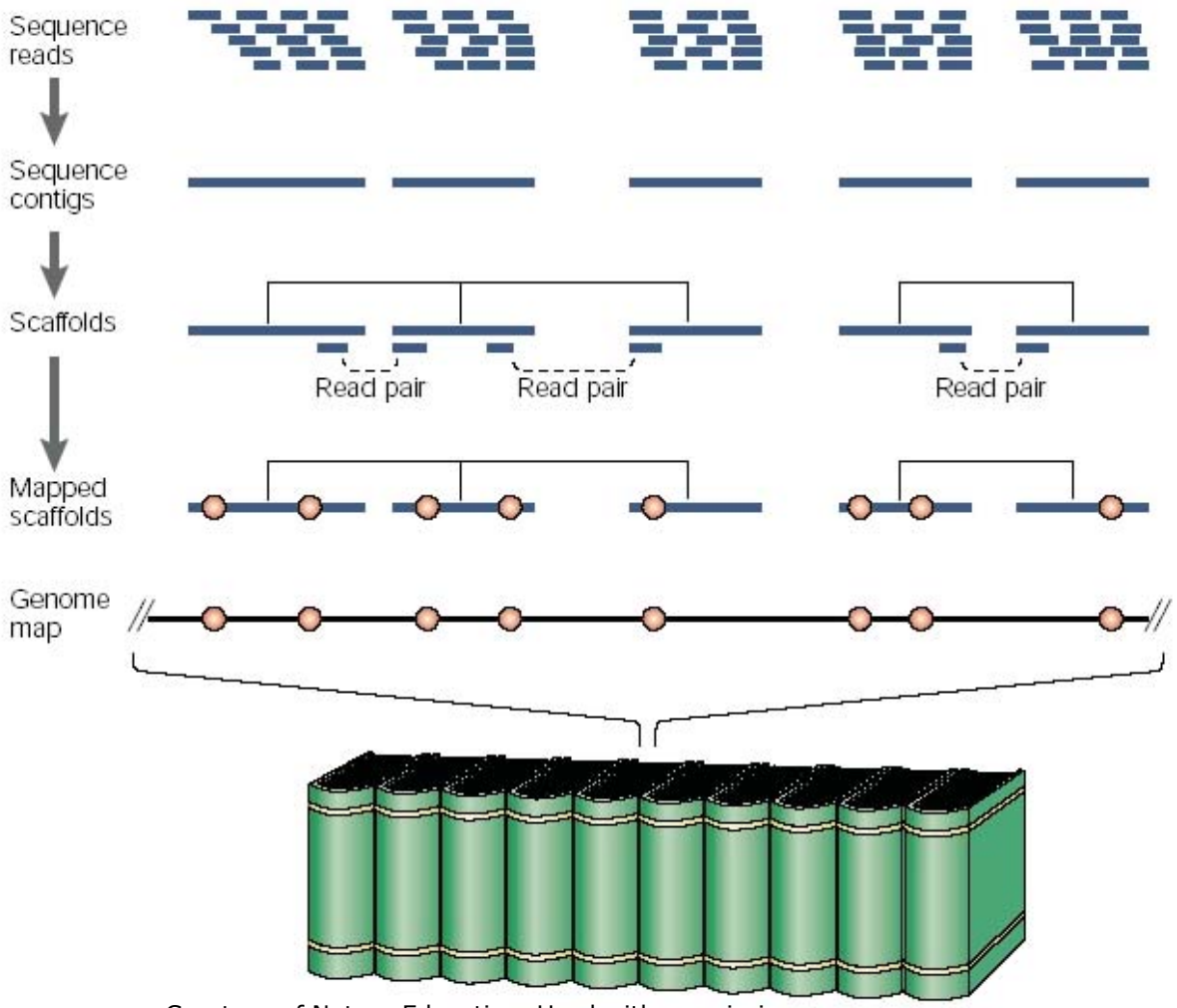
Lecture 6
Genome Assembly

Foundations of Computational Systems Biology

David K. Gifford

de novo whole-genome shotgun assembly



Source: Green, Eric D. "Strategies for the Systematic Sequencing of Complex Genomes." *Nature Reviews Genetics* 2, no. 8 (2001): 573-83.

Adams, J. (2008) Complex genomes: Shotgun sequencing. Nature Education 1(1)

Assembly

Whole-genome "shotgun" sequencing starts by copying and fragmenting the DNA

("Shotgun" refers to the random fragmentation of the whole genome; like it was fired from a shotgun)

Input: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Copy:
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Fragment:
GGCGTCTA TATCTCGG CTCTAGGCCCTC ATTTTTT
GGC GTCTATAT CTCGGCTCTAGGCCCTCA TTTTTT
GGCGTC TATATCT CGGCTCTAGGCCCT CATTTTTT
GGCGTCTAT ATCTCGGCTCTAG GCCCTCA TTTTTT

http://www.langmead-lab.org/teaching-materials/

Assembly

Assume sequencing produces such a large # fragments that almost all genome positions are *covered* by many fragments...

Reconstruct this

```
                    CTAGGCCCTCAATTTTT
                  CTCTAGGCCCTCAATTTTT
                GGCTCTAGGCCCTCATTTTTT
              CTCGGCTCTAGCCCCTCATTTT
          TATCTCGACTCTAGGCCCTCA
          TATCTCGACTCTAGGCC
     TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
```

From these

```
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Assembly

...but we don't know what came from where

Reconstruct this

CTAGGCCCTCAATTTTT
GGCGTCTATATCT
CTCTAGGCCCTCAATTTTT
TCTATATCTCGGCTCTAGG
GGCTCTAGGCCCTCATTTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
GGCGTCGATATCT
TATCTCGACTCTAGGCC
GGCGTCTATATCTCG

From these

GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Assembly

Key term: *coverage*. Usually it's short for *average coverage*: the average number of reads covering a position in the genome.

```
                    CTAGGCCCTCAATTTTT
                  CTCTAGGCCCTCAATTTTT
                GGCTCTAGGCCCTCATTTTTT
            CTCGGCTCTAGCCCCTCATTTT
        TATCTCGACTCTAGGCCCTCA
        TATCTCGACTCTAGGCC
    TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

177 nucleotides

35 nucleotides

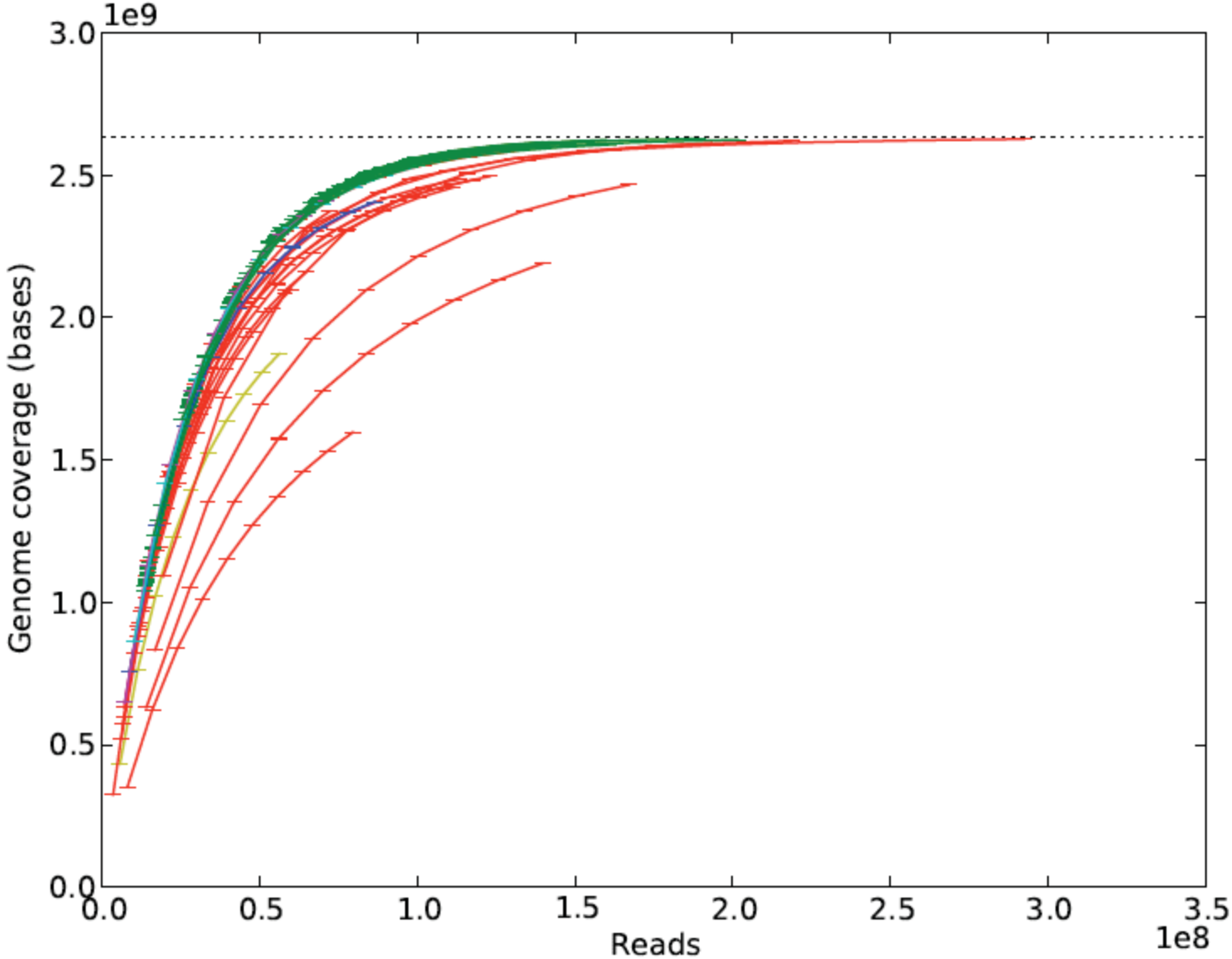
Average coverage = 177 / 35 ≈ 7x

Courtesy of Ben Langmead. Used with permission. http://www.langmead-lab.org/teaching-materials/

Estimating Uncovered Bases (Lander Waterman)

- G – genome size
- N - # of reads
- L – Length of read
- NL/G = reads/base = λ (coverage)
  - Poisson(0, λ) = $e^{-\lambda}$ =~ probability a base is not covered
  - # of uncovered bases =~ $Ge^{-\lambda}$
  - # of gaps= ~ $Ne^{-\lambda}$

Reads vs. coverage for 1000 Genomes Datasets



Assembly

Coverage could also refer to the number of reads covering a particular position in the genome:

```
                    CTAGGCCCTCAATTTTT
                  CTCTAGGCCCTCAATTTTT
               GGCTCTAGGCCCTCATTTTTT
           CTCGGCTCTAGCCCCTCATTTT
       TATCTCGACTCTAGGCCCTCA
       TATCTCGACTCTAGGCC
   TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

Coverage at this position = 6

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Assembly

Say two reads truly originate from overlapping stretches of the genome. Why might there be differences?

```
TATCTCGACTCTAGGCC
||||||| |||||||
TCTATATCTCGGCTCTAGG
```

1. Sequencing error
2. Difference between inhereted *copies* of a chromosome

   E.g. humans are diploid; we have two copies of each chromosome, one from mother, one from father. The copies can differ:

```
Read from Mother:      TATCTCGACTCTAGGCC
                       ||||||| |||||||
Read from Father:   TCTATATCTCGGCTCTAGG
Sequence from Mother:  TCTATATCTCGACTCTAGGCC
Sequence from Father:  TCTATATCTCGGCTCTAGGCC
```

We'll mostly ignore ploidy, but real tools must consider it

http://www.langmead-lab.org/teaching-materials/

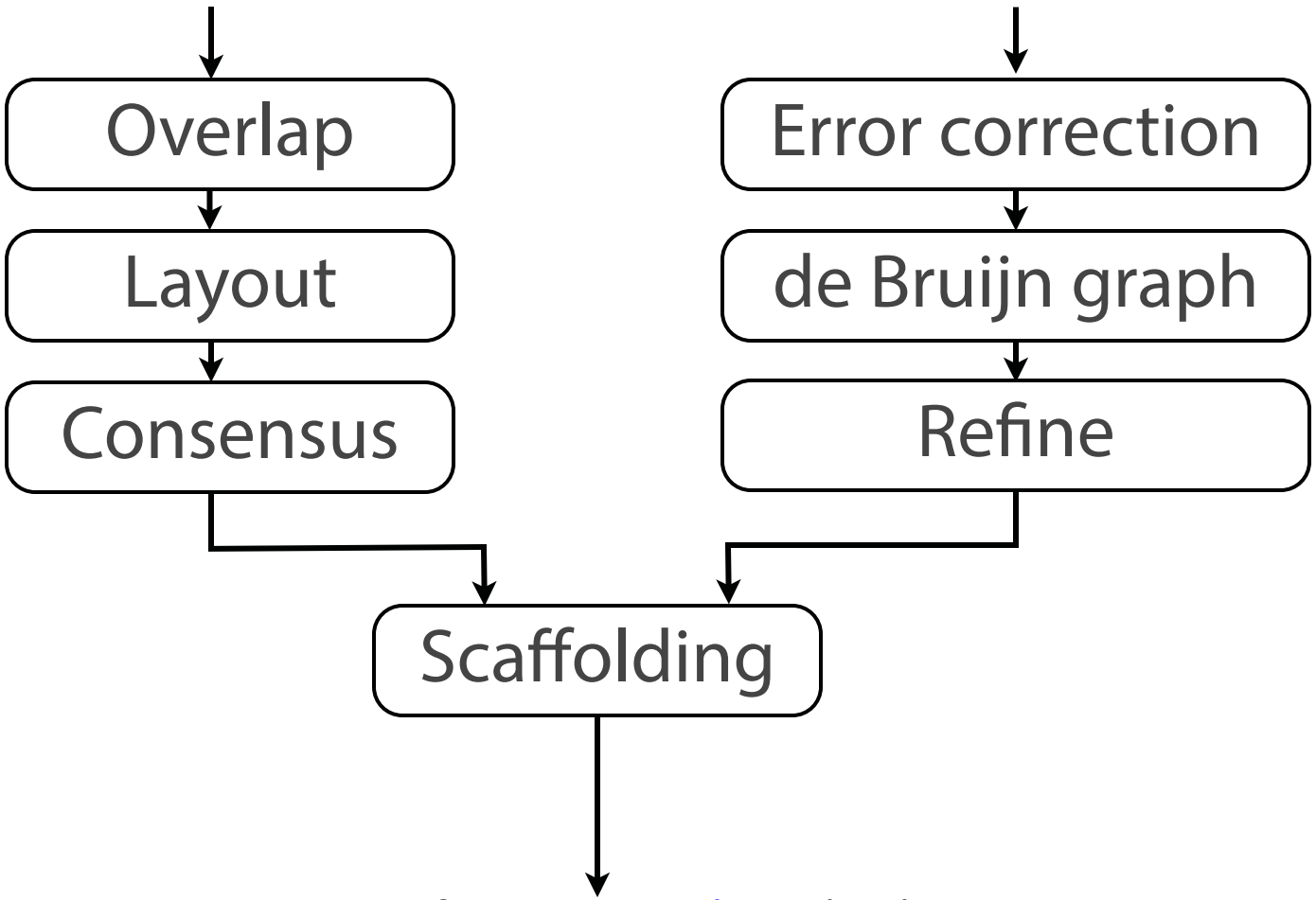
Two approaches to short read assembly

- Overlap Layout Consensus - String Graph Assemblers
  - Construct overlap graph directly from reads, eliminating redundant reads; trace path for assembly
  - Examples: SGA, Fermi
- de Bruijn graph-based assemblers
  - Construct k-mer graph from reads; original reads are discarded
  - Trace path in graph for assembly

Assembly alternatives

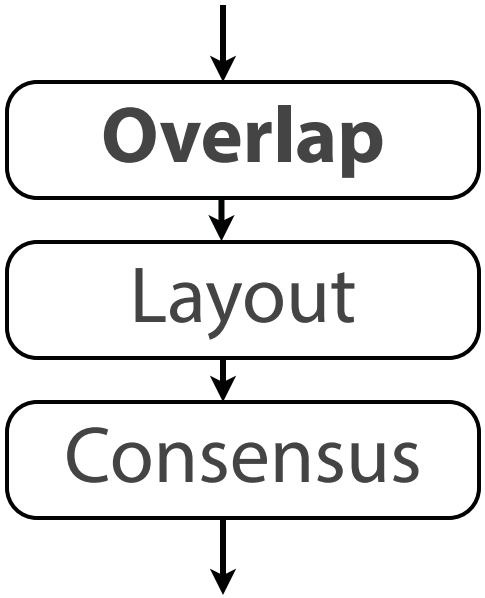
Alternative 1: Overlap-Layout-Consensus (OLC) assembly

Alternative 2: de Bruijn graph (DBG) assembly



Courtesy of Ben Langmead. Used with permission. http://www.langmead-lab.org/teaching-materials/

Overlap Layout Consensus



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

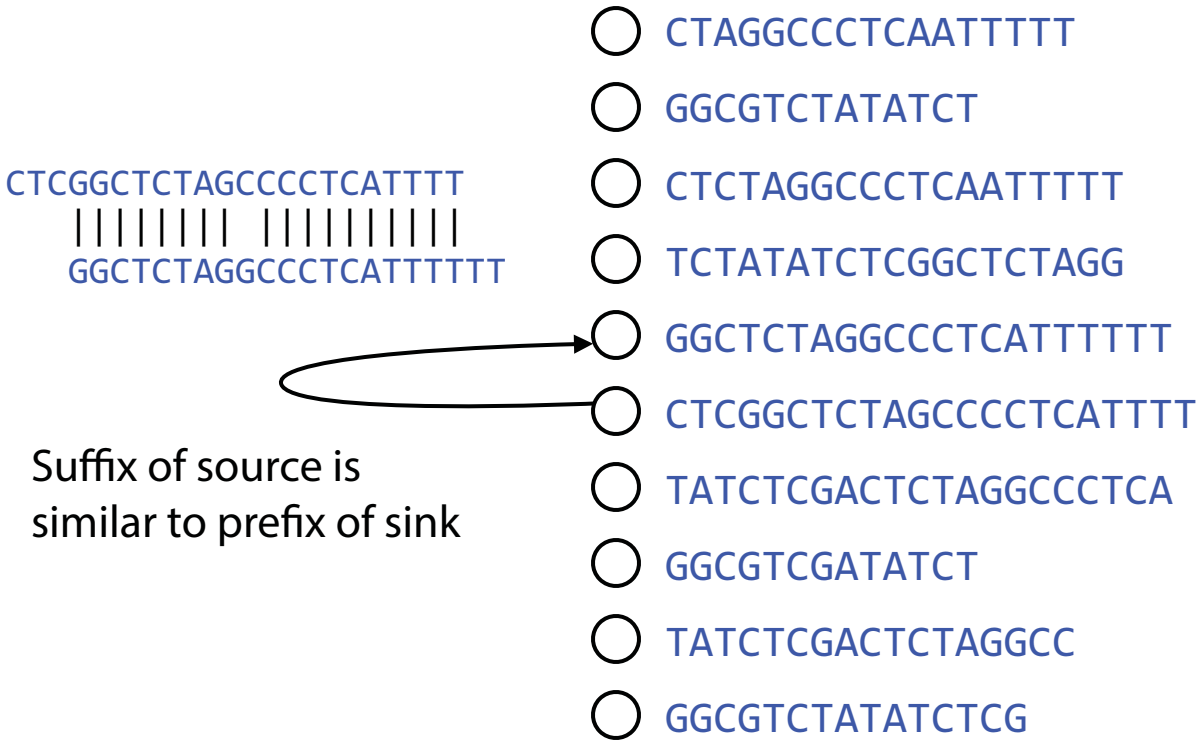
Pick most likely nucleotide sequence for each contig

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Overlaps

Finding all overlaps is like building a *directed graph* where directed edges connect overlapping nodes (reads)



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

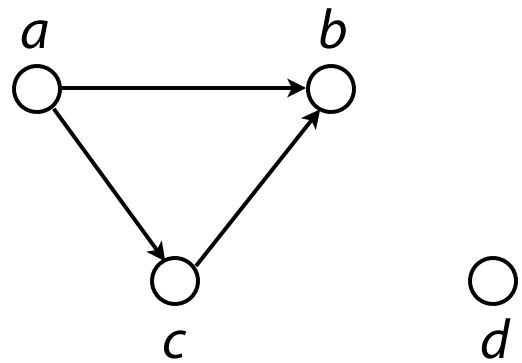
Directed graph review

Directed graph $G(V, E)$ consists of set of *vertices, V* and set of *directed edges, E*

Directed edge is an *ordered pair* of vertices.
First is the *source*, second is the *sink*.

Vertex is drawn as a circle

Edge is drawn as a line with an arrow connecting two circles



Vertex also called *node* or *point*

Edge also called *arc* or *line*

Directed graph also called *digraph*

$V = \{ a, b, c, d \}$

$E = \{ (a, b), (a, c), (c, b) \}$

Source Sink

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Overlap graph

Below: overlap graph, where an overlap is a suffix/prefix match of at least 3 characters

A vertex is a **read**, a directed edge is an overlap between suffix of source and prefix of sink

Vertices (reads): { *a:* CTCTAGGCC, *b:* GCCCTCAAT, *c:* CAATTTTT }

Edges (overlaps): { (*a*, *b*), (*b*, *c*) }

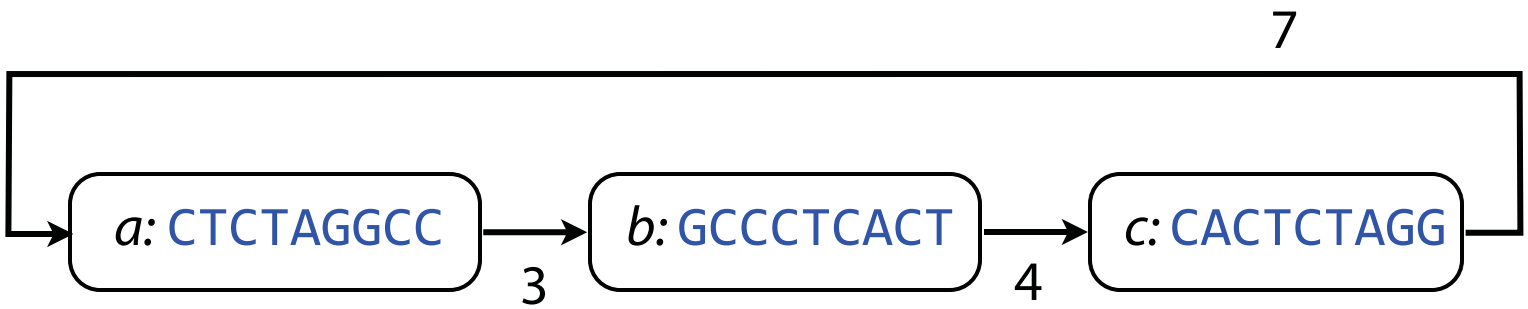
To keep our presentation uncluttered we will not show the treatment of read reverse complements


```
CTCTAGGCC          GCCCTCAAT
    |||                ||||
   GCCCTCAAT          CAATTTTT
```

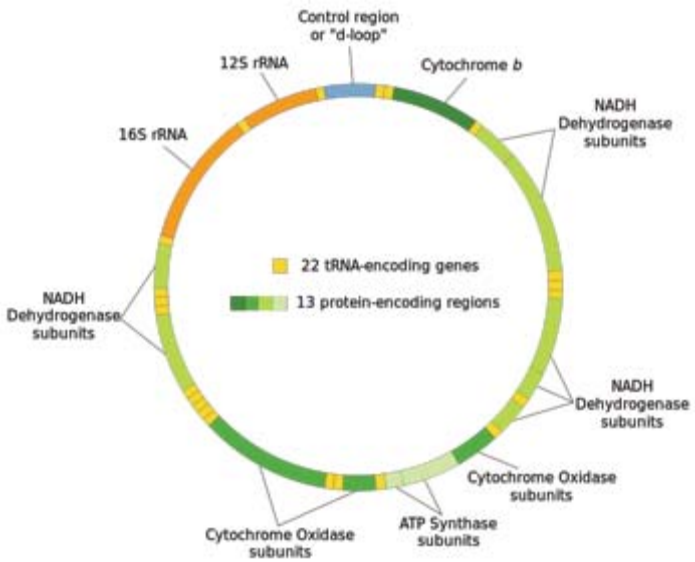
Courtesy of Ben Langmead. Used with permission. http://www.langmead-lab.org/teaching-materials/

Overlap graph

Overlap graph could contain *cycles*. A cycle is a path beginning and ending at the same vertex.



These happen when the DNA string itself is circular. E.g. bacterial genomes are often circular; mitochondrial DNA is circular.



Cycles could also be due to *repetitive* DNA, as we'll see

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Finding overlaps

How do we build the overlap graph?

Assume for now an "overlap" is when a suffix of *X* of length ≥ l exactly matches a prefix of *Y*, and *k* is the length of reads

```
Index:   CTCTAGGCC$GCCCTCAAT$CAATTTTT$$
```

A merged FM index of all reads allows us to match read prefixes and suffixes to discover overlaps. See SGA paper -

Efficient de novo assembly of large genomes using compressed data structures
Jared T Simpson and Richard Durbin Genome Res. 2012. 22: 549–556

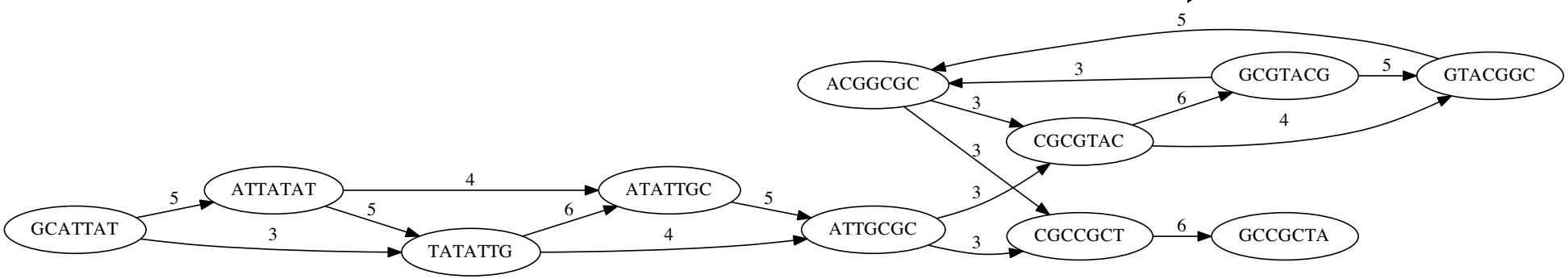
SGA algorithm excludes redundant (transitive) edges
A -> B -> C excludes A -> C

Finding overlaps

Example overlap graph with $l = 3$

$k = 7$

Edge label is overlap length

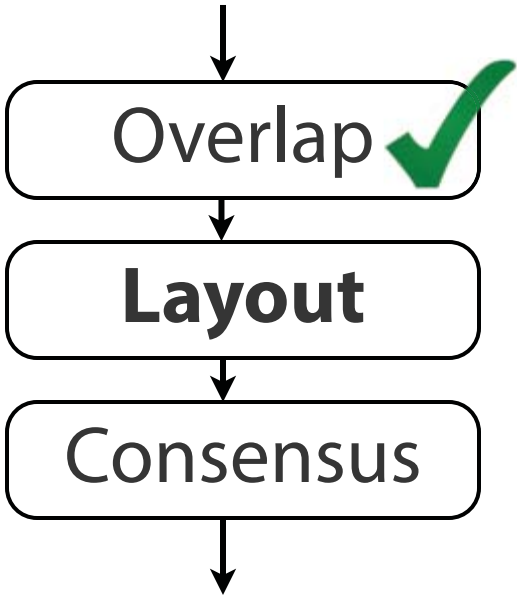


Original string: GCATTATATATTGCGCGTACGGCGCCGCTACA

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Overlap Layout Consensus



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

Pick most likely nucleotide sequence for each contig

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Formulating the assembly problem

Finding overlaps is important, and we'll return to it, but our ultimate goal is to recreate (assemble) the genome

How do we formulate this problem?

First attempt: the *shortest common superstring* (*SCS*) problem

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring

Given a collection of strings *S*, find *SCS*(*S*): the shortest string that contains all strings in *S* as substrings

Without requirement of "shortest," it's easy: just concatenate them

Example: *S*: BAA AAB BBA ABA ABB BBB AAA BAB

Concatenation: BAAAABBBAABAABBBBBAAABAB

24

SCS(S): AAABBBABAA

10

```
AAA
 AAB
  ABB
   BBB
    BBA
     BAB
      ABA
       BAA
```

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring

Can we solve it?

Imagine a modified overlap graph where each edge has cost = - (length of overlap)

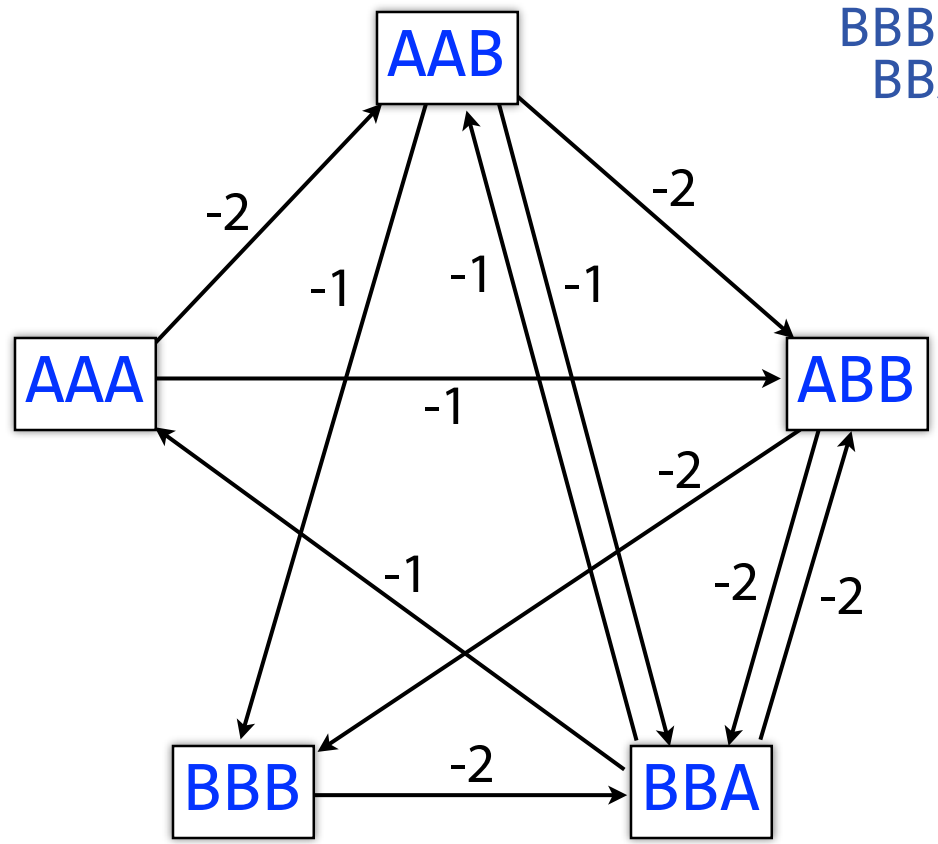
SCS corresponds to a path that visits every node once, minimizing total cost along path

That's the *Traveling Salesman Problem* (*TSP*), which is NP-hard!

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBA

AAA
AAB
ABB
BBB
BBA



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring

Say we disregard edge weights and just look for a path that visits all the nodes exactly once

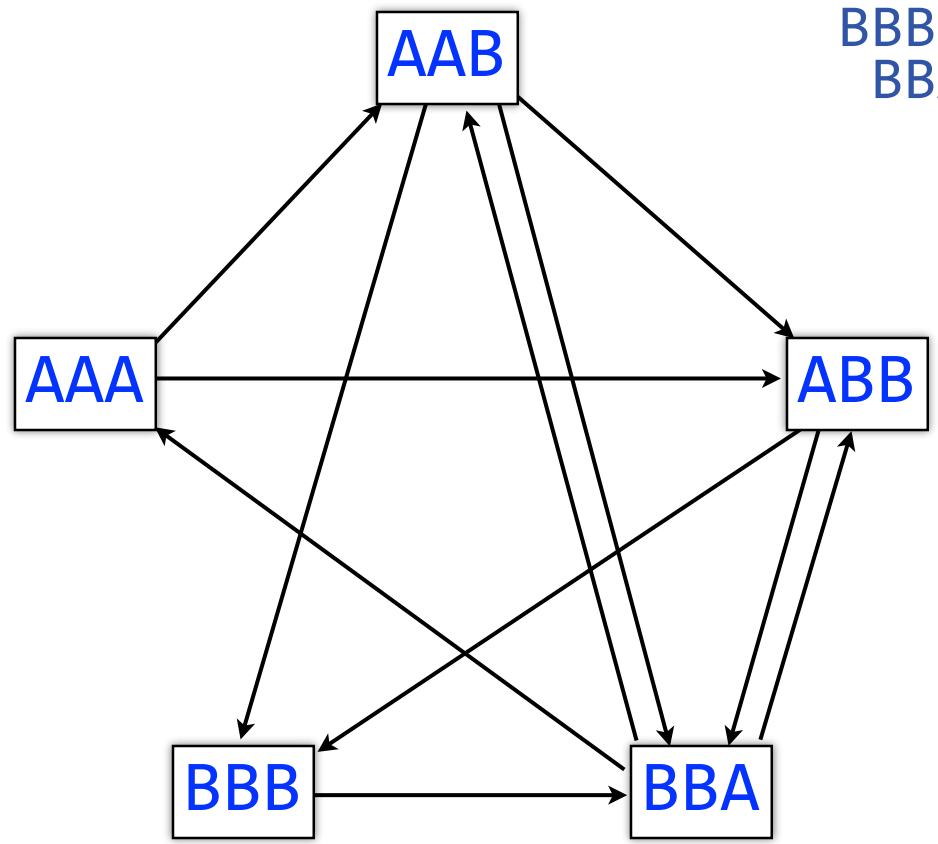
That's the *Hamiltonian Path* problem: NP-complete

Indeed, it's well established that SCS is NP-hard

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBA

```
AAA
 AAB
  ABB
   BBB
    BBA
```



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring

Let's take the hint give up on finding the *shortest possible* superstring

Non-optimal superstrings can be found with a *greedy* algorithm

At each step, the greedy algorithm "greedily" chooses longest remaining overlap, merges its source and sink

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring: greedy

Greedy-SCS algorithm in action ($l = 1$):

Input strings

	ABA ABB AAA AAB BBB BBA BAB BAA
2	BAAB ABA ABB AAA BBB BBA BAB
2	BABB BAAB ABA AAA BBB BBA
2	BBAAB BABB ABA AAA BBB
2	BBBAAB BABB ABA AAA
2	BBBAABA BABB AAA
2	BABBBAABA AAA
1	BABBBAABAAA
	BABBBAABAAA

Superstring

In red are strings that get merged before the next round

Greedy answer:
BABBBAABAAA

Actual SCS:
AAABBBABAA

Rounds of merging, one merge per line.
Number in first column = length of overlap merged before that round.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring: greedy

Greedy algorithm is *not* guaranteed to choose overlaps yielding SCS

But greedy algorithm is a good *approximation*; i.e. the superstring yielded by the greedy algorithm won't be more than ~2.5 times longer than true SCS (see Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, 16.17.1)

Shortest common superstring: greedy

Greedy-SCS algorithm in action again ($l = 3$):

Input strings

```
  ATTATAT CGCGTAC ATTGCGC GCATTAT ACGGCGC TATATTG GTACGGC GCGTACG ATATTGC
6 TATATTGC ATTATAT CGCGTAC ATTGCGC GCATTAT ACGGCGC GTACGGC GCGTACG
6 CGCGTACG TATATTGC ATTATAT ATTGCGC GCATTAT ACGGCGC GTACGGC
5 CGCGTACG TATATTGCGC ATTATAT GCATTAT ACGGCGC GTACGGC
5 CGCGTACGGC TATATTGCGC ATTATAT GCATTAT ACGGCGC
5 CGCGTACGGCGC TATATTGCGC ATTATAT GCATTAT
5 CGCGTACGGCGC GCATTATAT TATATTGCGC
5 CGCGTACGGCGC GCATTATATTGCGC
3 GCATTATATTGCGCGTACGGCGC
  GCATTATATTGCGCGTACGGCGC
```

Superstring

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring: greedy

Another setup for Greedy-SCS: assemble all substrings of length 6 from string a_long_long_long_time. $l = 3$.

```
  ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long g_time ng_tim
5 ng_time ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long
5 ng_time g_long_ ng_lon a_long long_l ong_ti ong_lo long_t
5 ng_time long_ti g_long_ ng_lon a_long long_l ong_lo
5 ng_time ong_lon long_ti g_long_ a_long long_l
5 ong_lon long_time g_long_ a_long long_l
5 long_lon long_time g_long_ a_long
5 long_lon g_long_time a_long
5 long_long_time a_long
4 a_long_long_time
  a_long_long_time
```

We only got back: a_long_long_time (missing a _long)

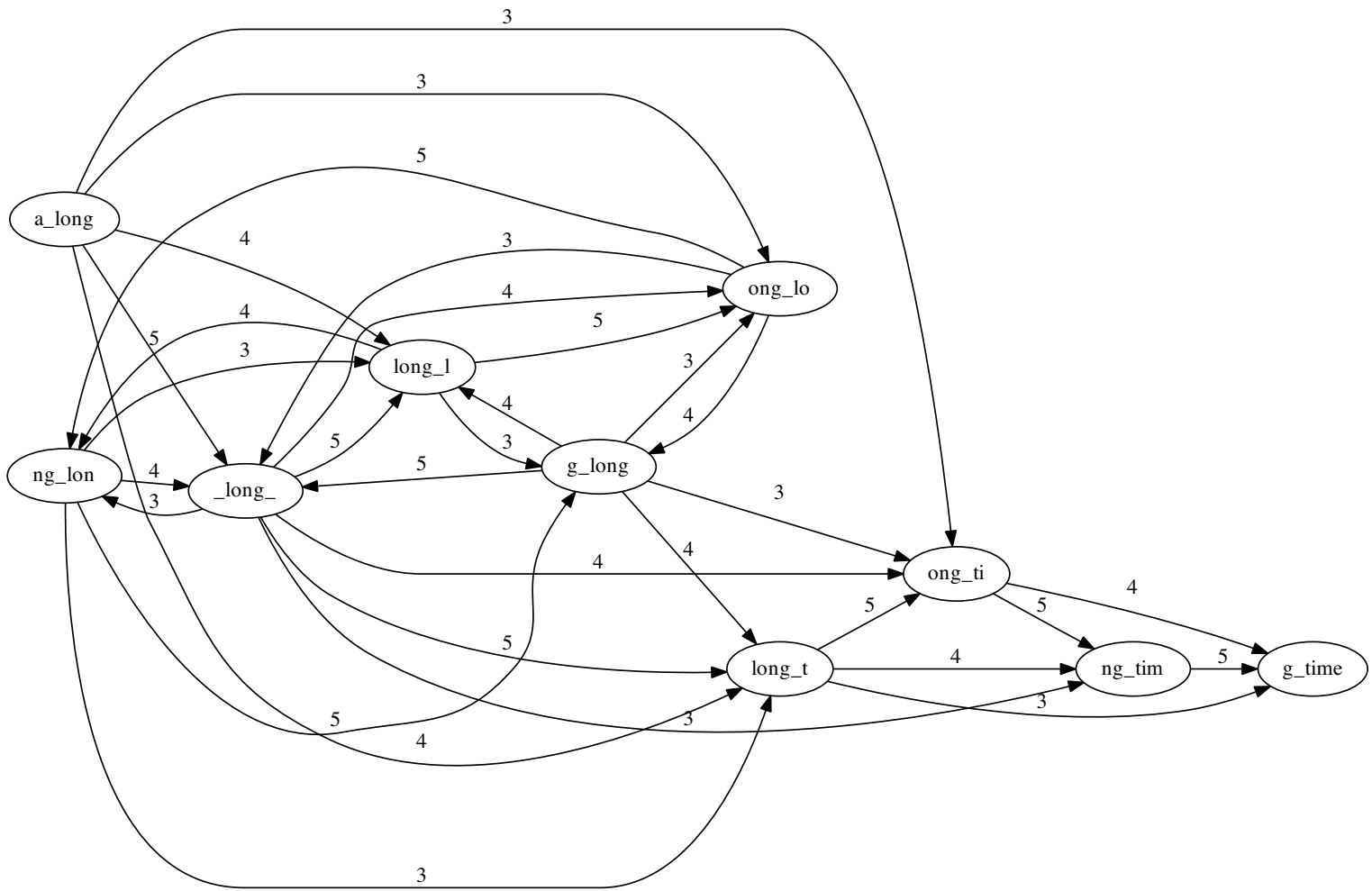
What happened?

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring: greedy

The overlap graph for that scenario ($l = 3$):

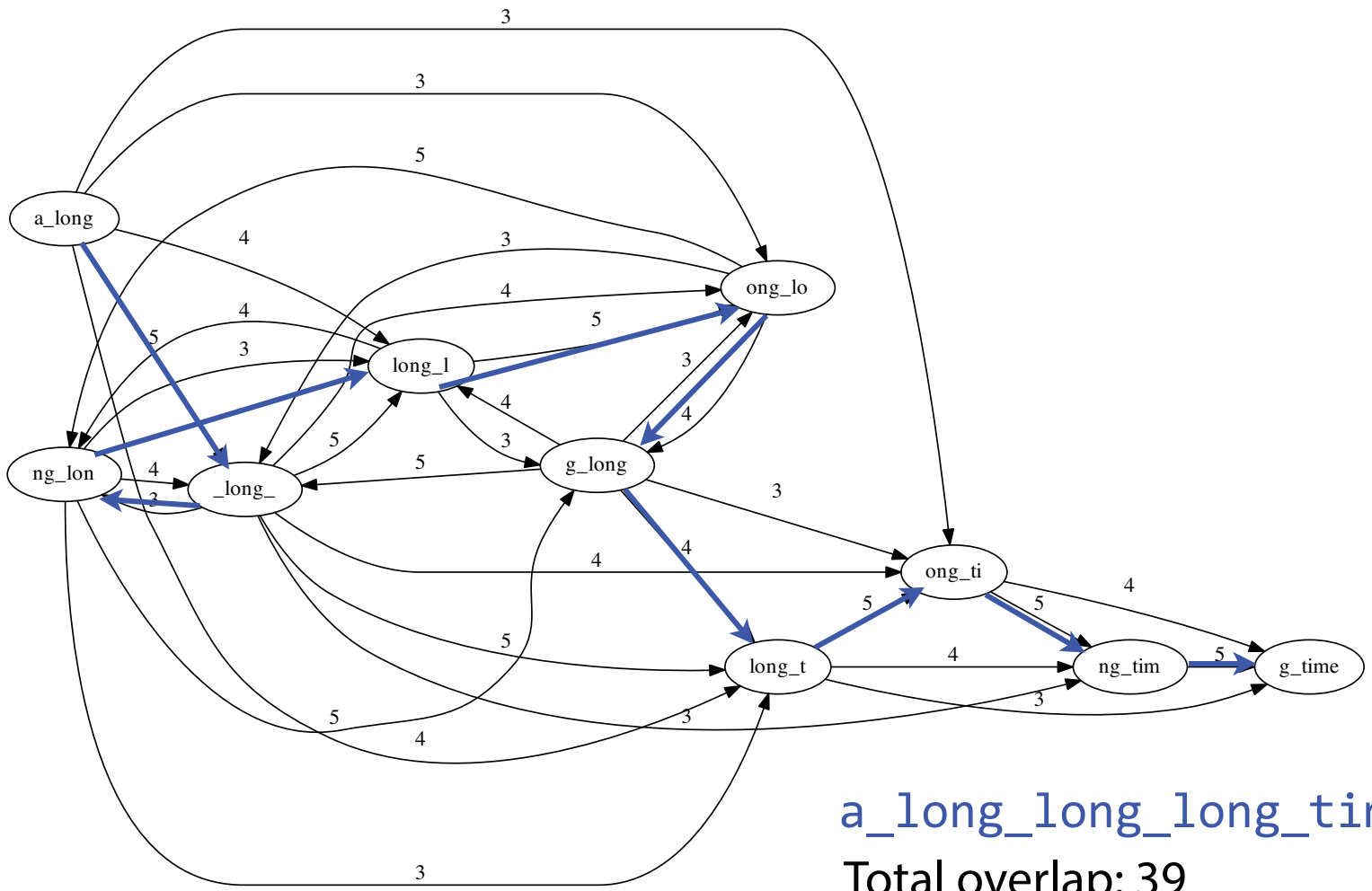


Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring: greedy

The overlap graph for that scenario ($l = 3$):



`a_long_long_long_time`

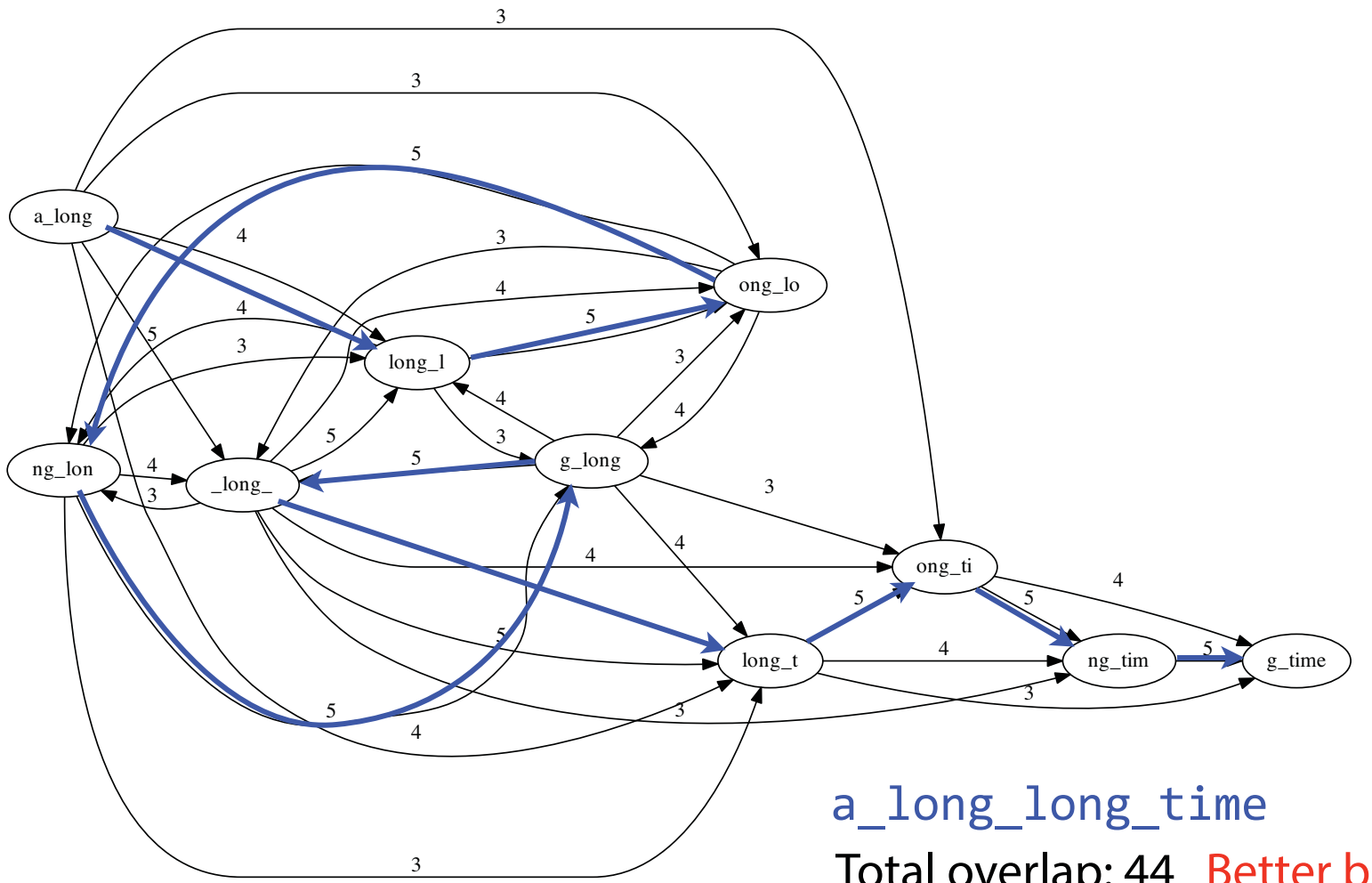
Total overlap: 39

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring: greedy

The overlap graph for that scenario ($l = 3$):



`a_long_long_time`

Total overlap: 44 Better but wrong!

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Shortest common superstring: greedy

Same example, but increased the substring length from 6 to 8

```
  long_lon ng_long_ _long_lo g_long_t ong_long g_long_l ong_time a_long_l _long_ti long_tim
7 long_time long_lon ng_long_ _long_lo g_long_t ong_long g_long_l a_long_l _long_ti
7 _long_time long_lon ng_long_ _long_lo g_long_t ong_long g_long_l a_long_l
7 _long_time a_long_lo long_lon ng_long_ g_long_t ong_long g_long_l
7 _long_time ong_long_ a_long_lo long_lon g_long_t g_long_l
7 g_long_time ong_long_ a_long_lo long_lon g_long_l
7 g_long_time ong_long_ a_long_lon g_long_l
7 g_long_time ong_long_l a_long_lon
7 g_long_time a_long_long_l
3 a_long_long_long_time
  a_long_long_long_time
```

Got the whole thing: `a_long_long_long_time`

Shortest common superstring: greedy

Why are substrings of length 8 long enough for Greedy-SCS to figure out there are 3 copies of `long`?

```
a_long_long_long_time
  g_long_l
```

One length-8 substring spans all three `long`s

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Repeats

Repeats often foil assembly. They certainly foil SCS, with its "shortest" criterion!

Reads might be too short to "resolve" repetitive sequences. This is why sequencing vendors try to increase read length.

Algorithms that don't pay attention to repeats (like our greedy SCS algorithm) might *collapse* them

```
a_long_long_long_time
        |
        | collapse
        v
a_long_long_time
```

The human genome is ~ 50% repetitive!

http://www.langmead-lab.org/teaching-materials/

Repeats

Basic principle: *repeats foil assembly*

Another example using Greedy-SCS:

Input: `it_was_the_best_of_times_it_was_the_worst_of_times`

Extract every substring of length k, then run Greedy-SCS.
Do this for various l (min overlap length) and k.

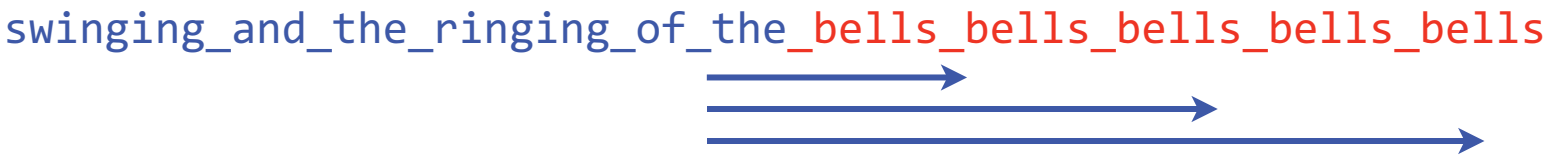
l, k	output
3, 5	`the_worst_of_times_it_was_the_best_o`
3, 7	`s_the_worst_of_times_it_was_the_best_of_t`
3, 10	`_was_the_best_of_times_it_was_the_worst_of_tim`
3, 13	`it_was_the_best_of_times_it_was_the_worst_of_times`

http://www.langmead-lab.org/teaching-materials/

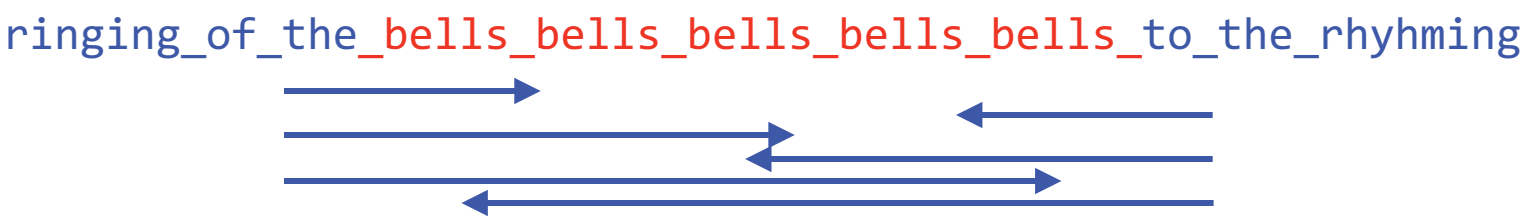
Repeats

Basic principle: *repeats foil assembly*

Longer and longer substrings allow us to "anchor" more of the repeat to its non-repetitive context:



Often we can "walk in" from both sides. When we meet in the middle, the repeat is resolved:



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Repeats

Basic principle: *repeats foil assembly*

Yet another example using Greedy-SCS:

Input: `swinging_and_the_ringing_of_the_bells_bells_bells_bells_bells`

l, k	output
3, 7	`swinging_and_the_ringing_of_the_bells_bells`
3, 13	`swinging_and_the_ringing_of_the_bells_bells_bells`
3, 19	`swinging_and_the_ringing_of_the_bells_bells_bells_bells_b`
3, 25	`swinging_and_the_ringing_of_the_bells_bells_bells_bells_bells`

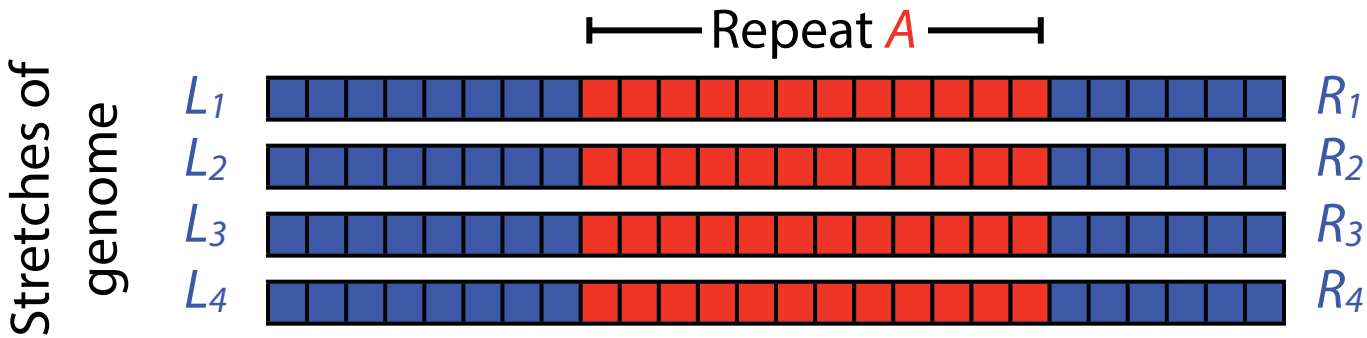
longer and longer substrings allow us to "reach" further into the repeat

Courtesy of Ben Langmead. Used with permission.

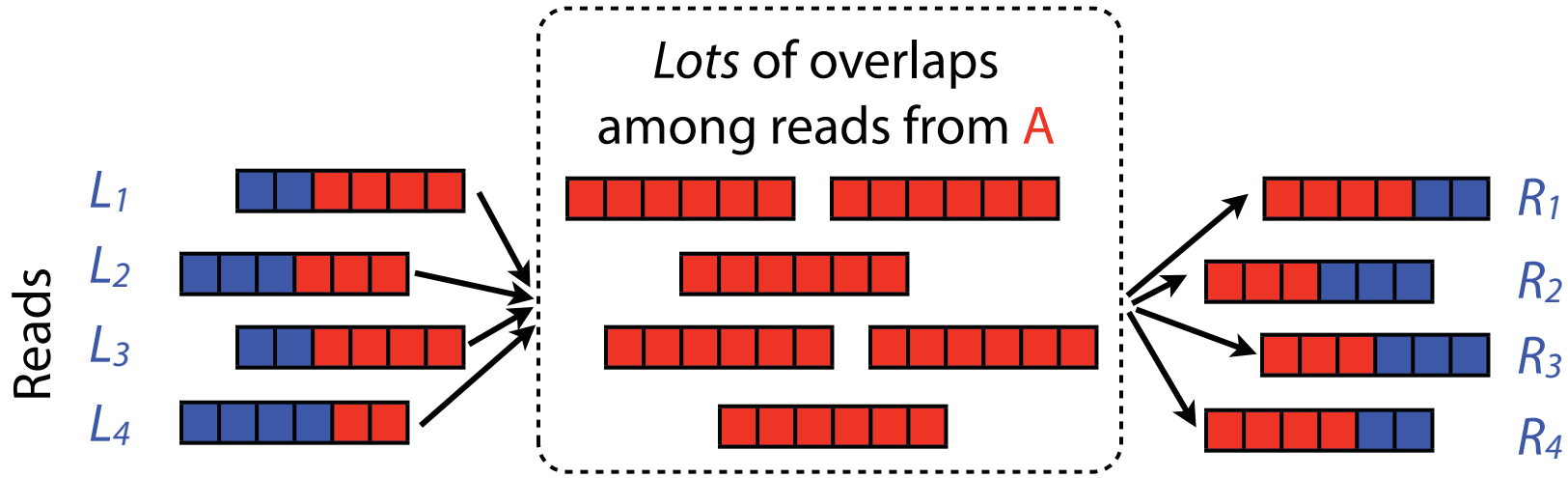
http://www.langmead-lab.org/teaching-materials/

Repeats

Picture the portion of the overlap graph involving repeat *A*



Assume *A* is longer than read length



Even if we avoid collapsing copies of *A*, we can't know which paths *in* correspond to which paths *out*

http://www.langmead-lab.org/teaching-materials/

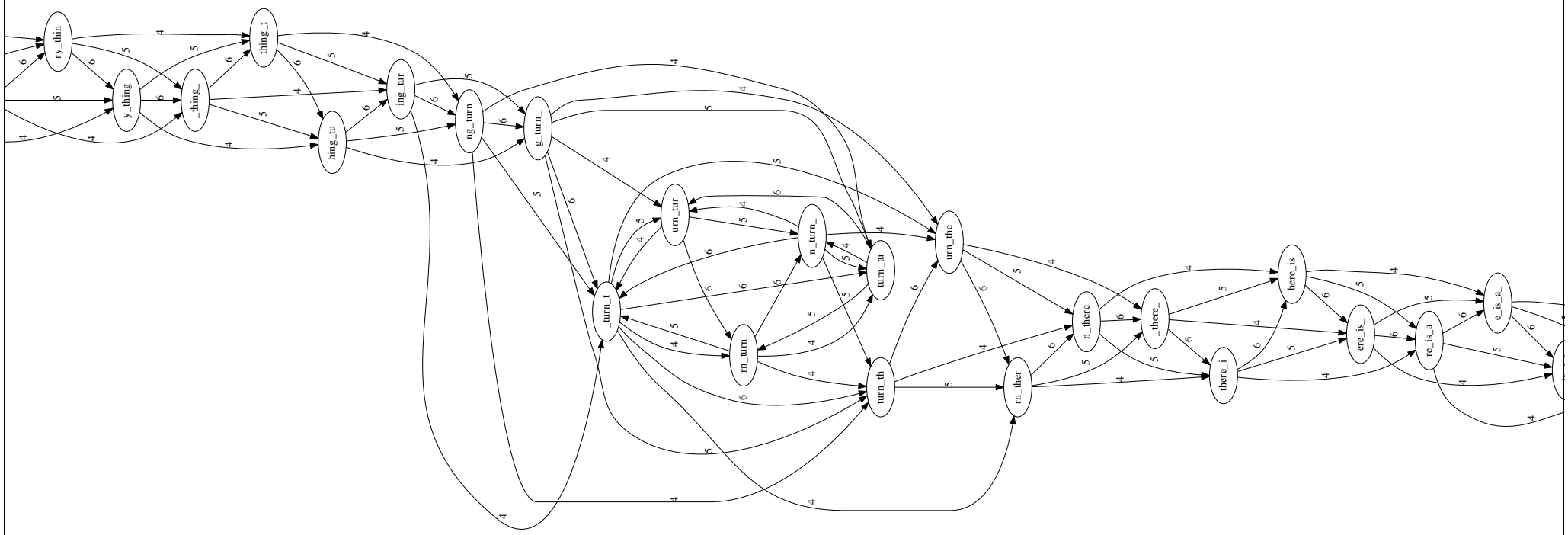
Layout

The overlap graph is big and messy. Contigs don't "pop out" at us.

Below: part of the overlap graph for

`to_every_thing_turn_turn_turn_there_is_a_season`

$l = 4, k = 7$

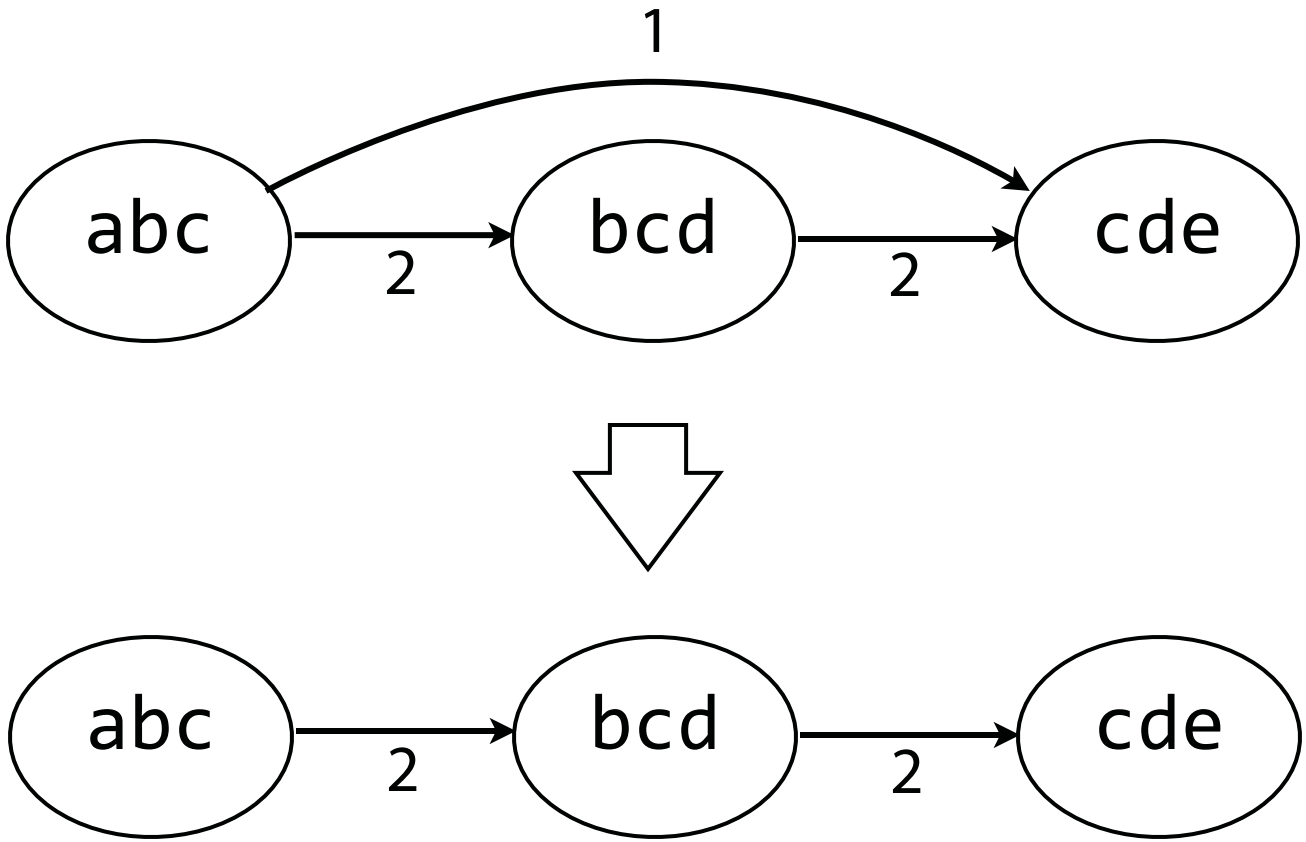


Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Layout

Picture gets clearer after removing some transitively-inferrible edges

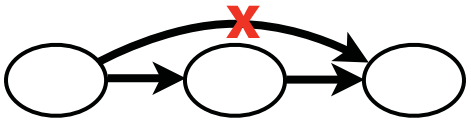


Courtesy of Ben Langmead. Used with permission.

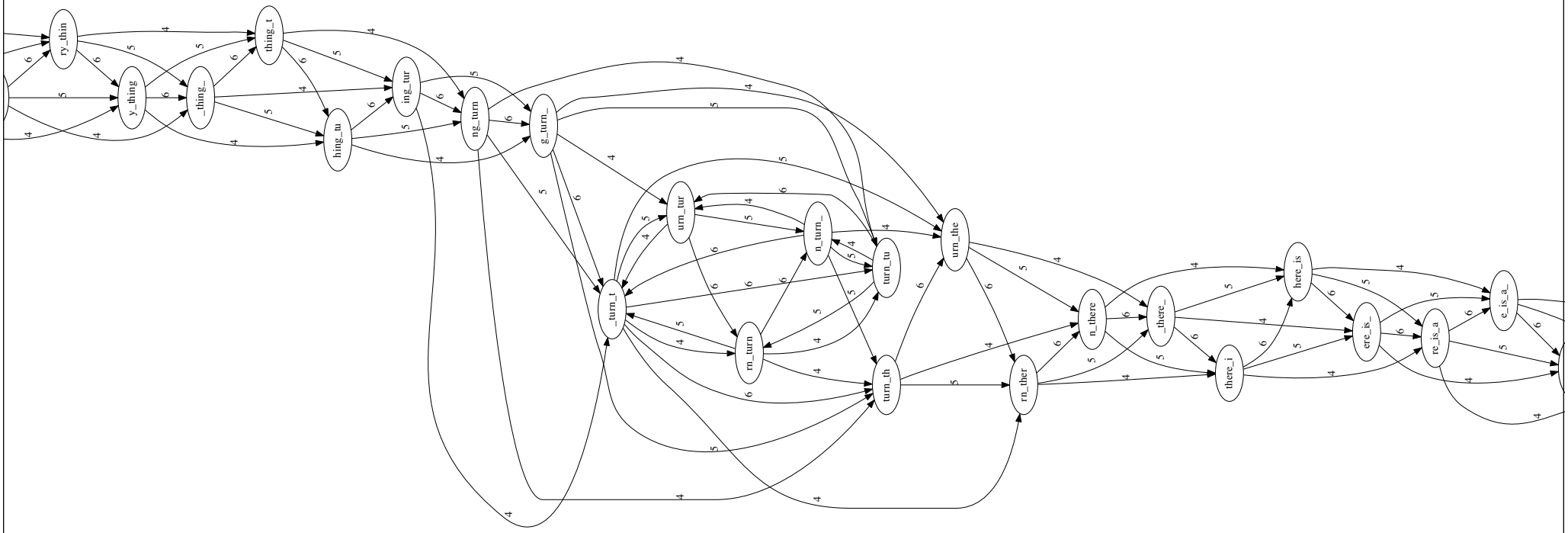
http://www.langmead-lab.org/teaching-materials/

Layout

Remove transitively-inferrible edges, starting with edges that skip one node:



Before:

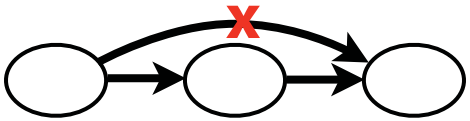


Courtesy of Ben Langmead. Used with permission.

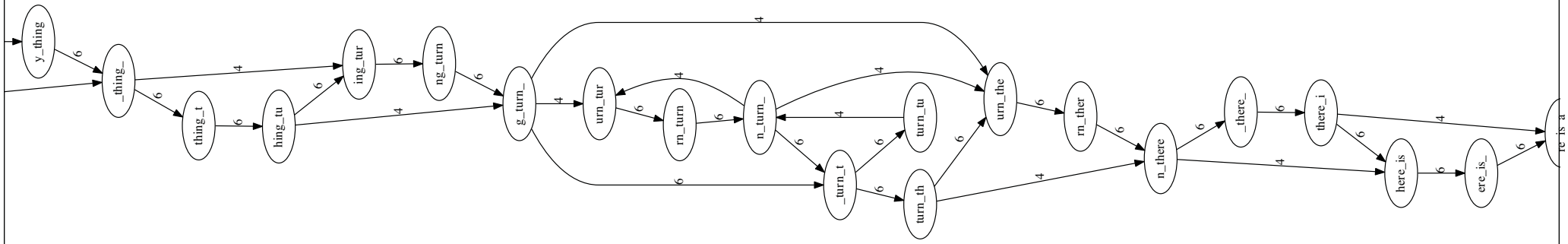
http://www.langmead-lab.org/teaching-materials/

Layout

Remove transitively-inferrible edges, starting with edges that skip one node:



After:

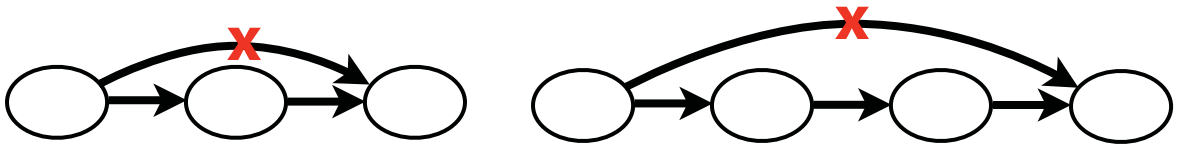


Courtesy of Ben Langmead. Used with permission.

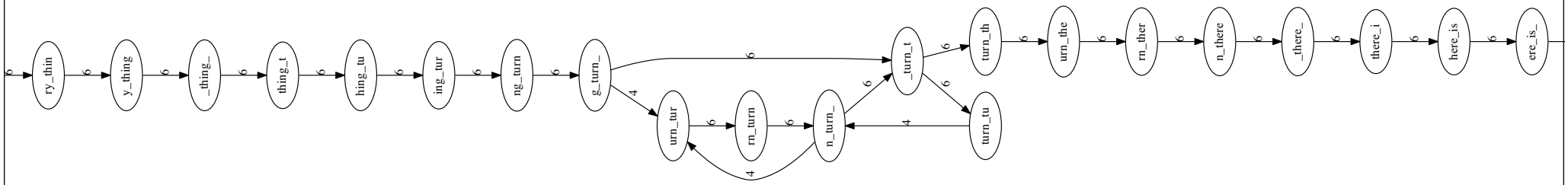
http://www.langmead-lab.org/teaching-materials/

Layout

Remove transitively-inferrible edges, starting with edges that skip one *or two* nodes:



After:



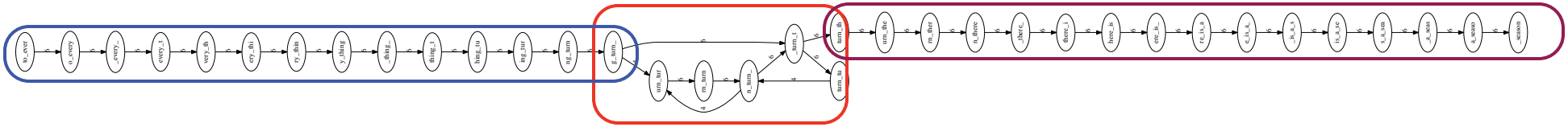
Even simpler

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Layout

Emit *contigs* corresponding to the non-branching stretches



Contig 1
`to_every_thing_turn_`

Contig 2
`turn_there_is_a_season`

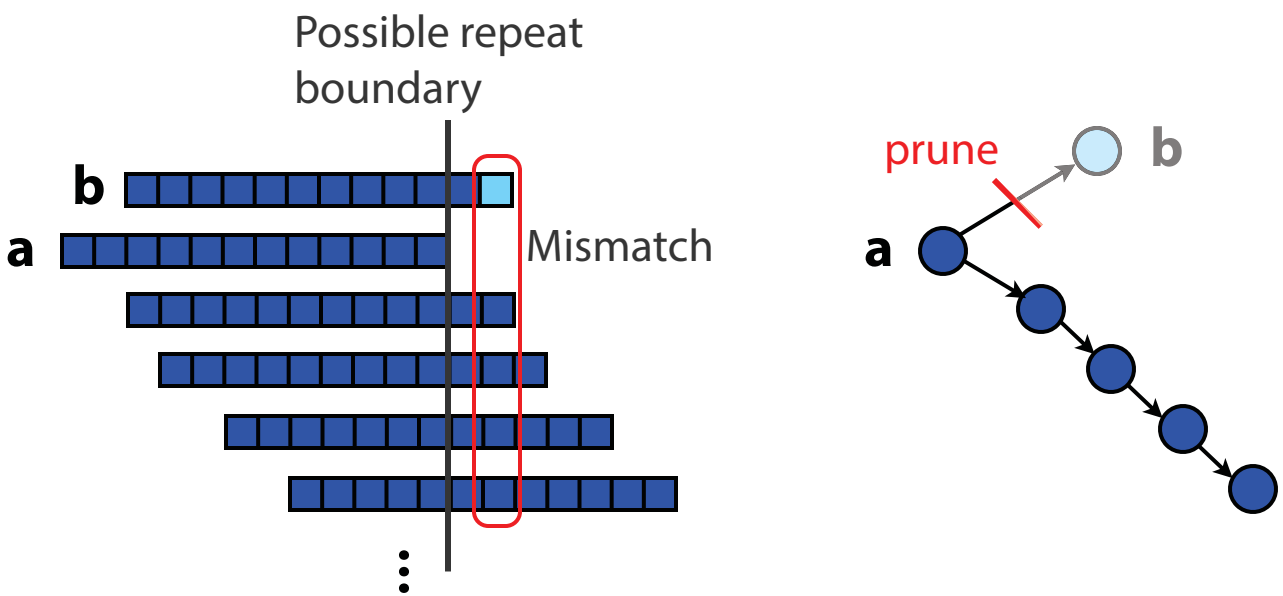
Unresolvable repeat

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Layout

In practice, layout step also has to deal with spurious subgraphs, e.g. because of sequencing error

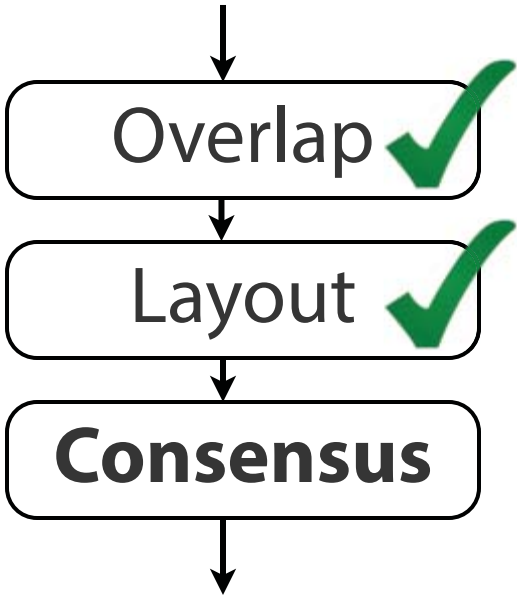


Mismatch could be due to sequencing error or repeat. Since the path through **b** ends abruptly we might conclude it's an error and prune **b**.

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Overlap Layout Consensus



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

Pick most likely nucleotide sequence for each contig

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Consensus

```
TAGATTACACAGATTACTGA TTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAAACTA
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

Take reads that make up a contig and line them up

```
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

Take *consensus*, i.e. majority vote

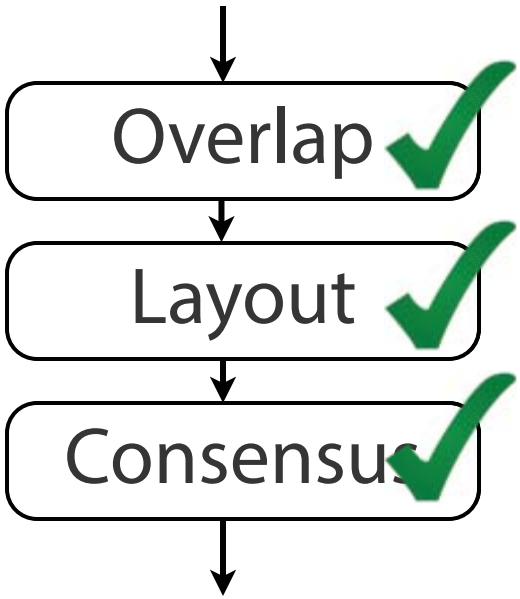
At each position, ask: what nucleotide (and/or gap) is here?

Complications: (a) sequencing error, (b) ploidy

Say the true genotype is AG, but we have a high sequencing error rate and only about 6 reads covering the position.

http://www.langmead-lab.org/teaching-materials/

Overlap Layout Consensus



Build overlap graph

Bundle stretches of the overlap graph into *contigs*

Pick most likely nucleotide sequence for each contig

What's the main drawback of OLC?

Building overlap graph can be *slow*.

2nd-generation sequencing datasets are ~ 100s of millions or billions of reads, hundreds of billions of nucleotides total

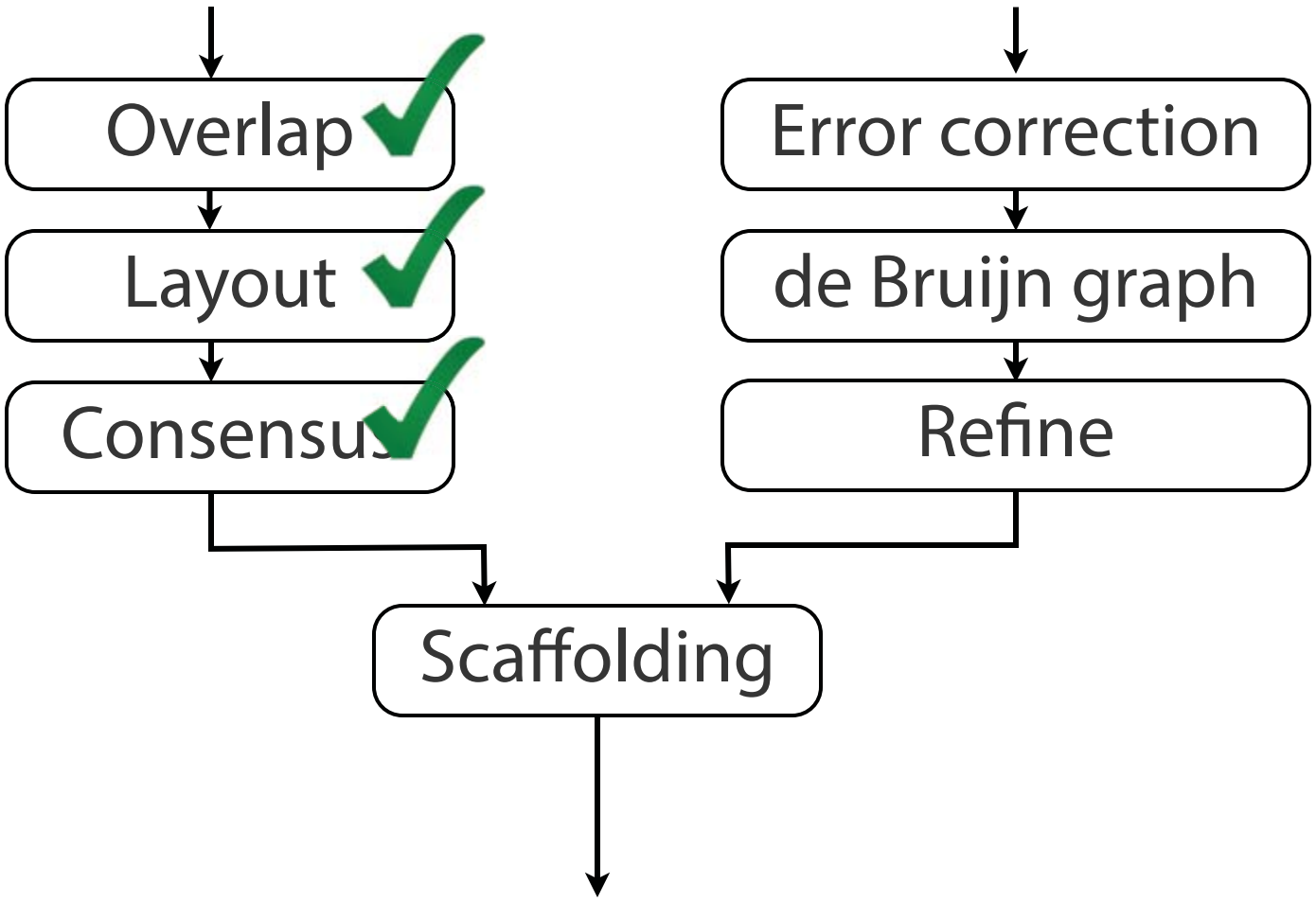
Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Assembly alternatives

Alternative 1: Overlap-Layout-Consensus (OLC) assembly

Alternative 2: de Bruijn graph (DBG) assembly



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Flow chart removed due to copyright restrictions.

Table removed due to copyright restrictions.

SGA contigs cover 95% of autosomes and chr X (non "N" bases)

NA12878 1.2 x 10^9 reads 40x coverage

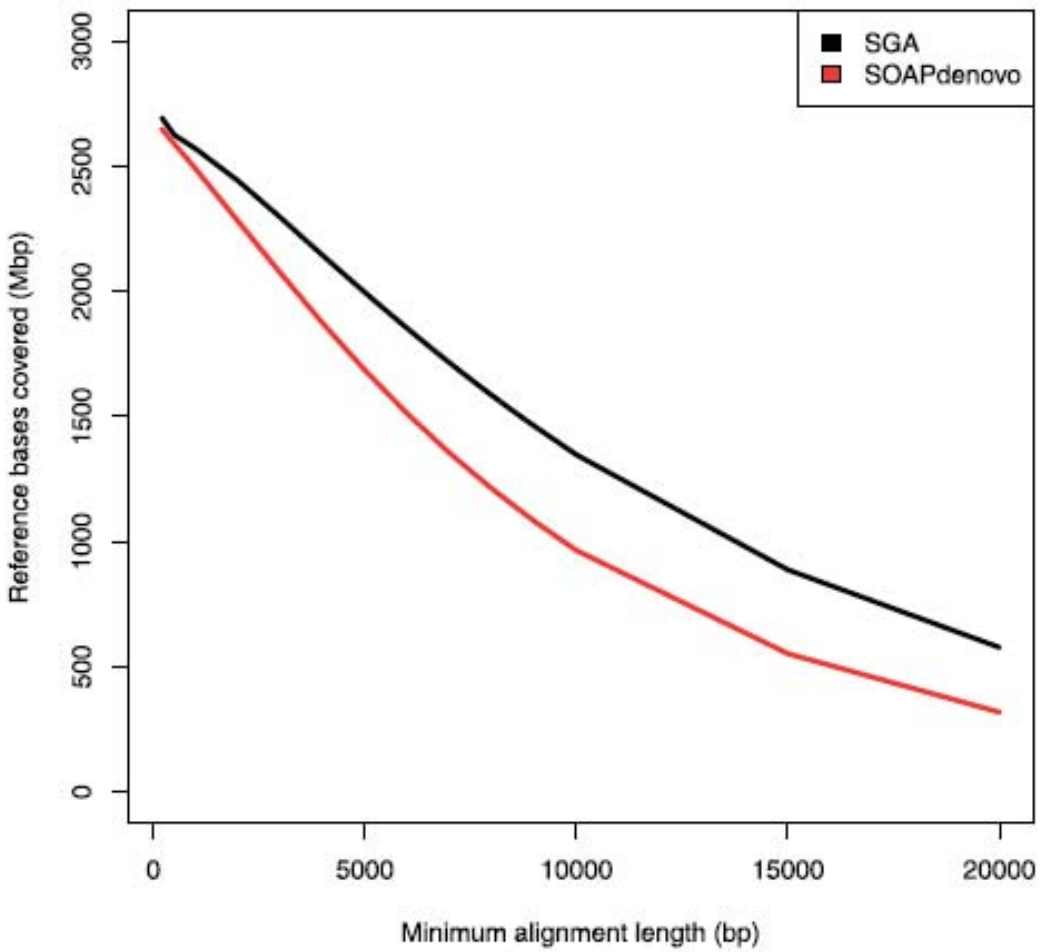


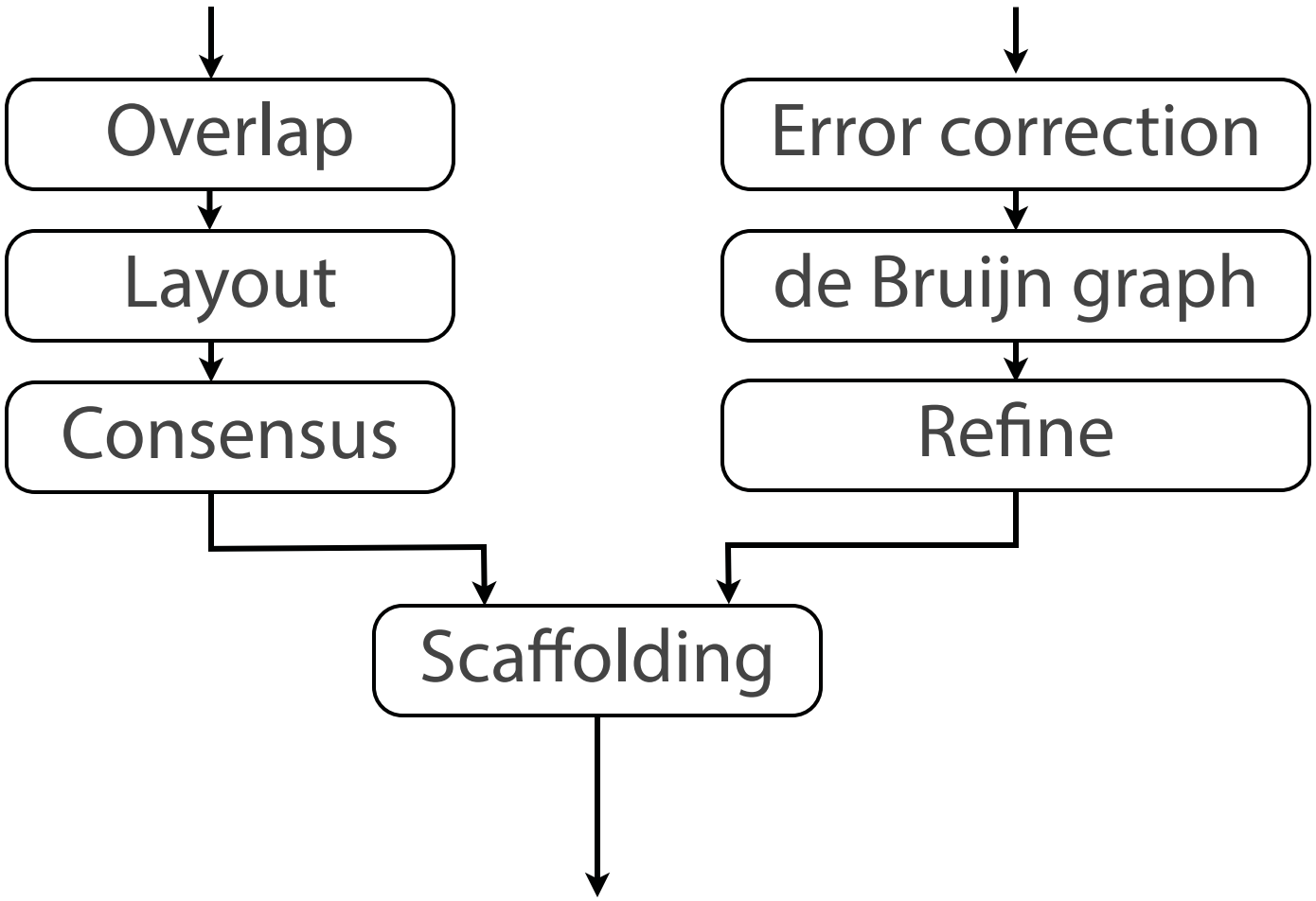
Figure 3. The amount of the human reference genome covered by a contig as a function of the minimum contig alignment length. For each length L on the *x*-axis, contig alignments less than L bp in length were filtered out and the amount of the reference genome covered by the remaining alignments was calculated.

Source: Simpson, Jared T., and Richard Durbin. "Efficient De Novo Assembly of Large Genomes using Compressed Data Structures." *Genome Research* 22, no. 3 (2012): 549-56.

Assembly alternatives

Alternative 1: Overlap-Layout-Consensus (OLC) assembly

Alternative 2: de Bruijn graph (DBG) assembly



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph assembly

A formulation conceptually similar to overlapping/SCS, but has some potentially helpful properties not shared by SCS.

http://www.langmead-lab.org/teaching-materials/

k-mer

"*k*-mer" is a substring of length *k*

S: GGCGATTCATCG

A 4-mer of *S*: ATTC

All 3-mers of *S*:

```
GGC
 GCG
  CGA
   GAT
    ATT
     TTC
      TCA
       CAT
        ATC
         TCG
```

mer: from Greek meaning "part"

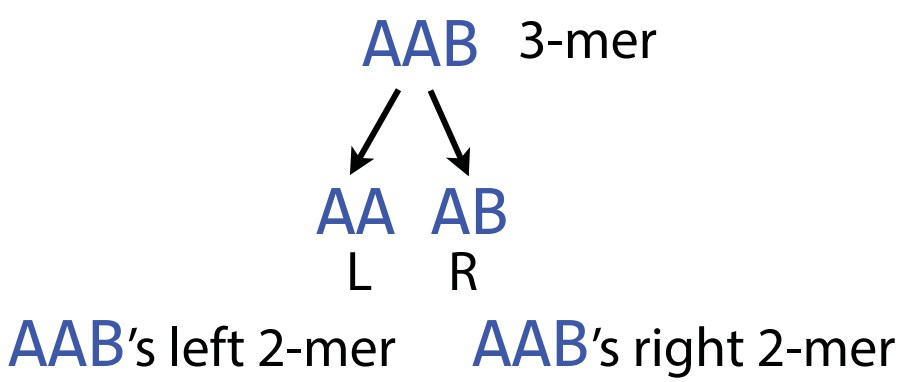
I'll use "*k*-1-mer" to refer to a substring of length *k* - 1

De Bruijn graph

As usual, we start with a collection of reads, which are substrings of the reference genome.

AAA, AAB, ABB, BBB, BBA

AAB is a *k*-mer ($k = 3$). AA is its *left* *k*-1-mer, and AB is its right *k*-1-mer.



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

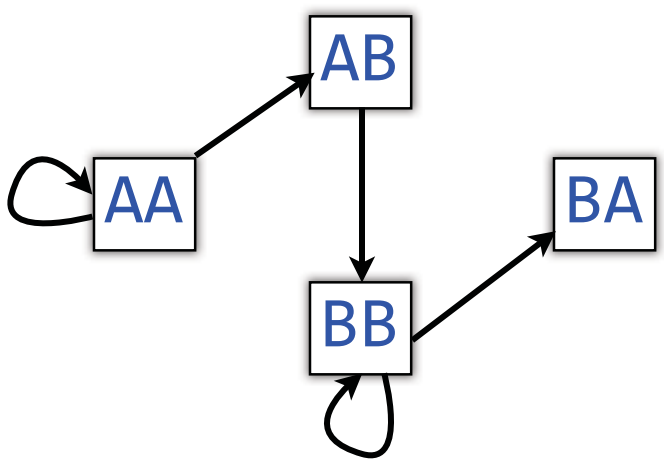
AAABBBA

take all 3-mers: AAA, AAB, ABB, BBB, BBA

form L/R 2-mers: AA, AA, AA, AB, AB, BB, BB, BB, BB, BA

L R L R L R L R L R

Let 2-mers be nodes in a new graph. Draw a directed edge from each left 2-mer to corresponding right 2-mer:

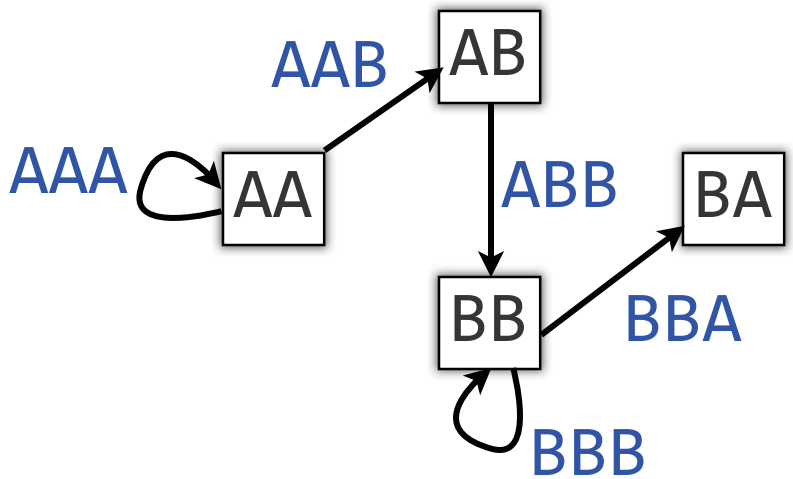


Each *edge* in this graph corresponds to a length-3 input string

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

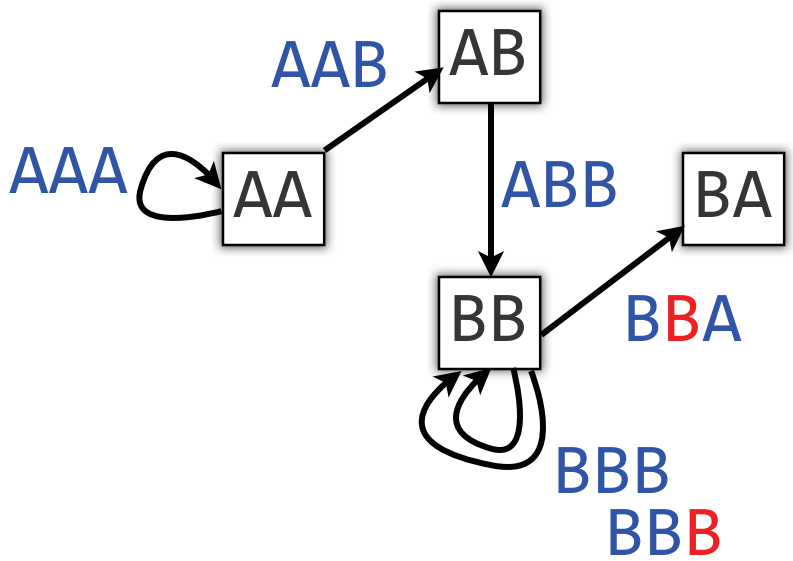


An edge corresponds to an overlap (of length *k*-2) between two *k*-1 mers.
More precisely, it corresponds to a *k*-mer from the input.

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph



If we add one more B to our input string: AAABBBBA, and rebuild the De Bruijn graph accordingly, we get a *multiedge*.

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

Eulerian walk definitions and statements

Node is *balanced* if indegree equals outdegree

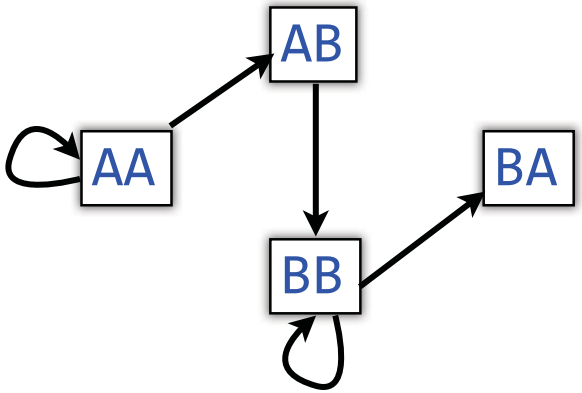
Node is *semi-balanced* if indegree differs from outdegree by 1

Graph is *connected* if each node can be reached by some other node

Eulerian walk visits each edge exactly once

Not all graphs have Eulerian walks. Graphs that do are *Eulerian*.
(For simplicity, we won't distinguish Eulerian from semi-Eulerian.)

A directed, connected graph is Eulerian if and only if it has at most 2 semi-balanced nodes and all other nodes are balanced

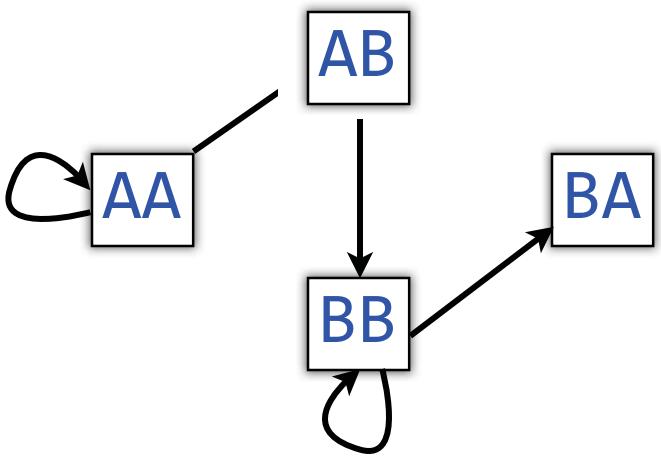


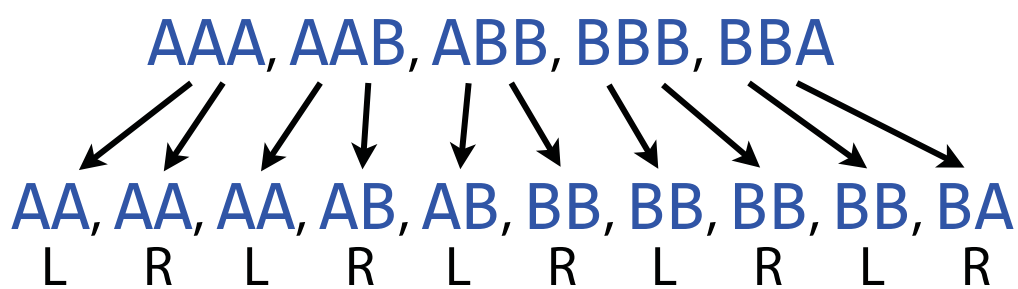
Jones and Pevzner section 8.8

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

Back to our De Bruijn graph





Is it Eulerian? Yes

Argument 1: AA → AA → AB → BB → BB → BA

Argument 2: AA and BA are semi-balanced, AB and BB are balanced

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

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

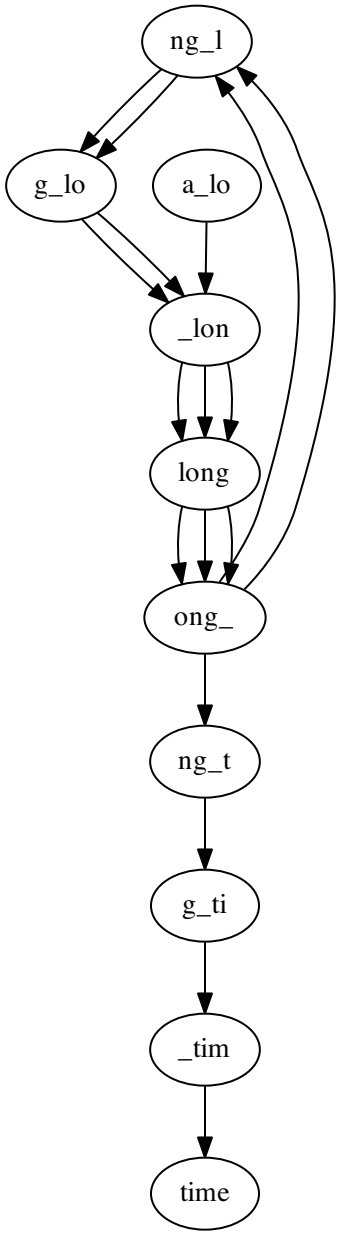
Start with each read: `a_long_long_long_time`

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

`long_`

`long` `ong_`

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

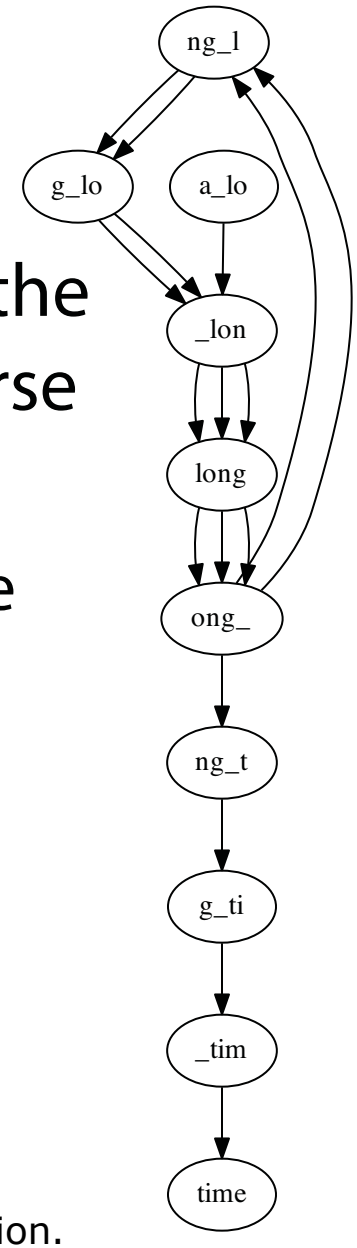


Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

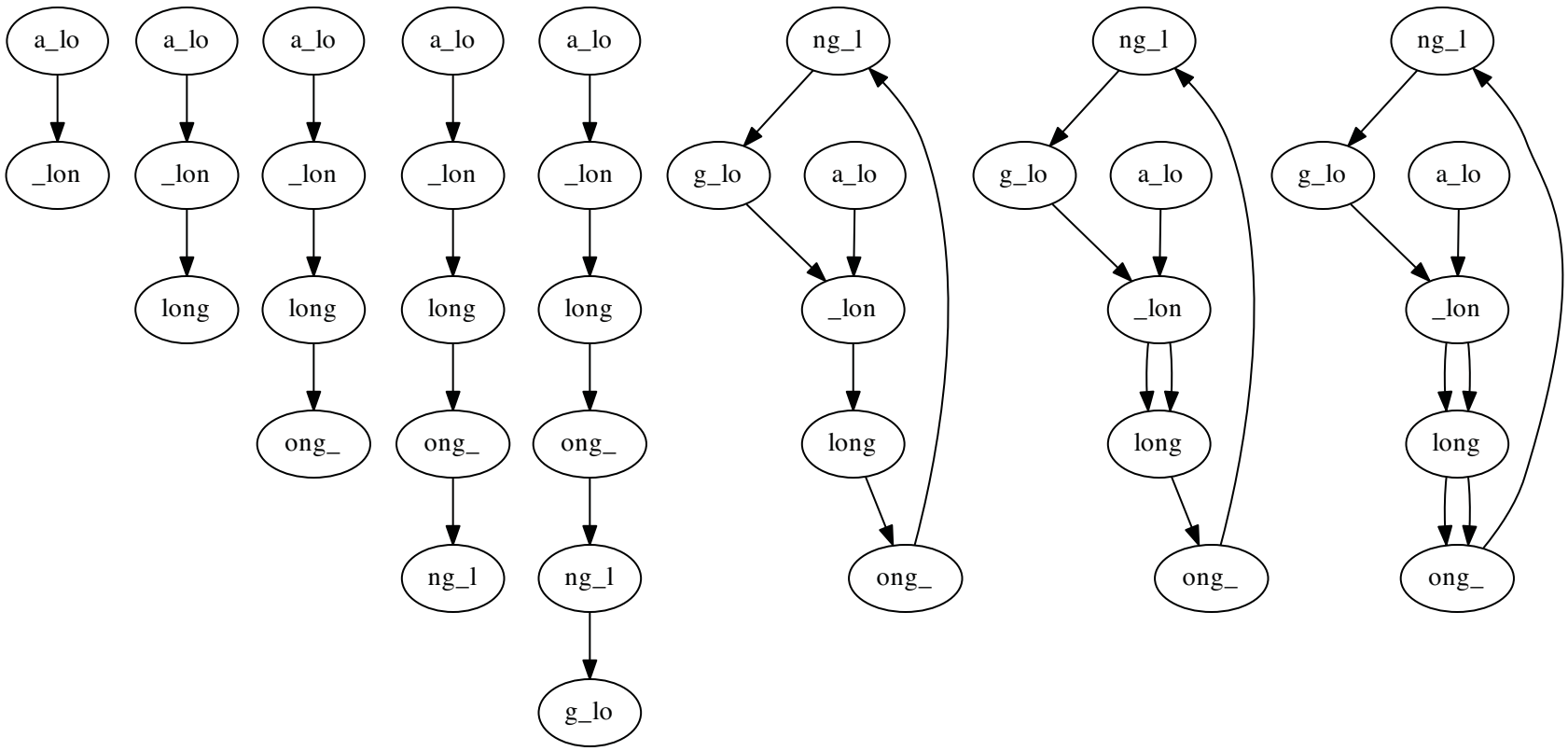
- For genome assembly each k-mer is recorded in "twin" nodes – one node in the forward direction and one node in reverse complement
  - k is odd so no node can be its own reverse complement
- We will not show reverse complement twin nodes to cut down on clutter



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph



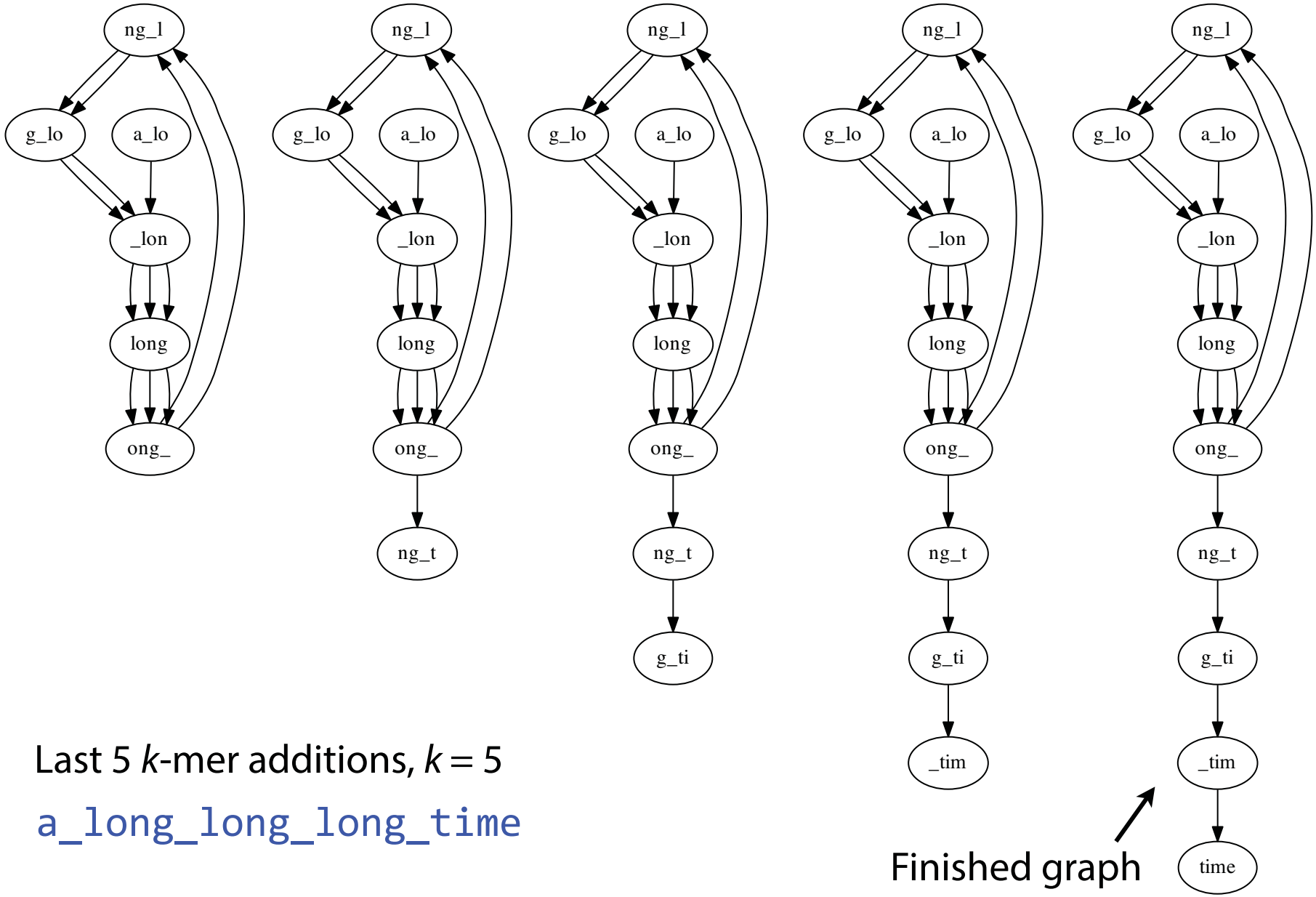
First 8 *k*-mer additions, $k = 5$

`a_long_long_long_time`

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph



Last 5 *k*-mer additions, $k = 5$

`a_long_long_long_time`

Finished graph

Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

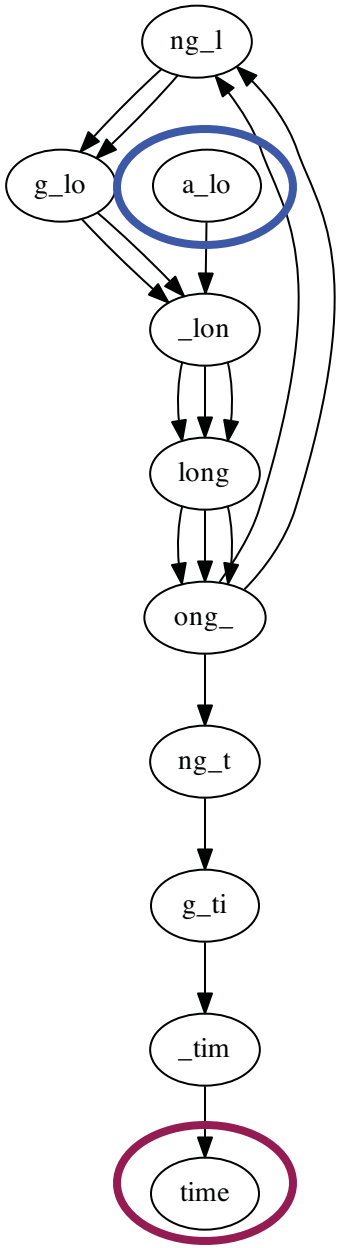
De Bruijn graph

With perfect sequencing, this procedure always yields an Eulerian graph. Why?

Node for *k*-1-mer from left end is semi-balanced with one more outgoing edge than incoming *

Node for *k*-1-mer at right end is semi-balanced with one more incoming than outgoing *

Other nodes are balanced since # times *k*-1-mer occurs as a left *k*-1-mer = # times it occurs as a right *k*-1-mer



* Unless genome is circular

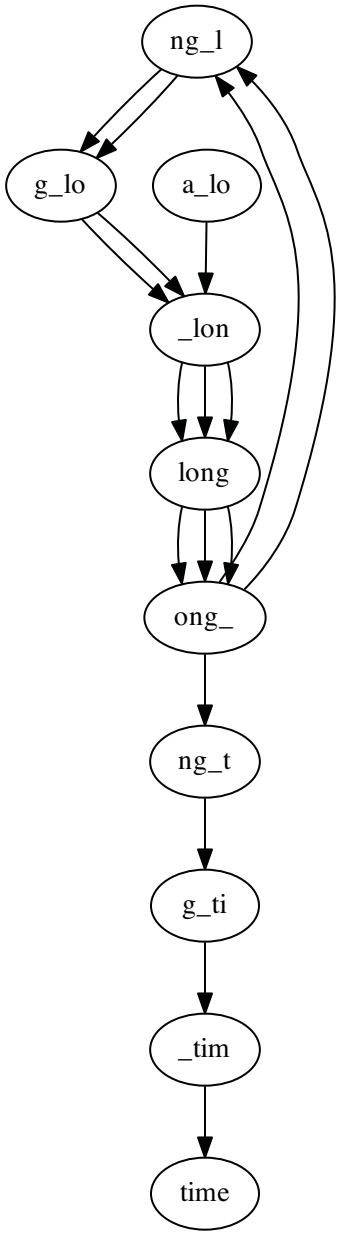
Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

Assuming perfect sequencing, procedure yields graph with Eulerian walk that can be found efficiently.

We saw cases where Eulerian walk corresponds to the original superstring. Is this always the case?



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

No: graph can have multiple Eulerian walks, only one of which corresponds to original superstring

Right: graph for ZABCDABEFABY, $k = 3$

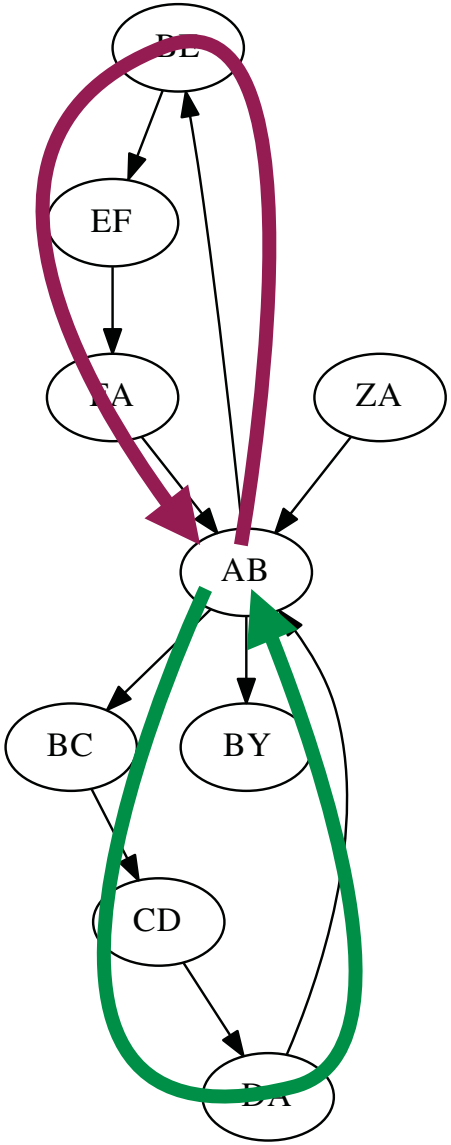
Alternative Eulerian walks:

ZA → AB → BE → EF → FA → AB → BC → CD → DA → AB → BY

ZA → AB → BC → CD → DA → AB → BE → EF → FA → AB → BY

These correspond to two edge-disjoint directed cycles joined by node AB

AB is a repeat: ZABCDABEFABY



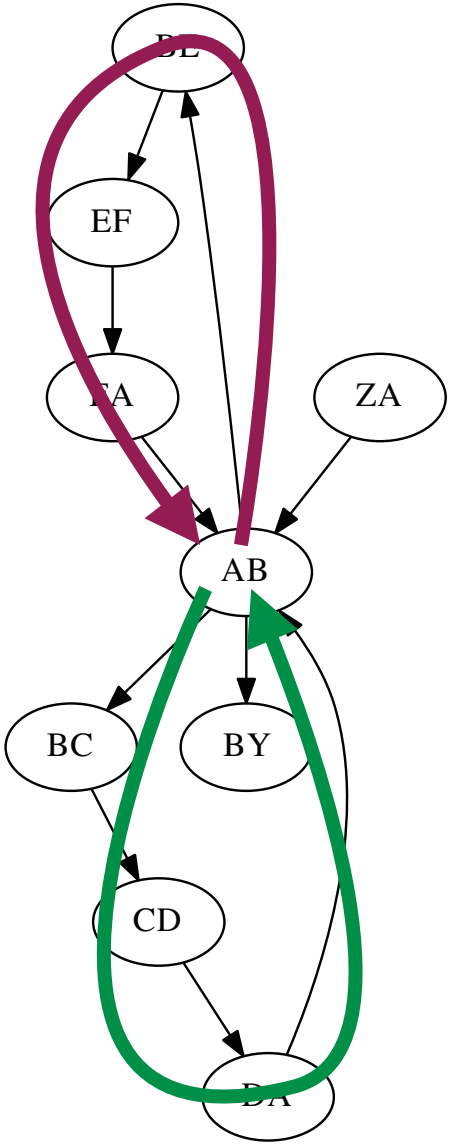
Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

This is the first sign that Eulerian walks can't solve all our problems

Other signs emerge when we think about how actual sequencing differs from our idealized construction



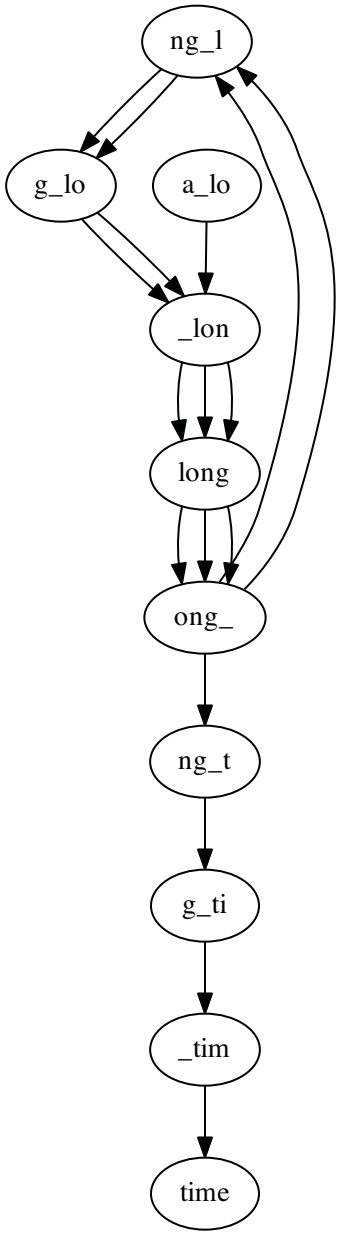
Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

Gaps in coverage can lead to *disconnected* graph

Graph for `a_long_long_long_time`, $k = 5$:



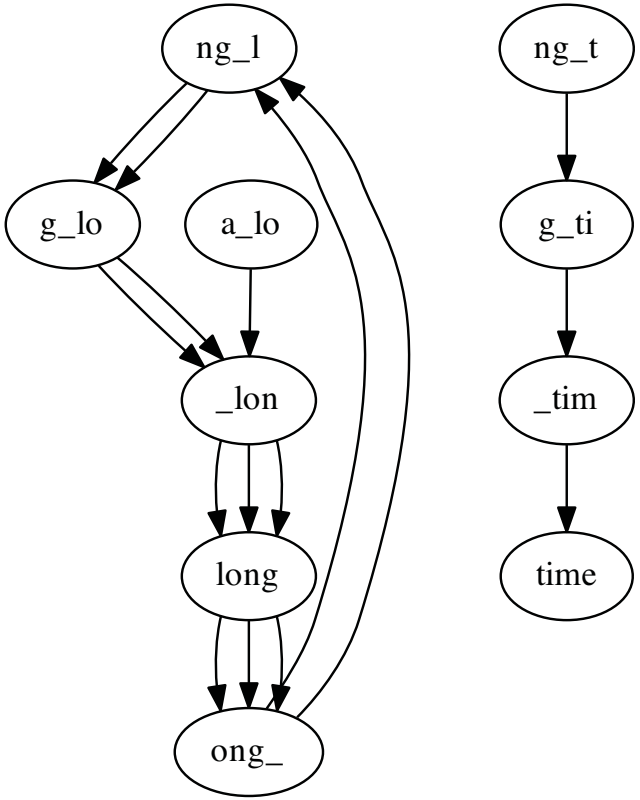
Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

Gaps in coverage can lead to *disconnected* graph

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



Connected components are individually Eulerian, overall graph is not

Courtesy of Ben Langmead. Used with permission.

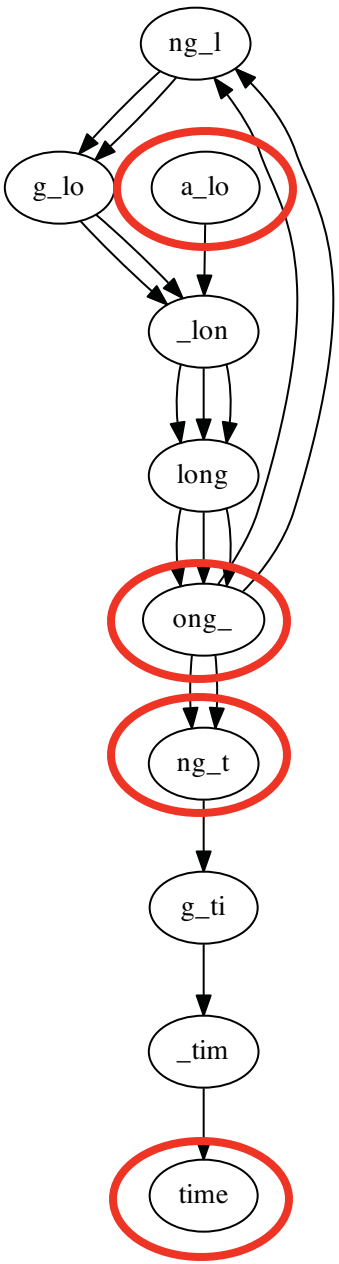
http://www.langmead-lab.org/teaching-materials/

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



Courtesy of Ben Langmead. Used with permission.

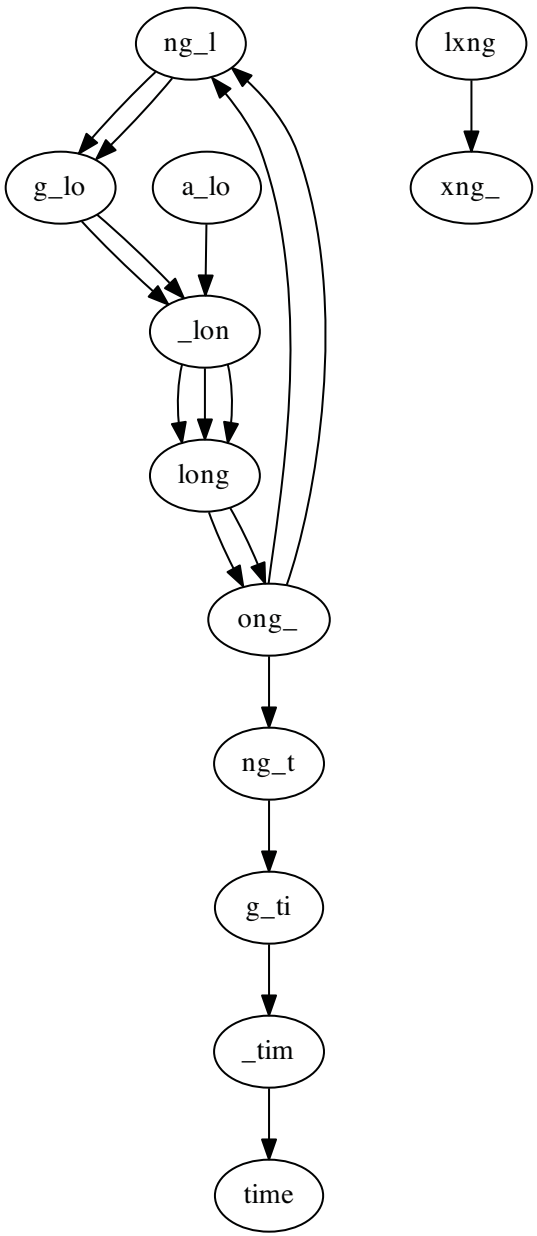
http://www.langmead-lab.org/teaching-materials/

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_`

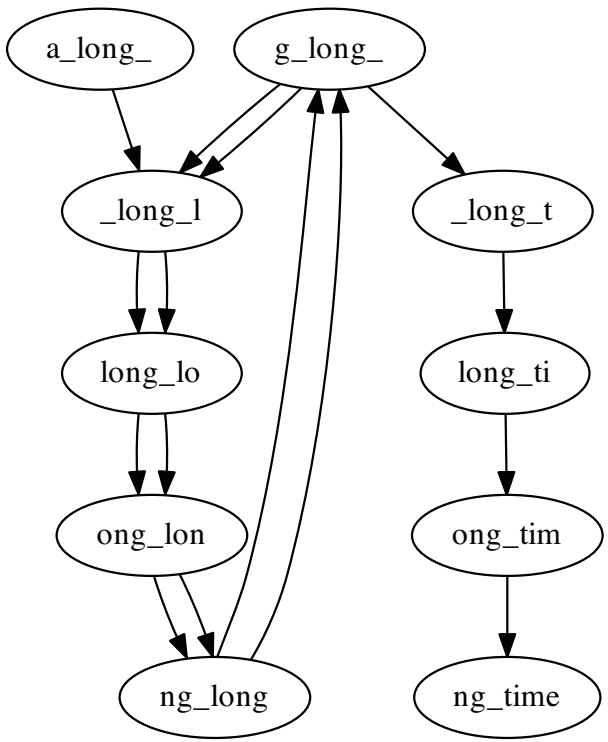
Graph is not connected; largest component is not Eulerian



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph



How much work to build graph?

For each *k*-mer, add 1 edge and up to 2 nodes

Reasonable to say this is O(1) expected work

Assume hash map encodes nodes & edges

Assume *k*-1-mers fit in O(1) machine words, and hashing O(1) machine words is O(1) work

Querying / adding a key is O(1) expected work

O(1) expected work for 1 *k*-mer, O(*N*) overall

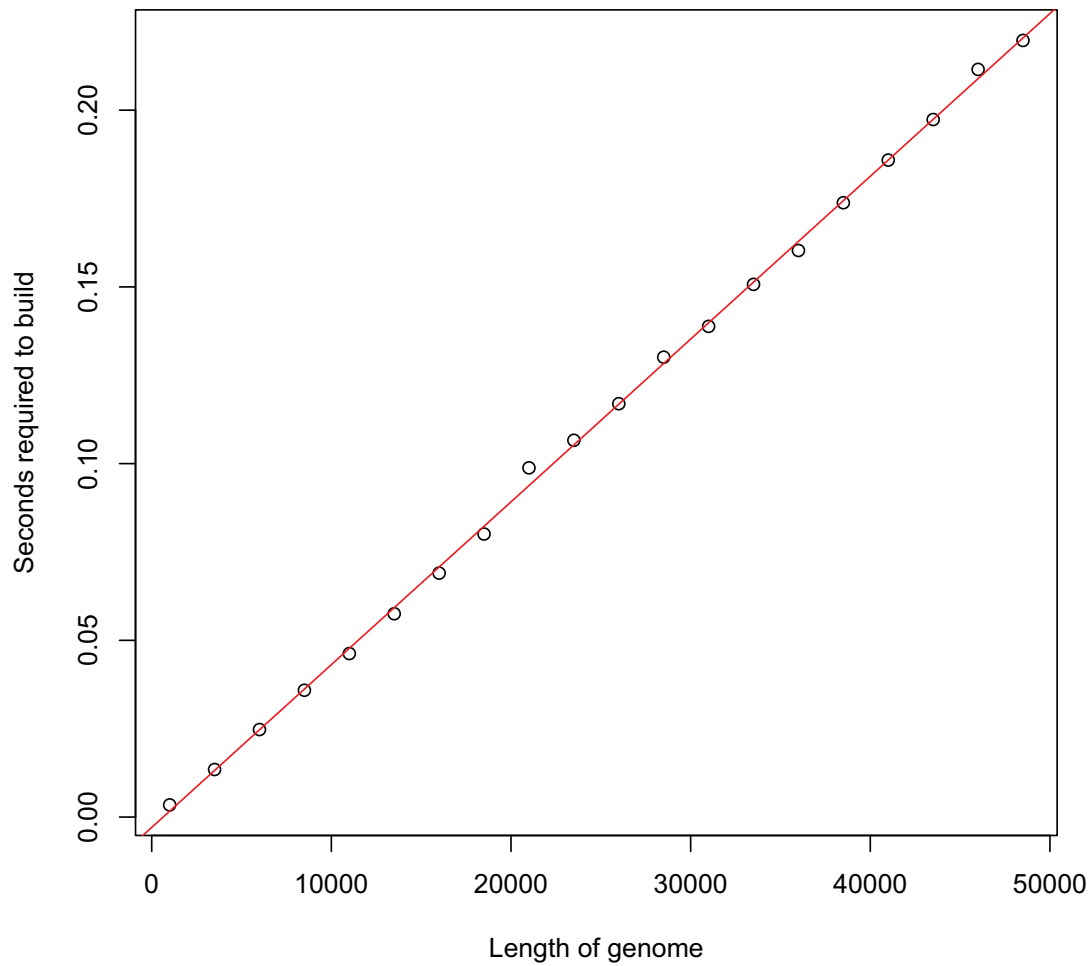
Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

Timed De Bruijn graph construction applied to progressively longer prefixes of lambda phage genome, $k = 14$

O(N) expectation appears to work in practice, at least for this small example

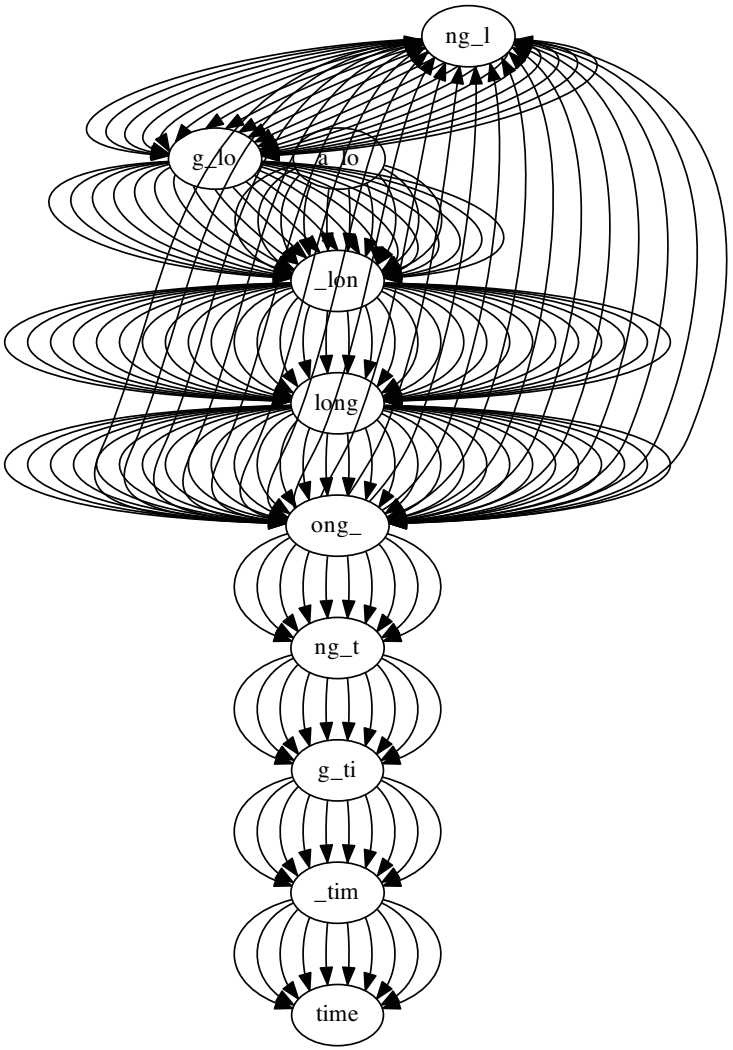


Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

De Bruijn graph

In typical assembly projects,
average coverage is ~ 30 - 50



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

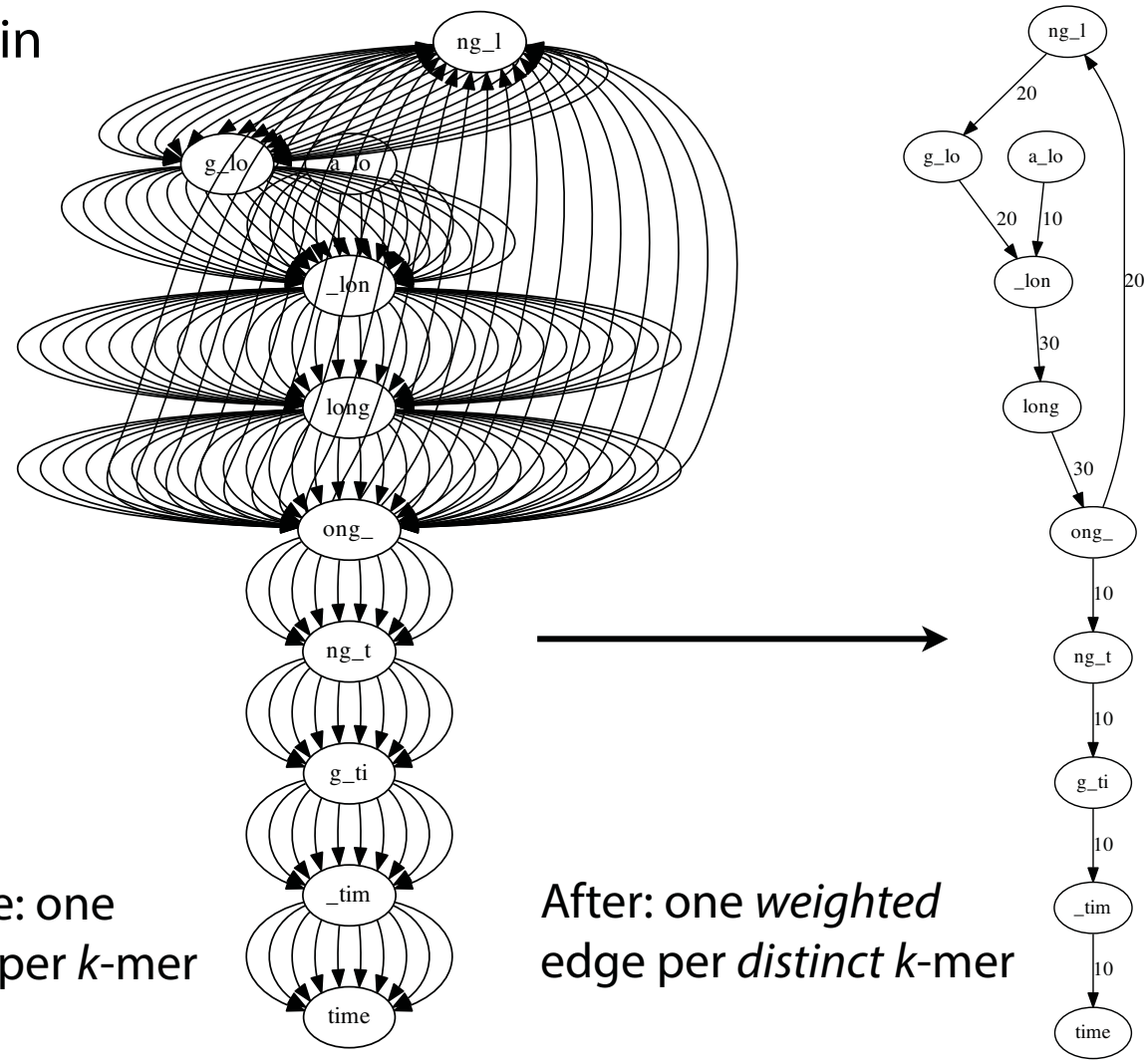
De Bruijn graph

In typical assembly projects, average coverage is ~ 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

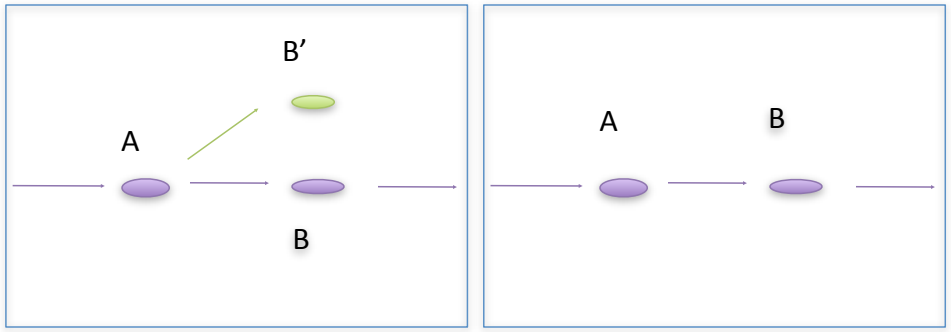
After: one *weighted* edge per *distinct k*-mer

Courtesy of Ben Langmead. Used with permission.

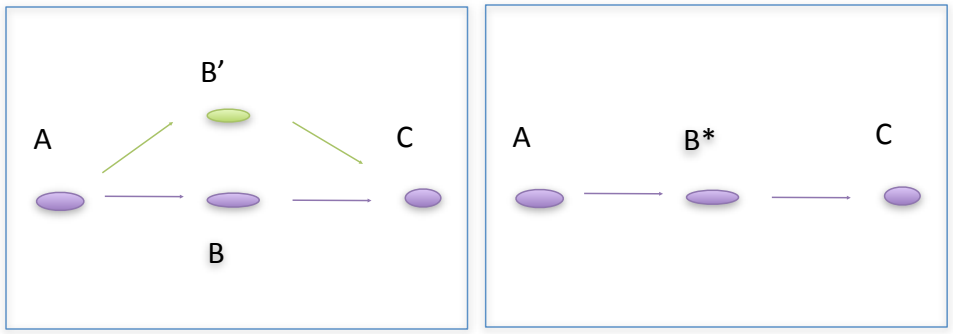
http://www.langmead-lab.org/teaching-materials/

Graph topology based error eorrection

– Errors at end of read
  - Trim off ‘dead-end’ tips



– Errors in middle of read
  - Pop Bubbles



– Chimeric Edges
  - Clip short, low coverage nodes

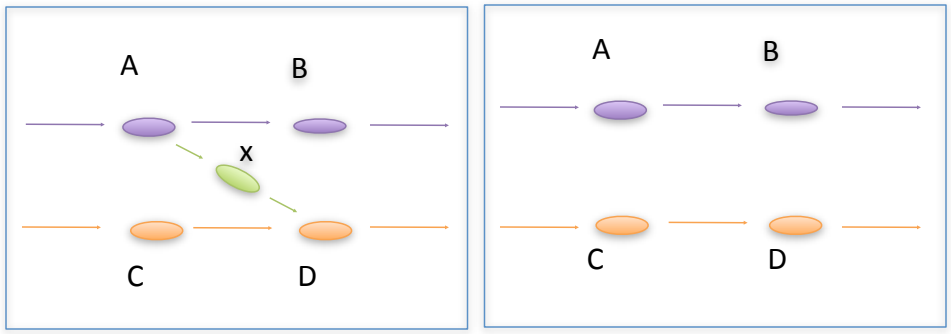


Figure adapted from presentation by Michael Schatz

De Bruijn graph

What are the limitations of De Bruijn graphs?

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. Need to thread reads though De Bruijn graph to recover information lost when reads are fragmented into k-mers.

This is the OLC $\longleftrightarrow$ DBG tradeoff

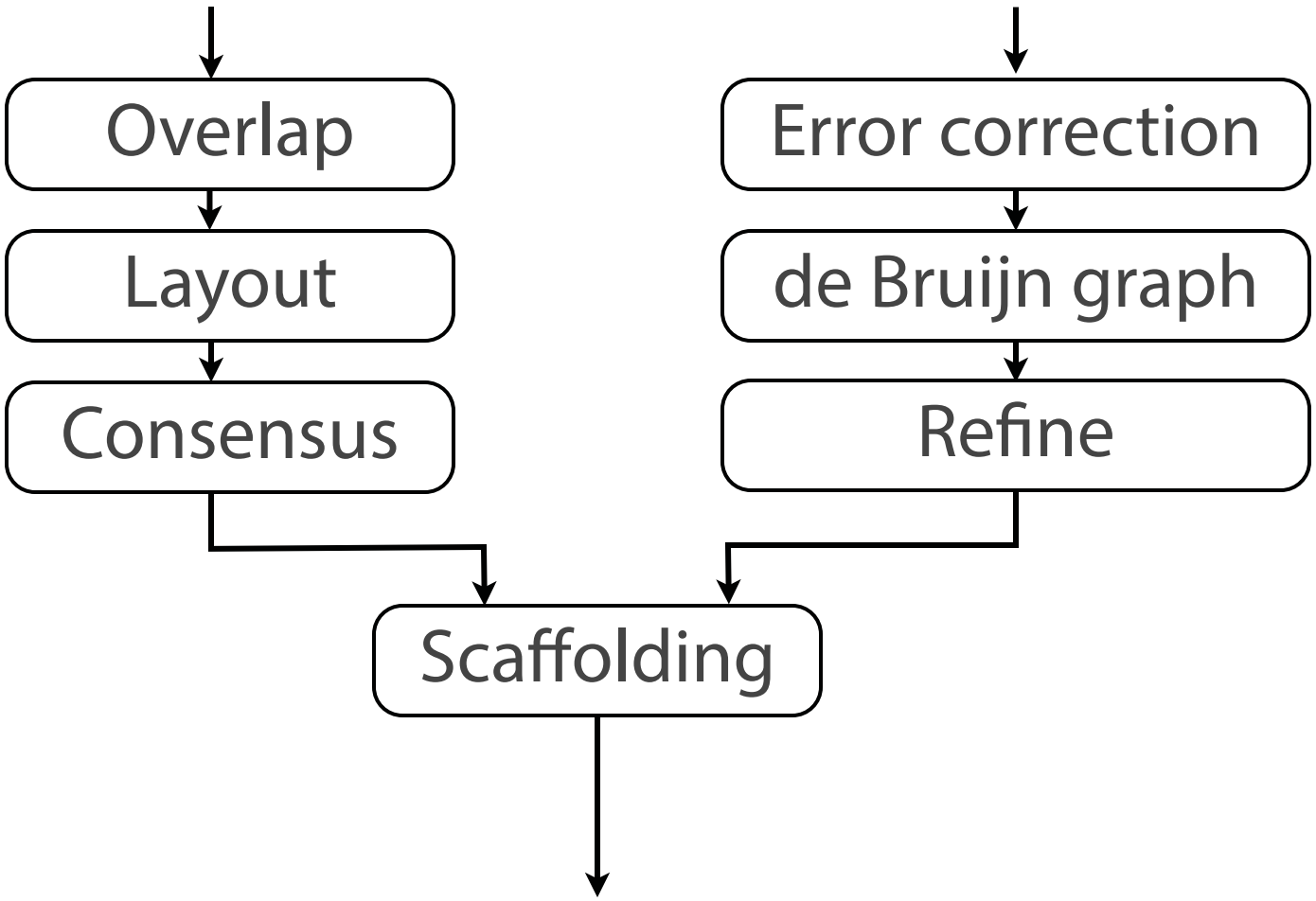
Single most important benefit of De Bruijn graph is speed and simplicity.

http://www.langmead-lab.org/teaching-materials/

Assembly alternatives

Alternative 1: Overlap-Layout-Consensus (OLC) assembly

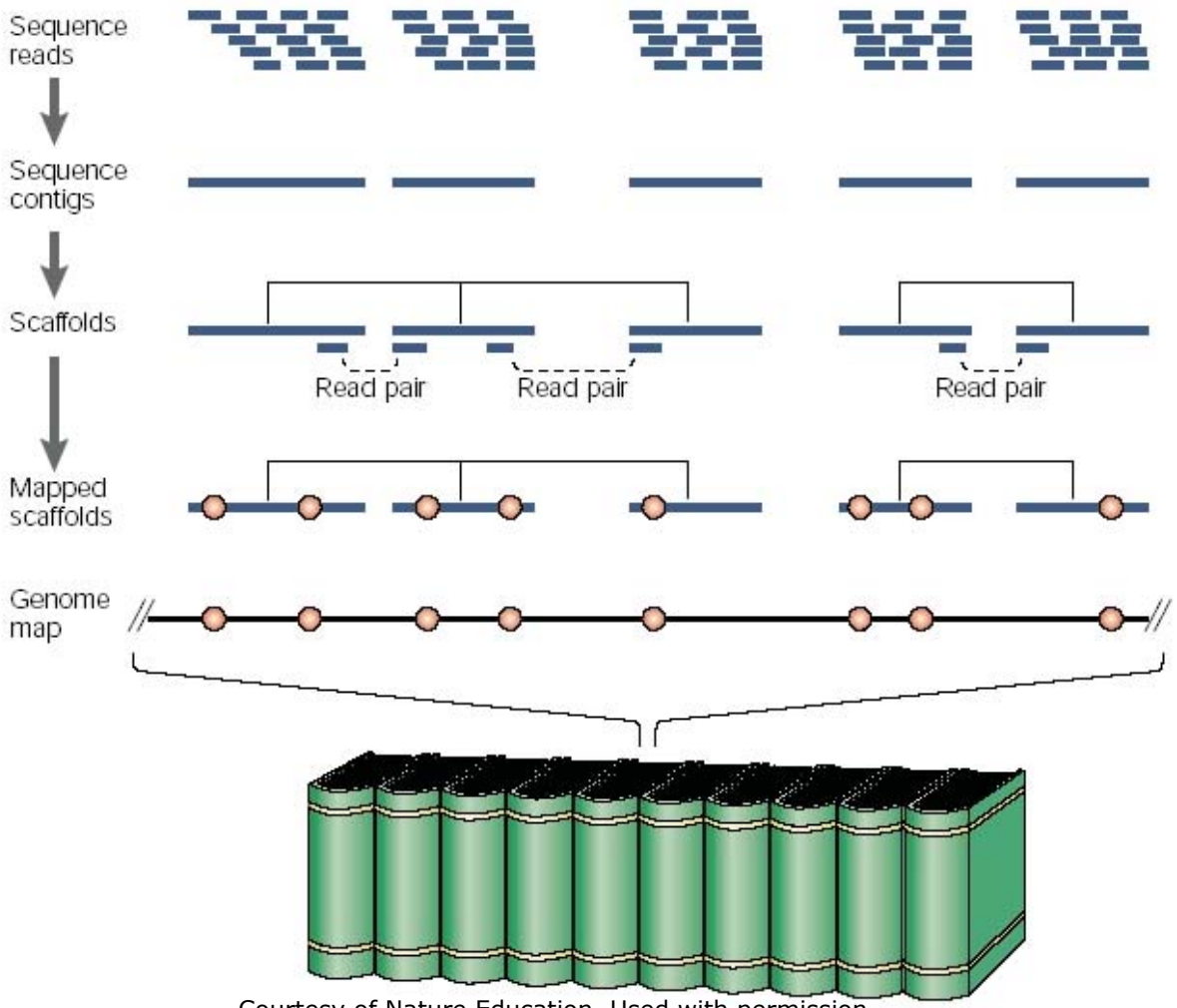
Alternative 2: de Bruijn graph (DBG) assembly



Courtesy of Ben Langmead. Used with permission.

http://www.langmead-lab.org/teaching-materials/

de novo whole-genome shotgun assembly



Source: Green, Eric D. "Strategies for the Systematic Sequencing of Complex Genomes." *Nature Reviews Genetics* 2, no. 8 (2001): 573-83.

Adams, J. (2008) Complex genomes: Shotgun sequencing. Nature Education 1(1)

N50 - contig/scaffold length or larger that contains 50% of bases

	OLC t >= 75	De Bruijn k = 61	De Bruijn k = 67	De Bruijn k = 59

Table 1. Assembly statistics for *C. elegans* data set

	SGA	Velvet	ABySS	SOAPdenovo
Scaffold N50 size	26.3 kbp	31.3 kbp	23.8 kbp	31.1 kbp
Aligned contig N50 size	16.8 kbp	13.6 kbp	18.4 kbp	16.0 kbp
Mean aligned contig size	4.9 kbp	5.3 kbp	6.0 kbp	5.6 kbp
Sum aligned contig size	96.8 Mbp	95.2 Mbp	98.3 Mbp	95.4 Mbp
Reference bases covered	96.2 Mbp	94.8 Mbp	95.9 Mbp	95.1 Mbp
Reference bases covered by contigs ≥1 kb	93.0 Mbp	92.1 Mbp	93.9 Mbp	92.3 Mbp
Mismatch rate at all assembled bases	1 per 21,545 bp	1 per 8786 bp	1 per 5577 bp	1 per 26,585 bp
Mismatch rate at bases covered by all assemblies	1 per 82,573 bp	1 per 18,012 bp	1 per 8209 bp	1 per 81,025 bp
Contigs with split/bad alignment (sum size)	458 (4.4 Mbp)	787 (7.2 Mbp)	638 (9.1 Mbp)	483 (4.4 Mbp)
Total CPU time	41 h	2 h	5 h	13 h
Max memory usage	4.5 GB	23.0 GB	14.1 GB	38.8 GB

100 MBase genome, 33.8M read pairs, 100bp reads each end, 250bp insert size

Efficient de novo assembly of large genomes using compressed data structures
Jared T Simpson and Richard Durbin Genome Res. 2012. 22: 549-556

FIN

7.91J / 20.490J / 20.390J / 7.36J / 6.802J / 6.874J / HST.506J Foundations of Computational and Systems Biology
Spring 2014