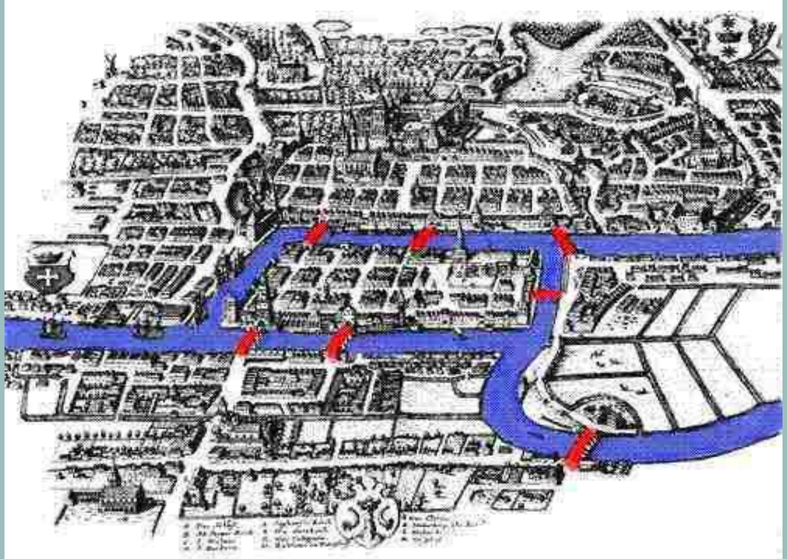


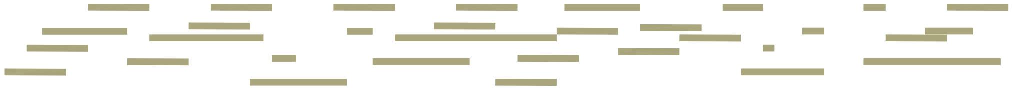
ALGORITHMS IN SEQUENCE ANALYSIS

De novo genome assembly

STRATEGIES TO SEQUENCE LONG DNA MOLECULES: SHOTGUN SEQUENCING

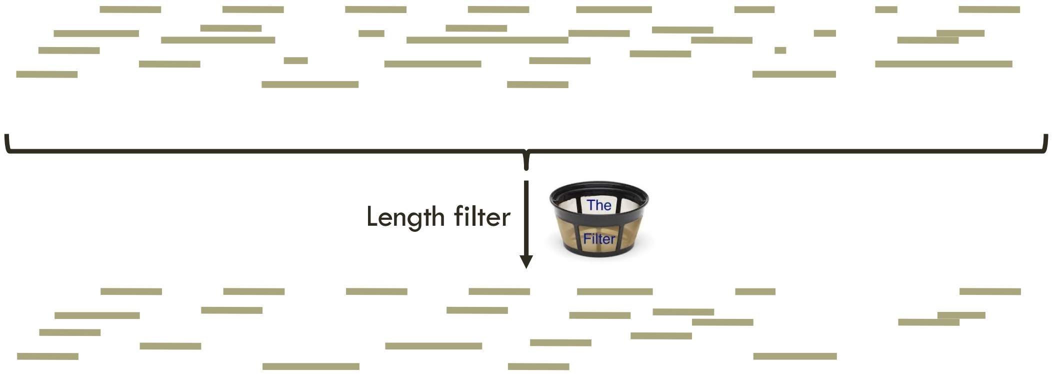
1. Processing of the template DNA
    1. Random fragmentation

STRATEGIES TO SEQUENCE LONG DNA MOLECULES: SHOTGUN SEQUENCING



1. Processing of the template DNA
    1. Random fragmentation

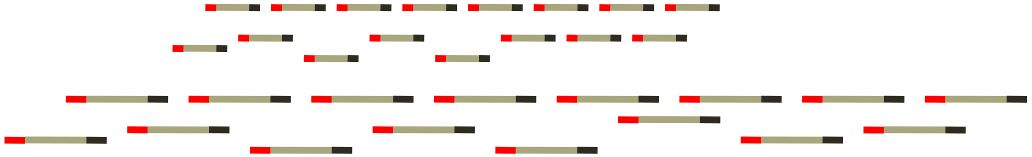
STRATEGIES TO SEQUENCE LONG DNA MOLECULES: SHOTGUN SEQUENCING



1. Processing of the template DNA
    1. Random fragmentation
    2. Size selection (-> Insert-size[1])

[1] typically several 100 Bp for ‚short-read-technologies (e.g.. Illumina)

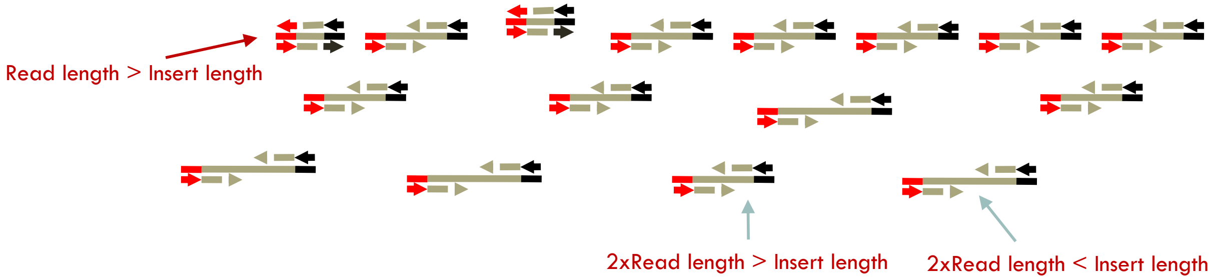
STRATEGIES TO SEQUENCE LONG DNA MOLECULES: SHOTGUN SEQUENCING



1. Processing of the template DNA
    1. Random fragmentation
    2. Size selection (-> Insert-size)
2. Append adapters[1] (DNA fragements with known sequence) that provide the necessary binding sites for downstream wet lab experiments (amplification, sequencing), as well as index sequences

[1] each fragment gets the same set of adaptors

STRATEGIES TO SEQUENCE LONG DNA MOLECULES: SHOTGUN SEQUENCING

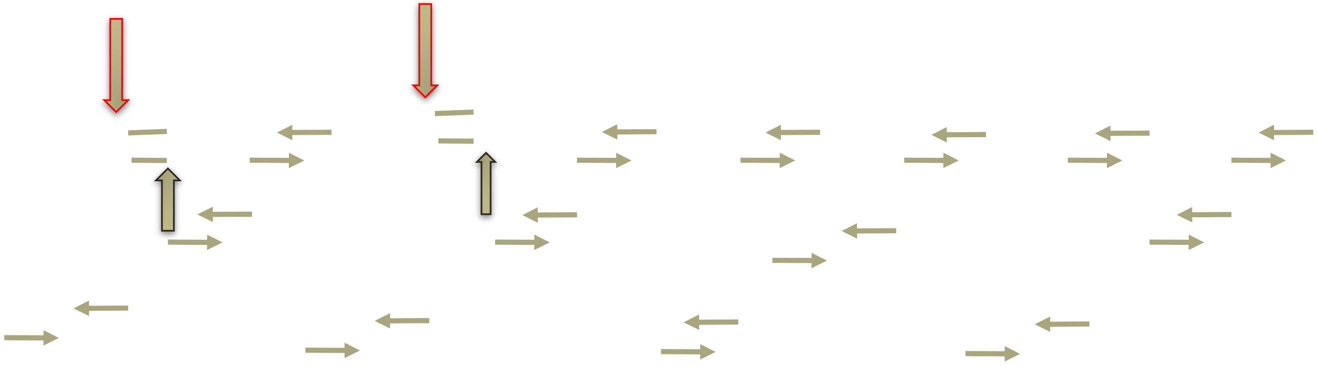

Read length > Insert length
2xRead length > Insert length
2xRead length < Insert length

1. Processing of the template DNA
   1. Random fragmentation
   2. Size selection (-> Insert-size)
2. Append adapters[1] (DNA fragements with known sequence) that provide the necessary binding sites for downstream wet lab experiments (amplification, sequencing), as well as index sequences
3. Sequence the insert ends

[1] typically, we sequence both ends of the insert -> Paired-End Reads

[2] if read length > Insert length, you will seqeunce into the adapter

STRATEGIES TO SEQUENCE LONG DNA MOLECULES: SHOTGUN SEQUENCING SOMETIMES ADAPTER SEQUENCES REMAIN!

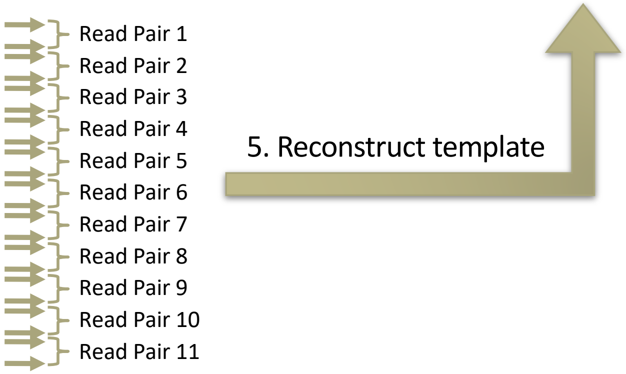


1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments

Identifying these sequences is simple when we ignore the complexity of the search

Clip adapter sequences

STRATEGIES TO SEQUENCE LONG DNA MOLECULES: SHOTGUN SEQUENCING



Read Pair 1

Read Pair 2

Read Pair 3

Read Pair 4

Read Pair 5

Read Pair 6

Read Pair 7

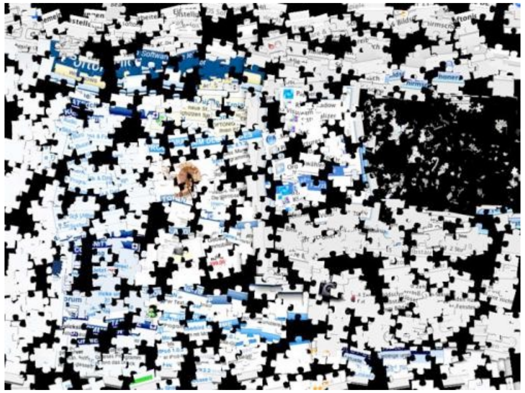
Read Pair 8

Read Pair 9

Read Pair 10

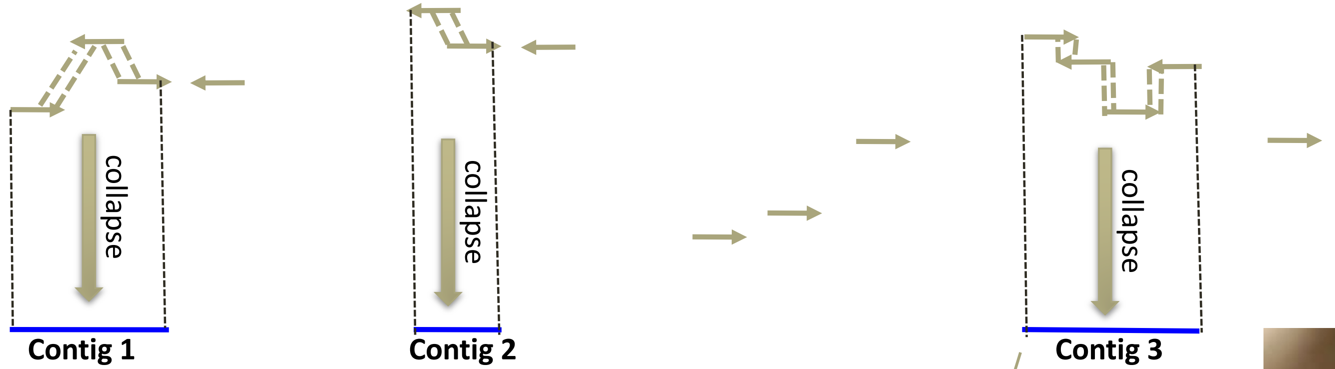
Read Pair 11

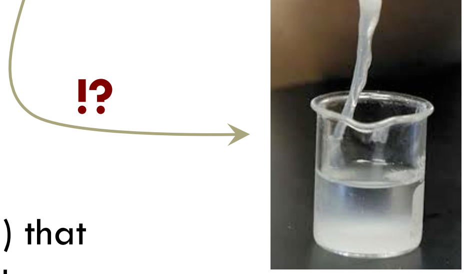
5. Reconstruct template



1. Processing of the template DNA
    1. Random fragmentation
    2. Size selection (-> Insert-size)
2. Append adapters[1] (DNA fragements with known sequence) that provide the necessary binding sites for downstream wet lab experiments (amplification, sequencing), as well as index sequences
3. Sequence the insert ends
4. Identify and remove adapters from the sequence reads

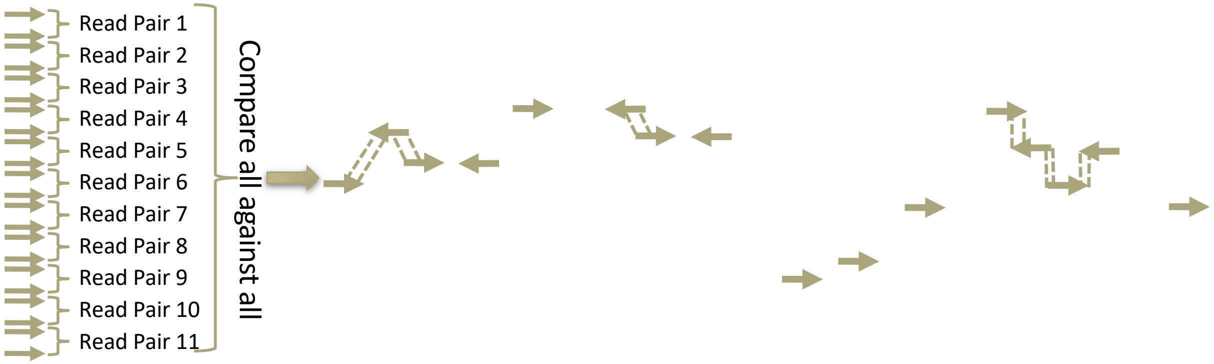
STRATEGIES TO SEQUENCE LONG DNA MOLECULES: **SEQUENCE ASSEMBLY**





1. Processing of the template DNA
   1. Random fragmentation
   2. Size selection (-> Insert-size)
2. Append adapters[1] (DNA fragements with known sequence) that provide the necessary binding sites for downstream wet lab experiments (amplification, sequencing), as well as index sequences
3. Sequence the insert ends
4. Identify and remove adapters from the sequence reads
5. Template reconstruction: (i) **de novo;** (ii) Reference sequence guided

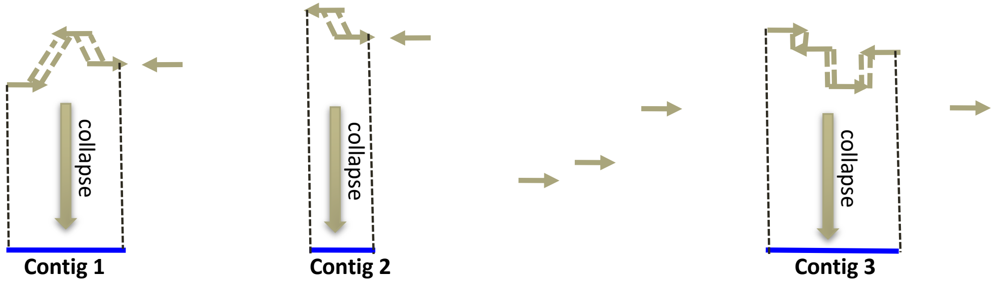
CLEANING UP THE MESS: CONTIG BUILDING[1] DURING DE-NOVO ASSEMBLY



2. …
3. Sequence the insert ends
4. Identify and remove adapters from the sequence reads
5. De-novo sequence assembly: determine overlap between sequence reads and assemble overlapping sequences into contigs.

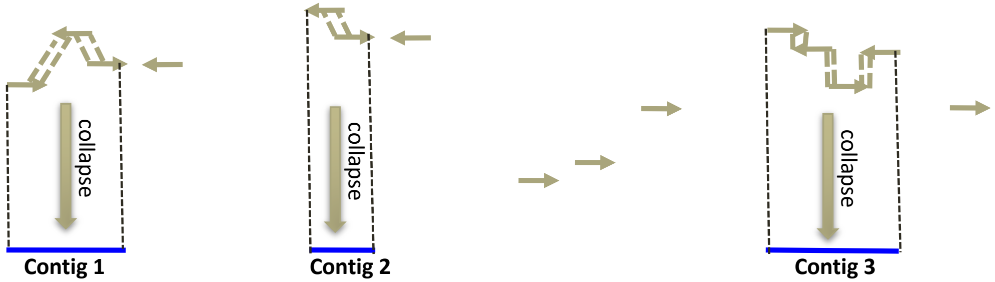
[1] only the general concept. Algorithmic solutions typically take a different approach as we will see later

CONTIG BUILDING



2. …
3. Sequence the insert ends
4. Identify and remove adapters from the sequence reads
5. De-novo sequence assembly: determine overlap between sequence reads and assemble overlapping sequences into contigs.

CONTIG BUILDING

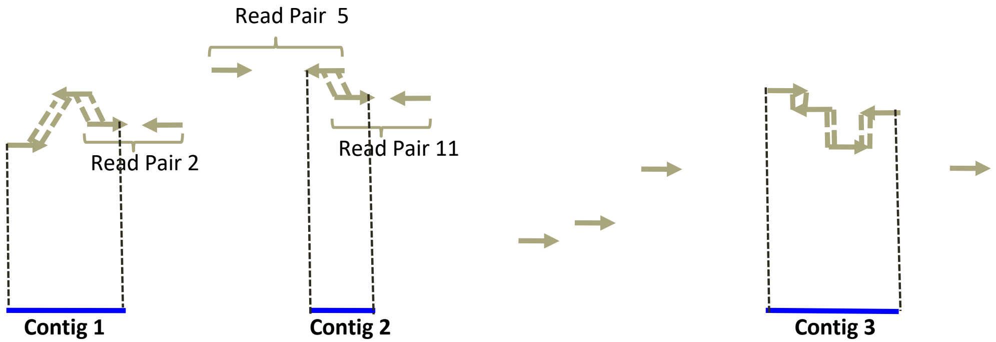


Contig[1]: *In order to make it easier to talk about our data gained by the shotgun method of sequencing we have invented the word "contig". A contig is a set of reads*[2] *that are related to one another by overlap of their sequences. All reads belong to one and only one contig, and each contig contains at least one read.*
The reads in a contig can be summed to form a contiguous consensus sequence and the length of this sequence is the length of the contig." (Comment by IE: The contig length resembles, in theory, the length of the corresponding DNA molecule)

[1] Definition from Staden, R (1980). Nucleic Acids Research. 8 (16): 3673–3694

[2] In the original text, Staden refers to ‚gel reading', a term no longer in use

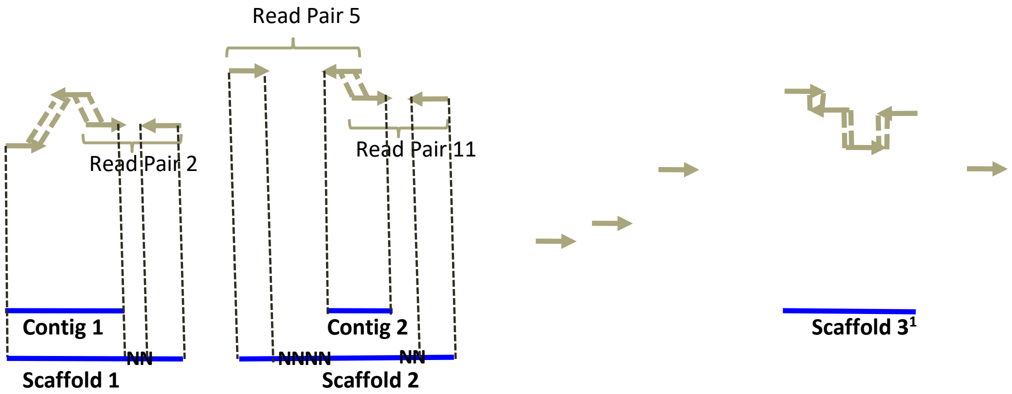
SCAFFOLDING – THE USE OF READ PAIRS FOR CONNECTING NON-OVERLAPPING FRAGMENTS



2. …
3. Sequence the insert ends
4. Identify and remove adapters from the sequence reads
5. De-novo sequence assembly: determine overlap between sequence reads and assemble overlapping sequences into contigs. **Read pair information** can then be used to build **scaffolds**[1] from physically non-overlapping contigs.

[1] scaffolds are sometimes also called ‚super-contigs‘

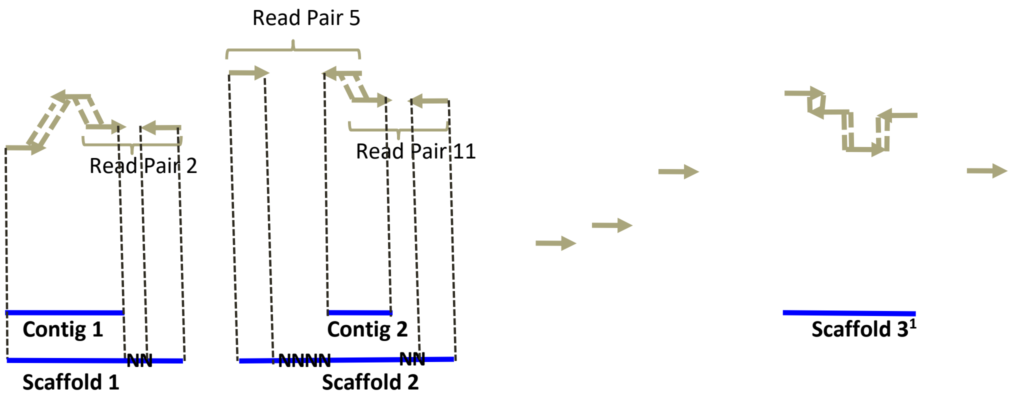
DE-NOVO ASSEMBLY – SCAFFOLDING



2. …
3. Sequence the insert ends
4. Identify and remove adapters from the sequence reads
5. De-novo sequence assembly: determine overlap between sequence reads and assemble overlapping sequences into contigs. **Read pair information** can then be used to build **scaffolds**[1] from physically non-overlapping contigs.

[1] A scaffold can consist only of a single contig

DE-NOVO ASSEMBLY – SCAFFOLDING

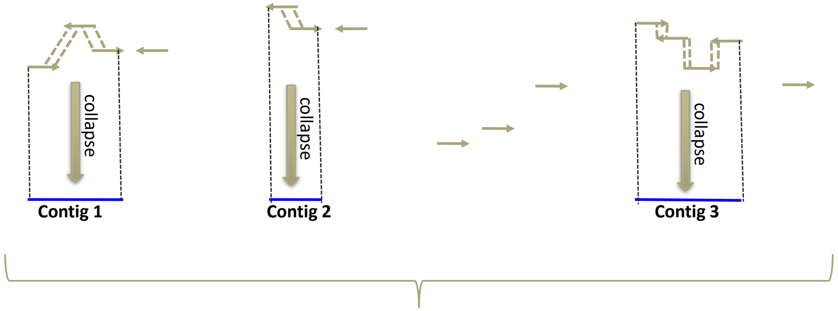


Scaffold: *A scaffold consists of ordered and oriented – but typically non-overlapping – contigs separated by gaps of approximately known length. Scaffolds are typically formed by identifying contig pairs that each contain one read of a ‚read pair'[1]. The contigs in a scaffold are combined to form a contiguous consensus sequence, and the length of this sequence is the length of the scaffold[2].*

[1] Note, subsequent to a scaffolding step, we typically refer to all consensus sequences as scaffolds, even if they consist of only one contig.

[2] The scaffold length is typically shorter than the corresponding DNA molecule, because the run of Ns between two contigs is limited

SUMMARY STATISTICS TO DESCRIBE ASSEMBLIES – READ COVERAGE



1. **Coverage:** The average number of reads covering a position in the sequenced template DNA.

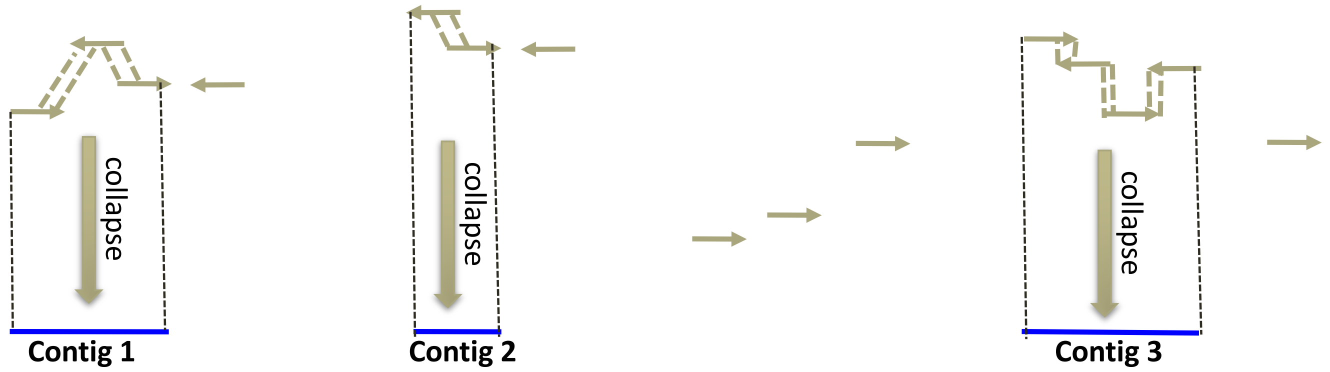
Length of genomic segment: L

Number of reads: n

Length of each read: l

Coverage C = n l / L

SUMMARY STATISTICS TO DESCRIBE ASSEMBLIES – READ COVERAGE



1. **Coverage:** The average number of reads covering a position in the sequenced template DNA.

Length of genomic segment: L
Number of reads: n
Length of each read: l

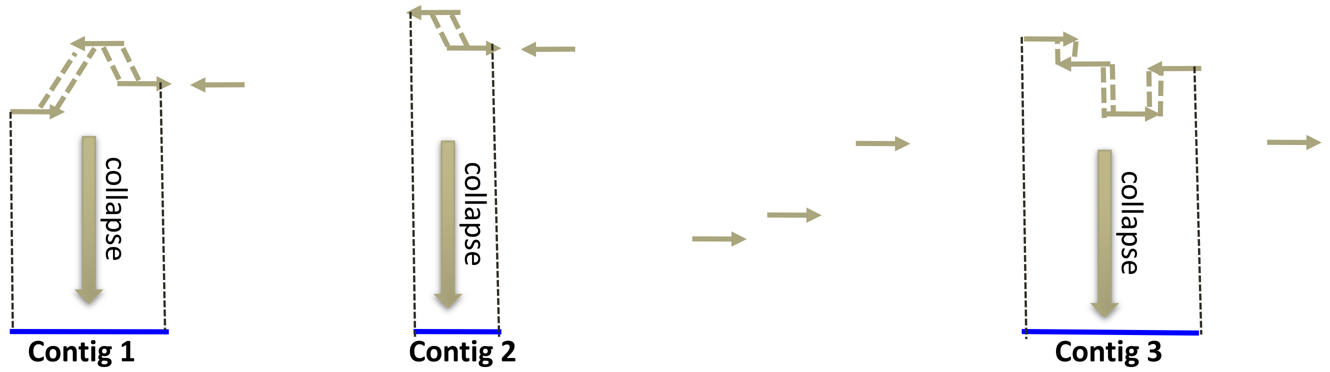
Coverage C = n l / L

How much coverage is enough?

Lander-Waterman model:

Assuming uniform distribution of reads, C=10 results in 1 gapped region per 1,000,000 nucleotides -> This is no more than a crude rule of thumb and greatly depends on read length, repeat composition of the template DNA, etc.

SUMMARY STATISTICS TO DESCRIBE ASSEMBLIES – READ COVERAGE



1. **Coverage:** The average number of reads covering a position in the sequenced template DNA.

Length of genomic segment: L

Number of reads: n

Length of each read: l

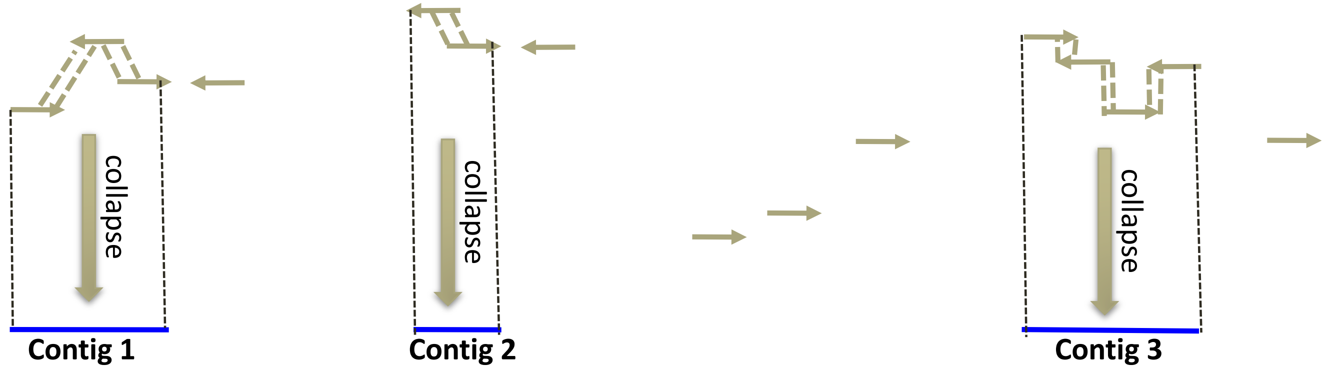
Coverage $C = n\,l / L$

The higher the coverage the better, provided unlimited computational resources[1]!

The more uniform the coverage distribution the better!

[1] watch out, some assemblers have a hard-coded upper limit of the allowed coverage. Anything above this limit will be treated as repeat…

SUMMARY STATISTICS TO DESCRIBE ASSEMBLIES – N50 SIZE



2. **N50-size:** More than 50% of the bases in your assembly reside in contigs/scaffolds with **at least** the size determined by the N50 value. **NOTE:** You can of course specify any other *N-value*

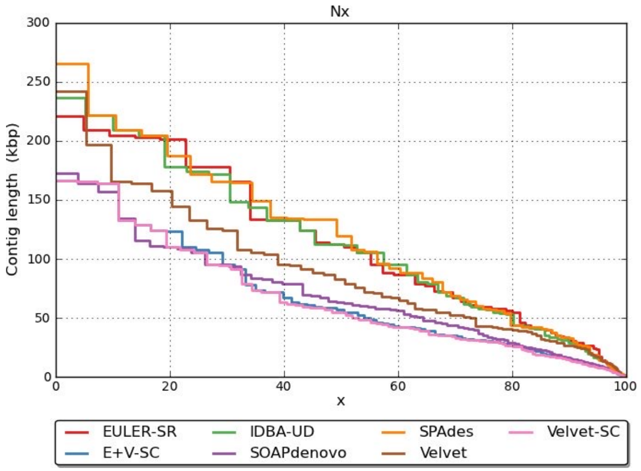
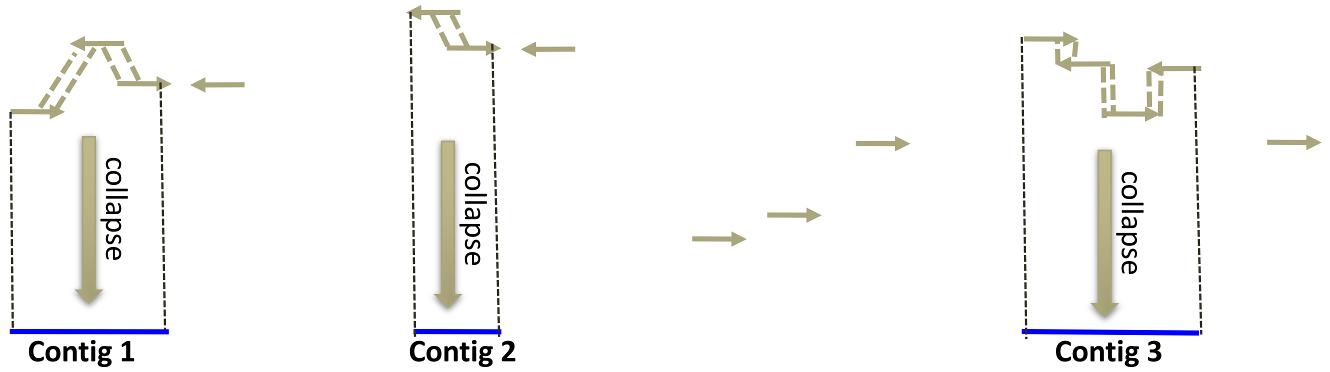


Image source: https://sourceforge.net/projects/quast/

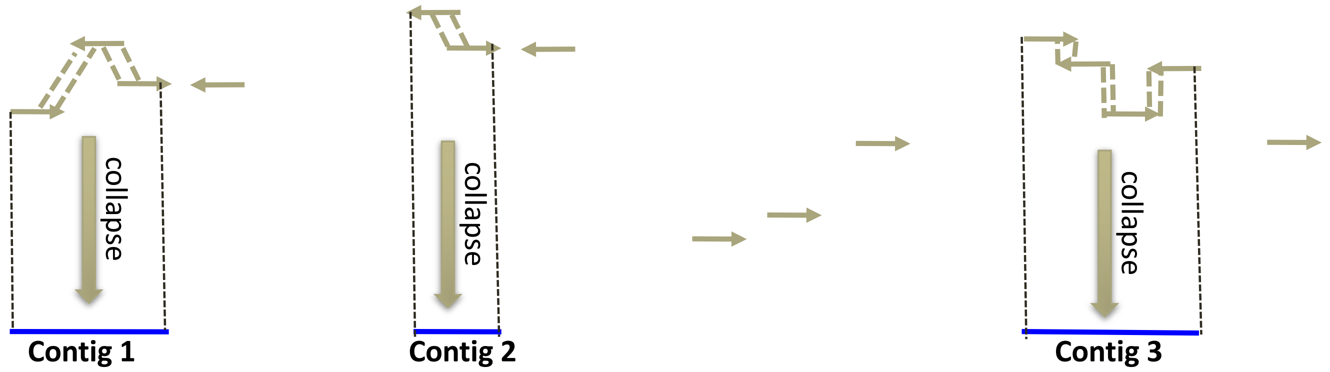
SUMMARY STATISTICS TO DESCRIBE ASSEMBLIES – N50 SIZE



2. **N50-size:** More than 50% of the bases in your assembly reside in contigs/scaffolds with **at least** the size determined by the N50 value. **NOTE:** You can of course specify any other *N-value*

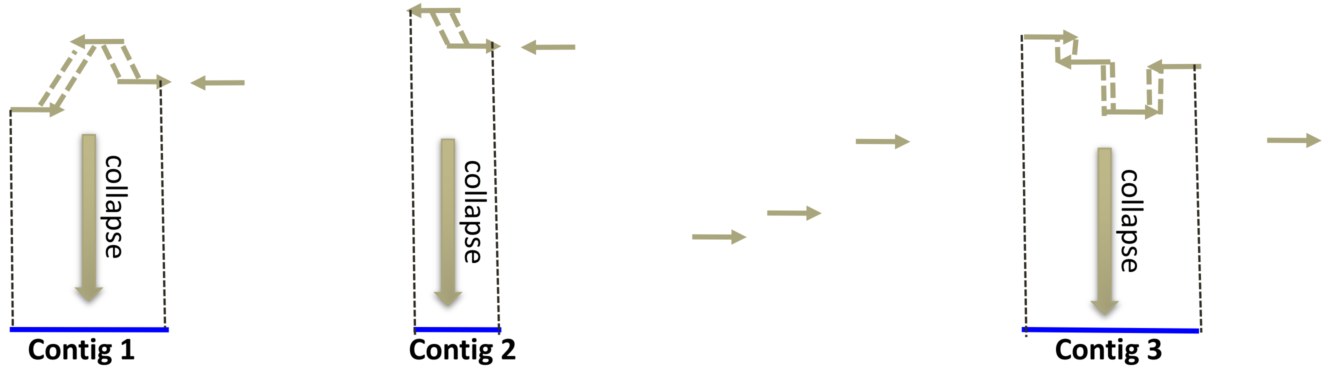
What now tells us the N50 size exactly?
Is it a quality measure as people frequently use it?
When does it make sense to mention the N50 size (just consider RNAseq assemblies)?

SUMMARY STATISTICS TO DESCRIBE ASSEMBLIES – N50, NG50, AND NGA50



2. **N50-size:** More than 50% of the bases in your assembly reside in contigs/scaffolds with **at least** the size determined by the N50 value. **NOTE:** You can of course specify any other *N-value*
3. **NG50-size**: More than 50% of the **genome sequence** reside in contigs/scaffolds with **at least** the size determined by the NG50 value
4. **NGA50-size**: More than 50% of the genome sequence reside in contigs/scaffolds of at least this size, which can be contiguously aligned to a reference sequence

SUMMARY STATISTICS TO DESCRIBE ASSEMBLIES – N50, NG50, AND NGA50



collapse

collapse

collapse

Contig 1

Contig 2

Contig 3

5. **Contig/scaffold length distribution**

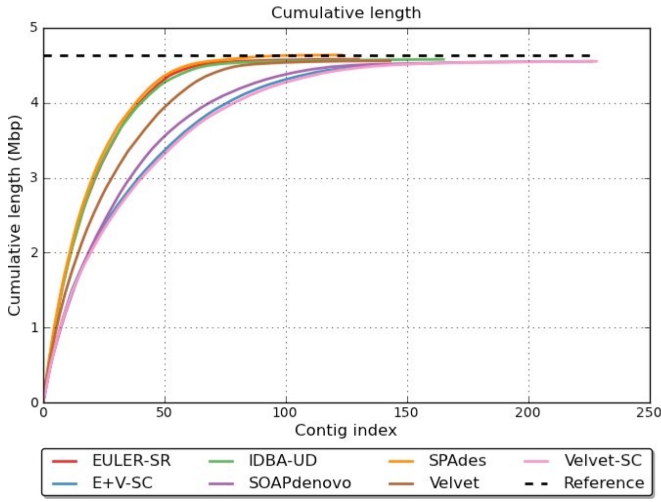
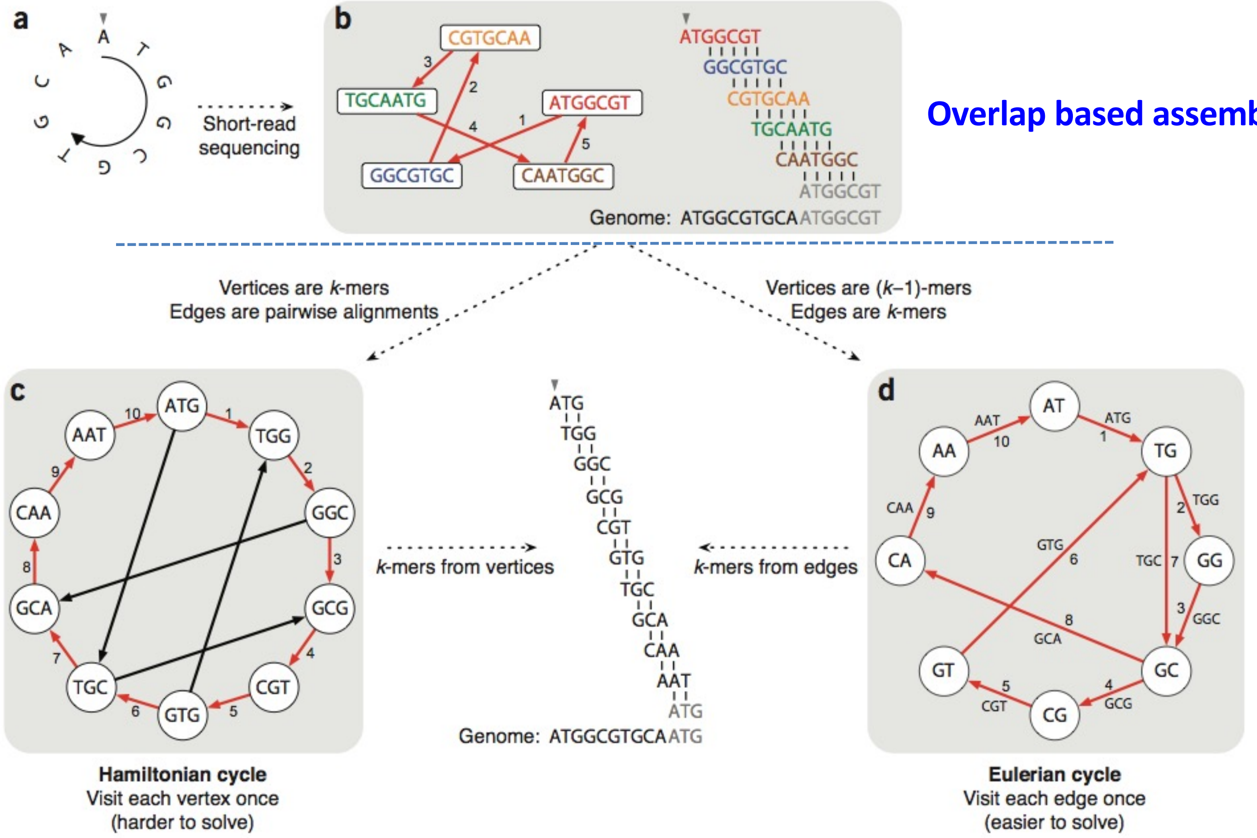


Image source: https://sourceforge.net/projects/quast/

THERE ARE TWO MAIN APPROACHES TO THE SEQUENCE ASSEMBLY PROBLEM



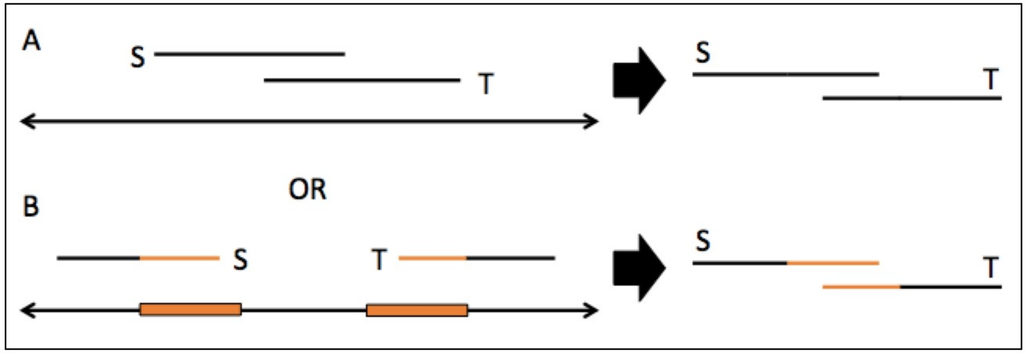
Overlap based assemblies

Word-based (Kmer) assemblies

modfied from Compeau et al. (2011) Nature Biotechnology **29**(11)

OVERLAP BASED ASSEMBLIES

Definition - An overlap is a region of high sequence similarity between the prefix of one read and the suffix of a second read[1]. Both maximum number of mismatches and minimum length need to be determined a priori



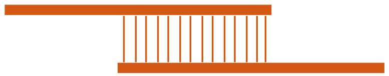
Interpretation - An overlap can indicate that two reads partially cover the same region of the template (A). Alternatively, repeats, i.e. the same or nearly the same sequence occurs more than once in the template sequence can induce overlaps.

[1] Note, reads have to be compared in both orientations

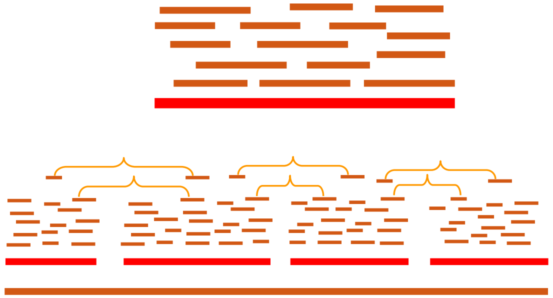
OVERLAP-LAYOUT-CONSENSUS ASSEMBLY

Assemblers: ARACHNE, PHRAP, CAP, TIGR, CELERA, CANU

Overlap: find potentially overlapping reads



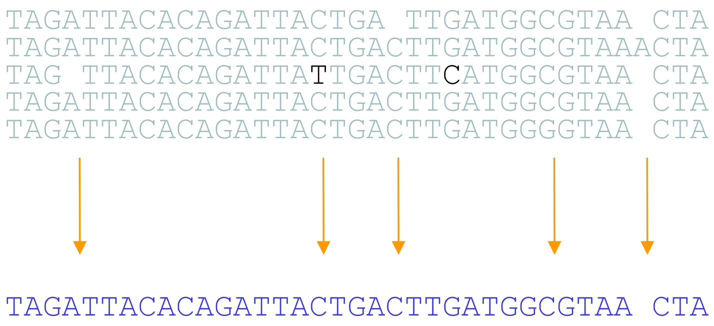
Layout: merge reads into contigs and
contigs into supercontigs



Consensus: derive the DNA sequence considering all read overlaps, and correct read errors

..ACGATTACAATAGGTT..

DERIVE CONSENSUS SEQUENCE



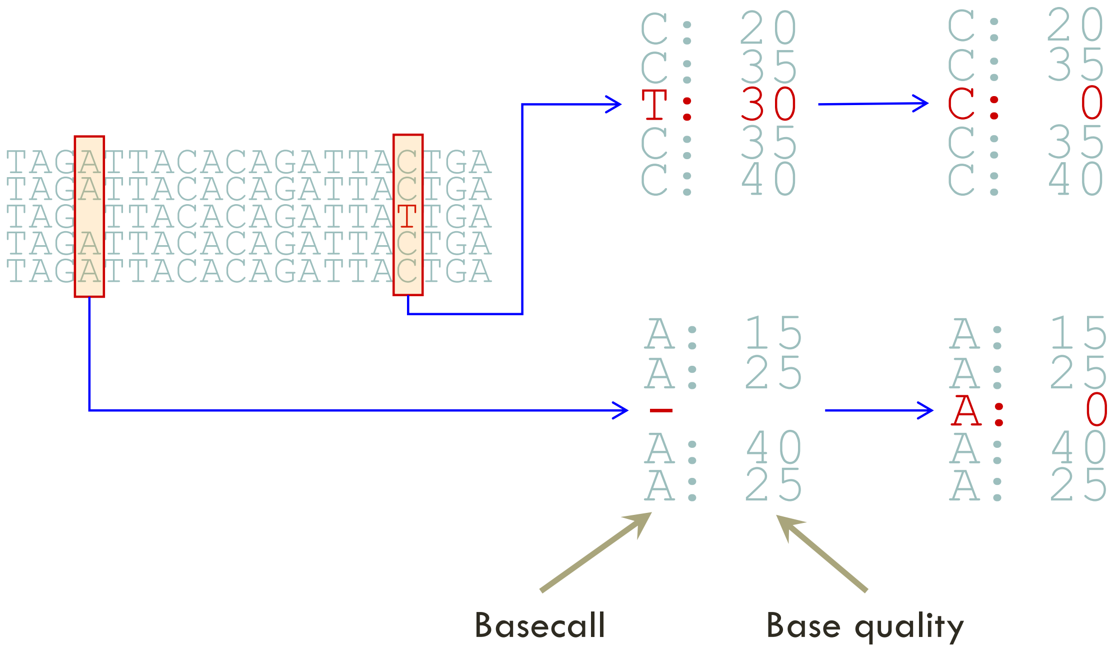
```
TAGATTACACAGATTACTGA TTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAAACTA
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGGGTAA CTA

TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

1. Derive multiple alignment from pairwise read alignments
2. Derive each consensus base by weighted voting

ERROR CORRECTION DURING CONSENSUS SEQUENCE FORMATION[1] USING A WEIGHTED VOTING

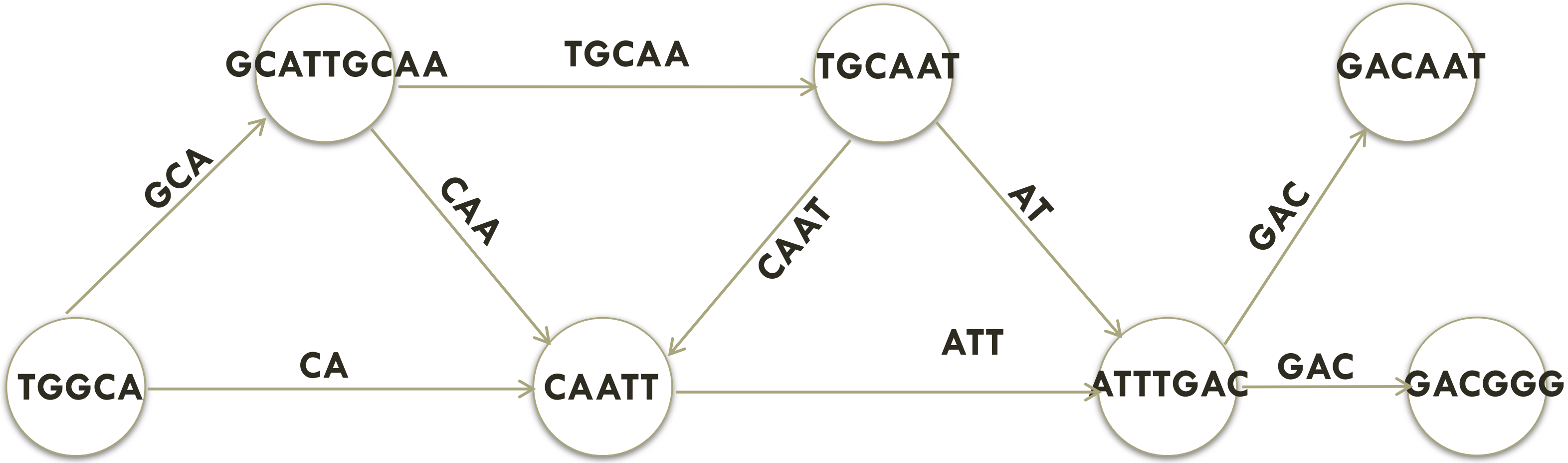
- Correct errors using multiple alignment



[1] There is also error correction on the read and the assembly level

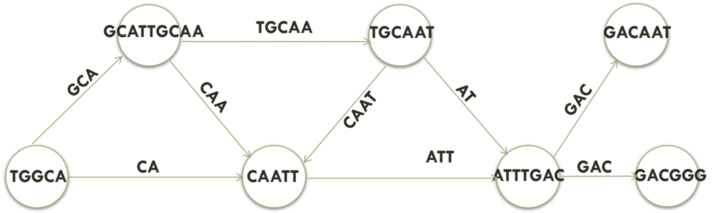
THE OVERLAP GRAPH

The construction of an **overlap** graph is in principle straightforward[1]. Reads constitute the nodes of the graph, and we draw an edge between to nodes if the reads overlap



[1] However, consider how many pair-wise comparisons you need to do, how graph complexity scales with coverage

THE OVERLAP GRAPH

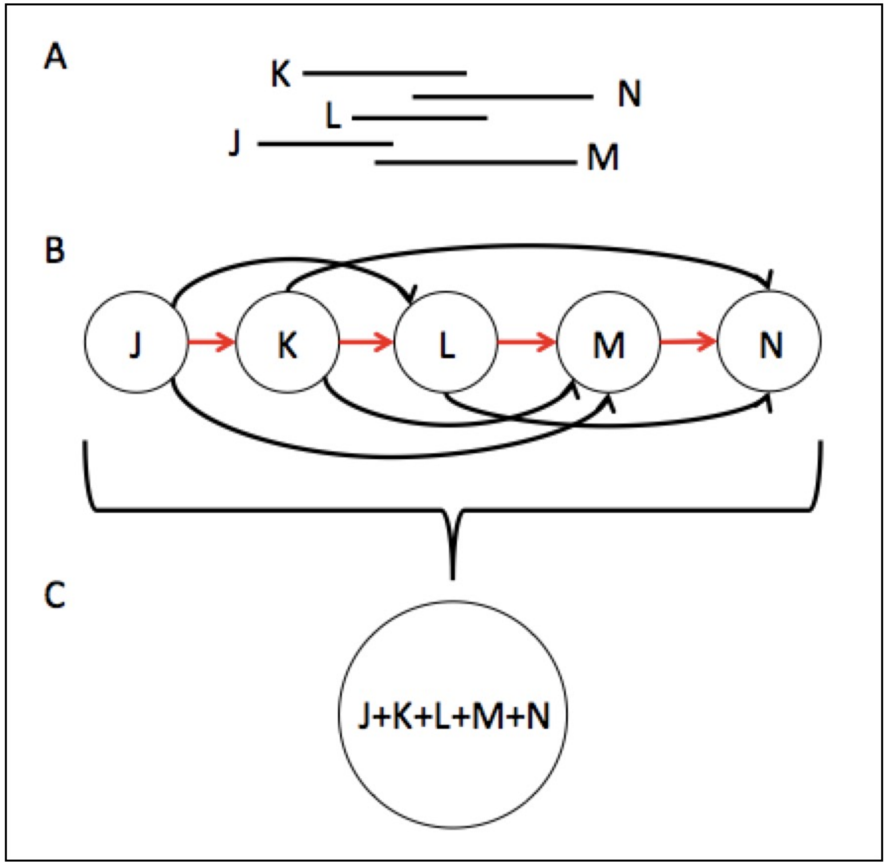


❖Traversing the graph such that each node is visited exactly once reconstructs the original sequence

❖Finding such a Hamiltonian path in a graph of millions of nodes and edges is computationally hard.

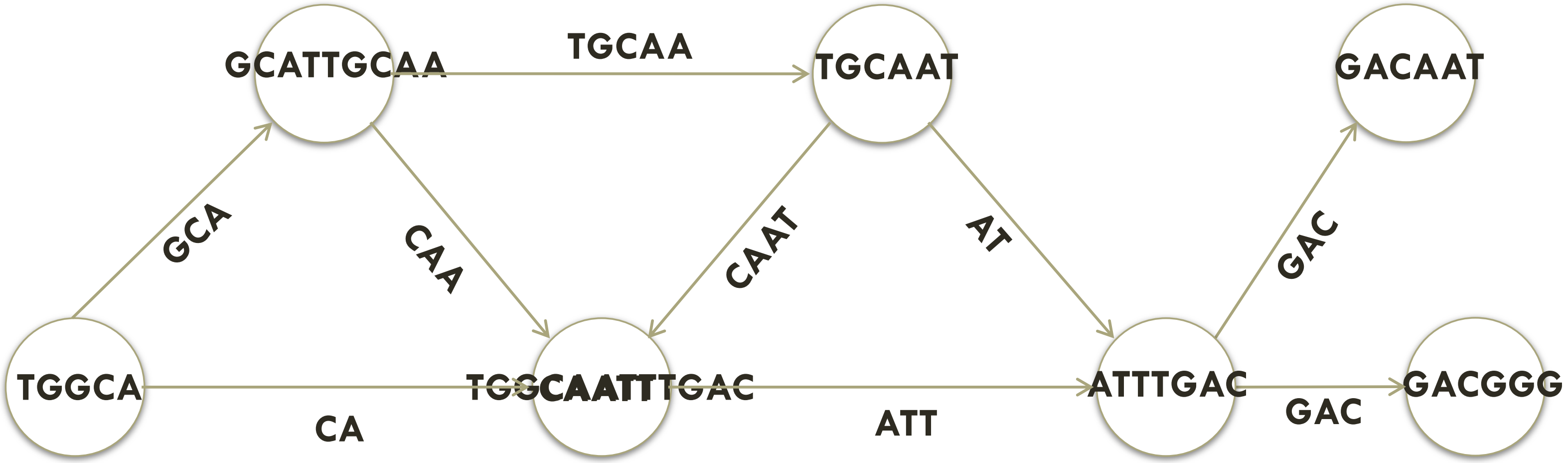
❖In order to decrease the search complexity the OLC assembly graph is simplified in the **layout stage,** where segments of the assembly graph are compressed into contigs

GRAPH SIMPLIFICATION

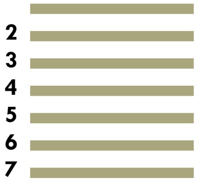


Graph simplification during the layout phase reduces the complexity of finding the best path through the overlap graph by summarizing unambiguous paths up to the next ‘fork’ into contigs.

GRAPH SIMPLIFICATION

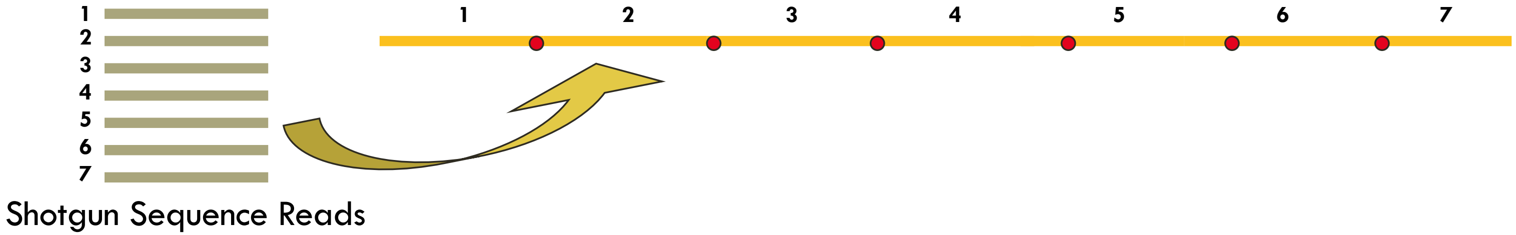


DE-NOVO SEQUENCE ASSEMBLY: CAP3



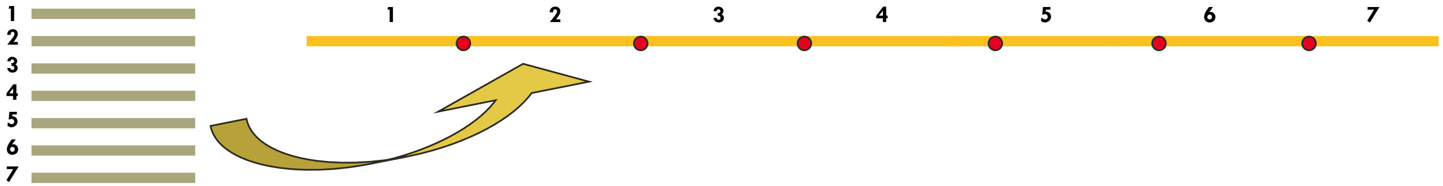
Shotgun Sequence Reads

DE-NOVO SEQUENCE ASSEMBLY: CAP3



1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS

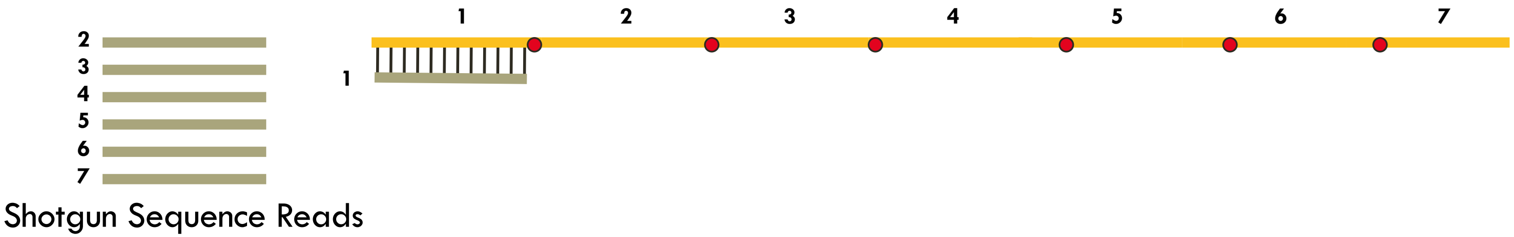


Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once

DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS

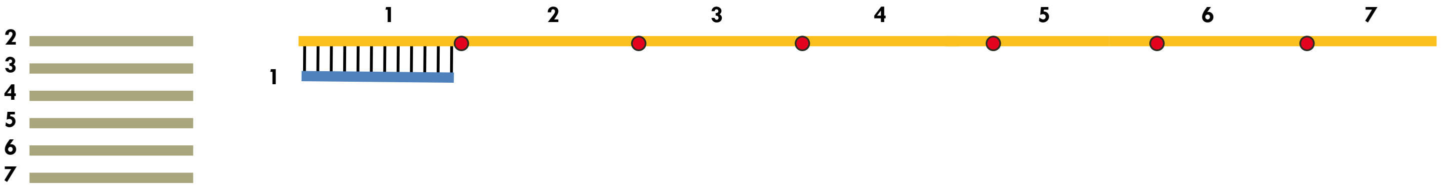


Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

DE-NOVO SEQUENCE ASSEMBLY (CAP3) SEARCH FOR LOCAL ALIGNMENTS



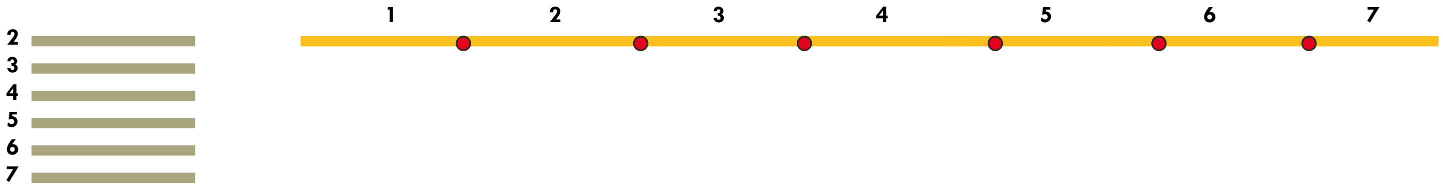
Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Remove the trivial solution (alignment against itself)

DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS

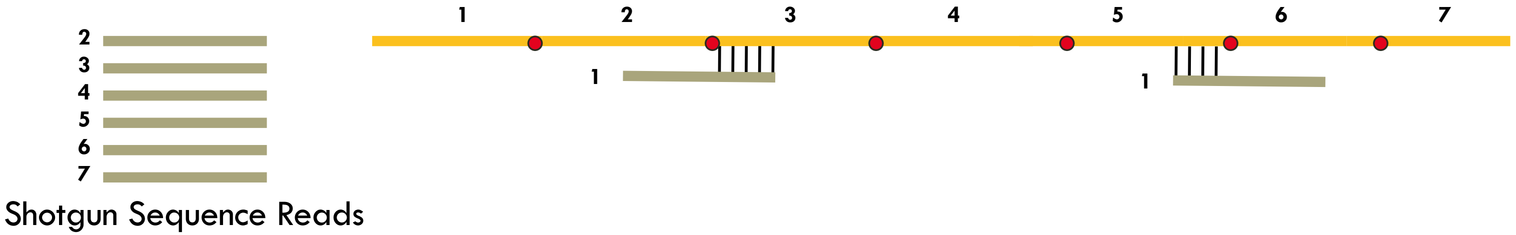


Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Remove the trivial solution (alignment against itself)

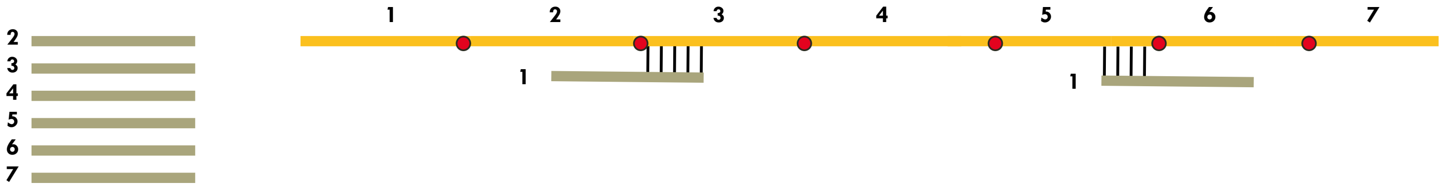
DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS



1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS

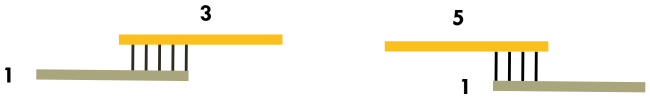


Shotgun Sequence Reads

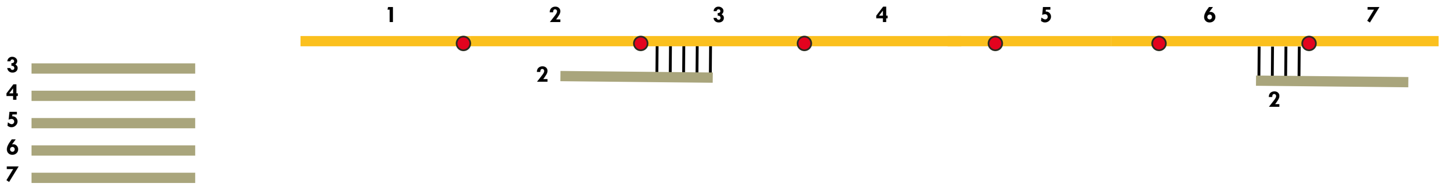
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Candidate pairs for read 1:



DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS

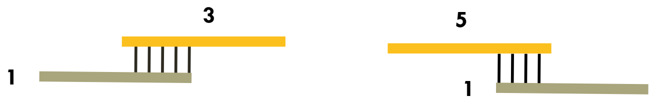


Shotgun Sequence Reads

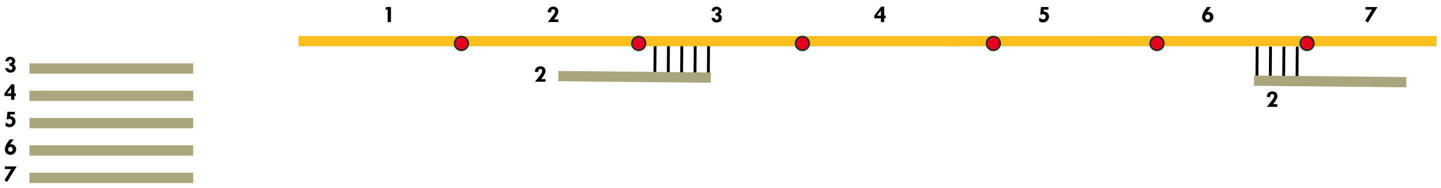
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

Candidate pairs for read 1:



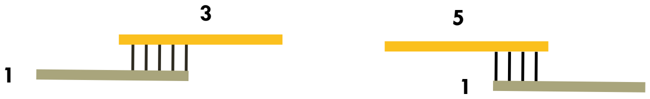
DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS



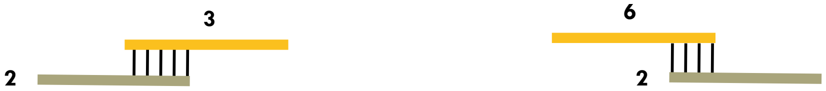
Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

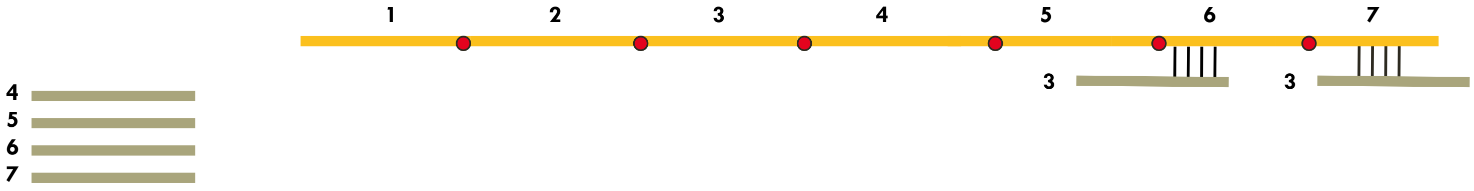
Candidate pairs for read 1:



Candidate pairs for read 2:



DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS

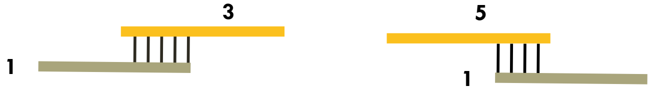


Shotgun Sequence Reads

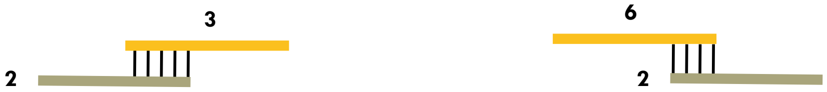
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

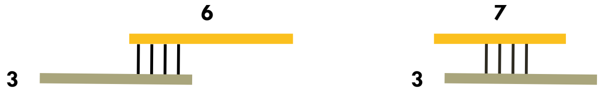
Candidate pairs for read 1:



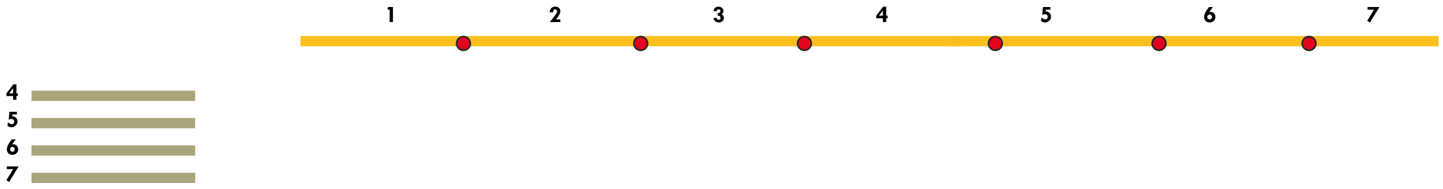
Candidate pairs for read 2:



Candidate pairs for read 3:



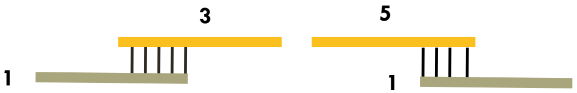
DE-NOVO SEQUENCE ASSEMBLY (CAP3) SEARCH FOR LOCAL ALIGNMENTS



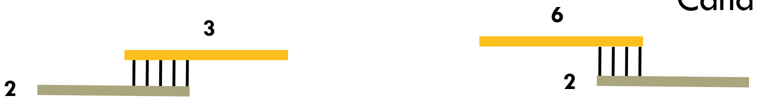
Shotgun Sequence Reads

1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

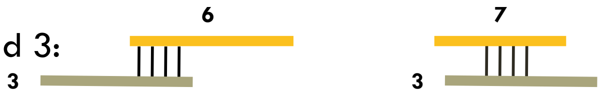
Candidate pairs for read 1:



Candidate pairs for read 2:



Candidate pairs for read 3:



⋮

Candidate pairs for read 7:

OVERLAPPING READS AND REPEATS

A *k*-mer that appears N times, initiates N^2 comparisons

For an *Alu* that appears 10^6 times → 10^{12} comparisons – too much

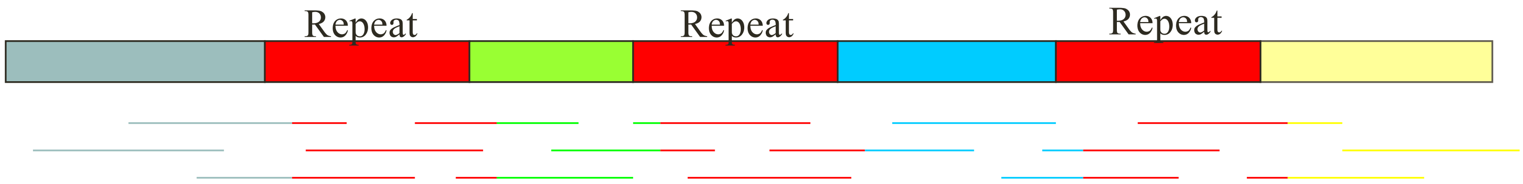
Solution:

Discard all *k*-mers that appear more than

$t \times$ Coverage, ($t \sim 10$)

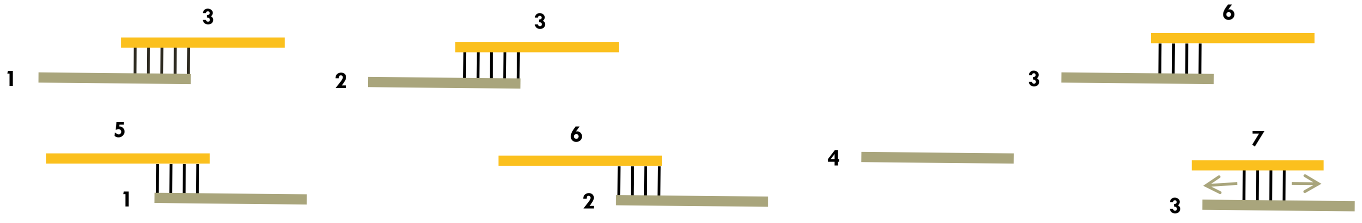
> 50% of human genome are repeats:

- over 1 million *Alu* repeats (about 300 bp)
- about 200,000 LINE repeats (1000 bp and longer)



Green and blue fragments are interchangeable when assembling repetitive DNA

DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS: POST-PROCESSING



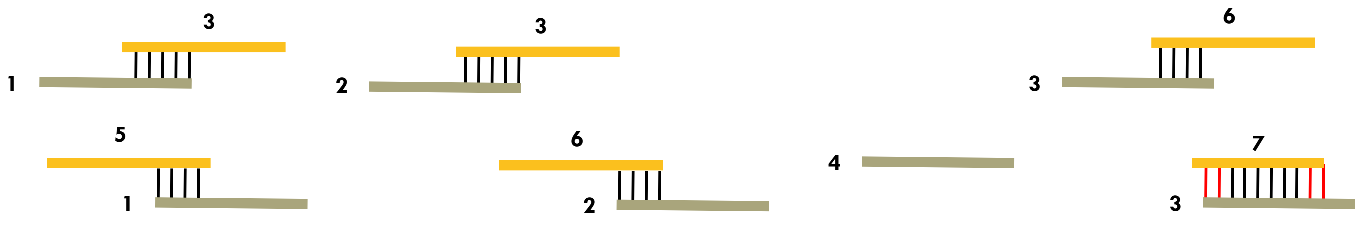
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.

2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.

3. Remove poor quality sequence ends

4. Compute global alignment for the high quality sequence pairs to verify overlaps.

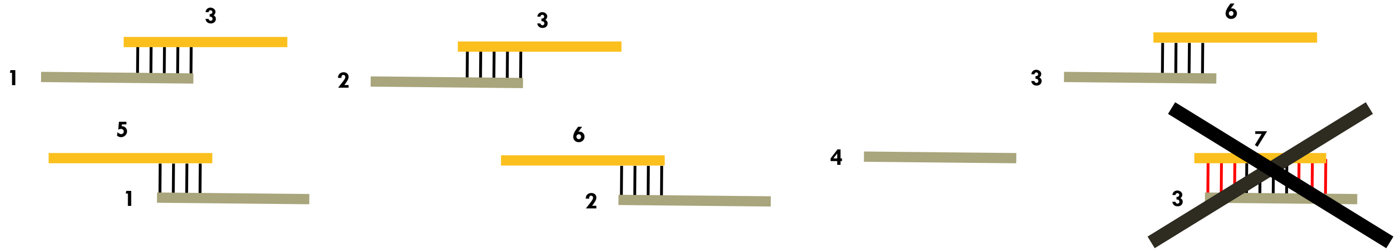
DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS: POST-PROCESSING



1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.
3. Remove poor quality sequence ends
4. Compute global alignment for the high quality sequence pairs to verify overlaps. Evaluate according to the following criteria:
    1. minimum length
    2. minimum identity
    3. minimum similarity
    4. number of high-quality mismatches

Remove sequence pairs that do not meet the thresholds for 4.1 to 4.4

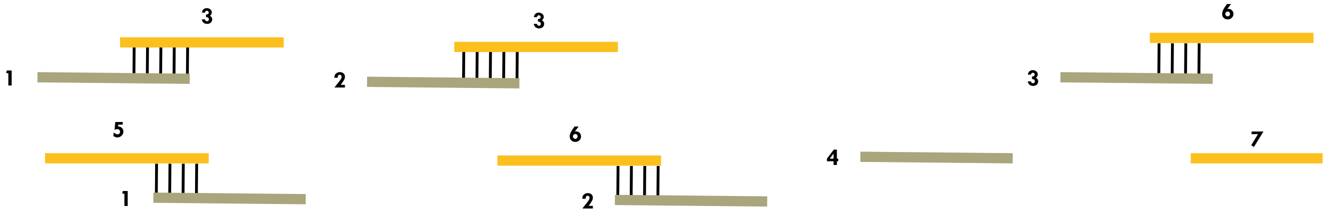
DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS: POST-PROCESSING



1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.
3. Remove poor quality sequence ends
4. Compute global alignment for the high quality sequence pairs to verify overlaps. Evaluate according to the following criteria:
    1. minimum length
    2. minimum identity
    3. minimum similarity
    4. number of high-quality mismatches

   Remove sequence pairs that do not meet the thresholds for 4.1 to 4.4

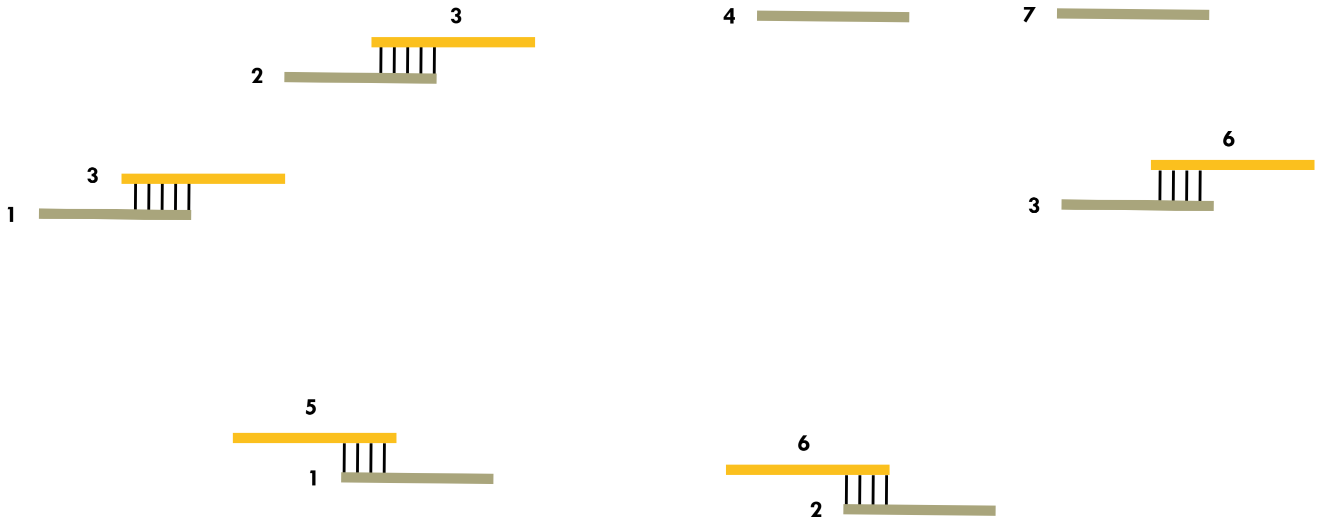
DE-NOVO SEQUENCE ASSEMBLY (CAP3)
SEARCH FOR LOCAL ALIGNMENTS: POST-PROCESSING



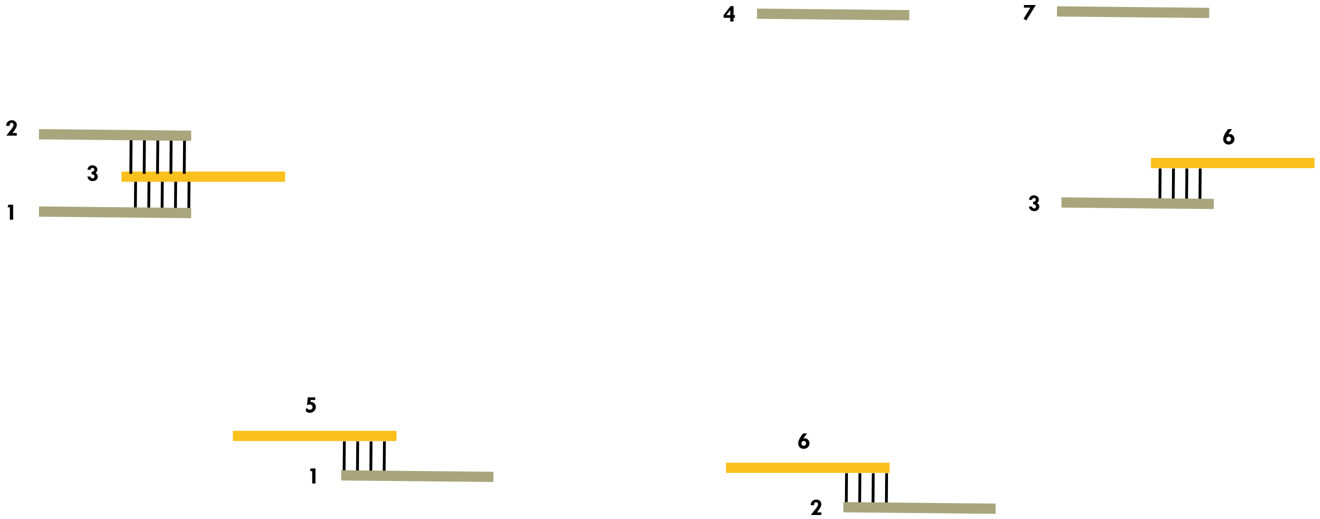
1. Concatenate sequences into a combined sequence. Reads are separated by a separation character.
2. Compute high scoring chains of segments between each read and the combined sequence using local alignment search tools. Identify candidate pairs. Every pair is counted only once.
3. Remove poor quality sequence ends
4. Compute global alignment for the high quality sequence pairs to verify overlaps. Evaluate according to the following criteria:
   1. minimum length
   2. minimum identity
   3. minimum similarity
   4. number of high-quality mismatches

   Remove sequence pairs that do not meet the thresholds for 4.1 to 4.4

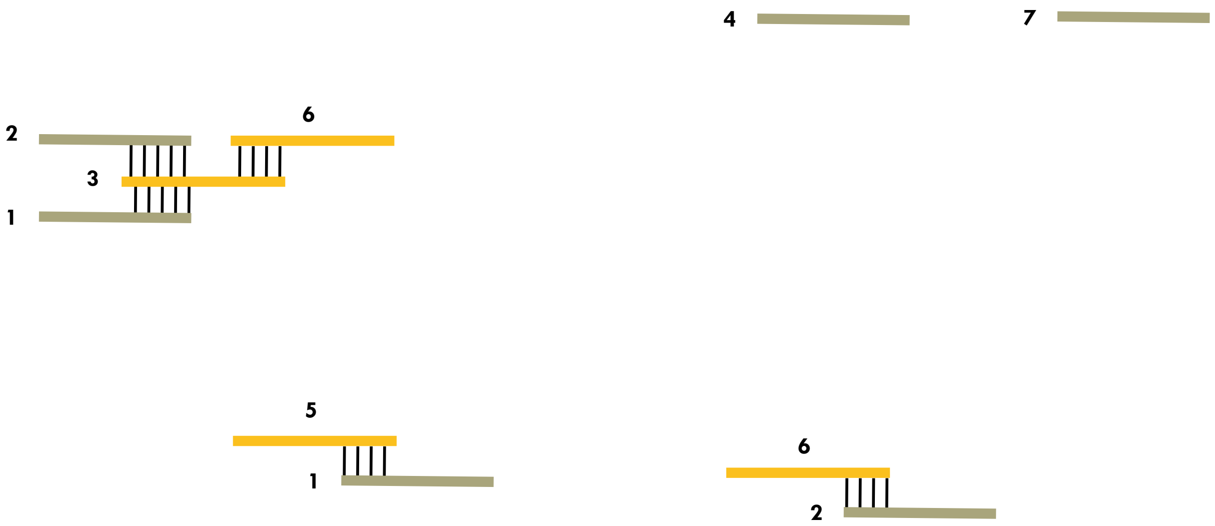
CAP3: CONTIG BUILDING



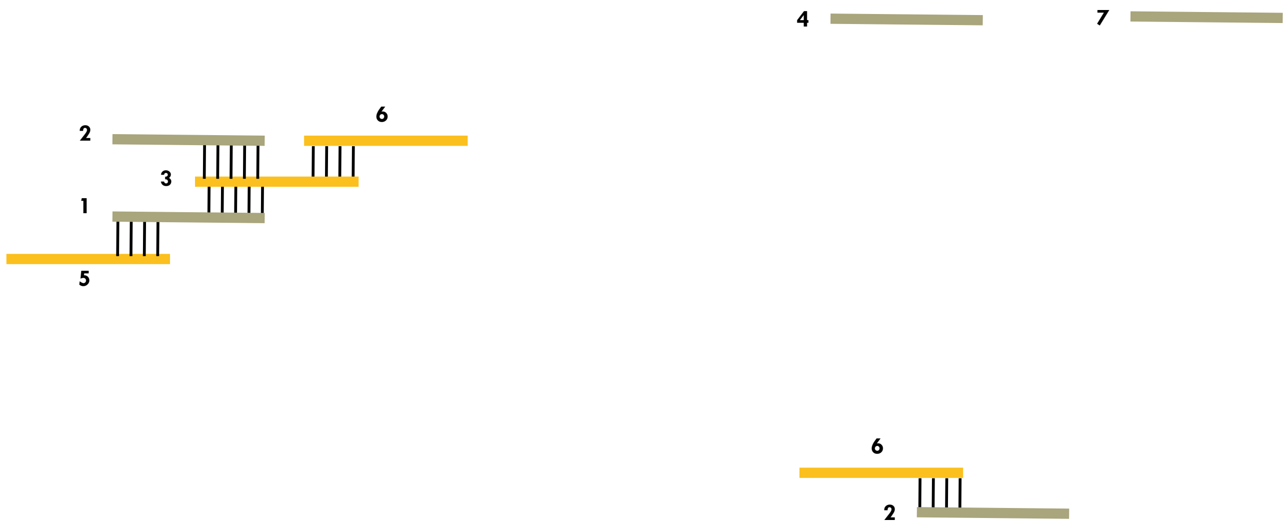
CAP3: CONTIG BUILDING



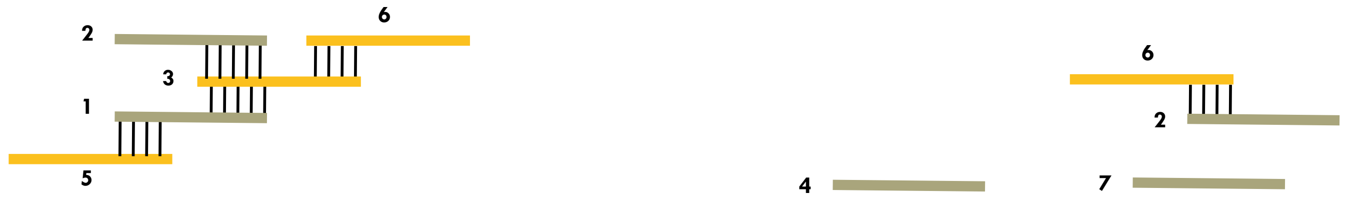
CAP3: CONTIG BUILDING



CAP3: CONTIG BUILDING

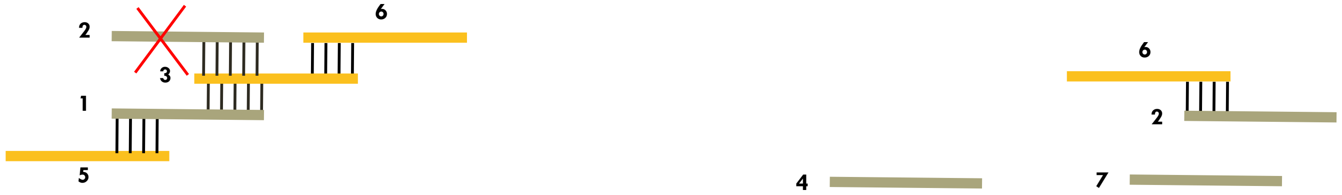


CAP3: CONTIG BUILDING



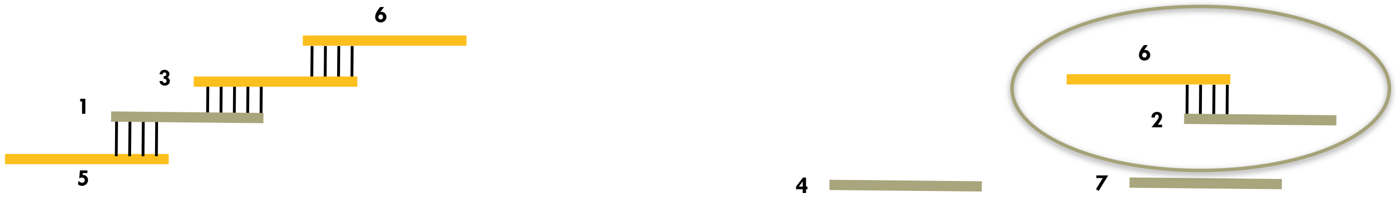
1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.

CAP3: CONTIG BUILDING



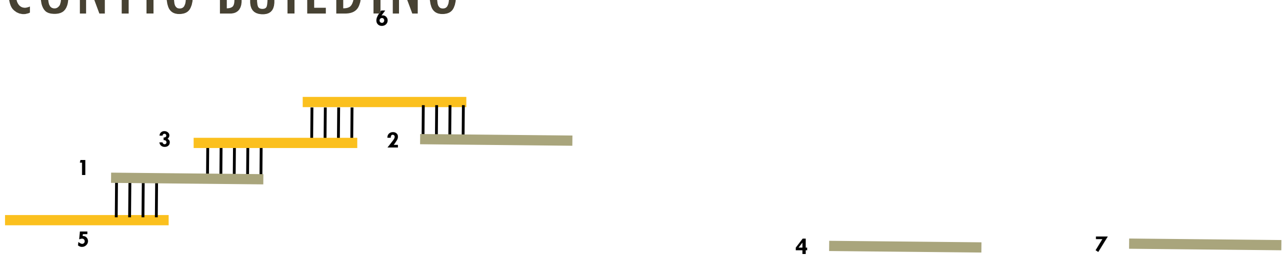
1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.
    1) sequence read 1 and 2 are incompatible since they could not be aligned.

CAP3: CONTIG BUILDING



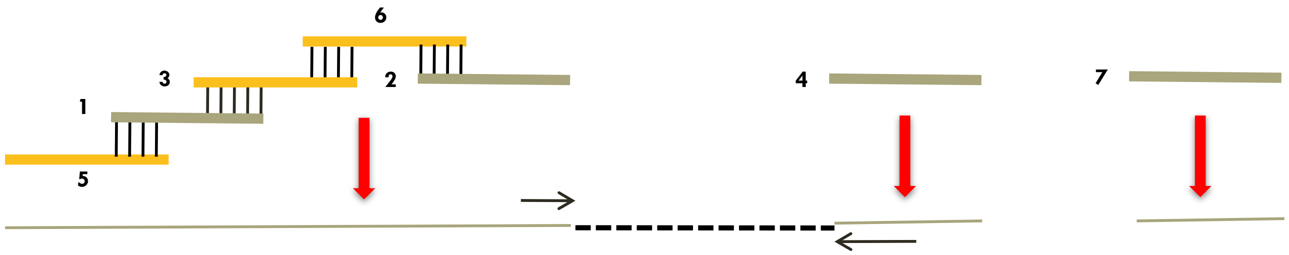
1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.
    1) sequence read 1 and 2 are incompatible since they could not be aligned.
    2) resolve incompatibility
    3) check for new possible layouts

CAP3: CONTIG BUILDING



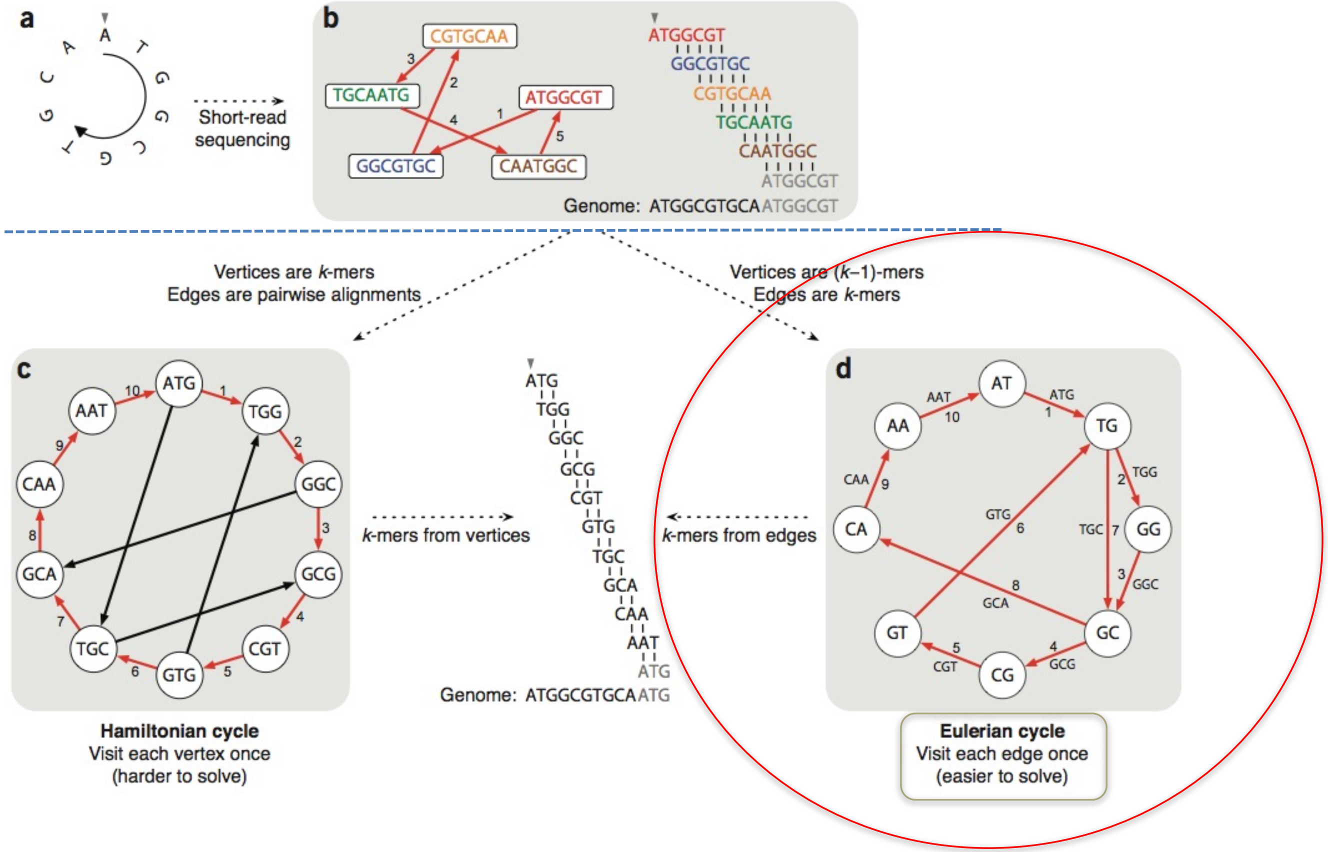
1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities.
    1) sequence read 1 and 2 are incompatible since they could not be aligned.
    2) resolve incompatibility
    3) check for new possible layouts

FURTHER STEPS - SCAFFOLDING



1) Generate a general layout using the overlapping reads from the pair-wise analysis (Greedy algorithm in decreasing order of overlap scores).
2) In a simple view: Check the layout for incompatibilities, remove incompatible reads and align.
3) Build a consensus sequence for each contigs.
4) Order and orient contigs if possible using additional information, e.g., paired end reads.

THE TWO BASIC CONCEPTS OF DNA SEQUENCE ASSEMBLY



modfied from Compeau et al. (2011) Nature Biotechnology **29**(11)

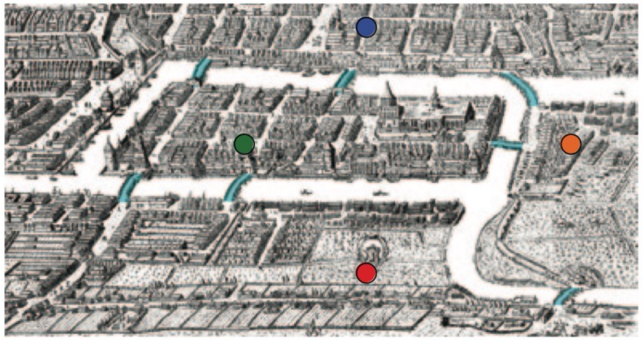
LETS LOOK AT THE SEQUENCE ASSEMBLY PROBLEM FROM A DIFFERENT PERSPECTIVE: THE SHORTEST SUPERSTRING PROBLEM (NICOLAS DE BRUIJN 1946)

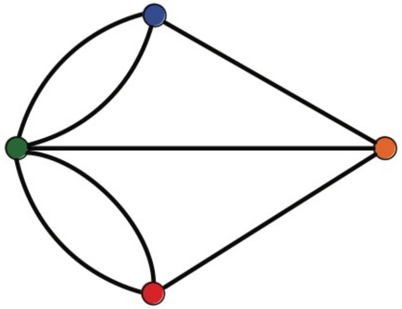
Problem: find the shortest (circular) superstring that contains all possible substrings of length *K* over a given alphabet.

for *K* = *4* and a two letter alphabet *A*={*0,1*} we have 16 different words:

0000, 0001, 0010, 0100, 1000, 0011, 0110, 1100, 1001, 1010, 0101, 0111, 1011, 1101, 1110, 1111

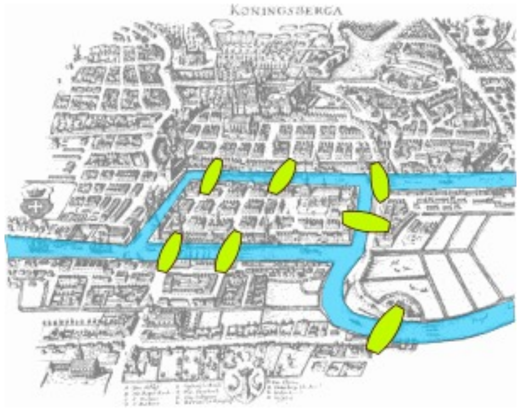
To solve this problem, de Bruijn borrowed from Euler who solved 1735 the 'Königsberg' problem, i.e. the question whether it is possible to visit each island by crossing each bridge exactly once (Eulerian cycle)

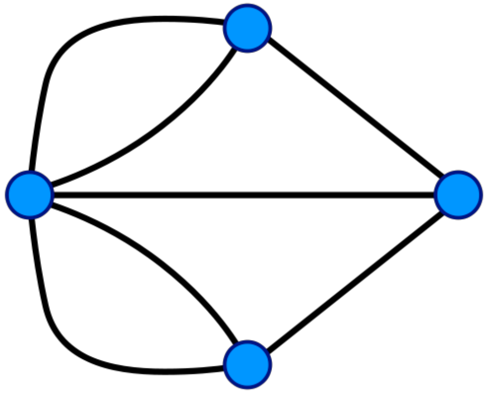




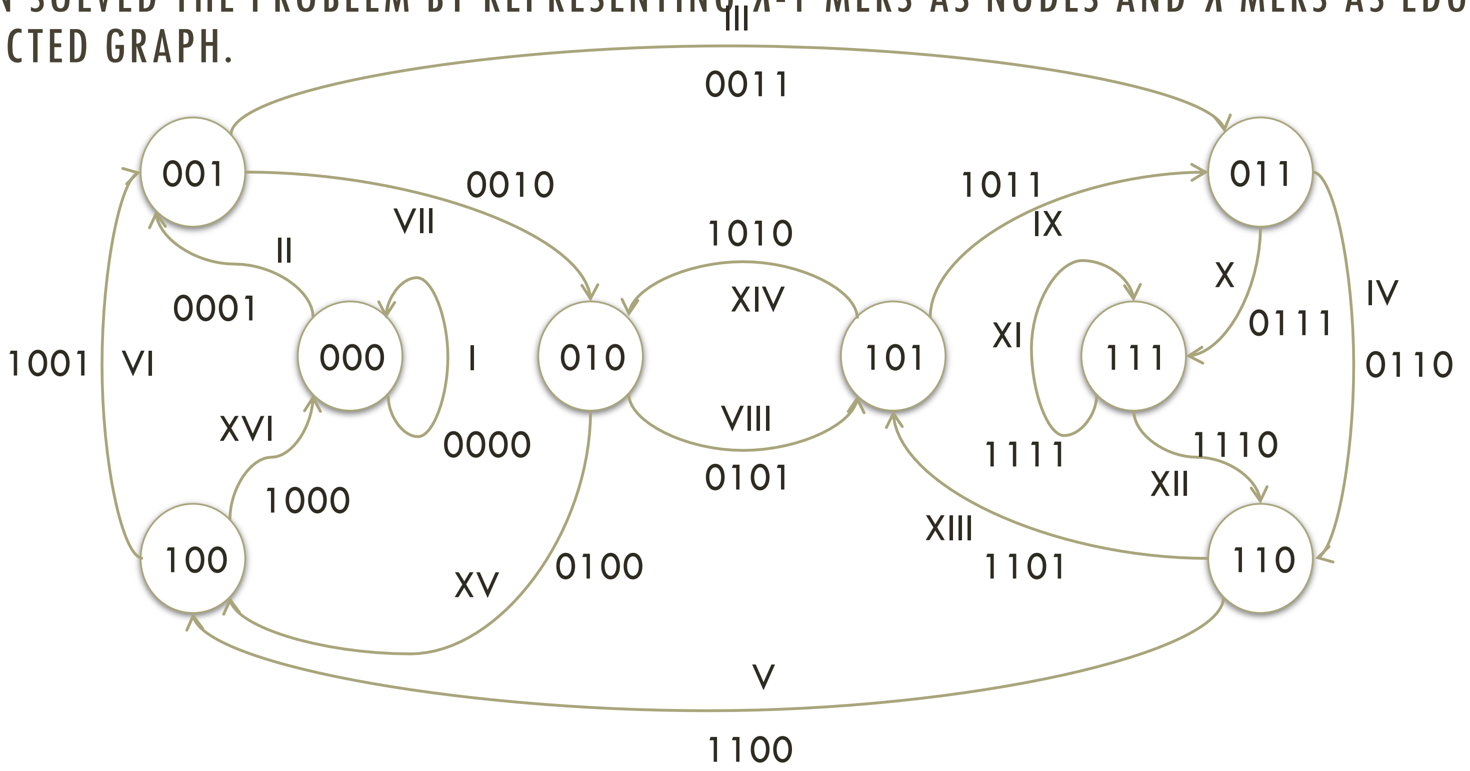
EULERIAN CYCLE PROBLEM

- Find a cycle that visits every ***edge*** exactly once (Linear time)
- An Eulerian Cycle exists if the number of ‘outgoing’ edges for a node equals the number of ‘incoming’ edges*.
- The graph may have 2 nodes with an odd number of edges connected to it. In this case an Eulerian path rather than an Eulerian cycle can be found.



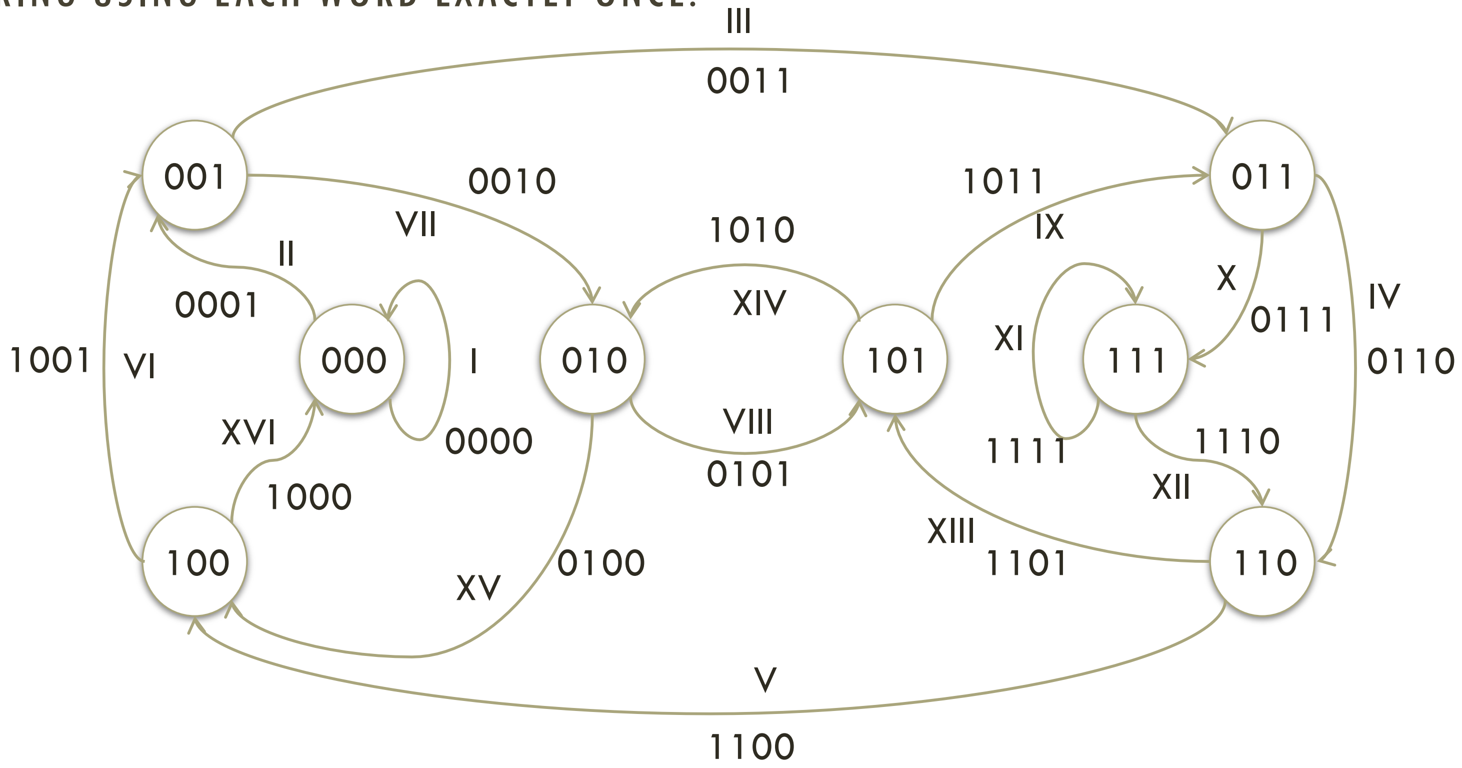


DE BRUIJN SOLVED THE PROBLEM BY REPRESENTING *K*-1 MERS AS NODES AND *K* MERS AS EDGES IN A DIRECTED GRAPH.



By doing so, he related the problem of finding a shortest common superstring to the already solved problem of finding an Eulerian cycle in a graph.

PASSING THROUGH THE EDGES BY FOLLOWING THE ROMAN NUMBERS RECONSTRUCTS THE SUPERSTRING USING EACH WORD EXACTLY ONCE!



I: 0000, II: 0001, III: 0011; IV: 0110; V: 1100; VI: 1001; VII: 0010; VIII: 0101; IX: 1011; X: 0111; XI: 1111; XII: 1110; XIII: 1101; XIV: 1010; XV: 0100; XVI: 1000

0000110010111101

BASIC CONCEPTS OF DE BRUIJN GRAPH BASED ASSEMBLERS

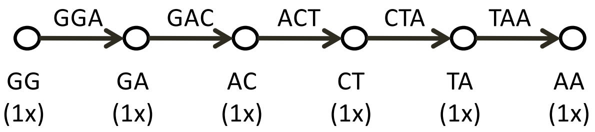
- ❖The sequence is treated as a consecutive string of words of length *K*
- ❖Sequence reads are no longer considered to represent a consecutive string of nucleotides. Thus read length as well as read overlap become, in principle, irrelevant.
- ❖Sequence reads are only used to identify words of length *K* occurring in the sequence.
- ❖Given perfect data – error-free K-mers providing full coverage and spanning every repeat – the K-mer graph would be a de Bruijn graph and it would contain an Eulerian path, that is, a path that traverses each edge exactly once.

DE BRUIJN GRAPH EXAMPLE

SHRED READS INTO K-MERS (K = 3)

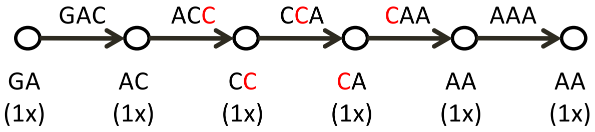
Read 1

G	**G**	**A**	**C**	**T**	**A**	**A**
G	G	A				
	G	A	C			
		A	C	T		
			C	T	A	
				T	A	A



Read 2

G	**A**	**C**	**C**	**A**	**A**	**A**
G	A	C				
	A	C	C			
		C	C	A		
			C	A	A	
				A	A	A



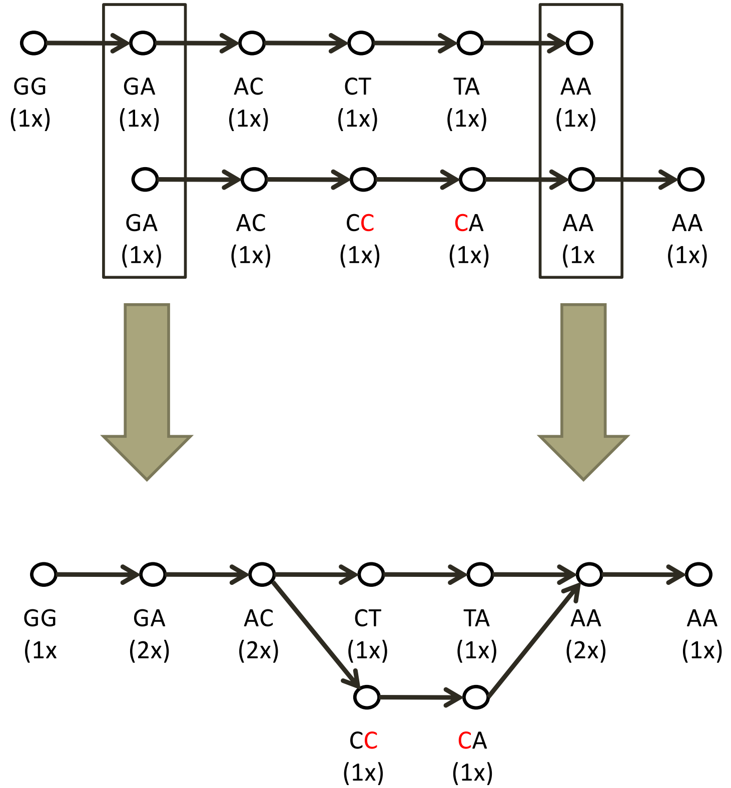
DE BRUIJN GRAPH EXAMPLE

MERGE VERTICES LABELED BY IDENTICAL (K-1)-MERS

Read 1:

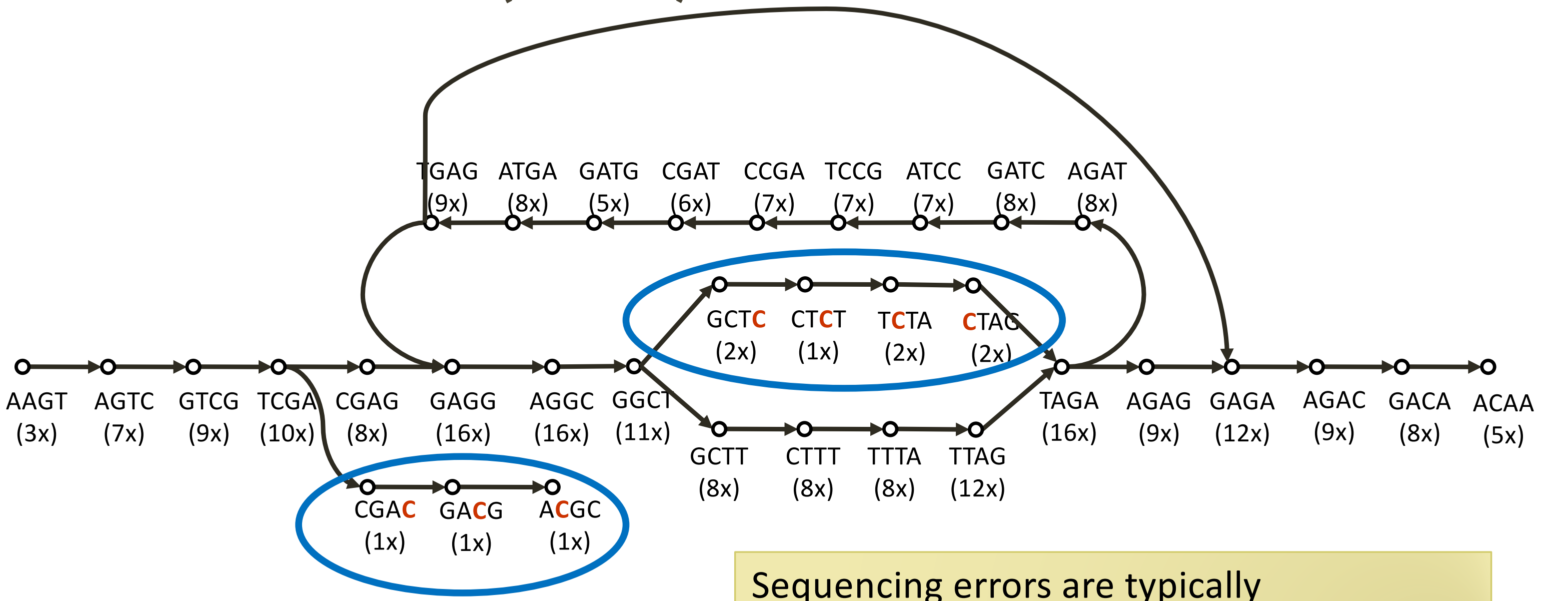
Read 2:

Resulting Graph:



ANOTHER EXAMPLE

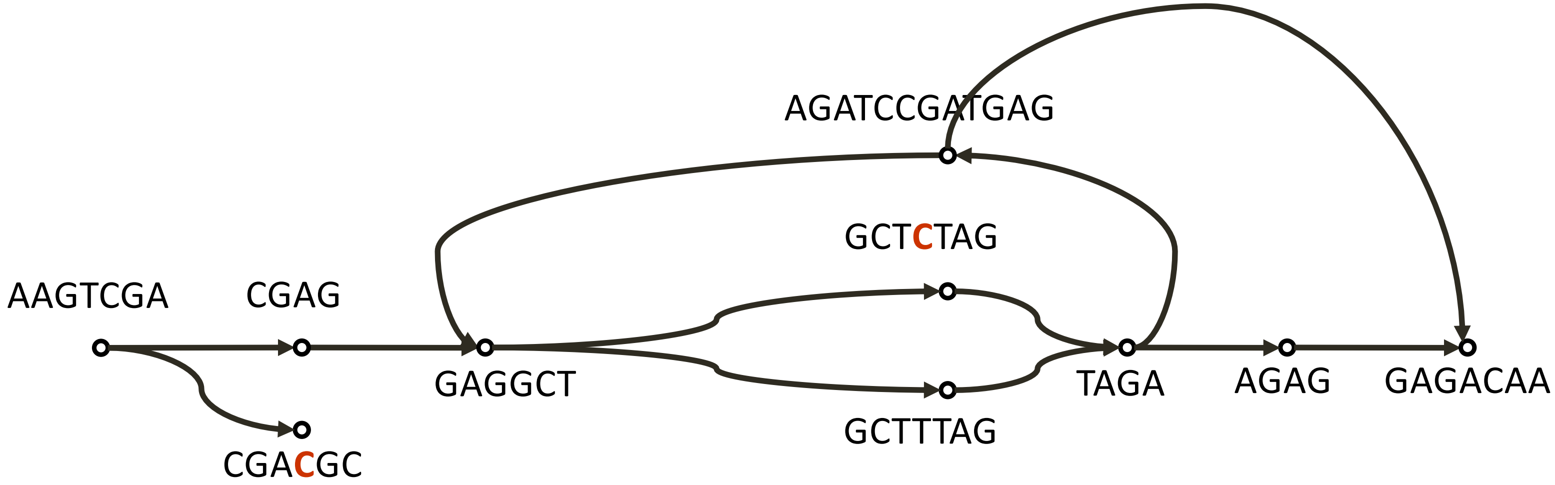
CONSTRUCT THE GRAPH (K = 5)



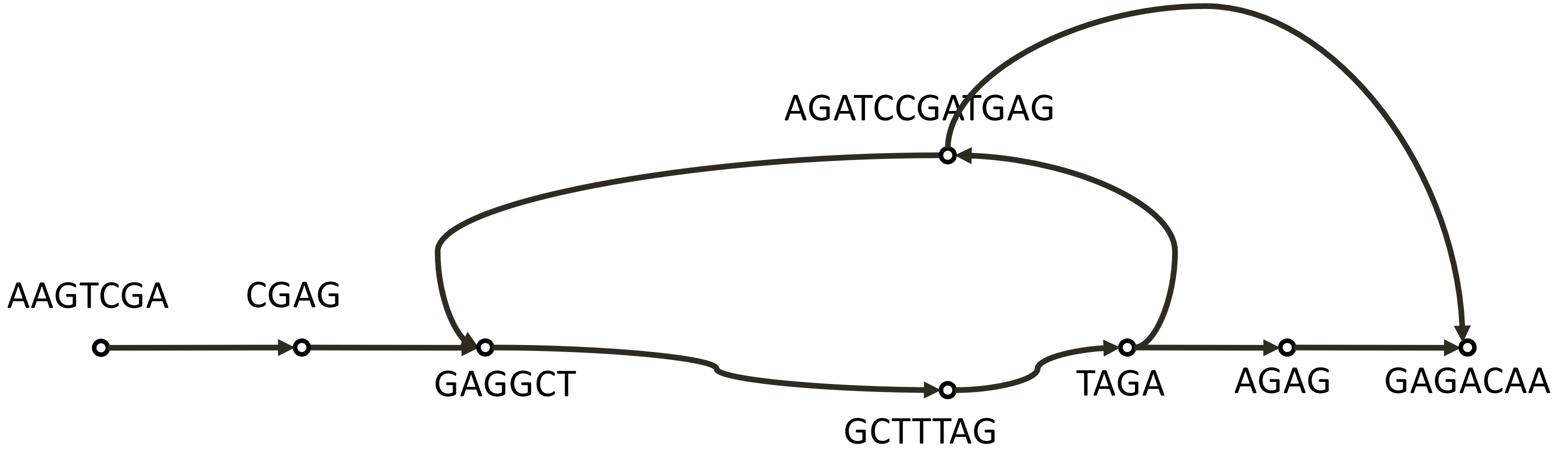
Sequencing errors are typically detected by a coverage cutoff threshold

A branching vertex is caused by either a repeat in the original sequence or a sequencing error

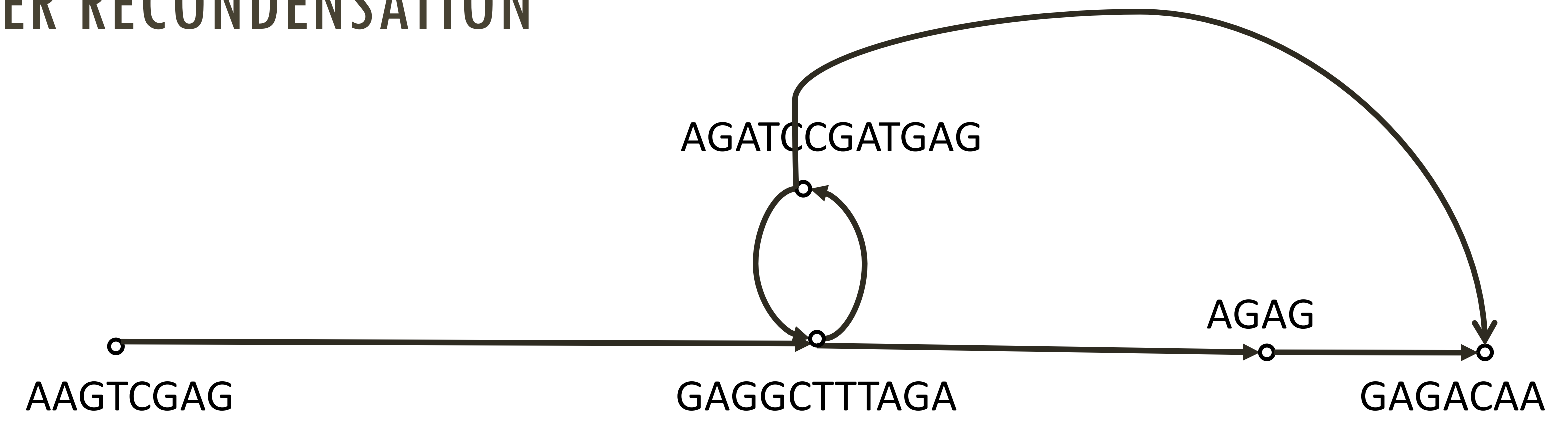
CONDENSE UNBRANCHED RUNS IN THE GRAPH



CORRECT SEQUENCING ERRORS USING A COVERAGE THRESHOLD



AFTER RECONDENSATION



Any non-branching path in this graph corresponds to a contig in the original sequence.

Contig 1: AAGTCGAG

Contig 2: GAGGCTTTAGA

Contig 3: AGATCCGATGAG

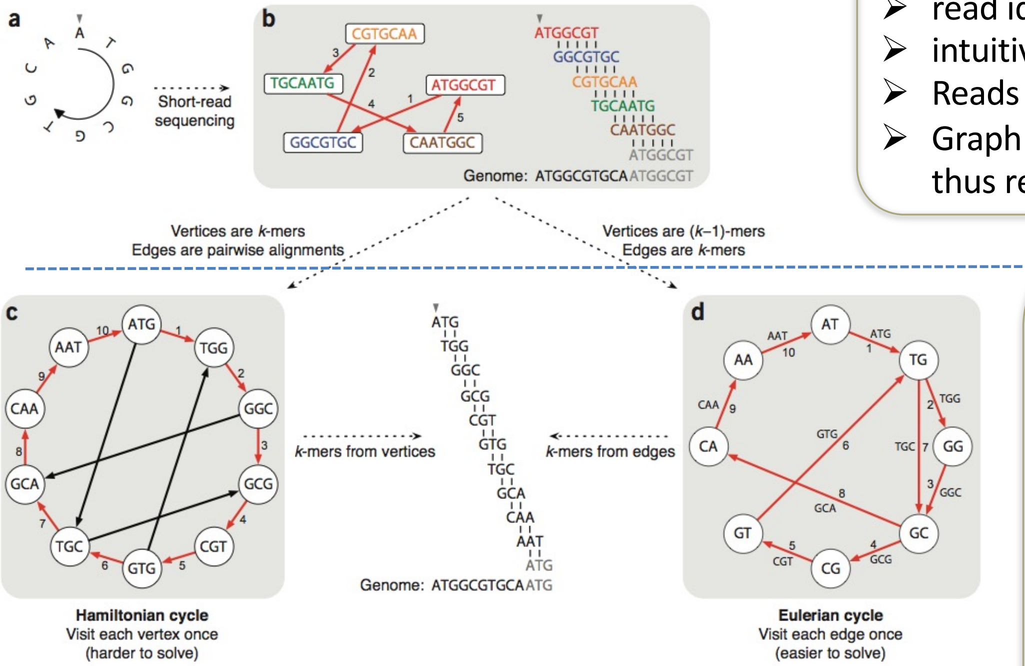
Contig 4: AGAG

Contig 5: GAGACAA

Taking the risk of following arbitrary branching paths may create chimeric species

Source: Serafim Batzoglou

SUMMARY: THERE ARE TWO MAIN APPROACHES TO THE SEQUENCE ASSEMBLY PROBLEM



modified from Compeau et al. (2011) Nature Biotechnology **29**(11)

Overlap based assembly

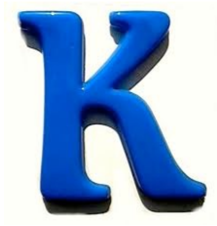
- ➢ read identity is maintained
- ➢ intuitive
- ➢ Reads can be organized in an overlap graph
- ➢ Graph complexity increases with coverage, thus read redundancy inflates the graph

Kmer approaches

- ➢ read identity is (temporarily) lost…
- ➢ Reads are organized in deBruijn graphs
- ➢ Graph complexity depends on Kmer size
- ➢ Graph complexity is (by and large) independent from coverage, read redundancy is naturally handled
- ➢ repeats are represented only once in the graph with explicit links to the different start and end points

THE MAGIC '*KMER*' GIVES MOST USERS OF GRAPH BASED ASSEMBLY ALGORITHMS A VERY HARD TIME AS THEY HAVE TO DECIDE ON THE SIZE OF *K.*

To give an informed statement we need to make sure to understand what *K* should represent and what the algorithmic requirements of de Bruijn graph assemblers are



K must represent a word that occurs **only once** in the sequence that should be assembled. Thus, *K* must be **sufficiently large.**

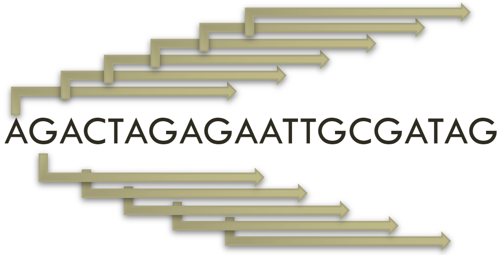
de Bruijn graph based assemblers assume that **each word** of length *K* occurring in the genome is also represented in the graph. As *Kmers* are collected from a finite set of sequence reads, ***K* must not be too large.**

consider a DNA word of *K*=2, how often does it on average occur in a string of 16 bp?

Take home message: If *K* is only sufficiently large the chance for any *Kmer* to occur more than once in a (repeat-free) genome approaches 0.

Why not using simply the read length as *K*?

WHY K MUST NOT BE TOO LARGE



AGACTAGAGAATTGCGATAG

A sequence of length 20 contains 11 different words of length 10!

Now, consider the sequence is spanned by 2 reads of length 13:

```
T:  AGACTAGAGAATTGCGATAG
R1: AGACTAGAGAATT
R2:        AGAATTGCGATAG
```

It is easy to see that not all 11 words of length 10 can be reconstructed with the two reads.
This violates the key assumption of the de Bruijn graphs

It is also easy to see that reducing *K* ameliorates the problem and eventually gets rid of it (just consider *K*=1…)

ÜBUNG

1. Skizzieren Sie die Vorgehensweise bei der ‘Single Molecule Real Time (SMRT)’ Sequenzierung. Worin liegen die wesentlichen Vor- und worin die Herausforderungen dieser Methode? (8 P)

2. Sehen Sie ein Problem darin, wenn die verwendete Index-Sequenz in Ihrem Insert, das Sie sequenzieren wollen, vorkommt. Begründen Sie! (2 P)

3. Erläutern Sie das FASTQ-Sequenzformat anhand eines beliebigen Beispiels. (2 P)

4. Die Software FASTQC durchsucht Ihr Sequenzdatensatz nach überrepräsentierten (Teil-)Sequenzen.

    1. Skizzieren Sie einen einfachen Algorithmus, der diese Aufgabe leisten kann. (3 P)

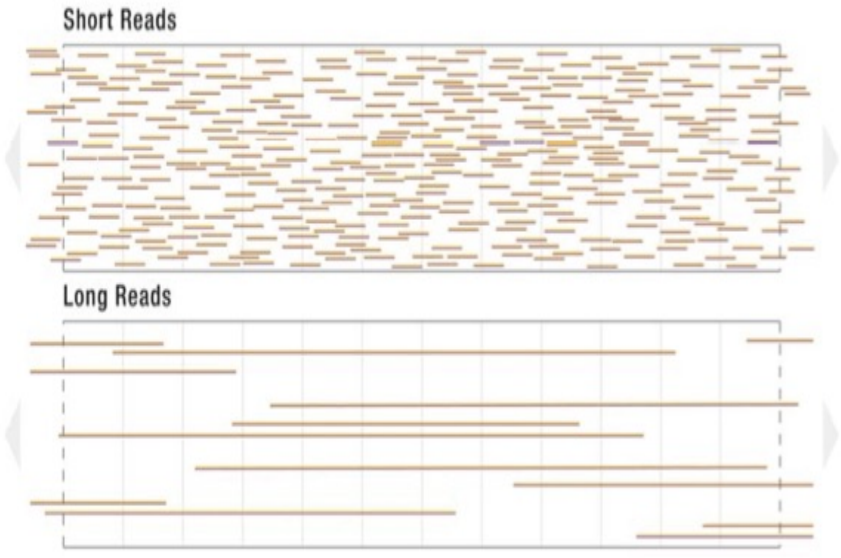
    2. Mit welcher Speicherkomplexität läuft Ihr Algorithmus und worin sehen Sie das Problem bei der Analyse von Datensätzen, die von den heute gängigen Sequenzierungsmaschinen generiert werden? (1 P)

    3. Mittels welchem Ansatz löst FASTQC dieses Problem und welche Gefahren birgt dieser? (2 P)

    4. Beurteilen Sie die Höhe des Risikos, das sich aus dem Ansatz von FASTQC für Ihre Sequenzanalyse ergeben könnte. (2 P)

Single Molecule Real Time Sequencing (SMRT)

- Sequencing by synthesis
- Parallelized
- Uses DNA polymerase
- Readlength ~ 15 kbp
- Individual reads have a substantial sequencing error (~15%)



SMRT – Technology

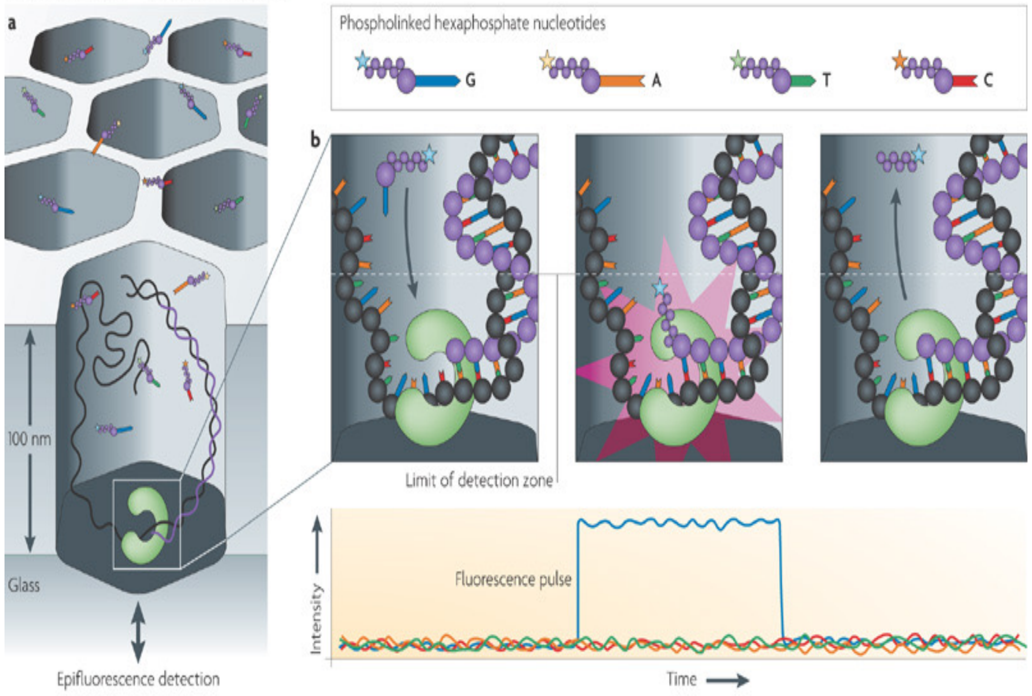
- zero-mode waveguide (ZMW) reaction chamber

- immobile polymerase

- 150.000 ZMWs per sequencing cell

- fluorescent labeled phosphate chain

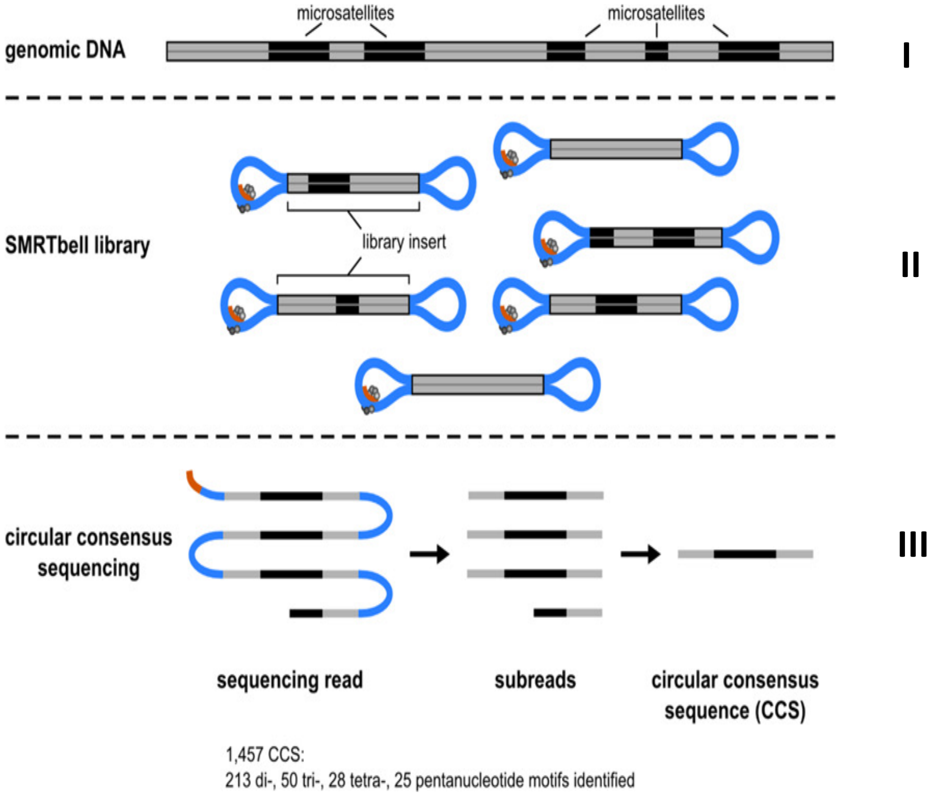
Pacific Biosciences — Real-time sequencing



Nature Reviews | Genetics

SMRT – library preparation

- library of overlapping inserts
- hairpin adaptors create a circular molecule
- adaptors contain binding site for DNA polymerase
- sequencing results in a long sequencing read
- generate multiple subreads from one long sequencing read
- combine subreads to create consensus read


microsatellites
microsatellites
genomic DNA
I
SMRTbell library
library insert
II
circular consensus sequencing
III
sequencing read
subreads
circular consensus sequence (CCS)
1,457 CCS:
213 di-, 50 tri-, 28 tetra-, 25 pentanucleotide motifs identified

* single read accuracy ~85%

FASTQ Format – Human readable text format for sequence reads

```
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCACG
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

Header Information:

EAS139 the unique instrument name

136 the run id

FC706VJ the flowcell id

2 flowcell lane

2104 tile number within the flowcell lane

15343 'x'-coordinate of the cluster within the tile

197393 'y'-coordinate of the cluster within the tile

1 the member of a pair, 1 or 2 *(paired-end or mate-pair reads only)*

Y Y if the read is filtered, N otherwise

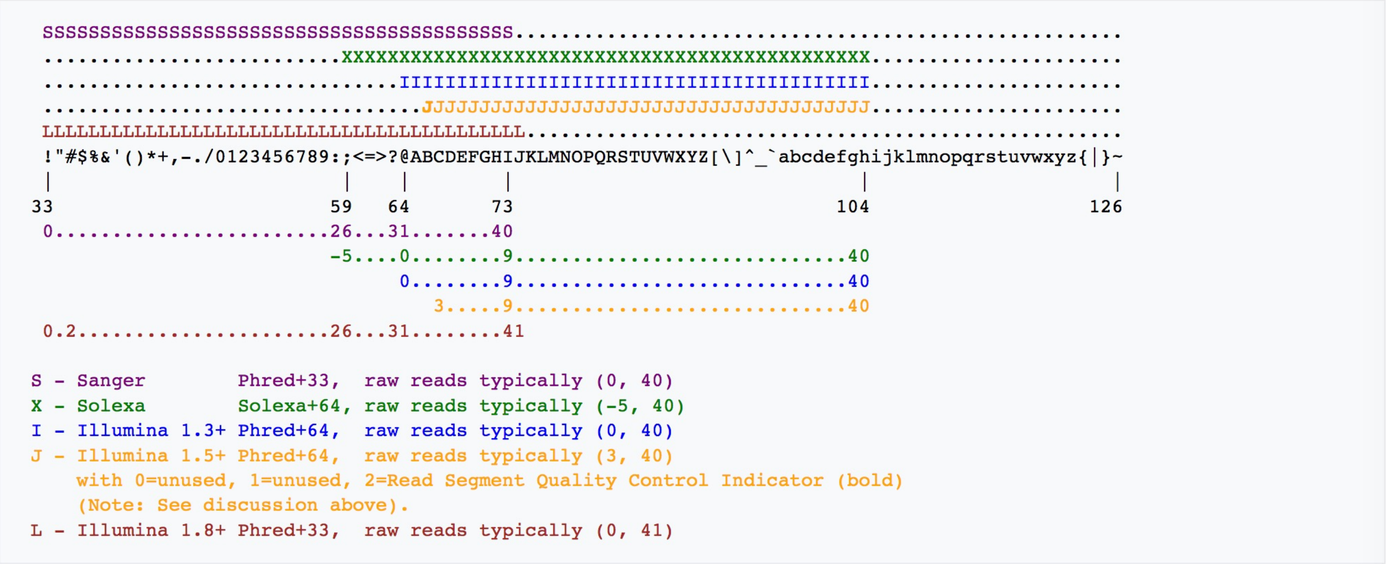
18 0 when none of the control bits are on, otherwise it is an even number

ATCACG index sequence

FASTQ Format – Human readable text format for sequence reads

```
@EAS139:136:FC706VJ:2:2104:15343:197393 1:Y:18:ATCACG
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

Sequence quality information:



```
  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....................................................
  ..........................XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX......................
  ...............................IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII......................
  .................................JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ......................
  LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL....................................................
  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
  |                         |    |        |                              |                     |
 33                        59   64       73                            104                   126
  0........................26...31.......40
                         -5....0........9.............................40
                               0........9.............................40
                                  3.....9.............................40
  0.2......................26...31........41

 S - Sanger        Phred+33,  raw reads typically (0, 40)
 X - Solexa        Solexa+64, raw reads typically (-5, 40)
 I - Illumina 1.3+ Phred+64,  raw reads typically (0, 40)
 J - Illumina 1.5+ Phred+64,  raw reads typically (3, 40)
     with 0=unused, 1=unused, 2=Read Segment Quality Control Indicator (bold)
     (Note: See discussion above).
 L - Illumina 1.8+ Phred+33,  raw reads typically (0, 41)
```

FASTQC – Overrepresented Sequences

- **Assumption:** A normal high-throughput library will contain a diverse set of sequences, with no individual sequence making up more than a tiny fraction of the whole
- **Question:** Are there any (sub-)sequences violating the assumption? If so, we call them overrepresented sequences in the set
- **Why are we interested in these?**
  - Biologically significant?!
  - indicate that the library is contaminated?!
  - indicate that the library is not as diverse as you expected?!
- FASTQC module lists all of the sequence which make up more than 0.1% of the total
- To conserve memory only sequences which appear **in the first 100,000 sequences** are tracked to the end of the file
- duplication detection requires an **exact sequence match** over the whole length of the sequence
- any reads over 75bp in length are **truncated to 50bp**