. 3 gigabase genome

• For the moment, let’s ignore inexact matches and just look at how to find exact matches efficiently

• How could we do this?
Solving the Scale Problem

- 100 million reads (or more) vs. 3 gigabase genome

- Comparing strings is computationally expensive, so you can’t do every possible comparison directly!
Solving the Scale Problem

• 100 million reads (or more) vs. 3 gigabase genome

• Comparing strings is computationally expensive, so you can’t do every possible comparison directly!

• Solutions:
  – Hashing: an efficient way to look for exact-matching strings
  – Seeding: a way to cut down on the size of comparisons
The basic idea of hashing is to transform character strings (reads) into keys that can be easily compared.
Hashing Improves Search

• A hash function assigns a unique key to each unique data element (DNA sequence in our case)

\[
\text{hash(“ATGCTG”)} = \text{key1} \\
\text{hash(“TTTCTG”)} = \text{key2} \\
\ldots
\]

• Keys encode strings in a short, easily comparable format (e.g. a number)
Hashing Improves Search

- A **hash function** assigns a unique key to each unique data element (DNA sequence in our case)

- The **hash table** is an associative array that describes the relationship between the key and the sequence

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1</td>
<td>“GCTAGC”</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>KeyN</td>
<td>“TTTAGC”</td>
</tr>
</tbody>
</table>
Hashing Improves Search

• A hash function assigns a unique key to each unique data element (DNA sequence in our case)

• The hash table is an associative array that describes the relationship between the key and the sequence

• Hashing is useful because, rather than traversing the entire data set (the genome) looking for a matching string we can just use the hash of the string we are looking for (the read) to see if it exists
Hashing Improves Search

The basic idea of hashing is to transform character strings into keys that can be easily compared (in fact, the key can point directly to a physical memory address).
Hashing Improves Search

A hashed reference genome

A read CAGGTCGCA
Its key 0x4ad8
What next?
Hashing Improves Search

A hashed reference genome

A read
CAGGTCGCA

Its key
0x4ad8

We go looking for this key is in our hashed reference!
Seeded Alignment

Now we have a way to efficiently search for exact-matching strings
Seeded Alignment

Now we have a way to efficiently search for exact-matching strings.

We can’t deal with any sort of inexactness (i.e. mutations, gaps, etc) – for that we need to do traditional alignment.
Seeded Alignment

Now we have a way to efficiently search for exact-matching strings

We can’t deal with any sort of inexactness (i.e. mutations, gaps, etc) – for that we need to do traditional alignment

However, we know we can’t use our traditional alignment algorithms that allow for mismatches for all the reads…

Thoughts on how to proceed?
Seeded Alignment

We cannot just align every read to every position in the genome using a regular alignment strategy.
We can make our search much more efficient by using small **seed sequences** to identify potential hits in the genome. Then, we can do a full alignment of just the hits.
Seeded Alignment

The basic assumption is that even if the full-length read has many mismatches or gaps some substring (seed) of that read will not
Seeded Alignment

This cuts down massively on the number of full alignments we’ll need to do!
How Do We Find Seeds in The Reference?
Using Hashing to Map Seeds

For seed mapping we:

1) ?
Using Hashing to Map Seeds

For seed mapping we:

1) Create a hash table for the genome

<table>
<thead>
<tr>
<th>Key</th>
<th>Genome Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1</td>
<td>“GCTAGC”</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>KeyN</td>
<td>“TTTAGC”</td>
</tr>
</tbody>
</table>
Using Hashing to Map Seeds

For seed mapping we:

1) Create a hash table for the genome
2) ?

<table>
<thead>
<tr>
<th>Key</th>
<th>Genome Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1</td>
<td>“GCTAGC”</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>KeyN</td>
<td>“TTTAGC”</td>
</tr>
</tbody>
</table>
Using Hashing to Map Seeds

For seed mapping we:

1) Create a hash table for the genome
2) Hash each seed
3) ?

Read = GCTAGC TGATGCCCTAATC

hash(GCTAGC) = Key1

<table>
<thead>
<tr>
<th>Key</th>
<th>Genome Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1</td>
<td>“GCTAGC”</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>KeyN</td>
<td>“TTTAGC”</td>
</tr>
</tbody>
</table>
Using Hashing to Map Seeds

For seed mapping we:

1) Create a hash table for the genome
2) Hash each seed
3) See if the seed key exists in the genome hash table (much faster than comparing the sequences directly)

Read = GCTAGC

hash(GCTAGC) = Key1

<table>
<thead>
<tr>
<th>Key</th>
<th>Genome Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key1</td>
<td>“GCTAGC”</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>KeyN</td>
<td>“TTTAGC”</td>
</tr>
</tbody>
</table>
Using Hashing to Map Seeds

For seed mapping we:

1) Create a hash table for the genome
2) Hash each seed
3) See if the seed key exists in the genome hash table (much faster than comparing the sequences directly)
4) ?
Using Hashing to Map Seeds

For seed mapping we:

1) Create a hash table for the genome
2) Hash each seed
3) See if the seed key exists in the genome hash table (much faster than comparing the sequences directly)
4) We then extend each seed alignment and score them, picking the best one as the correct alignment of the read
Hash Alignment Example

• For example… “hash” every 28-mer in the human genome to a unique location in memory that is a function of the 28-mer sequence itself.
Hash Alignment Example

- For example… “hash” every 28-mer in the human genome to a unique location in memory that is a function of the 28-mer sequence itself.
- Go through every 28-mer subsequence of every read
- Look up whether that 28-mer exists in the genome
- If it does, try to extend the read & report alignment if meets threshold
Hash Alignment Example

• For example… “hash” every 28-mer in the human genome to a unique location in memory that is a function of the 28-mer sequence itself.

• Go through every 28-mer subsequence of every read

• Look up whether that 28-mer exists in the genome

• If it does, try to extend the read & report alignment if meets threshold

• For exact-match 28-mer to occur by chance unlikely.

• Most of things you seed are going to be productive alignments
Hashing and Seeding

We now have a general solution to finding exact match seeds.

And by searching for seeds and doing full alignments only for hits, we’ve mostly solved the mismatch problem.

We’re still missing something…
Hashing and Seeding

We now have a general solution to finding **exact match** seeds

And by searching for seeds and doing full alignments only for hits, we’ve mostly solved the mismatch problem

What if the seeds themselves contain a mismatch?
Refinement: Spaced seeds

• How can we keep the benefits of hashing and seeding while allowing for up to two mismatches in our seeds?
Refinement: Spaced seeds

- Split each big seed into four “spaced” seeds
Refinement: Spaced seeds

- Split each seed into four “spaced” seeds
- Any read with two (or fewer) mismatches will have **at least two seeds that match** the reference
Refinement: Spaced seeds

- Split each seed into four “spaced” seeds
- Any read with two (or fewer) mismatches will have **at least two seeds that match the reference**
- As usual, these hits can be further examined to determine how well they align

 Seeds whose key is in reference (matches)
How, Exactly, Do We Do This?

How can we use hashing to search for any two spaced seed matches for a read?
How, Exactly, Do We Do This?

Rather than 1 hash table, store multiple hash tables that correspond to all the possible ways to get two mismatches amongst our four spaced seeds.
How, Exactly, Do We Do This?

• Rather than 1 hash table, store 6 hash tables
• Imagine we have 8 bp seed
• Create 6 hash tables (as below) where only positions with 1 are used (and 0 positions ignored).
• Even with mismatches at any 2 positions, 1+ hash tables will have a match.
• Follow up hits by counting total # of mismatches across seed. Extend alignment to full read. Report best hit for each read.

1: 11110000  All possible pairs of positions
2: 00001111  (representing potential mismatch locations) are masked in at least 1 of the 6
3: 11000011
4: 00111100
5: 11001100
6: 00110011

All possible pairs of positions (representing potential mismatch locations) are masked in at least 1 of the 6 hash tables.
Hashing + spaced seed alignment = Maq

• Maq is a tool that incorporates hashing and spaced-seed alignment

• Has been a workhorse of short read alignment, but has been superseded