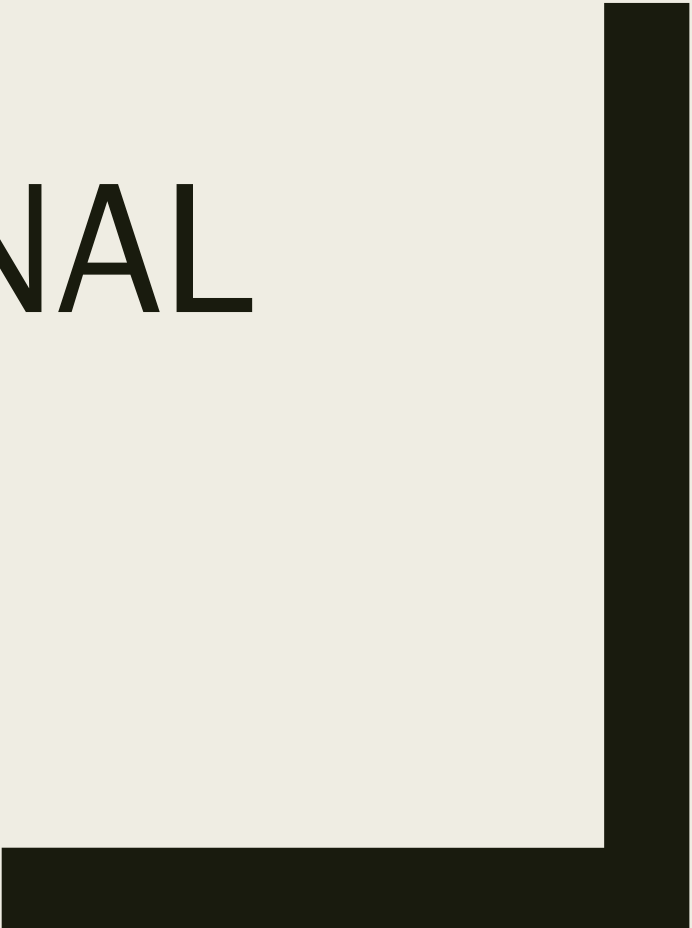


CS 364
COMPUTATIONAL
BIOLOGY

Sara Mathieson
Haverford College



Lab 1 runtime notes

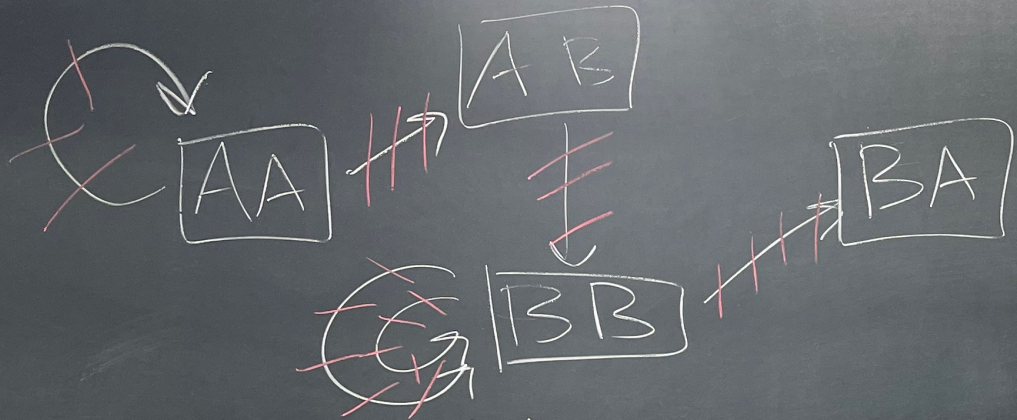
- Naïve average case: $O(nm)$
- Boyer-Moore average case: $O(n)$
- Both in worst-case: $O(nm)$

Outline

- Finish de Bruijn graphs and their practical applications
- Velvet assembler (uses DBGs)
- Begin: pairwise sequence alignment

Reading: Durbin 2.1-2.3
(on hold in the library)

de Bruijn graphs in practice



Orig = AAAABBBBA

R=3

function
Stack

~~f+(AA)~~
~~f+(BB)~~
~~f+(BB)~~
~~f+(BA)~~
~~f+(BB)~~
~~f+(AB)~~
~~f+(AA)~~

tour
stack

AA
 AA
 AB
 BB
 BB
 BB
 BB
 BA

add
one
base
for
each
node

find_tour(u):

for edge $e(u, v)$
 remove e
 find_tour(v)
 push u onto stack

AAAABBBBA



De Bruijn graph

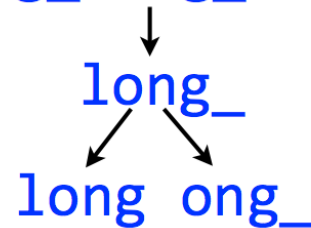
A procedure for making a De Bruijn graph for a genome

Assume *perfect sequencing* where each length- k substring is sequenced exactly once with no errors

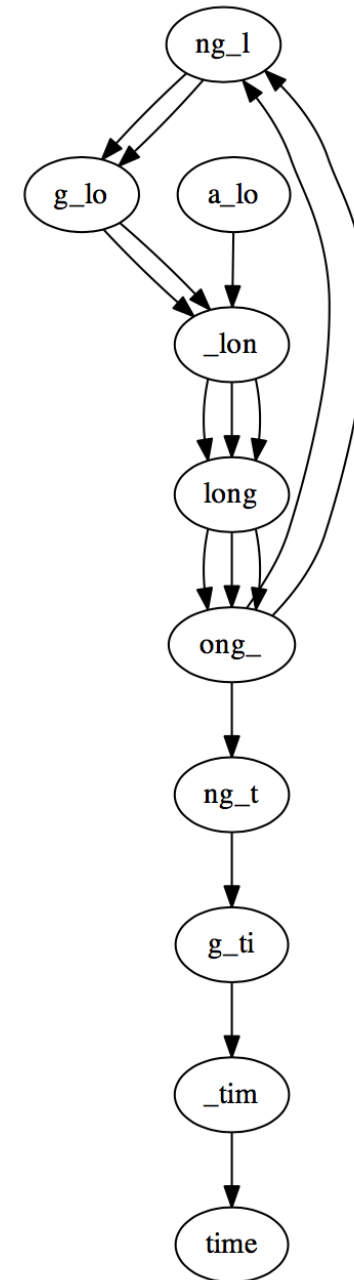
Pick a substring length k : 5

Start with an input string: `a_long_long_long_time`

Take each k mer and split into left and right $k-1$ mers



Add $k-1$ mers as nodes to De Bruijn graph (if not already there), add edge from left $k-1$ mer to right $k-1$ mer



De Bruijn graph

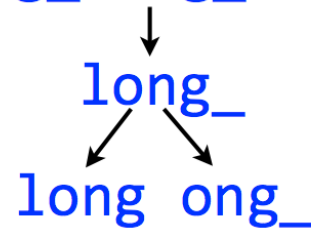
A procedure for making a De Bruijn graph for a genome

Assume *perfect sequencing* where each length- k substring is sequenced exactly once with no errors

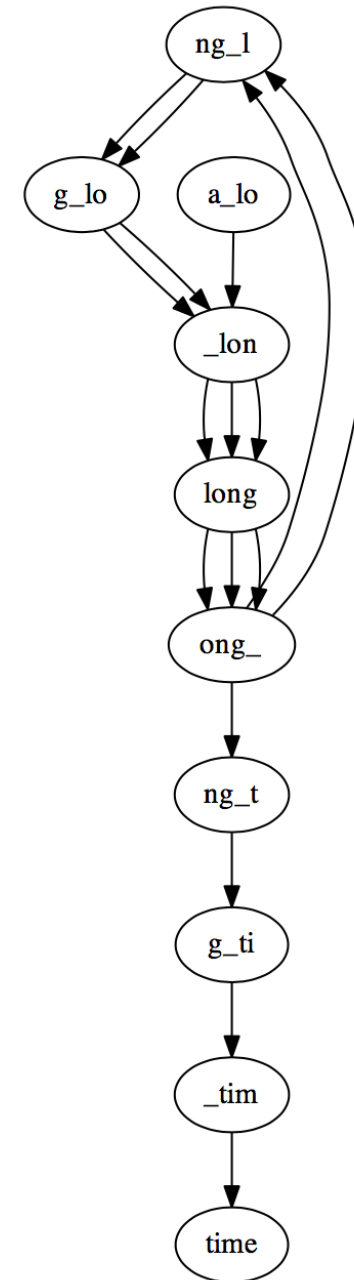
Pick a substring length k : 5

Start with an input string: `a_long_long_long_time`

Take each k mer and split into left and right $k-1$ mers



Add $k-1$ mers as nodes to De Bruijn graph (if not already there), add edge from left $k-1$ mer to right $k-1$ mer



Will always give an Eulerian graph. Why?

Building k-mer graph with reads

$R = 6 \times 10^9$ reads
 $m = 100$ bp
 $n = 3 \times 10^9$ bp (humans)

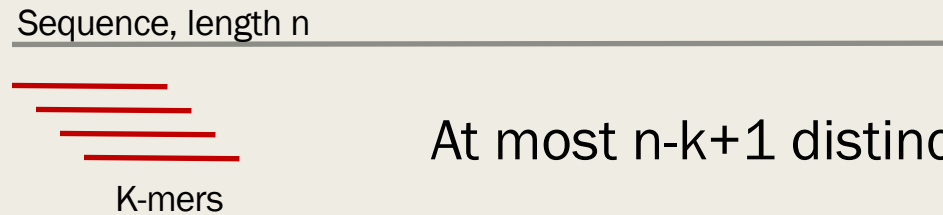
- * Pick k (for $m = 100$ bp, $k = 21-41$ is common)
- * For each read:
 - For each k-mer in read:
 - add L&R (k-1)-mers to graph as nodes (if not already there)
 - add edge from L \rightarrow R

From last time: # k-mers is $O(n)$ \Rightarrow # nodes is $O(n)$, and # edges is $O(n)$

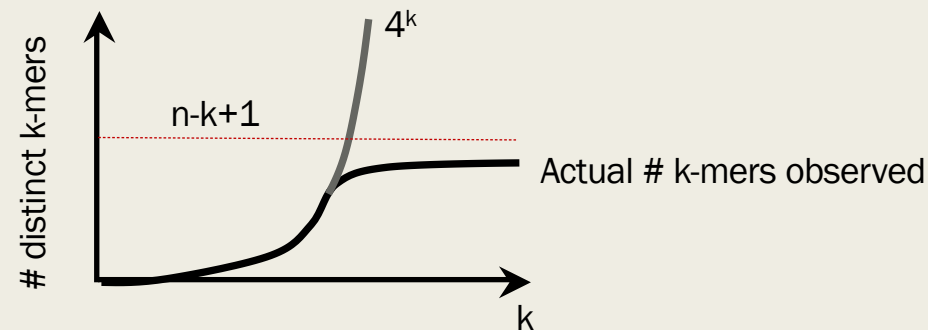
But: how do we know if (k-1)-mer is already in our graph or not? Do we have to compare with all nodes?

How many k-mers are there?

There are 4^k possible k-mers



So $\min(4^k, n-k+1)$ k-mers in a sequence



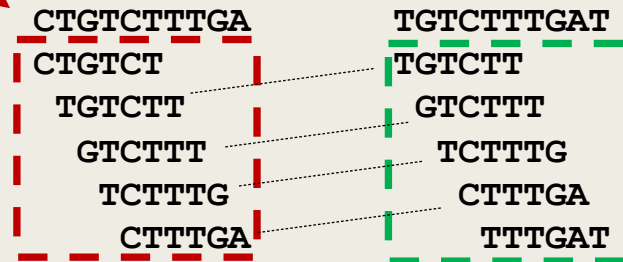
(Assuming $n \gg k$)

Multiple reads

Sequence: ...AGTATCTGTCTTTGATTCCCTAACTCATCCTATTATTTATCGCACCTACGTTCAATATT...

Reads: AGTATCTGTC
TATCTGTCTT
CTGTCTTTGA
TGTCTTTGAT
TCTTTGATTG

Split each read
into k-mers:



Build de Bruijn graph from these k-mers. Key: The number of nodes in the graph does is bounded by the number of k-mers in the sequence [$\max(4^k, n-k+1)$], so it does not grow indefinitely with the number of reads like the overlap graph.

Implementation considerations

- 1) In practice, k-mers must be hashed so we can easily compare them
 - * for us we will use sets which will hash implicitly
- 2) When graphs become larger, recursive solutions are no longer practical
 - * for us the examples are small enough we can use recursion

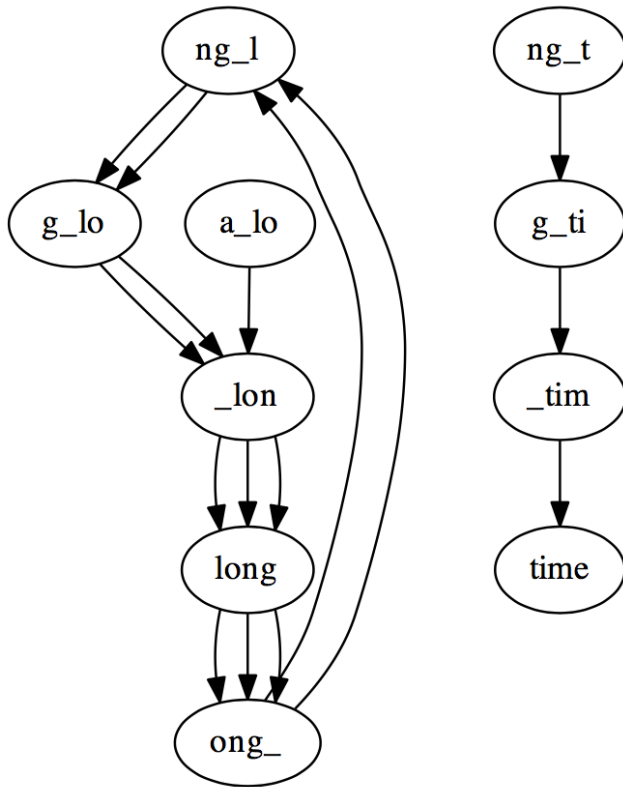
What “messes” up our DBG?

- 1) Repeats of length $(k-1)$ or longer
- 2) Gaps in coverage
- 3) Differences in coverage
- 4) Sequencing errors

Issues with DBGs

Gaps in coverage can lead to *disconnected* graph

Graph for `a_long_long_time`, $k = 5$ but *omitting* `ong_t`:



Connected components are individually Eulerian, overall graph is not

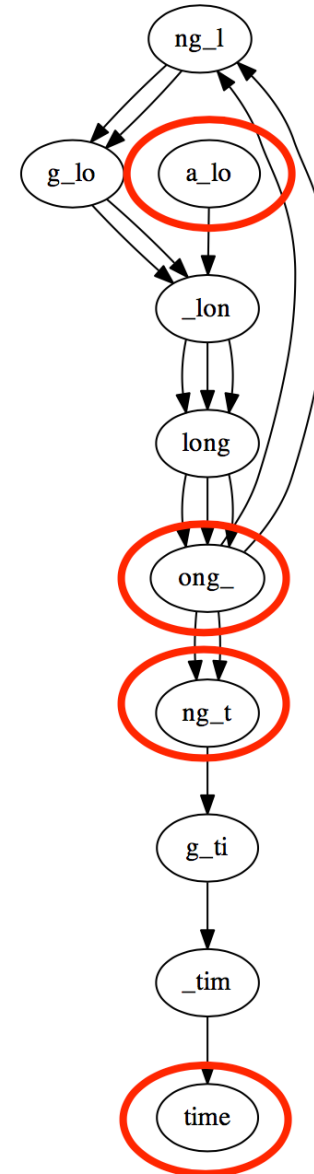
Issues with DBGs

De Bruijn graph

Differences in coverage also lead to non-Eulerian graph

Graph for `a_long_long_long_time`,
 $k = 5$ but with *extra copy* of `ong_t`:

Graph has 4 **semi-balanced** nodes,
isn't Eulerian



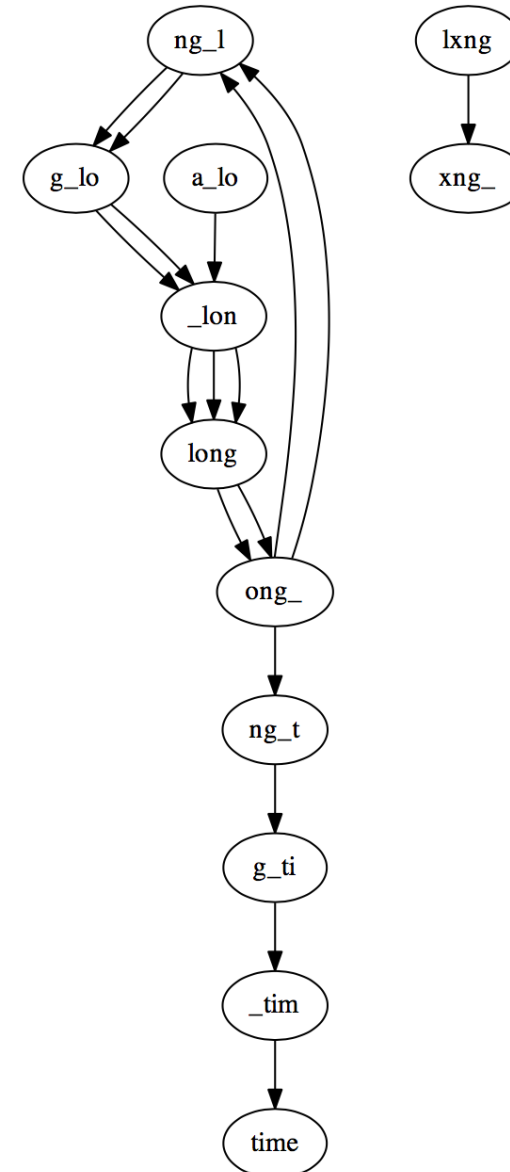
Issues with DBGs

De Bruijn graph

Errors and differences between chromosomes also lead to non-Eulerian graphs

Graph for `a_long_long_long_time`, $k = 5$ but with error that turns a copy of `long_` into `lxng_`

Graph is not connected; largest component is not Eulerian



One workaround for coverage issues:

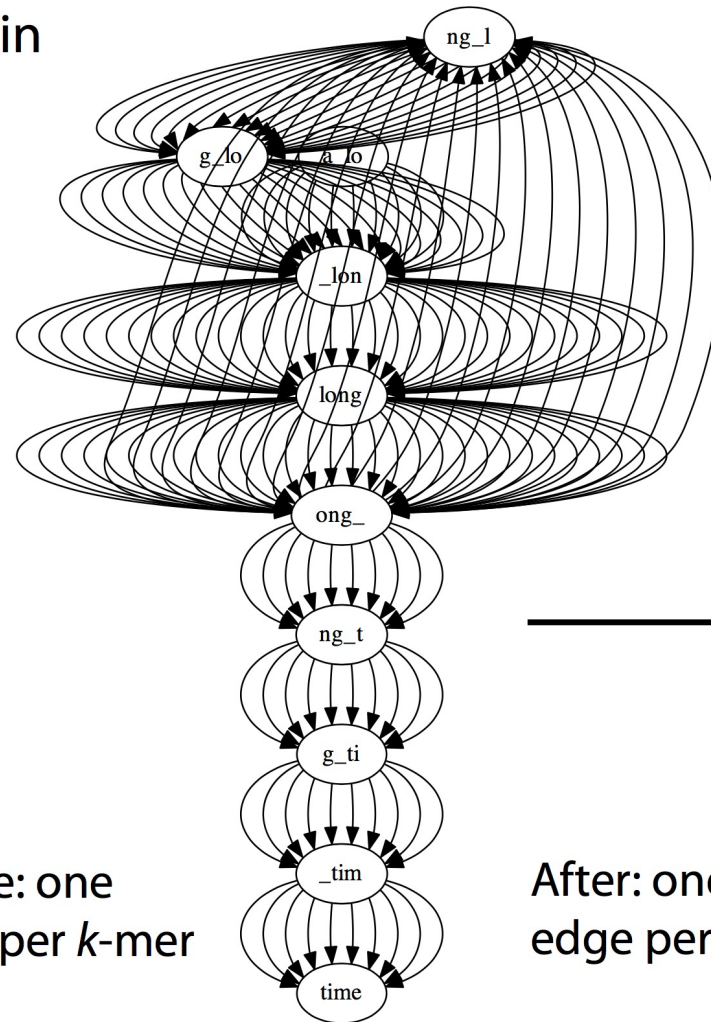
In typical assembly projects, average coverage is $\sim 30 - 50$

Same edge might appear in dozens of copies; let's use edge *weights* instead

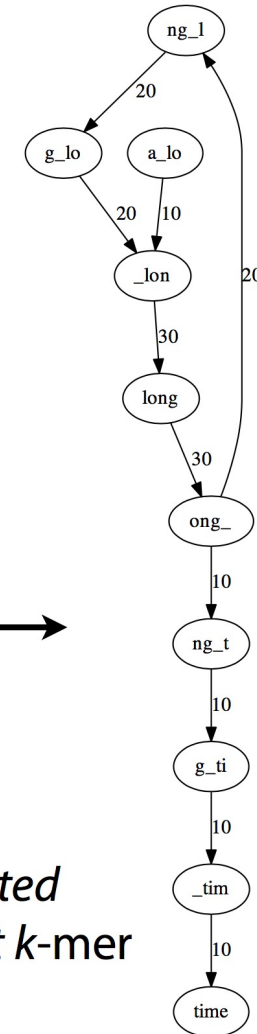
Weight = # times *k*-mer occurs

Using weights, there's one *weighted* edge for each *distinct k*-mer

Before: one edge per *k*-mer



After: one *weighted* edge per *distinct k*-mer



What did we give up by going from OLC to DBG?

Reads are immediately split into shorter k -mers; can't resolve repeats as well as overlap graph

Only a very specific type of "overlap" is considered, which makes dealing with errors more complicated

Read coherence is lost. Some paths through De Bruijn graph are inconsistent with respect to input reads.

Issues with DBGs

Casting assembly as Eulerian walk is appealing, but not practical

Uneven coverage, sequencing errors, etc make graph non-Eulerian

Even if graph were Eulerian, repeats yield many possible walks

Kingsford, Carl, Michael C. Schatz, and Mihai Pop. "Assembly complexity of prokaryotic genomes using short reads." *BMC bioinformatics* 11.1 (2010): 21.

De Bruijn Superwalk Problem (DBSP) is an improved formulation where we seek a walk over the De Bruijn graph, where walk contains each read as a *subwalk*

Proven NP-hard!

Medvedev, Paul, et al. "Computability of models for sequence assembly." *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2007. 289-301.

But we still have advantages...

Building the de Bruijn graph:

- $O(Rm)$ since we go through each read and k -mers along the length of the read
- $O(n)$ space to store since both # edges and # nodes are $O(n)$

Finding paths through the graph:

- assuming Eulerian, $O(n)$ to traverse since # edges is $O(n)$

Velvet Assembler

Velvet Assembler (Zerbino & Birney, 2008)

- The first truly practical de Bruijn graph assembler
- Combines several algorithms to simplify the raw k-mer graph

Resource

Velvet: Algorithms for de novo short read assembly using de Bruijn graphs

~11,000 citations

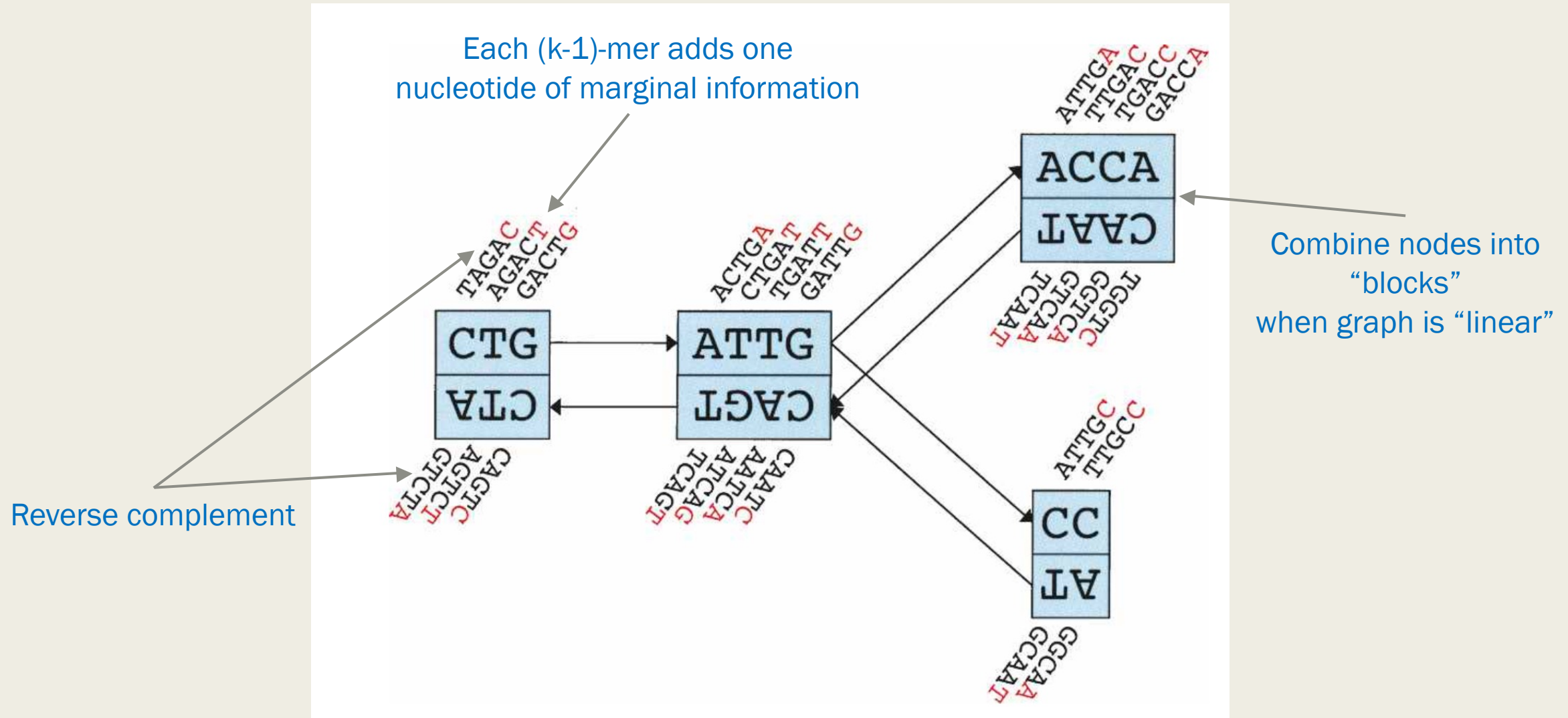
Daniel R. Zerbino and Ewan Birney¹

EMBL-European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, United Kingdom

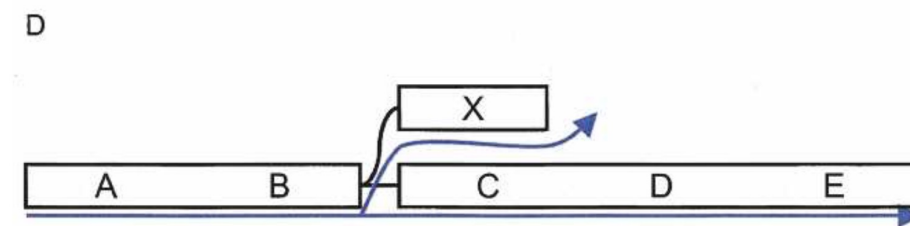
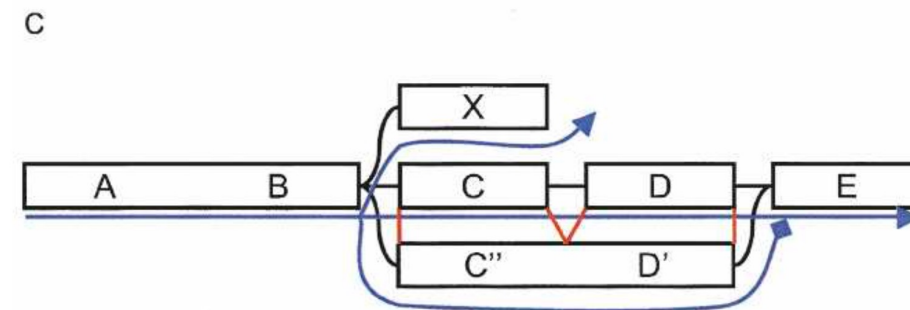
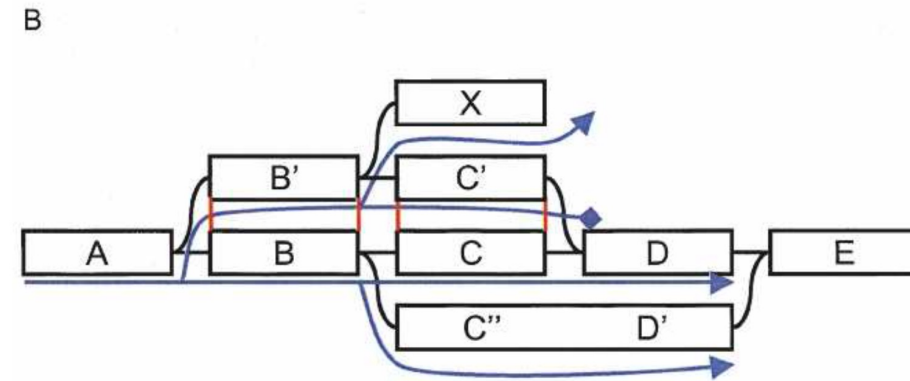
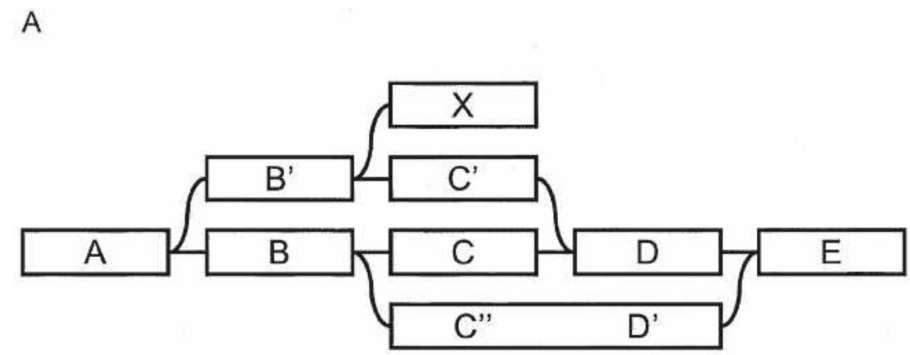
We have developed a new set of algorithms, collectively called “Velvet,” to manipulate de Bruijn graphs for genomic sequence assembly. A de Bruijn graph is a compact representation based on short words (k -mers) that is ideal for high coverage, very short read (25–50 bp) data sets. Applying Velvet to very short reads and paired-ends information only, one can produce contigs of significant length, up to 50-kb N50 length in simulations of prokaryotic data and 3-kb N50 on simulated mammalian BACs. When applied to real Solexa data sets without read pairs, Velvet generated contigs of ~8 kb in a prokaryote and 2 kb in a mammalian BAC, in close agreement with our simulated results without read-pair information. Velvet represents a new approach to assembly that can leverage very short reads in combination with read pairs to produce useful assemblies.

[Supplemental material is available online at www.genome.org. The code for Velvet is freely available, under the GNU Public License, at <http://www.ebi.ac.uk/~zerbino/velvet>.]

Velvet paper: Figure 1

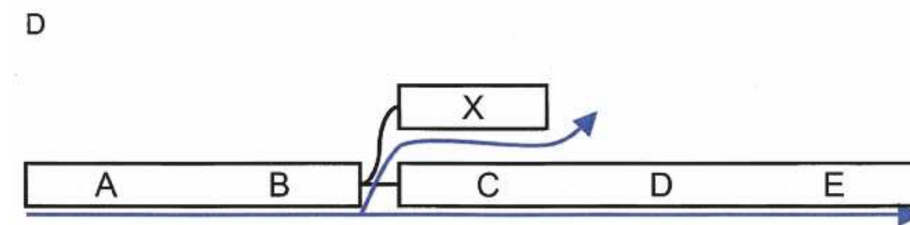
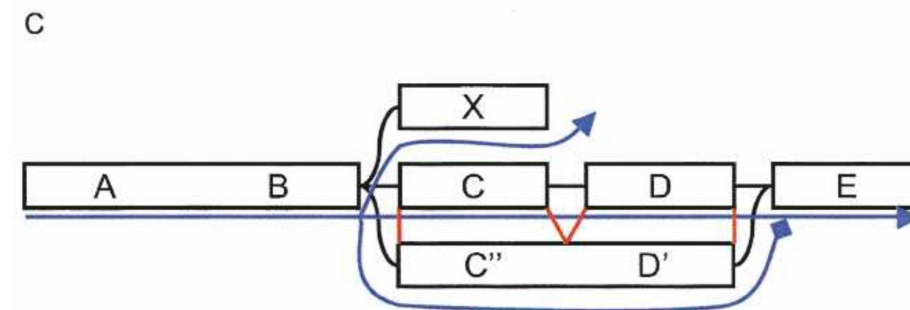
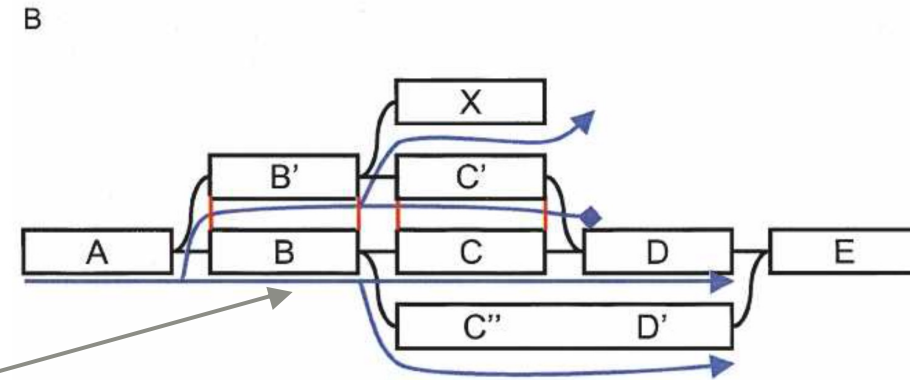
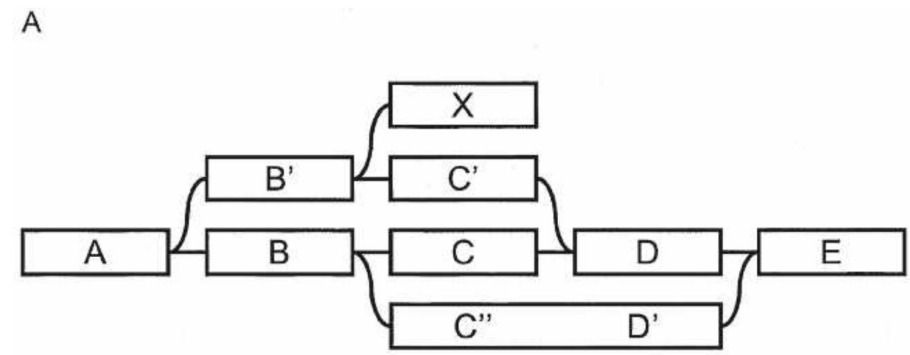


Velvet paper: Figure 2 (Tour Bus algorithm)



Velvet paper: Figure 2 (Tour Bus algorithm)

When we hit D for the second time, compare the sequences of the two ways we got there: BC and B'C'. If they are judged similar, error correct and merge



Error correction

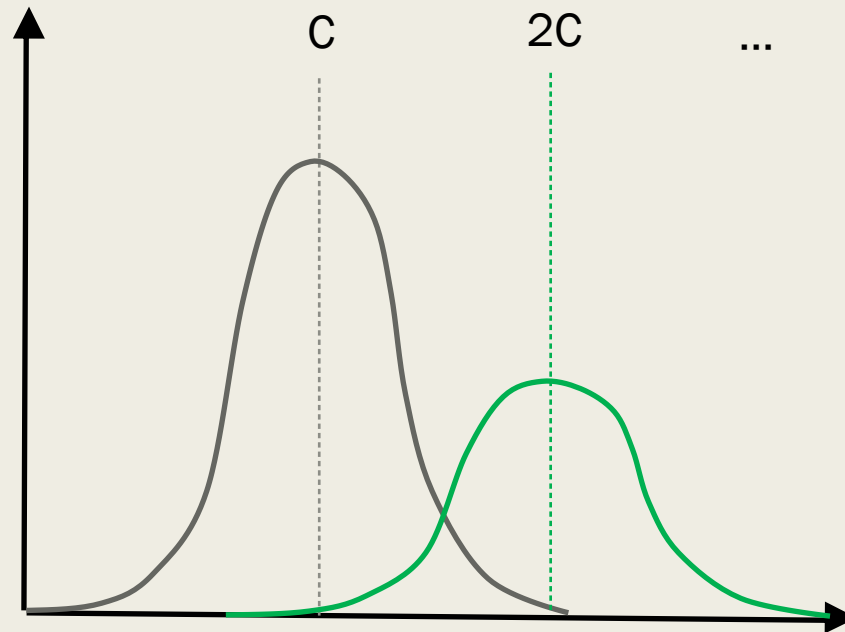
Genome: ...AGTATCTGTCCTTTGATTC...

Reads:
AGTATCTGTC
TATCTGTCTT
CTGTCCTTTGA
TGTCTTTGAT
TCTTTGATTC

Split each read
into k-mers:

CTGTCCTTTGA	TGTCTTTGAT
CTGTCCT	TGTCTT
TGTCTT	GTCCTT
GTCCTT	TCTTTG
TCTTTG	CTTTGA
CTTTGA	TTTGAT

Density



times we see each k-mer

k-mers that occur once in genome

k-mers that occur twice in genome

...

Error correction

Genome: ...AGTATCTGTCCTTTGATTC...

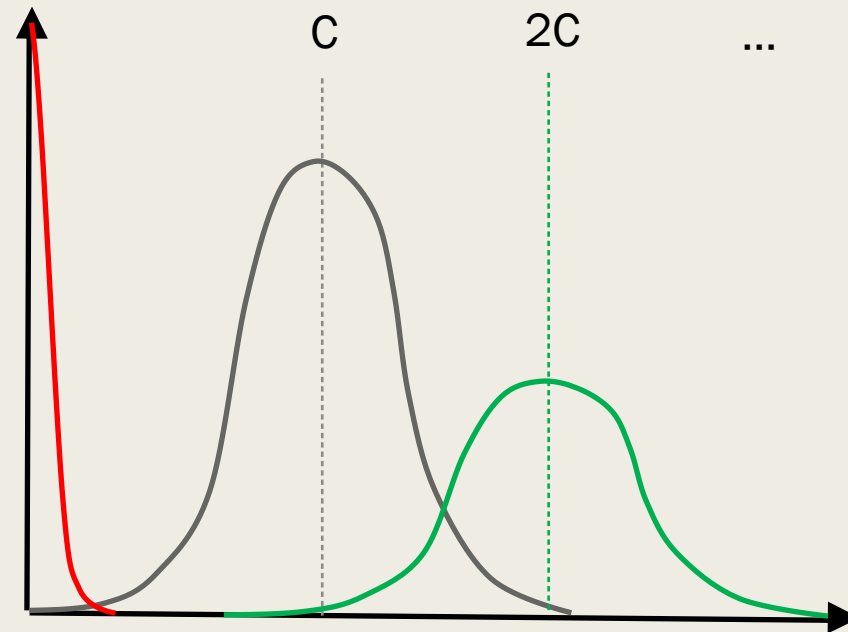
Reads:
 AGTATCTGTC
 TATCTGTCTT
 CTGT**A**TTTGA
 TGTCTTTGAT
 TCTTTGATTC

Split each read
 into k-mers:



This is bad because the de Bruijn Graph will include all these erroneous k-mers that are not in the reference so it will grow $\gg O(n)$

Density



times we see each k-mer

k-mers that occur once in genome

k-mers that occur twice in genome

...

k-mers with errors

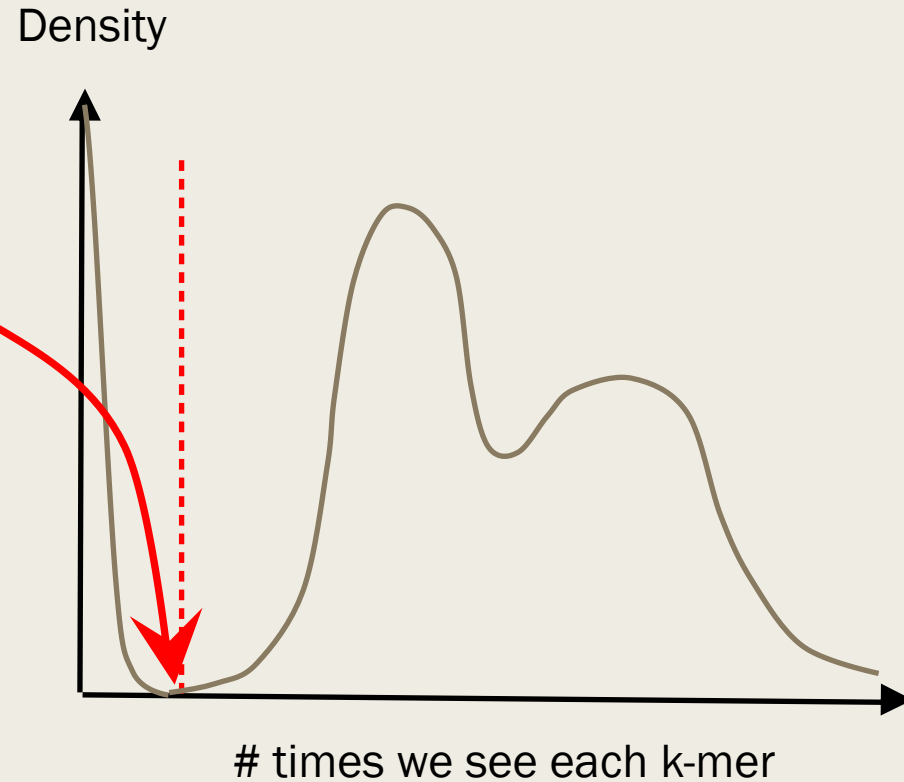
Error correction

Fortunately there is a simple solution

Chose some cutoff and filter out k-mers that occur less than the cutoff

In fact, we can do even better:

If we chose k large enough that $n \ll 4^k$ and assume that errors are rare, then we can actually correct the errors by replacing each k-mer less than the cutoff by the closest (in terms of edit distance) k-mer that is above the cutoff.



Error correction

Fortunately there is a simple solution

Chose some cutoff and filter out k-mers that occur less than the cutoff

In fact, we can do even better:

If we chose k large enough that $n \ll 4^k$ and assume that errors are rare, then we can actually correct the errors by replacing each k-mer less than the cutoff by the closest (in terms of edit distance) k-mer that is above the cutoff.

k-mer seen once

TGTCCTT



k-mers seen multiple times

AAAAAA

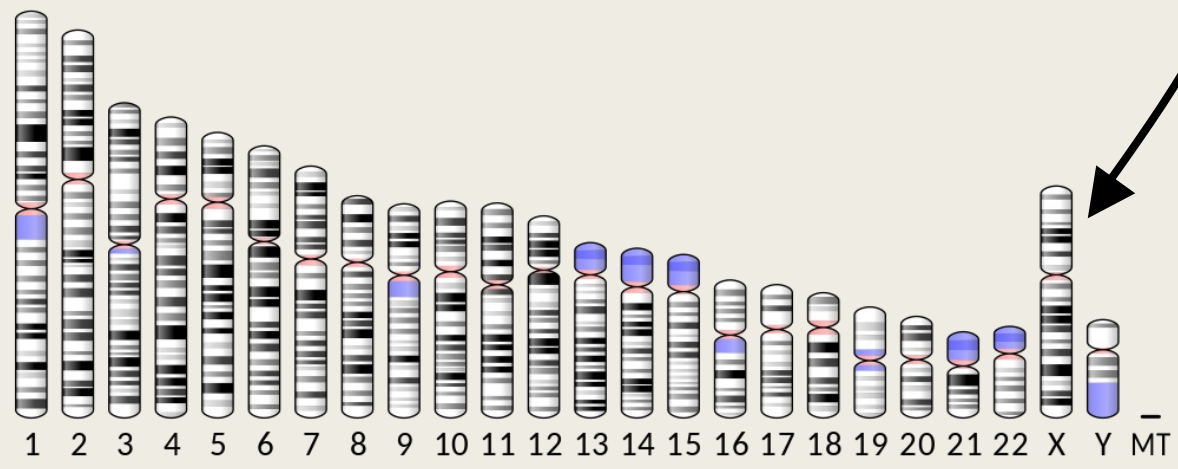
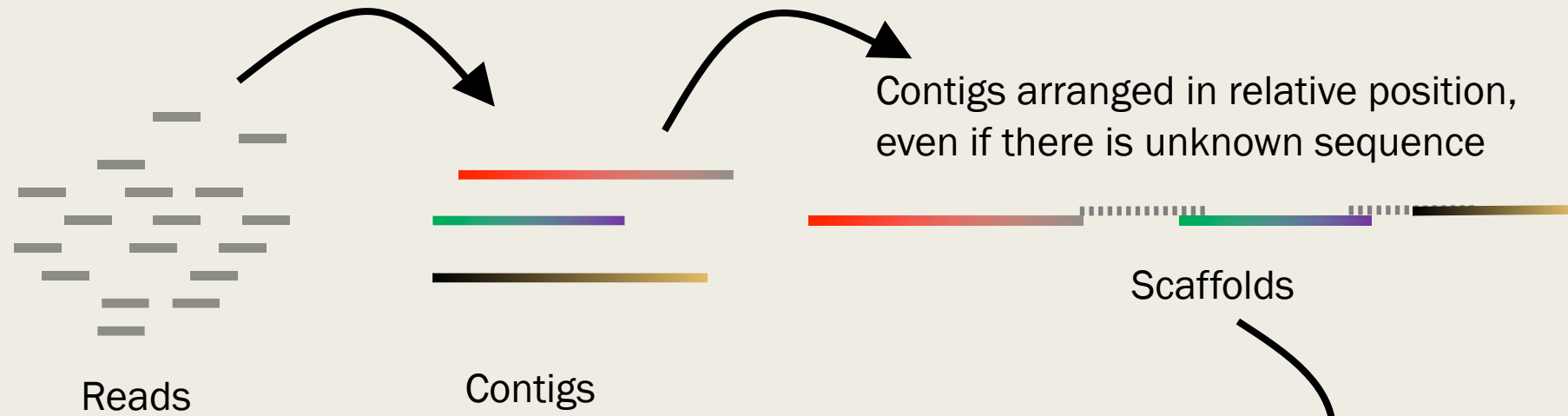
ACGAAT

TGTATT

CAATGT

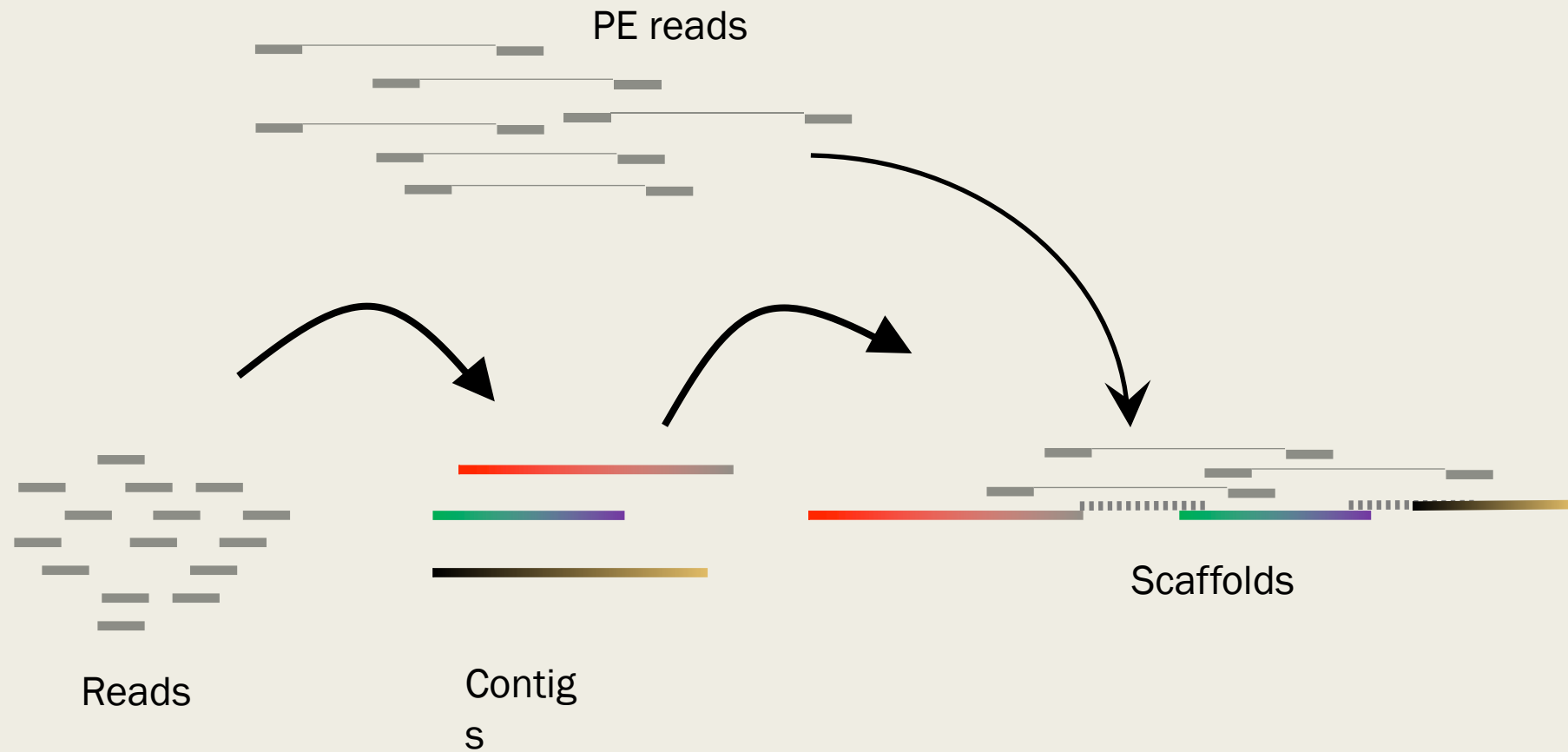
.....

Scaffolding



Scaffolding

- Use paired-end reads to arrange



Evaluating Assemblies

Assembly evaluation

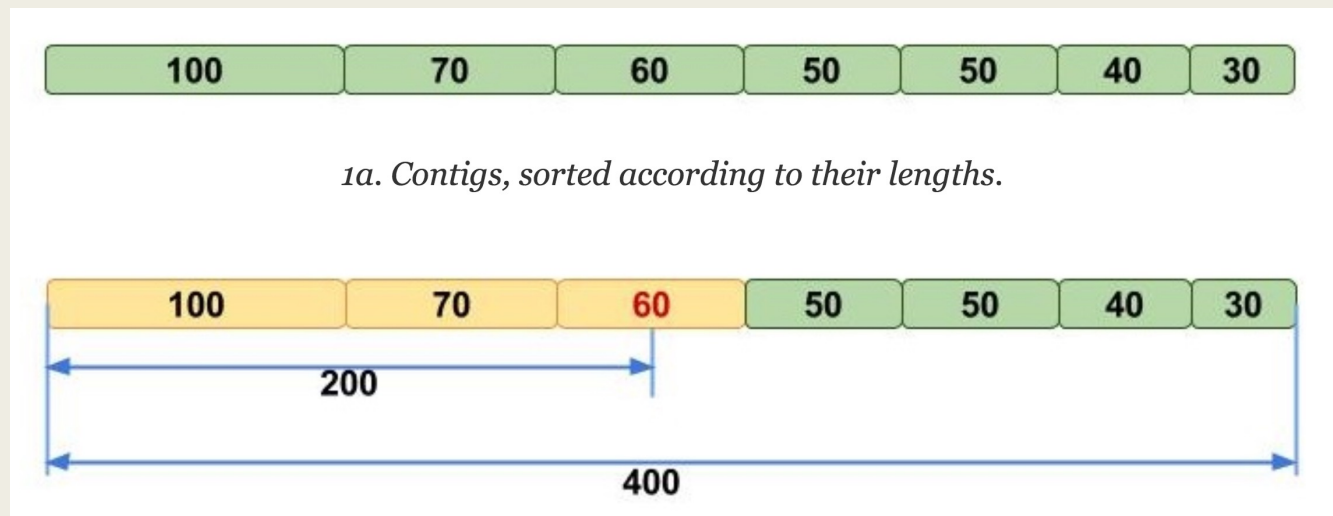
- **N50:** for a set of contigs, N50 is the greatest length such that at least half the bases of the assembly are in a contig with length N50 or longer



1a. Contigs, sorted according to their lengths.

Assembly evaluation

- **N50:** for a set of contigs, N50 is the greatest length such that at least half the bases of the assembly are in a contig with length N50 or longer



=> N50 = 60

① total bases = 400 half = 200

$\boxed{NSO = 60}$

② $NSO = 10,000$

③ $\{ \underline{100}, \underline{100}, \boxed{100}, 100, 100, 100 \}$
 $NSO = 100$

④ $NSO = 1000$

⑤ $NSO = 250$

⑥ - e

⑦

- (6) - easy to compute
- fewer larger contigs \Rightarrow higher N50
 - doesn't require ground truth

- (7) - incentivize long meaningless contigs

Why is N50 a bad evaluation metric?

- We could just loop through cycles in our a graph over and over, generating large (incorrect) contigs
- We need a better way to evaluate the quality of assemblies
- Take away: simulated data is every valuable. Take an existing genome, simulate random reads, then try to reconstruct.

Sequence Alignment

Next Topic: sequence alignment

- Goal: given two sequences, what is the best match or “alignment” between them?
- Global alignment: align the entire sequences start to finish
- Local alignment: find portions of the two sequences with high similarity
- Homologous: sequences that are similar due to descent from a common ancestor
- Usually we are aligning homologous sequences (not sequences from completely different regions of the genome)

Example alignments: human, chimp, macaque + other species



Today, we are only considering two sequences (“pairwise alignment”)

```
Human  FMPFDIKAEMTLKTNFFATRNM CNELL PIMKPHGRVVNISSLQCLRAFEN-CSEDLQERFHSETLTEGDLVDLMKKF-VEDTKNEVHEREGWPNSPYGVS
Chimp  FMPFDIKAEMTLKTNFFATRNM CNELL PIMKPHGRVVNISSLQCLRAFEN-CSEDLQERFHSETLTEGDLVDLMKKF-VEDTKNEVHEREGWPNSPYGVS
```

If time next week: many sequences (“multiple alignment”)

```
Human  FMPFDIKAEMTLKTNFFATRNM CNELL PIMKPHGRVVNISSLQCLRAFEN-CSEDLQERFHSETLTEGDLVDLMKKF-VEDTKNEVHEREGWPNSPYGVS
Chimp  FMPFDIKAEMTLKTNFFATRNM CNELL PIMKPHGRVVNISSLQCLRAFEN-CSEDLQERFHSETLTEGDLVDLMKKF-VEDTKNEVHEREGWPNSPYGVS
Macaque FMPFDIKAEMTLKTNFFATRNM CNELL PIMKPHGRVVNISSLQCLRAFEN-CSEDLQEKFRSDTLTEGDLVDLMKKF-VEDIKNEVHEREGWPNSPYGVS
Mouse  PTPFDIQAEVTLKTNFFATRNVCTELL PIMKPHGRVVNISSLQGLKALEN-CREDLQEKFRCDTLTEVDLVDLMKKF-VEDTKNEVHEREGWPDSAYGVS
Rat    PTPFDVQAEVTLKTNFFATRNVCTELL PIMKPHGRVVNVSSLQGLKALEN-CSEDLQERFRCDTLTEGDLVDLMKKF-VEDTKNEVHEREGWPDSAYGVS
Cow    PTPFDIQAEVTLKTNFFATRNVCTELL PIVKPHGRVVNVSSSQGSALEN-CSEDLQEKFRCDTLTEEDLVDLMKKF-VEDTKNEVHEREGWPNSAYGVS
Cat    PTPFDIRAEITLKTNFFATRNVCIELL PIIKPHGRVVNISSLQGLKALEN-CSPDLQKKFRCDTLTEGDLVDLMKKF-VEDANNEVHEREGWPNSAYGVS
Chick  RTPFAVQAEVTLKTNFFGTRNICTELL PLIKPYGRVVNVSSMVSISALGG-CSEQLQKFRSDTITEDELVELMTKF-VEDTKKSVHEKEGWPNTAYGVS
Zebrafish TTPFGTQADVTLKTNFFATRDMCNVFLPI IKPGGRLVNVSSGMGSMALGR-CSEPLQARFRSDITEEELNGLMERF-VREAQEGVHSERGWPSAYGIS
X_tropicalis TTPFGTQAEVTLKTNFFATRDCHELL PLIKPRGRVVNVSSMASYMALGRCCSPELQKVRSDTITEEELVTLMEKF-VEDAKKGAHQKEGWPNTAYGVS
```


Why sequence alignment?

- Understand evolutionary relationships between different species
- In particular: understanding fast-evolving bacterial and viral strains is important for health
- Understand protein function
- Understand diversity at the species level (important for diseases with a genetic component)

Example

■ **ACGGCTAGTTACG**

■ **TCGTAGTATACCGA**

- How should we “line them up” to get the best overlap?

$X = \boxed{A} \boxed{C} \boxed{G} G C T A G T - T A - C G -$
 $Y = \boxed{T} \boxed{C} \boxed{G} - - T A G T A T A C C G A$

mismatch (pointing to A/T)
 matches (under AC and CG)
 gaps (under - in X)
 deletion (under - in Y)
 gap (under - in Y)
 insertion (under A in Y)

$\approx O(2^n)$

$\binom{2n}{n}$

alignment score

match: +1

mismatch: -1

(g) gap: -2

$\longrightarrow 10 \cdot 1$

$\longrightarrow 1 \cdot (-1)$

$\longrightarrow 5 \cdot (-2)$

Score $> 0 \Rightarrow$ biological meaning

Score $< 0 \Rightarrow$ none

$S(x, y) = -1$

ACG...T

chose n gaps

Goal: find alignment with highest score

$\approx O(2^n)$ exponential

$\binom{2n}{n}$ ← total # bases
← max # gaps in one seq. determined

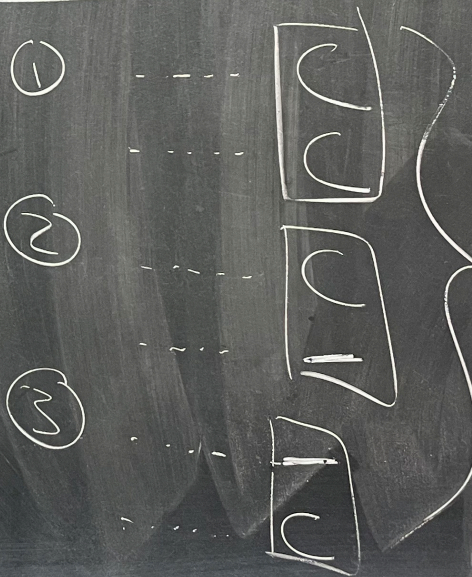
biological meaning
none

ACG --- T ---
--- TGC ---
chose n gaps

Can we do better?

X = AAAC
Y = AGC

3 ways to end



choose best

base

recur



Needleman-Wunsch

algorithm

$S(i, j) = \text{best alignment score}$
for $x[1 \dots i] + y[1 \dots j]$

base case

$$S(i, 0) = g \cdot i$$

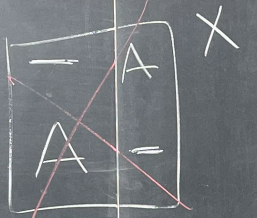
$$S(0, j) = g \cdot j$$

g is gap penalty

recursion

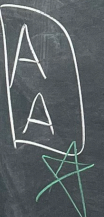
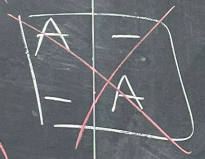
$$S(i, j) = \max \left\{ \begin{array}{l} S(i-1, j-1) + \underbrace{m(x_i, x_j)}_{\text{match score for } x_i, x_j} \\ S(i-1, j) + g \\ S(i, j-1) + g \end{array} \right.$$

X ~~---~~
 Y AG



X A A A

Y ~~---~~



$y_j = 0$
 0 -
 1 A
 2 A
 3 A
 4 C

	0	1	2	3
		A	G	C
0	0	-2	-4	-6
1	-2	(-4) (1)		
2	-4			
3	-6			
4	-8			

