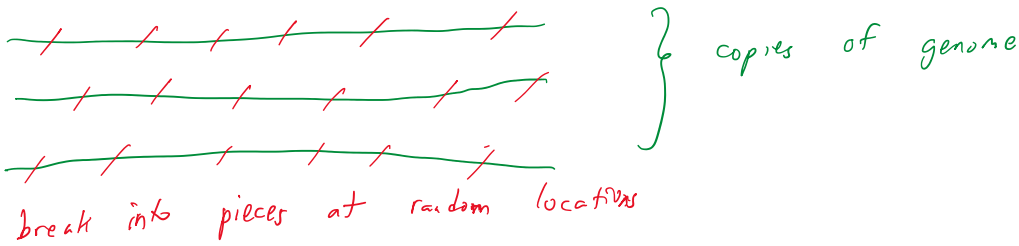
2-assembly-part1

Monday, January 19, 2026 5:56 PM

Recall: From last week, we know how to map reads to references. But how do/did we get the reference in the first place?

Define: Shotgun sequencing takes many copies of a genome, and breaks it randomly into smaller pieces.



(1kb = 1000 bp)

Some 3rd-gen technologies allow reading entire fragment ~ 1 – 100 kbp
But older and cheaper 2nd generation technologies like Illumina can only do ~ 100–200 bp at a time before errors pile up.

Define: Paired-end sequencing allows reading from both sides of fragment.

(actually reading forward on reverse complement)

Slowly being obsoleted by 3rd-gen technologies but at the moment, vast majority of sequencing done still 2nd-gen.

We will ignore this for now to make presentation cleaner, but be aware that this is a thing.

Intuition: Given two reads AACCGGTT & GGTTCCAA, assuming no errors, what possible genomes could they have come from?

AACCGGTTGGTTCCAA } no overlap

GGTTCCAAAACCGGTT ← shortest

AACCGGTTCCAA

GGTTCCAAACCGGTT } varying overlap

GGTTCCAACCGGTT

All five are possibilities, but the shortest one is "most parsimonious" by some metric

Define: Shortest Common Superstring

Given strings $s_1, \ldots, s_n$, find the shortest string T such that each s_i is a substring of T.

Aside: In CS a substring is contiguous, whereas a subsequence need not be.

e.g. PPL is a substring & subsequence of APPLE

but ALE is a subsequence but not a substring

Thm: Shortest Common Superstring is NP-hard. (with arbitrary size alphabet)

proof. Reduction from Vertex Cover [Virginia Vassilevska 2005]

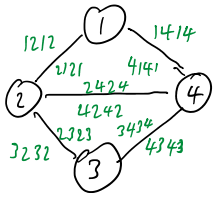
Recall: Given a graph $G=(V,E)$ with $|V|=n$ and $|E|=m$, the vertex cover problem asks if $\exists$ a set $S \subseteq V$ of size at most k such that every has an endpoint in S. Vertex Cover is known NP-complete.

Let the alphabet $\Sigma = V$ so each vertex a is associated with a unique letter a.

Let edge $(a,b) \in E$ be represented by two strings abab & baba.

This procedure converts a graph into a set of strings.

Ex



1212 2323
2121 3232
1414 3434
4141 4343
2424
4242

4242

Suppose G has a vertex S of size k.

Assign every edge to its covering vertex (or arbitrarily if both vertexes in S)

Then if a is the assigned vertex for (a, b), overlap the strings to get ababa.

" " b " " " " " " " " " babab.

Then $\forall c \in S$, we can overlap all assigned edge strings by 1 to get

$$c a_1 c a_1 c a_2 c a_2 c a_3 c a_3 c \cdots c a_{k_c} c a_{k_c} c \quad \text{of length } 4k_c + 1.$$

(k_c = # edges assigned to c)

By concatenating all such strings together, we get a superstring of length $4m + k$.

Exercise: Prove that no superstring has length $< 4m + 1$.

Lemma: If a superstring does not overlap abab & baba into either ababa or babab, we can shorten it by doing that overlap.

pf. Consider $T =$ ~~~~ abab ~~~~ baba ~~~~

Then $T' =$ ~~~~ ababa ~~~~ ba ~~~~ is shorter. □

Furthermore, if the b isn't part of an edge in ~~~~, we can remove it.

Same for a in ~~~~.

Thus, we can reduce any superstring into a concatenation of covering node strings above.

$\Rightarrow$ we can get a vertex cover from superstrings.

$\Rightarrow$ If we have a SCS of length $4m+k$, that immediately gives us a vertex cover of size k. ∎

Exercise: Extend this theorem to prove NP-hardness of SCS even for $\Sigma = \{A, C, G, T\}$

OK, so we can't exactly compute SCS. What if we could? Is SCS even the right way to think about the problem?

Intuition: Consider a set of reads

Counts	Read
1	TTGATTAC
1	TGATTACA
5	GATTACAG
5	ATTACAGA

SCS: TTGATTACAGATTACATT

```
1   TGATTACAGATTACATT
5    GATTACAG
5     ATTACAGA
5      TTACAGAT
5       TACAGATT
5        ACAGATTA
5         CAGATTAC
5          AGATTACA
1           GATTACAT
1            ATTACATT
```

SCS: TTGATTACAGATTACATT

But is this the "right" assembly?

SCS is parsimonious in that all of these substrings can be produced from a single string.
But it might not explain other features of the data like read counts

Intuition: Consider a set of reads

```
AAAAAAAC
CAAAAAAG
GAAAAAAT
TAAAAAAT
```

} SCS chains all of them together

But should a single character overlap be enough for us to conclude these regions were originally together on the genome?

Could this happen purely by chance?

How long of an overlap do we need?

So, even if we could compute SCS, it might not be the right choice.

Question: Is it possible to recover a full genome using shotgun sequencing of a single copy?

No! There's no way to know which pieces come after one another. Not like a puzzle, where pieces fit together. Need overlap instead.

CCTTGATTACAACCTTGG

CCTTGATTA
ACAACCTTGG

} Can we fit these two pieces together? No. not enough overlap.

CCTTGATTACAACC

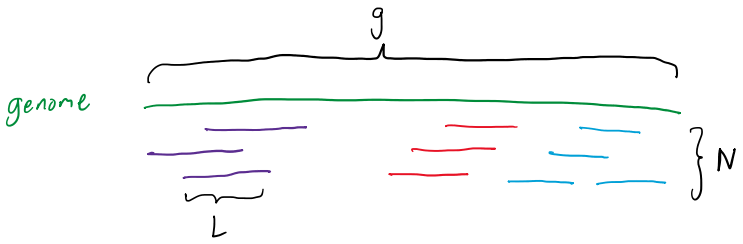
} With more overlap, maybe

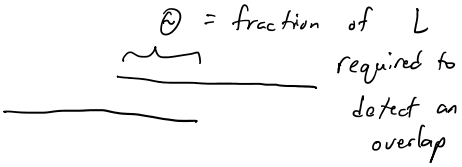
CCTTGATTACAACC
GATTACAACCTTGG
} With more overlap, maybe good enough to fit together pieces

We'll come back to algorithms for assembly. Let's first compute if it is even theoretically possible.

Define: Lander-Waterman Statistics

How many reads do we need to cover the genome?





An island is a contiguous group of reads that are connected by overlaps of $\geq \theta L$. (colored above)

Want: Expression for expected number of islands given N, g, L, θ.

Notice: This is a balls & bins problem, where we divide up N reads into g start positions.

$1/g$ probability first read starts at a given position (assuming uniform random sampling)

In expectation, N/g reads start at any given position. (linearity of expectation)

Bernoulli approximation: Instead of dividing up N reads to g locations, we assume each location has a $\lambda = \frac{N}{g}$ probability of a read spawning.

Passing to binomial distribution:

Let S_n be a random variable denoting the # of reads starting in an interval of length n.

From binomial distribution $\Pr(S_n = k) = \binom{n}{k} \lambda^k (1-\lambda)^{n-k}$

- $\binom{n}{k}$: number of configurations of k successes in x trials
- λ^k: prob from k successes
- $(1-\lambda)^{n-k}$: prob from $x-k$ failures

Recall: Poisson limit theorem

Let p_n be a sequence of real numbers in $[0,1]$ s.t. $np_n \to \beta < \infty$.

Then $\lim \binom{n}{k} p_n^k (1-p_n)^{n-k} = e^{-\beta} \frac{\beta^k}{}$

Let p_n be a sequence ... p_n

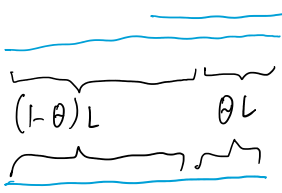
Then $\lim_{n\to\infty} \binom{n}{k} p_n^k (1-p_n)^{n-k} = e^{-\beta} \frac{\beta^k}{k!}$

Here, $\beta = \lambda n$, the expected number of reads in interval of length n.

Poisson approximation: $Pr(S_n = k) \approx e^{-\lambda n} \cdot \frac{(\lambda n)^k}{k!}$

Then the expected # islands = $N \cdot Pr$(read ends an island)

A read ends an island if no later read starts before the final θL bp of the read.



← would continue the island

← does not continue the the island

$= N \cdot Pr(S_{(1-\theta)L} = 0)$ ← no reads start in first $(1-\theta)L$ of read

$\approx N e^{-\lambda(1-\theta)L} \cdot \frac{(\lambda(1-\theta)L)^0}{0!}$

$= N e^{-\lambda(1-\theta)L}$

$= N e^{-(1-\theta)\cdot\frac{LN}{g}}$ → Let $c = \frac{LN}{g}$

Define Coverage $c = \frac{LN}{g}$

(sometimes called depth of coverage)

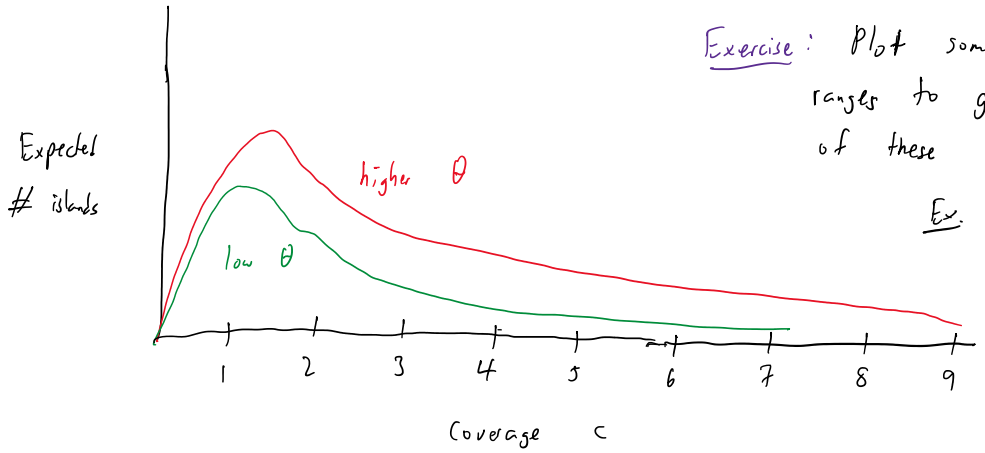
Rewrite in terms of c & θ: ⇒ $N = \frac{cg}{L}$

$= N e^{-(1-\theta)c}$

$= \frac{cg}{L} e^{-(1-\theta)c}$

Increasing c decreases # islands (once past coverage of ~1)

Decreasing θ decreases # islands (since less overlap required)



Exercise: Plot some example parameter ranges to get an intuitive understanding of these curves.

Ex. $L = 1000$, $g = 10^6$, $\theta = 0.1 - 0.3$

To get a full genome, we want ~~# islands~~ to be 1.

Exercise: Consider the approximations & assumptions we made. Where could you improve the analysis?

Overlap alignment (lengths m & n)

Brute force (exact)

Given two strings S & T, check if suffix of length j of S matches prefix of length j of T.

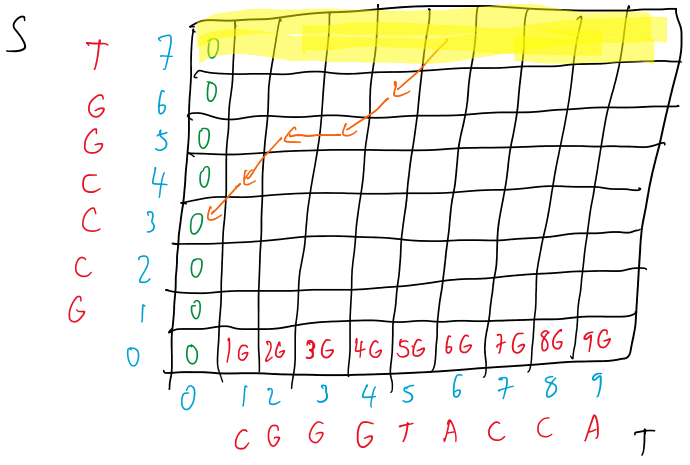
Repeat for increasing values of j.

Runtime: $O(mn)$

Need to recheck each position over & over.

But what if there are errors? Then need alignment matrix again.

Dynamic Programming (overlap alignment)



Initialize 1st col to 0
(basically saying gaps on T's right are free)

Ans is maximum score in top row
(start traceback there & stop when hit left side)

S: G C C C G − G T
C G G G T A C C A : T

Runtime: $O(mn)$ still but now can deal with errors.

For the exact match case, determine a linear time algorithm $O(m+n)$

Runtime: $O(mn)$ still but now can ...

Exercise: For the exact match case, determine a linear time algorithm $O(m+n)$ to find the longest exact overlap.

Hint: Think about suffix trees.

Exercise: Can you use something like seeding to speed up the DP?