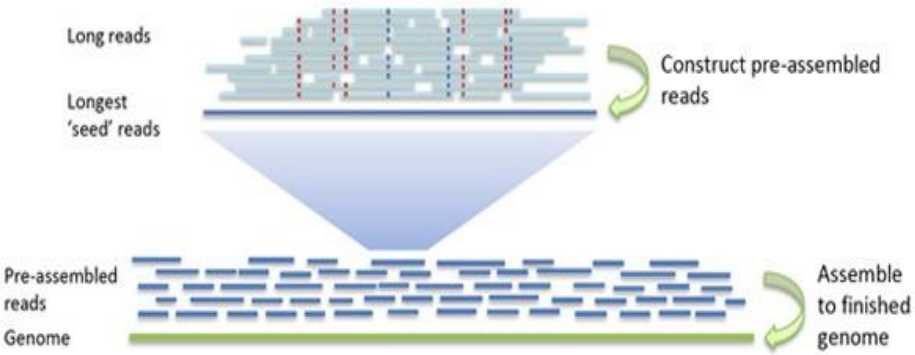
Assembly
Obtención de genomas

- **Assembling reads assisted by a reference genome – Mapping reads (Algorithms)**
- **De novo Assembly – overlapping and graph strategies (Algorithms)**

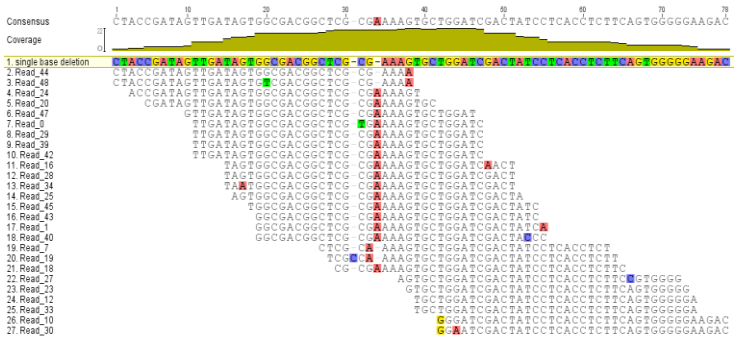
Hay dos problemas fundamentales en el análisis bioinformático de datos de NGS

Ensamblado: armar un genoma de novo, solo a partir de las lecturas obtenidas por secuenciación



Mapeo y llamado de variantes: ubicar las lecturas sobre un genoma de referencia (conocido) y determinar las variantes.

Ensamblar usando el genoma de referencia teniendo en cuenta las variantes



Recreating the genome : Sequence assembly

Sequence assembly: **refers to merging fragments of a much longer DNA sequence in order to reconstruct the original sequence.**

Recreate the genome with no prior knowledge using *de novo* sequence assembly

Recreate the genome using prior knowledge with reference based alignment/mapping

What do we need?

- **Is my data enough?**
- **Computational resources**
- **I don't have close organism**
- **Goals**

- **Close organism to compare with**
- **Quality of reference genome**
- **Goals**

***De novo* short read assembly** is the process whereby we merge together individual sequence reads to form long contiguous sequences **'contigs'**, sharing the same nucleotide sequence as the original template DNA from which the sequence reads were derived.

***Comparative assembly*:** assembling reads against an existing backbone or reference sequence, building a sequence that is similar but not necessarily identical to the backbone sequence.

Assembly

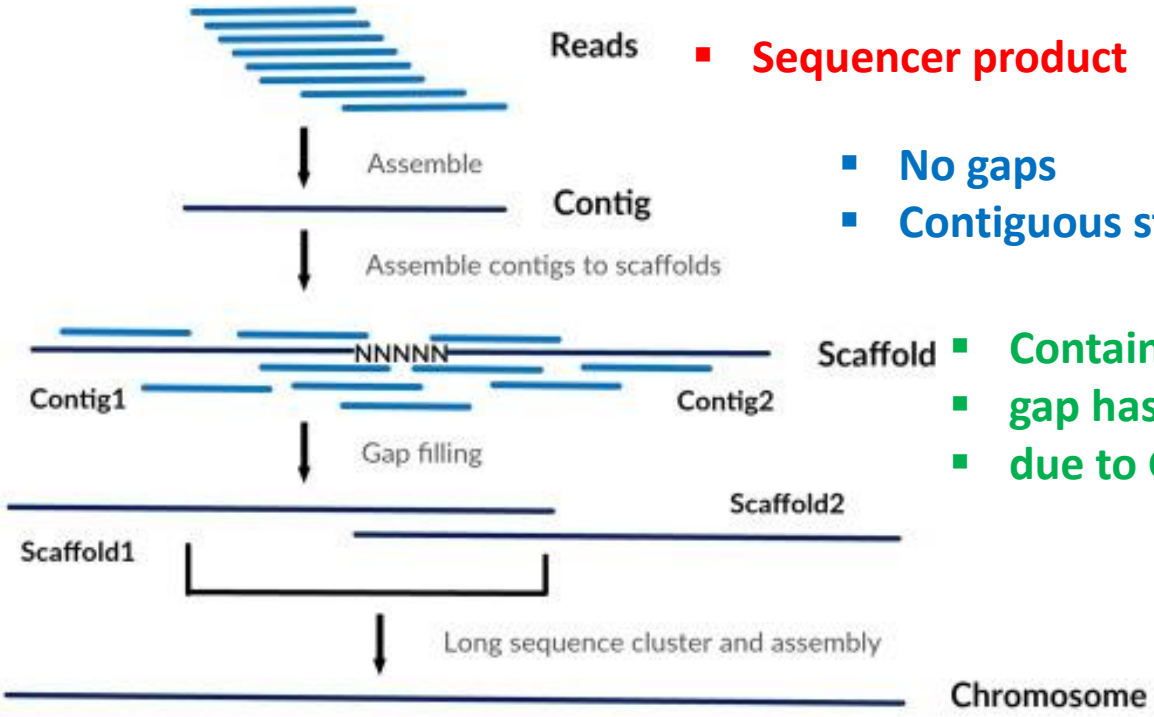
There are different levels of assemblies:

Contigs are continuous stretches of sequence containing only A, C, G, or T bases without gaps.

Scaffolds are created by chaining contigs together using additional information about the relative position and orientation of the contigs in the genome.

¿Cómo reconstruyo el genoma a partir de los fragmentos (reads- lecturas)?

La idea gráficamente..



- **Sequencer product**

- **No gaps**
- **Contiguous stretches**

- **Contain Gaps (missing genomic information)**
- **gap has no relation to the true gap size**
- **due to GC-bias or read-length limitations**

- **Original sequence**

Pasos generales del ensamblado

1. **“Recorte” de las lecturas**
2. **Filtrado y eliminación de lecturas de baja calidad**
3. **Generación de “contigs” a partir de las lecturas (pueden usarse long reads)**
4. **Validación del ensamblado**
5. **“Scaffolding”: Utilización de información de “pair-ends”, mate-pairs**
6. **Finalización: Uso de long reads**

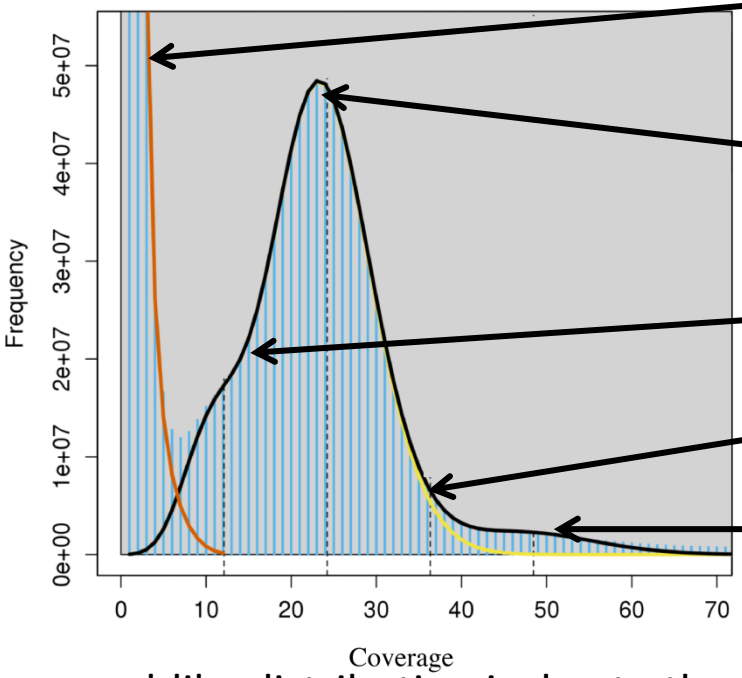
Was our sequencing enough for assembling?

$$\text{Coverage} = \frac{28.000.000 \text{ reads} * 130 \text{ bp}}{100.000.000 \text{ bp}} = 36{,}4X$$

(The number of times a site in the genome is represented by a read)

Is this coverage real?

Coverage kmers distribution follows poisson distribution



PCR errors

homozygous portions of the genome that account for the identical 21-mers from both strands of the DNA

heterozygous portions of the genome

duplicated heterozygous regions

duplicated homozygous regions

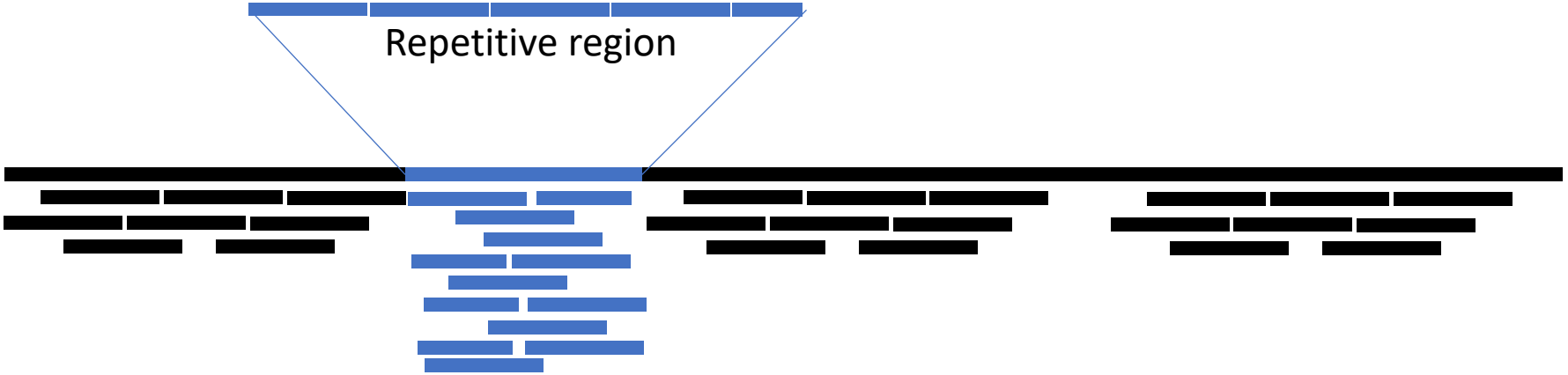
The normal-like distribution is due to the fact that we don't get perfect coverage of the genome. There are some regions with a little less coverage and some regions with a little more coverage but the average coverage depth is around 25.

Common assembly Problems

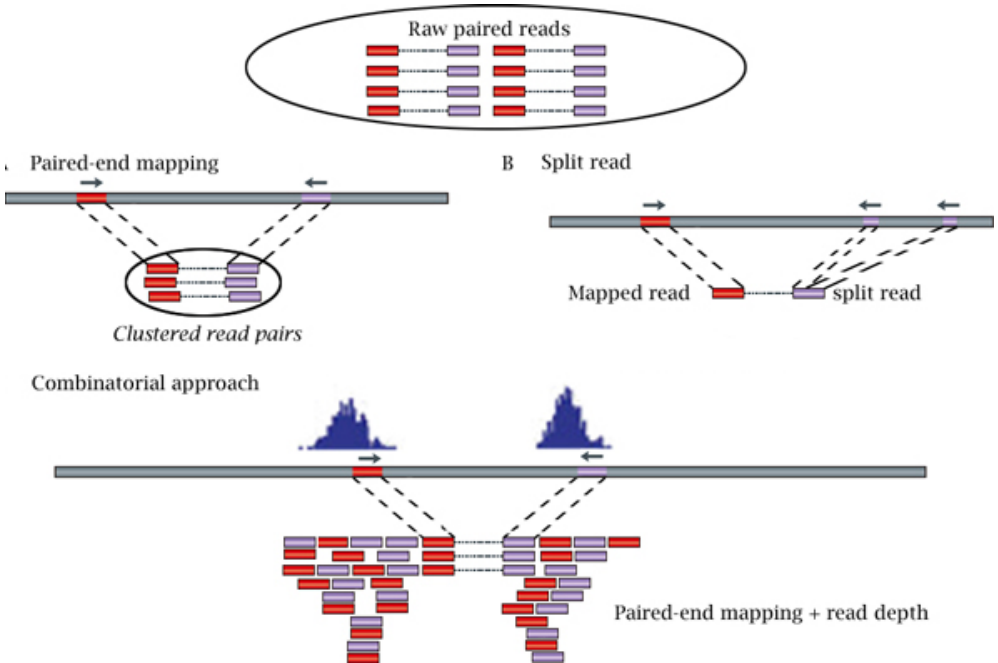
- Errors from sequencing machines, e.g. missing a base, or misreading a base
- Even at 8-10 X coverage, there is a probability that some portion of the genome remains unsequenced
- Repeat problem lead to Misassembly and Gaps
- Chimeric reads - When two fragments from two different parts of genome are combined together

Why repeats are a Problem?

- Ability of an assembly program to produce 1 contig for a chromosome is limited by regions of the genome that occur in multiple near-identical copies throughout the genome (repeats).
- Assembler incorrectly collapses the two copies of the repeat leading to the creation of 2 contigs instead of 1.
- Thus, number of contigs increase with the number of repeats.
- Repeated sequences within a genome also produce problems with higher level ordering.



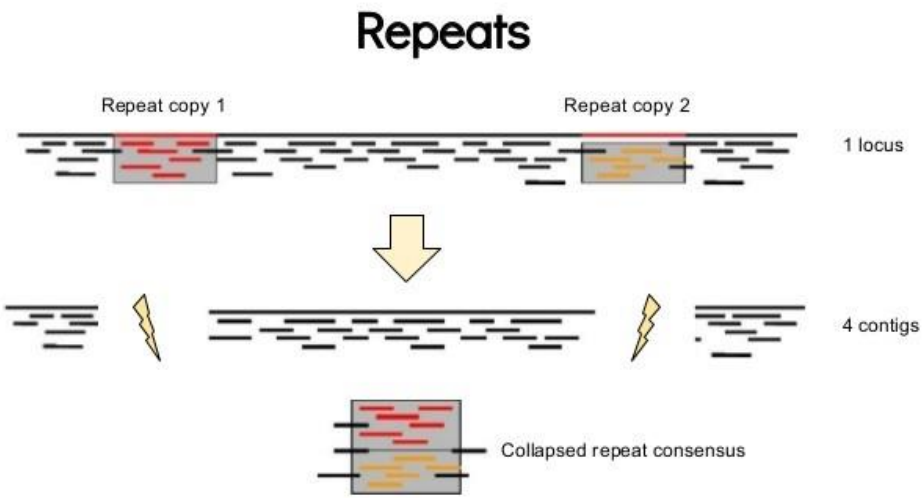
Recordemos el concepto de “Pair ends”



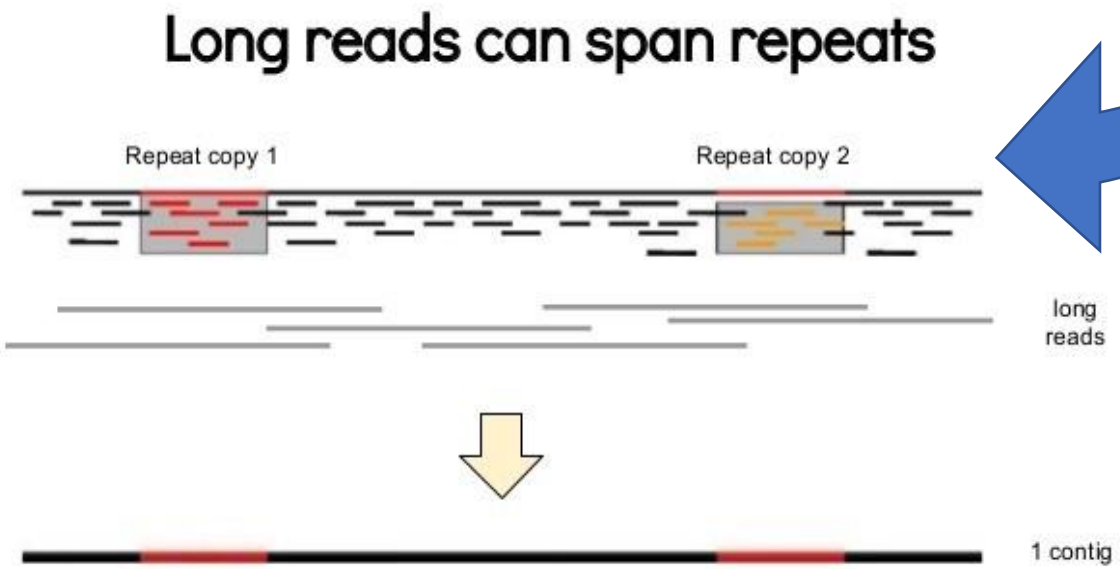
Usar paired-end o mate-pairs puede resolver el problema de las repeticiones siempre y cuando las regiones no sean tandem repeats o regiones muy largas

¿Entonces como resulevo el problema de los repeats?

Ventaja de usar “lecturas largas”



PacBio o Nanopore



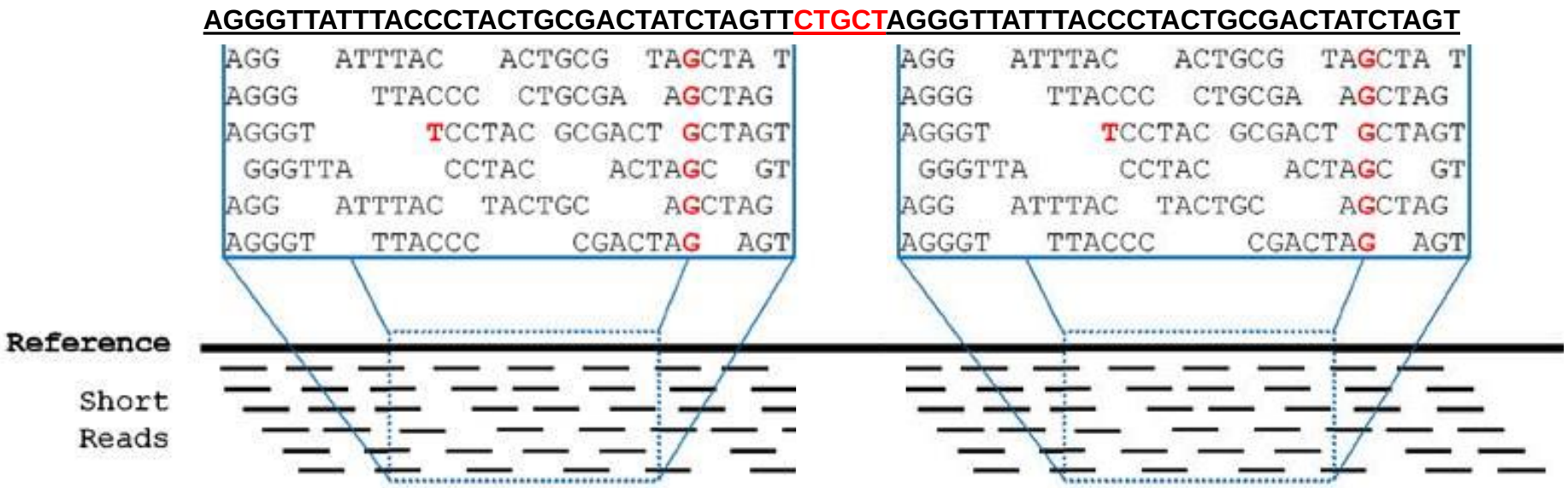
Assembling reads assisted by a reference genome

De novo Assembly - Algorithms

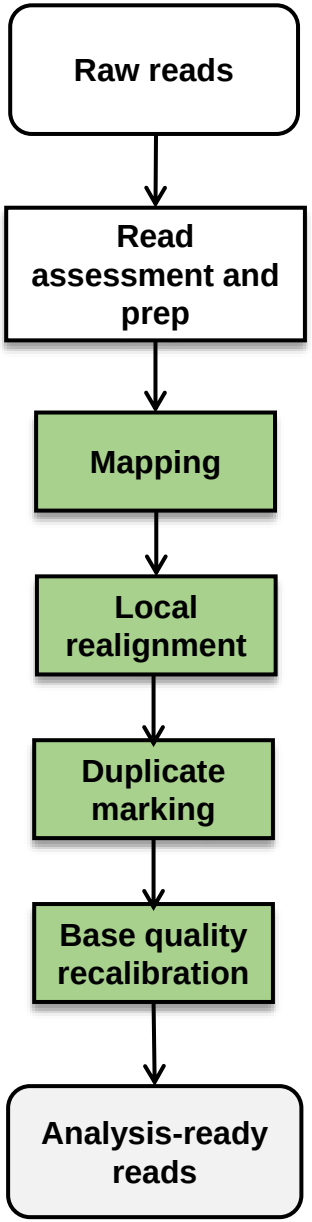
Assembling reads assisted by a reference genome

Assembling reads assisted by a reference genome

Recreate the genome using prior knowledge with reference based alignment/mapping



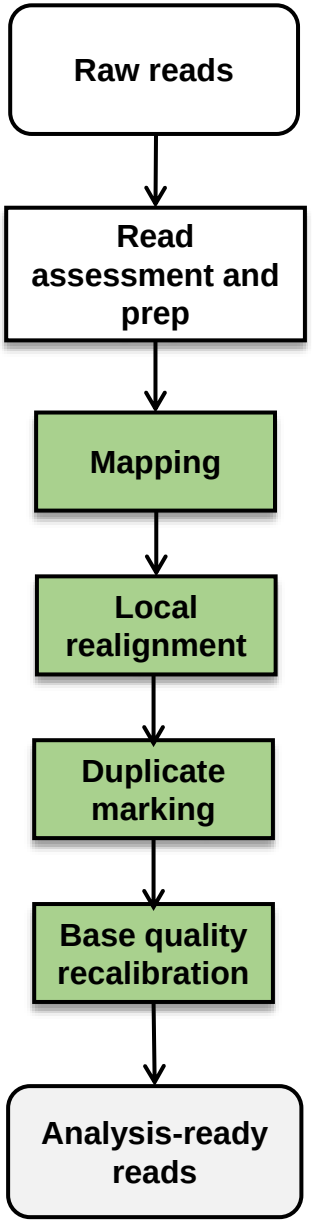
Mapping Reads Back



❖ Algorithms for mapping reads

- **Hash Table (Lookup table)**
  - FAST, but requires perfect matches. [O(m n + N)]
- **Array Scanning**
  - Can handle mismatches, but not gaps. [O(m N)]
- **Dynamic Programming (Smith Waterman)**
  - Indels
  - Mathematically optimal solution
  - Slow (most programs use Hash Mapping as a prefilter) [O(mnN)]
- **Burrows-Wheeler Transform (BW Transform- Ferragina and Manzini matching algorithm)**
  - FAST. [O(m + N)] (without mismatch/gap)
  - Memory efficient.
  - But for gaps/mismatches, it lacks sensitivity

Assembling reads against an existing reference



Short-read Mapping softwares

Software	Technique	Developer	License
Eland	Hashing reads	Illumnia	?
SOAP	Hashing refs	BGI	Academic
Maq	Hashing reads	Sanger (Li, Heng)	GNUPL
Bowtie	BWT	Salzberg/UMD	GNUPL
BWA	BWT	Sanger (Li, Heng)	GNUPL
SOAP2	BWT & hashing	BGI	Academic

Burrows-Wheeler Transform - Ferragina and Manzini matching algorithm

- **Reversible permutation of characters of a string, used originally for compression**
- **Easy to sort and group data**
- **Easy to compress**

We invent a new symbol **$** ("terminator"), defined to be alphabetically less

tan all others:

Burrows-Wheeler Transform - Ferragina and Manzini matching algorithm

- **Reversible permutation of characters of a string, used originally for compression**
- **Easy to sort and group data**
- **Easy to compress**

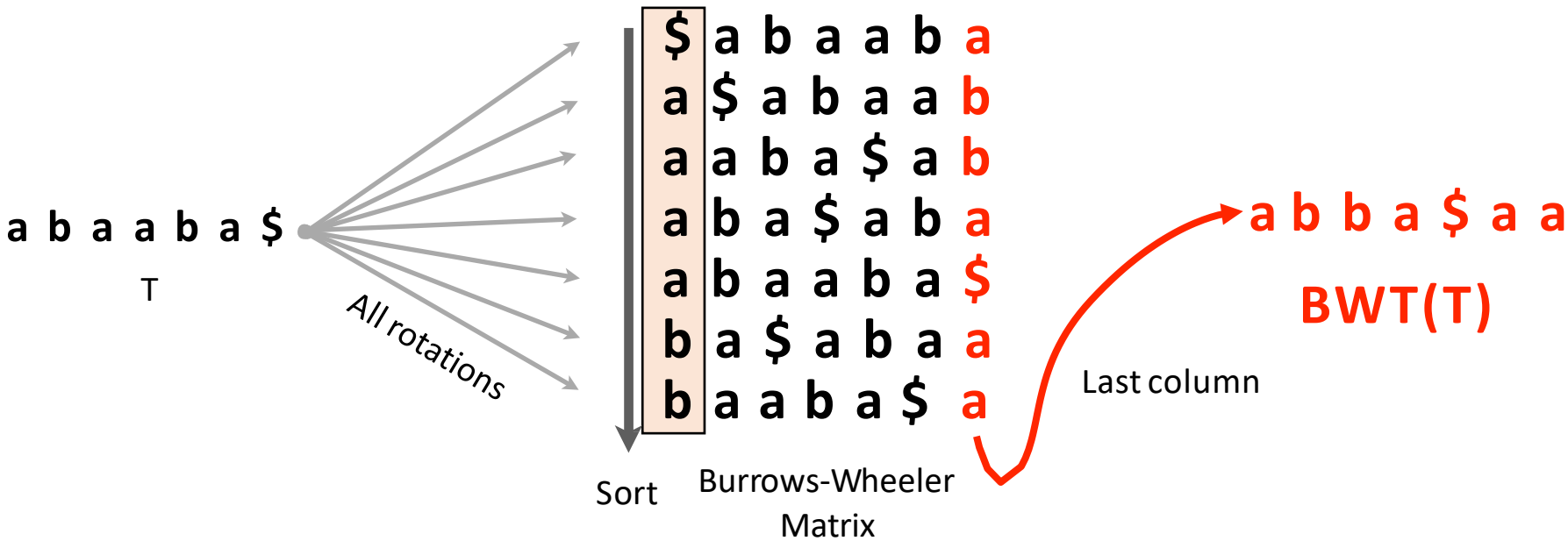
We invent a new symbol **$** ("terminator"), defined to be alphabetically less tan all others:

a b a a b a $

a	b	a	a	b	a	$
b	a	a	b	a	$	a
a	a	b	a	$	a	b
a	b	a	$	a	b	a
b	a	$	a	b	a	a
a	$	a	b	a	a	b
$	a	b	a	a	b	a

Burrows-Wheeler Transform

Last column is the Burrow Wheeler Transform



- Useful for compression
- It becomes an index that allow to acelerate searches
- It is reversible: allow us to retrieve the original string (sequences)

FM Index

FM Index: an index combining the BWT *with a few small auxiliary data structures*

Core of index is ***F*** and ***L*** from BWM:

L is the same size as *T*

F can be represented as array of |Σ| integers

L is compressible (but even uncompressed, it's small compared to suffix array)

We're discarding *T*

F						***L***
$	a	b	a	a	b	**a**
a	$	a	b	a	a	**b**
a	a	b	a	$	a	**b**
a	b	a	$	a	b	**a**
a	b	a	a	b	a	**$**
b	a	$	a	b	a	**a**
b	a	a	b	a	$	**a**

Not stored in index

We only keep the first and last columna of the ordered BWT matrix

FM Index: querying

How to query?

$	a	b	a	a	b	a
a	$	a	b	a	a	b
a	a	b	a	$	a	b
a	b	a	$	a	b	a
a	b	a	a	b	a	$
b	a	$	a	b	a	a
b	a	a	b	a	$	a

FM Index: querying

We can query like the suffix array?

$	a	b	a	a	b	a
a	$	a	b	a	a	b
a	a	b	a	$	a	b
a	b	a	$	a	b	a
a	b	a	a	b	a	$
b	a	$	a	b	a	a
b	a	a	b	a	$	a

6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

We don't have these columns, and we don't have the string T.

Therefore Binary search not posible because we don't have this

FM Index: querying

1. Look for range of rows of BWM(T) that have *P pattern* as a prefix

Start with shortest suffix, then match successively longer suffixes until the range becomes empty

P = **aba**

Fnd all the rows beginning with **a**

F						*L*
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	**$**
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

Subindices in red distinguish which are the same characters in L and F

FM Index: querying

1. Look for range of rows of BWM(T) that have *P pattern* as a prefix

Start with shortest suffix, then match successively longer suffixes until the range becomes empty

P = **aba**

Fnd all the rows beginning with **a**

F						*L*
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	**$**
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

Subindices in red distinguish which are the same characters in L and F

FM Index: querying

We have rows beginning with **a** → now we want rows beginning with **ba**

P = **aba**

If we look at the L column we can see the b preceding the a0,a1,a2,a3

P = **aba**

F						*L*
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	**$**
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

Look at those rows in *L*.

b_0, b_1 are **b**s occuring just to left.

FM Index: querying

We have rows beginning with **a** → now we want rows beginning with **ba**

P = **aba**

If we look at the L column we can see the b preceding the a0,a1,a2,a3

F						L
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	$
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

Look at those rows in *L*.
b_0, b_1 are **b**s occuring just to left.

Use LF Mapping. Let new range delimit those **b**s

P = **aba**

F						L
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	$
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

Now we have the rows with prefix **ba**

Those bs correspond to the bs in F column idex

FM Index: querying

Now we have rows beginning with **ba** → now we seek rows beginning with **aba**

P = **aba**

If we look at the L column we can see the a preceding the b0 and b1

P = **aba**

F						*L*
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	$
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

← a_2, a_3 occur just to left.

FM Index: querying

Now we have rows beginning with **ba** → now we seek rows beginning with **aba**

If we look at the L column we can see the a preceding the b0 and b1

P = **aba**

F						*L*
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	$
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

← a_2, a_3 occur just to left.

Use LF Mapping →

P = **aba**

F						*L*
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	$
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

Now we have the rows with prefix **aba**

Now we now where our reads **aba** match…. Do we?

P = **aba**

Got the same range, [3, 5), we would have got from suffix array

F						L
$	a	b	a	a	b	a_0
a_0	$	a	b	a	a	b_0
a_1	a	b	a	$	a	b_1
a_2	b	a	$	a	b	a_1
a_3	b	a	a	b	a	$
b_0	a	$	a	b	a	a_2
b_1	a	a	b	a	$	a_3

[3, 5)

6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

[3, 5)

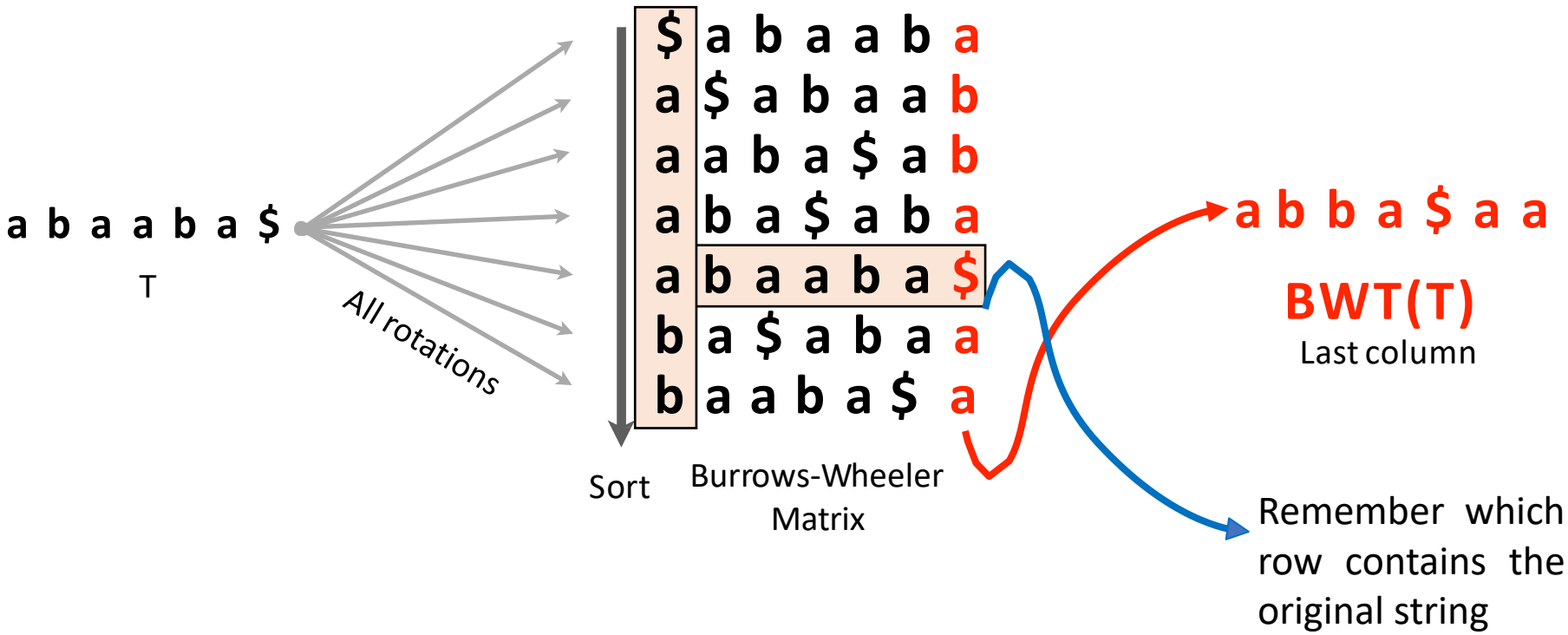
Where are these?

Unlike suffix array, we don't immediately know *where* the matches are in T…

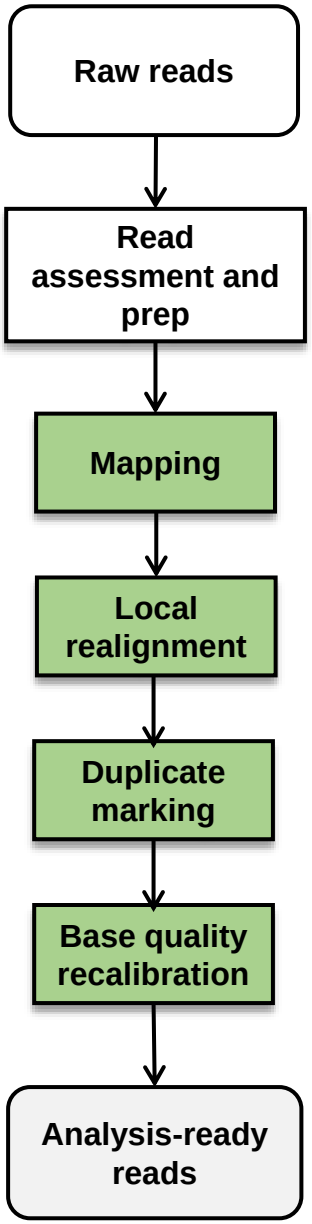
Burrows-Wheeler Transform

- Useful for compression
- It becomes an index that allow to acelerate searches
- **It is reversible**: allow us to **retrieve the original** string (sequences)

Last column is the Burrow Wheeler Transform



Mapping Reads

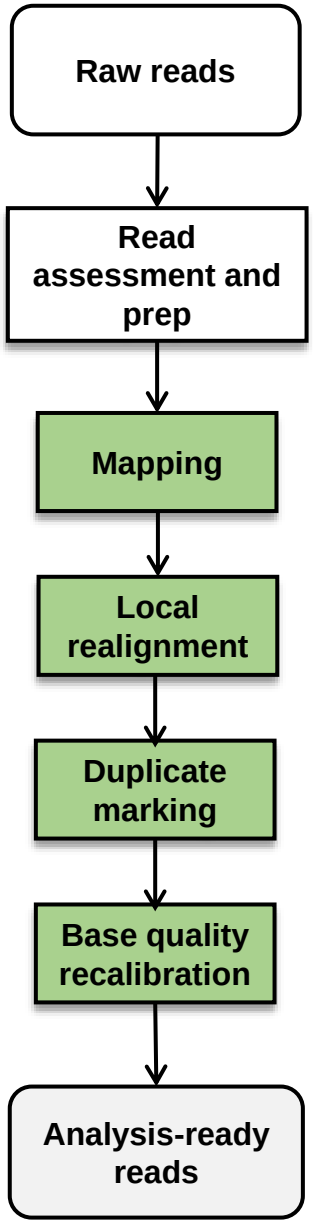


❖ BWA/Bowtie Features

- ❖ Uses Burrows Wheeler Transform
  - ❖ fast
  - ❖ modest memory footprint (<4GB)
- ❖ Accurate
- ❖ Tolerates base mismatches
  - ❖ increased sensitivity
  - ❖ reduces allele bias
- ❖ Gapped alignment for both single- and paired-end reads
- ❖ Automatically adjusts parameters based on read lengths and error rates
- ❖ Native BAM/SAM output (the *de facto* standard)
- ❖ Large installed base, well-supported
- ❖ Open-source (no charge)

Resume of How it works

- Store entire reference genome/Create index.
- Align tag base by base from the end.
- When tag is traversed, all active locations are reported.
- If no match is found, then back up and try a substitution.



Assembling reads against an existing reference

Format Files

- **SAM (Sequence Alignment/Map) format**

Single unified format for storing read alignments to a reference genome

- **BAM (Binary Alignment/Map) format:**

Binary equivalent of SAM – Advantages – Supports indexing – Compact size Remove duplicates • Local realignment • Base quality recalibration

Softwares to manipulate sequence alignment files:

- **SAMtools**
- **BAMtools**
- **PicardTools**
- **BCFtools**

The SAM format and the SAMtools

Aligner output formats

- Most aligners use their own format to output the alignments.
- Hence, downstream tools cannot be exchanged between aligners.

- To resolve this issue, Li et al. have suggested a standardized file format:

  the Sequence Alignment/Map (SAM) format
- SAM is increasingly used in newest tools.
- Converters from legacy formats are included with the SAMtools.

A SAM file

```
[...]

HWI-EAS225_309MTAAXX:5:1:689:1485    0    XIII    863564    25    36M    *
    0    0    GAAATATATACGTTTTTATCTATGTTACGTTATATA
CCCCCCCCCCCCCCCCCCCCCCC4CCCB4CA?AAA<  NM:i:0    X0:i:1    MD:Z:36

HWI-EAS225_309MTAAXX:5:1:689:1485    16    XIII    863766    25    36M    *
    0    0    CTACAATTTTGCACATCAAAAAAGACCTCCAACTAC
=8A=AA784A9AA5AAAAAAAAAAA=AAAAAAAAAA    NM:i:0    X0:i:1    MD:Z:36

HWI-EAS225_309MTAAXX:5:1:394:1171    0    XII    525532    25    36M    *
    0    0    GTTTACGGCGTTGCAAGAGGCCTACACGGGCTCATT
CCCCCCCCCCCCCCCCCCCCCC?CCACCACA7?<???    NM:i:0    X0:i:1    MD:Z:36

HWI-EAS225_309MTAAXX:5:1:394:1171    16    XII    525689    25    36M    *
    0    0    GCTGTTATTTCTCCACAGTCTGGCAAAAAAAAGAAA    7AAAAAA?
AA<AA?AAAAA5AAA<AAAAAAAAAAAA    NM:i:0    X0:i:1    MD:Z:36

HWI-EAS225_309MTAAXX:5:1:393:671    0    XV    440012    25    36M    *
    0    0    TTTGGTGATTTTCCCGTCTTTATAATCTCGGATAAA
AAAAAAAAAAAAAAAA<AAAAAAAA<AAAA5<AAAA3    NM:i:0    X0:i:1    MD:Z:36

HWI-EAS225_309MTAAXX:5:1:393:671    16    XV    440188    25    36M    *
    0    0    TCATAGATTCCATATGAGTATAGTTACCCCATAGCC    ?9A?A?CC?
<ACCCCCCCCCCCCCCCCCCACCCCCCC    NM:i:0    X0:i:1    MD:Z:36

[...]
```

The SAM format

A SAM file consists of two parts:

- Header
  - contains meta data (source of the reads, reference genome, aligner, etc.)
  - Most current tools omit and/or ignore the header.
  - All header lines start with "@".
  - Header fields have standardized two-letter codes for easy parsing of the information
- Alignment section
  - A tab-separated table with at least 11 columns
  - Each line describes one alignment

SAM format: Alignment section

The columns are:

- QNAME: ID of the read ("query")
- FLAG: alignment flags
- RNAME: ID of the reference (typically: chromosome name)
- POS: Position in reference (1-based, left side)
- MAPQ: Mapping quality (as Phred score)
- CIGAR: Alignment description (gaps etc.) in CIGAR format
- MRNM: Mate reference sequence name [for paired end data]
- MPOS: Mate position [for paired end data]
- ISIZE: inferred insert size [for paired end data]
- SEQ: sequence of the read
- QUAL: quality string of the read
- extra fields

SAM format: Flag and extra fields

FLAG field: A number, encoding

- whether the read is from a paired-end run, and if so, which one
- if so, whether the read and/or its mate are mapped
- whether the read mapped to the forward or the reverse strand
- whether the read passed platform quality checks
- [and a few more things]

Extra fields:

- Always triples of the format TAG : VTYPE : VALUE
- may encode number of mismatches ("NM"), number of alignments for the same read, extra informations on quality, aligner-specific data etc.

SAM format: extended CIGAR strings

- Alignments contain gaps (e.g., in case of an indel, or, in RNA-Seq, when a read straddles an intron).
- Then, the CIGAR string gives details. Example: "M10 I4 M4 D3 M12" means

- the first 10 bases of the read map ("M10") normally (not necessarily perfectly)
- then, 4 bases are inserted ("I4"), i.e., missing in the reference
- then, after another 4 mapped bases ("M4"), 3 bases are deleted ("D4" i.e., skipped in the query.
- Finally, the last 12 bases match normally.

There are further codes (N, S, H, P), which are rarely used.

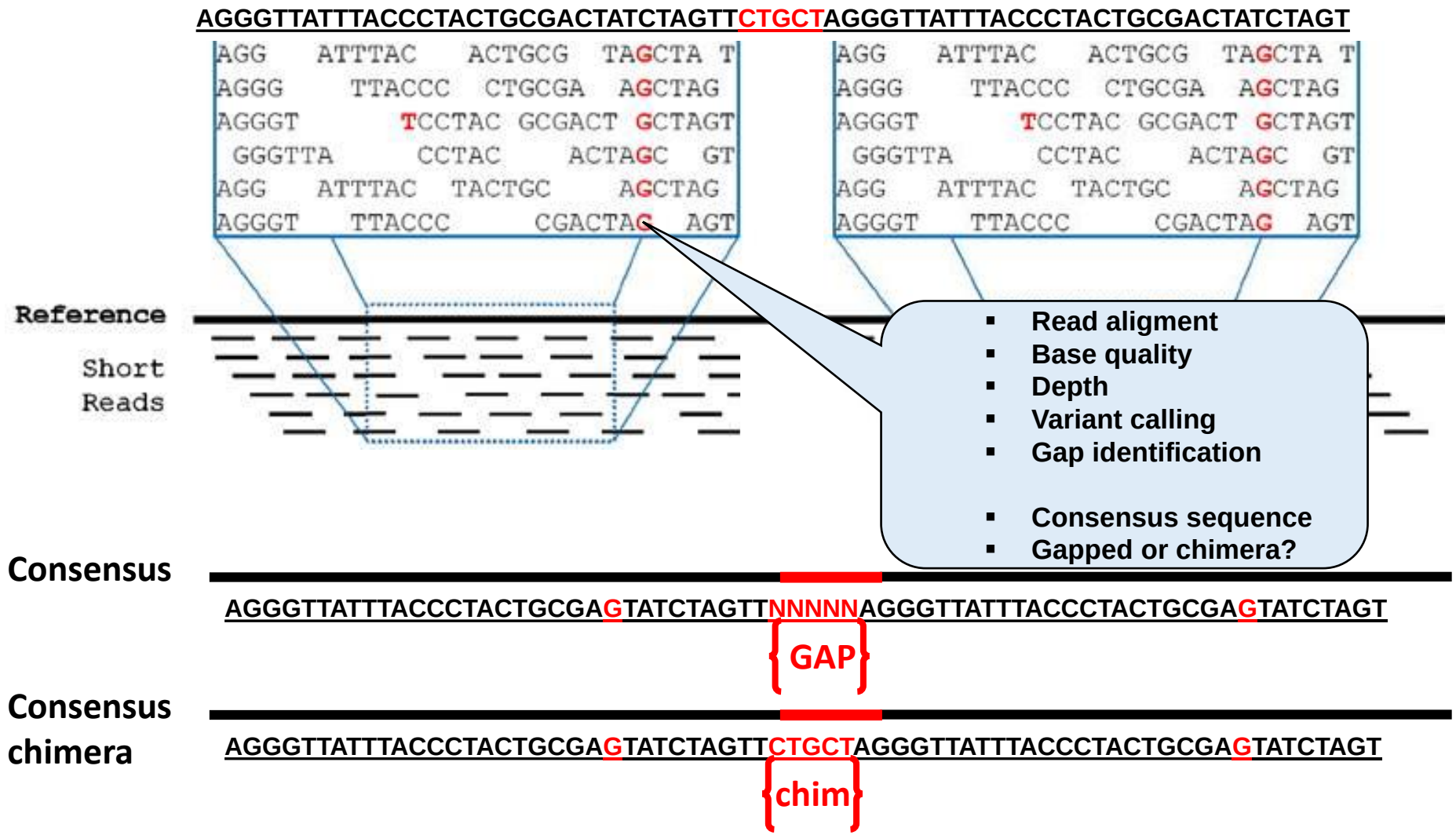
SAMtools

- The SAMtools are a set of simple tools to
  - convert between SAM and BAM
    - SAM: a human-readable text file
    - BAM: a binary version of a SAM file, suitable for fast processing
  - sort and merge SAM files
  - index SAM and FASTA files for fast access
  - view alignments ("tview")
  - produce a "pile-up", i.e., a file showing
    - local coverage
    - mismatches and consensus calls
    - indels

- The SAMtools C API facilitates the development of new tools for processing SAM files.

Final step: obtaining the consensus sequence

consensus sequence: sequence derived from the multiple alignment of reads in a contig



FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS
DEPARTAMENTO DE BIOTECNOLOGÍA Y TECNOLOGÍA ALIMENTARIA
UNIVERSIDAD ARGENTINA DE LA EMPRESA

Bioinformática

ANÁLISIS COMPUTACIONAL DE SECUENCIAS

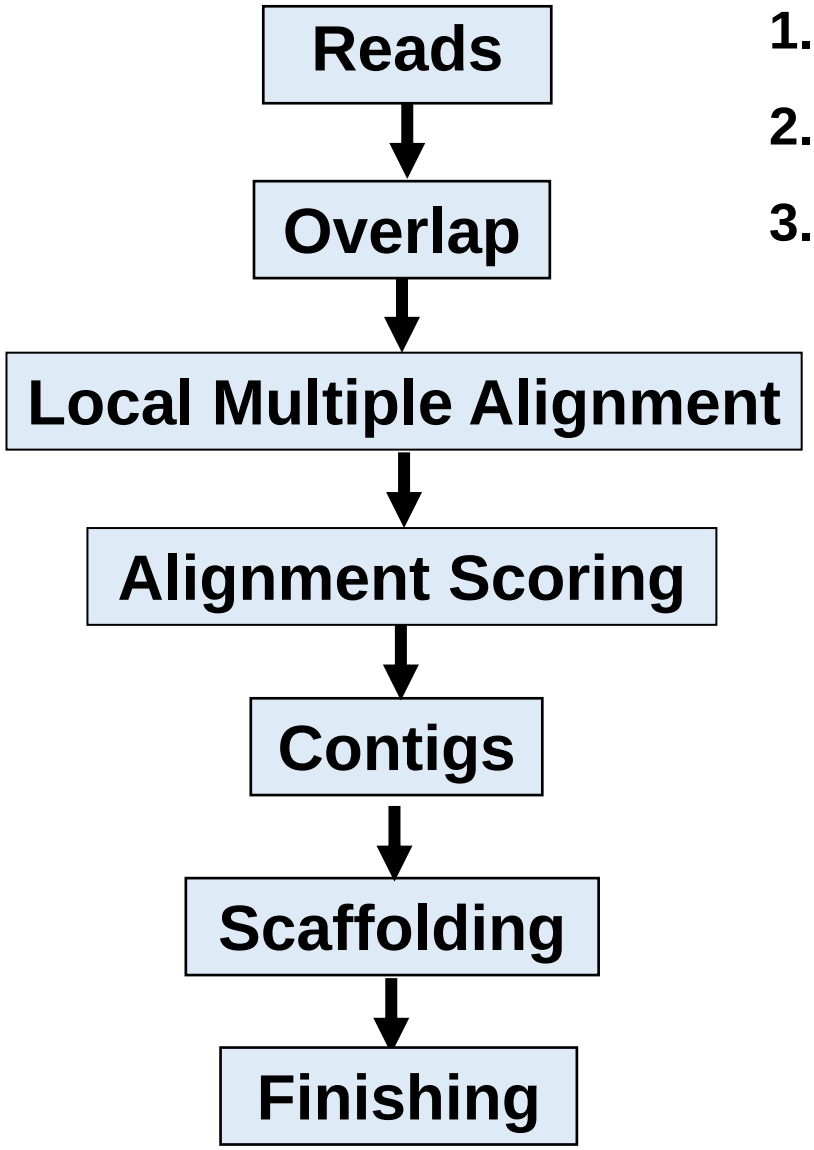
Lucas L. Maldonado (PhD)
Lic. en Biotecnología y Biología Molecular
PDRA Bioinformática y Genómica

CONICET
Instituto de investigaciones en Microbiología y Parasitología Medica – Fac. de Medicina - UBA
Instituto Multidisciplinario – Fac. de Ciencias Exactas y Naturales – UBA

lucamaldonado@uade.edu.ar
lmaldonado@fmed.uba.ar
lucas.l.maldonado@gmail.com.ar

De novo Assembly - Algorithms

De novo Assembly - Algorithms



1. Greedy Algorithm
2. Overlap-Layout-Consensus Algorithm
3. Bruijn graph Algorithm

Assembly Problems:

- Repeats
- Chimerism
- Gaps
- Computational requirements (RAM memory)

Methods

Different approaches	Basic idea
▪ **greedy assembly**	▪ Find all overlaps between reads
▪ **Overlap-layout-consensus**	▪ Build a graph based on overlaps
▪ **de bruijn graphs**	▪ Simplify the graph (sequencing errors)

Challenges

- ❖ Sequencing error
- ❖ Complexity reducing
- ❖ Repeat resolving
- ❖ Uneven depth
- ❖ RAM memory

- ❖ NGS platform and library preparation method
- ❖ Topological complexity of repetitive elements in genomes
- ❖ results from polymerase chain reaction (PCR), cloning, extreme GC bias, sequencing errors and copy number variations

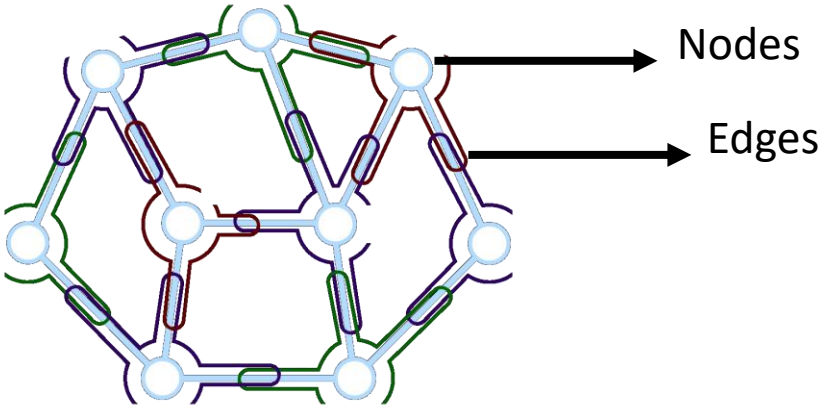
http://www.langmead-lab.org/teaching-materials/

Graphs: are mathematical structures used to model pairwise relations between objects. In an assembly the relation is the overlaps between sequences of symbols

NGS assemblers can be organized into three categories, all based on graphs:

1. **The Overlap/Layout/Consensus (OLC) methods rely on an overlap graph.**
2. **The de Bruijn Graph (DBG) methods use some form of K-mer graph.**
3. **The greedy graph algorithms may use OLC or DBG.**

A graph is an abstraction used widely in computer science. It is a set of nodes plus a set of edges between the nodes.
If the edges may only be traversed in one direction, the graph is known as a directed graph. The graph can be conceptualized as balls in space with arrows connecting them.
Collections of edges form paths that visit nodes in some order, such that the sink node of one edge forms the source node for any subsequent nodes.
A special kind of path, called a simple path, is one that contains only distinct nodes (each node is visited at most once).



Nodes are related with other nodes through the edges under a particular characteristic.

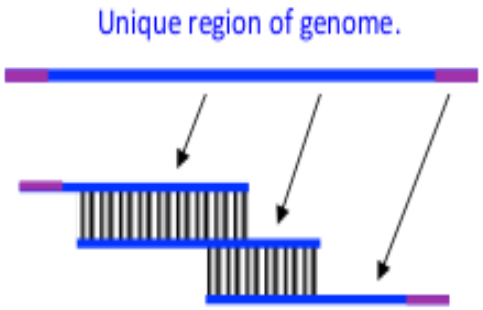
Graphs try to find the shortest or fastest pathway comunicating the Nodes in order to find a solution to our problem

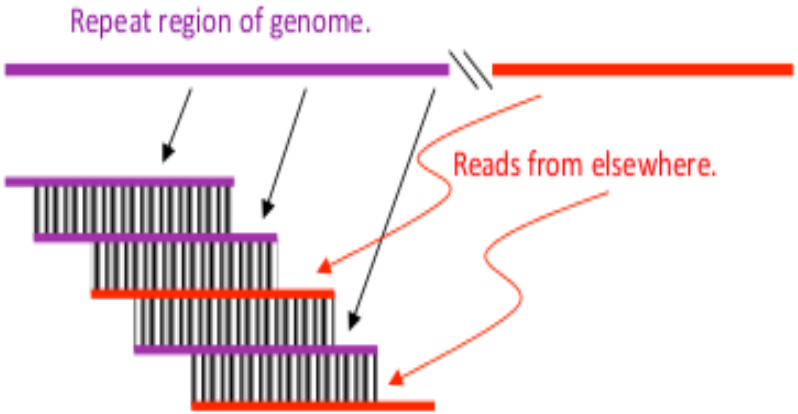
El concepto de overlap

Los falsos positivos pueden producirse por azar o por repeticiones.

Estos se pueden evitar aumentando la astringencia:

- Las superposiciones deben ser suficientemente largas
- La similitud de las secuencias debe ser alta (identity threshold)
- Los solapamientos deben incluir ambos extremos de las lecturas
- Ignorar los solapamientos en alta frecuencia (suelen representar regiones repetitivas)

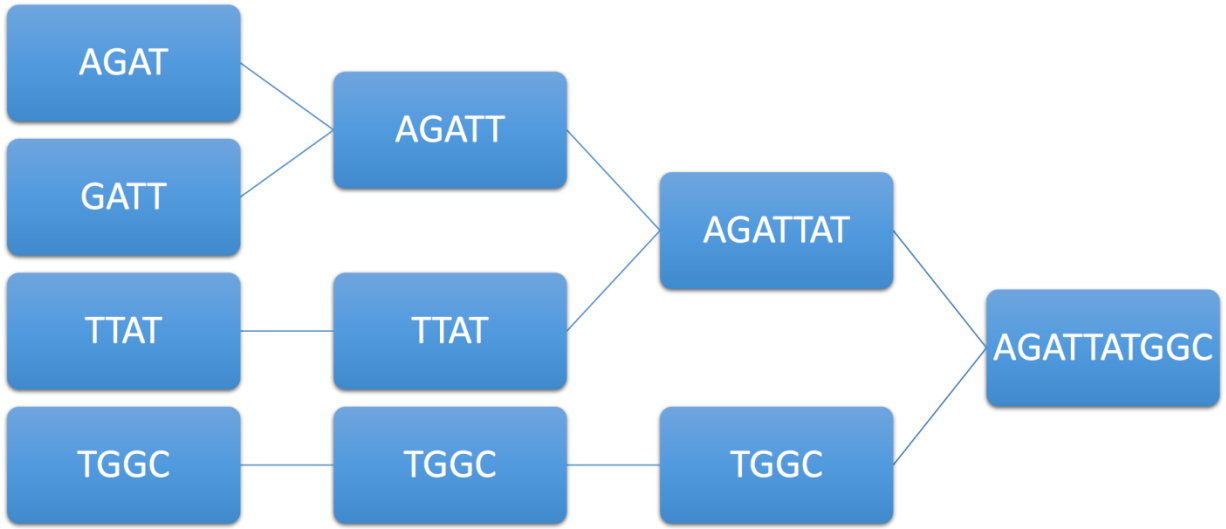




De novo Assembly

Greedy algorithm

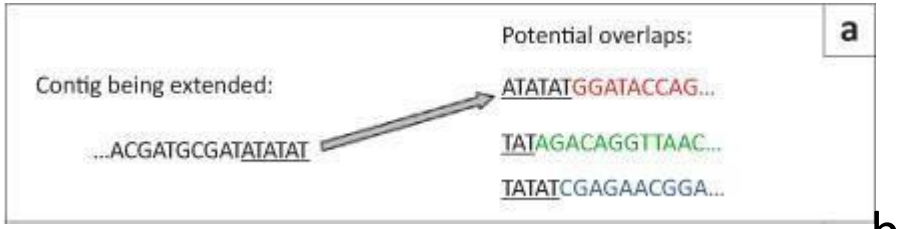
Assume that the reads are perfectly clean (no errors in the characters of the read). Repeatedly it finds a pair of sequences that have a large amount of overlap and merges the two sequences into one longer sequence. The sequences are initialized to be the observed reads.

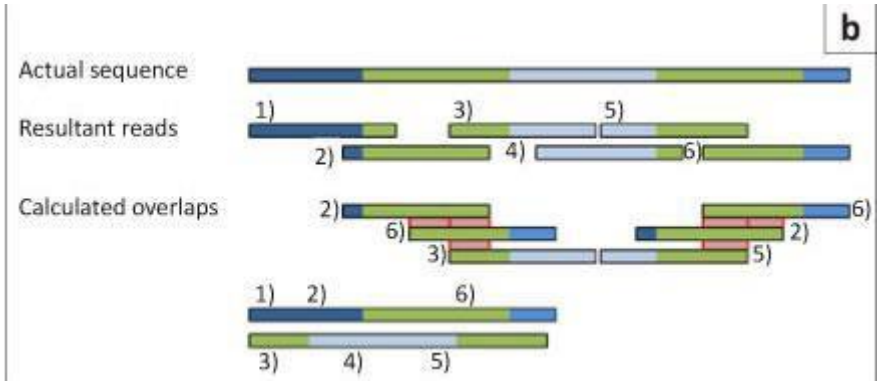


Greedy approach: We pick two strings of sequences with largest overlap and replace them with their merged overlapping. Stop when there is only one string left.

Greedy

El algoritmo “voraz” une una lectura con la lectura con el mejor puntaje de alineamiento hasta que no se puedan unir más lecturas





Limitaciones del método

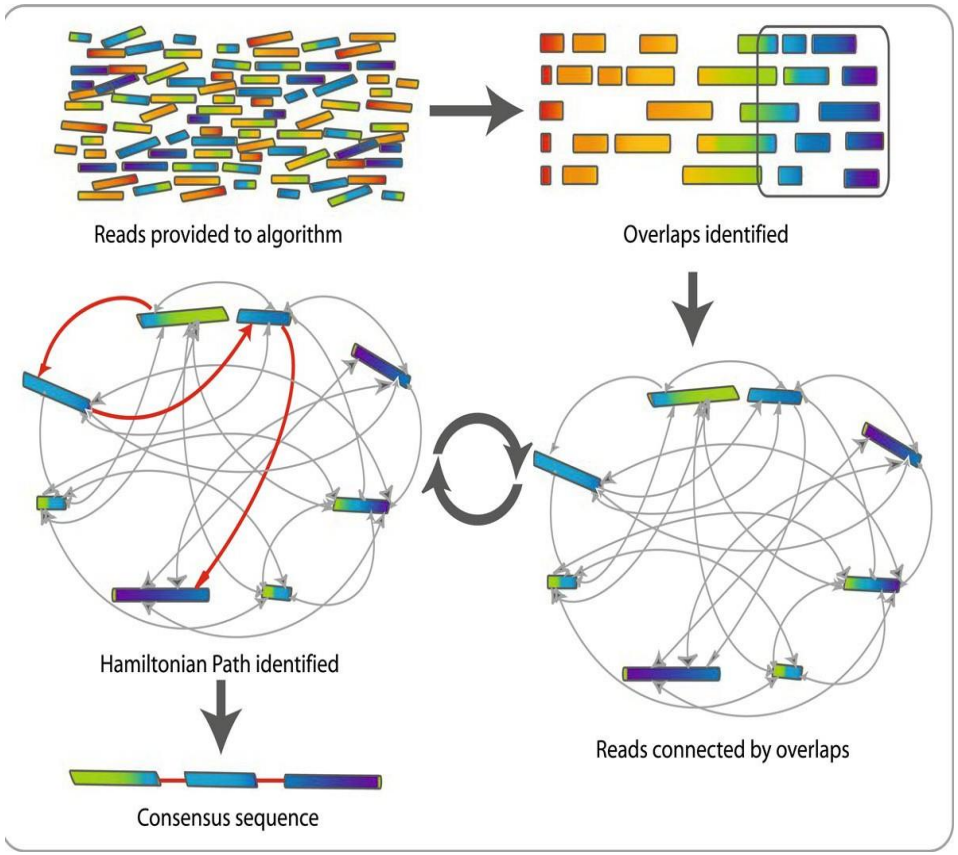
a) Elementos repetitivos comunes a lo largo del genoma conducen a puntajes altos de alineamiento y por consiguiente unión de secuencias no relacionadas.

b) Regiones duplicadas pueden producir una quimera cuando el alineamiento de dos lecturas pertenecientes a los extremos opuestos de las duplicaciones poseen un mejor puntaje que las lecturas correctas.

De novo Assembly

El método OLC genera un grafo utilizando lecturas y solapamientos. **Los nodos del grafo son las lecturas** y **las aristas las superposiciones de las lecturas**. De esta manera, el proceso de ensamblado consiste en encontrar el camino a través del grafo que visite todos los nodos una única vez.

Overlap-Layout-consensus (OLC)



Overlap : In the first step an overlap graph is created by joining all the reads by their respective best overlapping reads.

Layout : merge reads into contigs and simplify the graph (ex: Identify unique contigs).

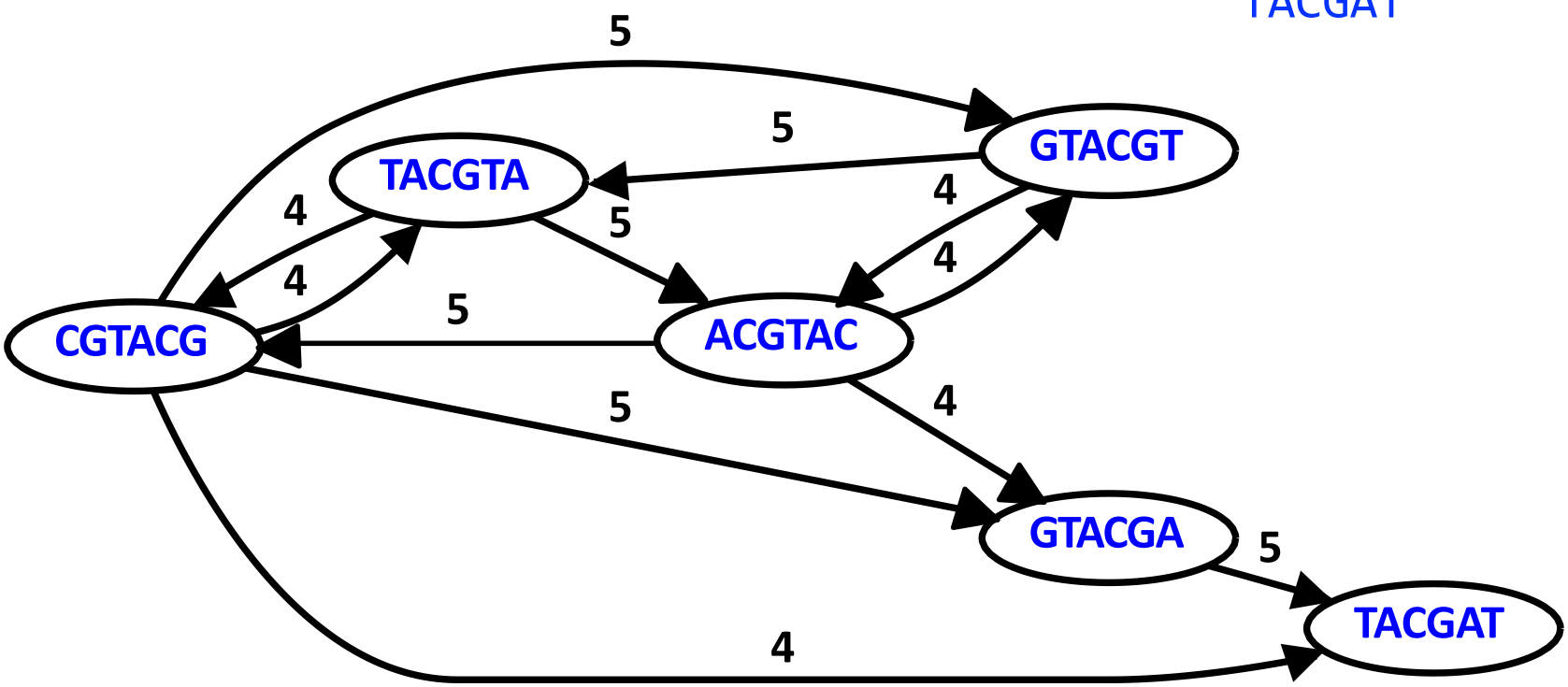
Consensus : Derive the DNA sequence and correct read errors. Groups of reads that overlapped now cast their votes in order to identify which base should be present at a particular location of the novel genome.

Overlap graph

K-mer=6
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA
TACGAT

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4



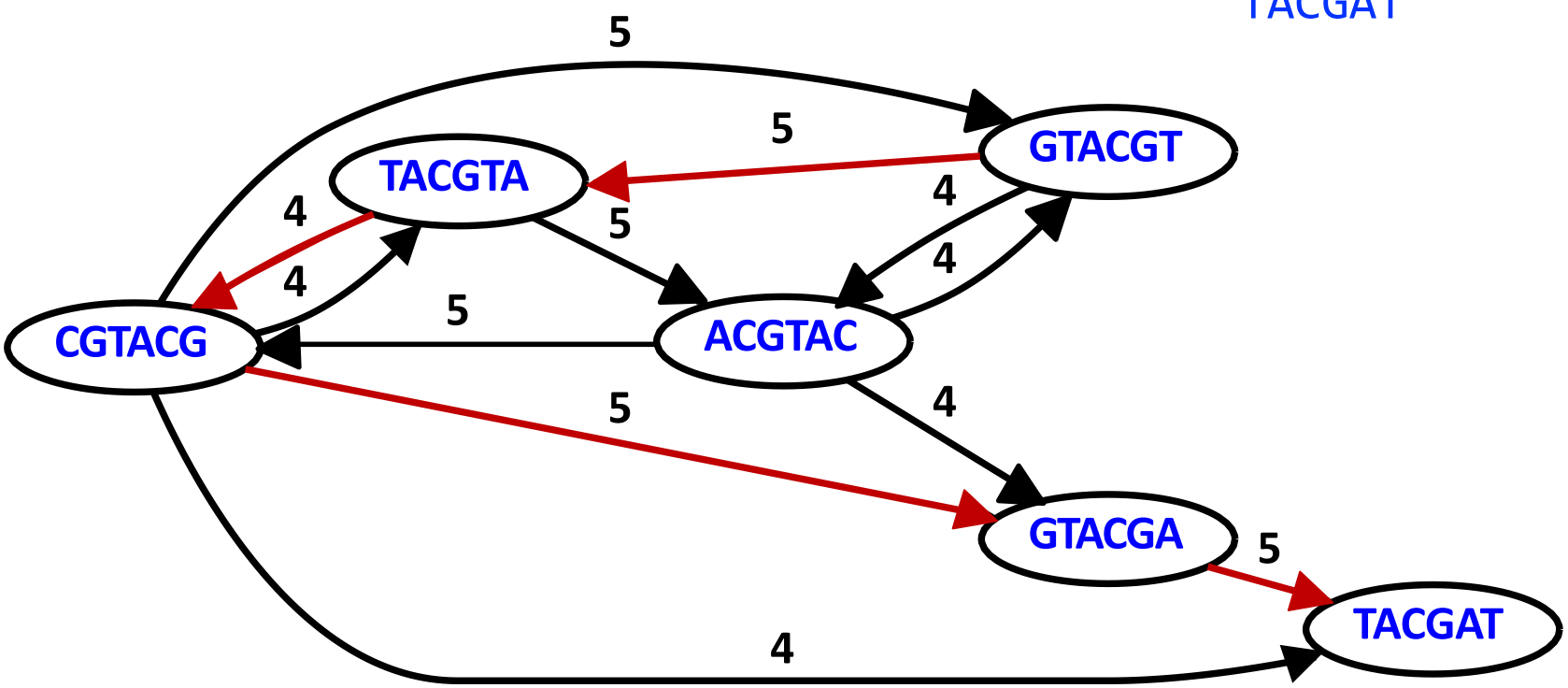
The highest number of overlapping characters defines the pathways through the graph

Overlap graph

K-mer=6
GTACGT
TACGTA
ACGTAC
CGTACG
GTACGA
TACGAT

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4



The highest number of overlapping characters defines the pathways through the graph

De novo Assembly

Overlap graph

Idea: pick order for strings in *S* ***and*** construct superstring

order 1: AAA AAB ABA ABB BAA BAB BBA BBB

AAABABBAABABBABBB ⟵ superstring 1

order 2: AAA AAB ABA BAB ABB BBB BAA BBA

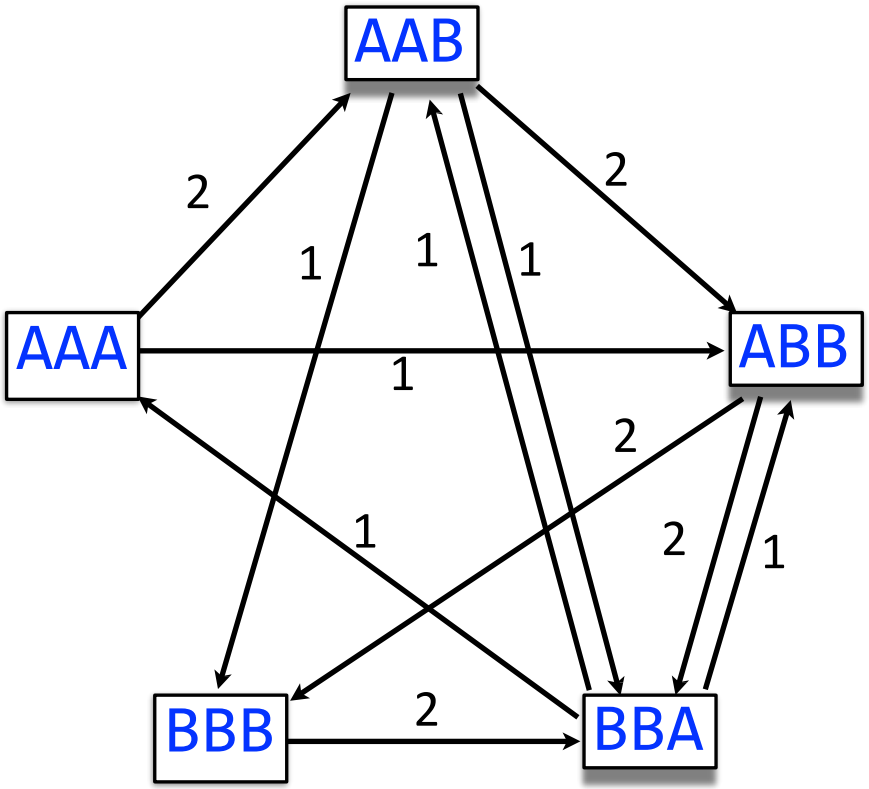
AAABABBBAABBA ⟵ superstring 2

Try all possible orderings and pick shortest superstring

If *S* contains *n* strings, *n* !(*n* factorial) orderings possible

De novo Assembly

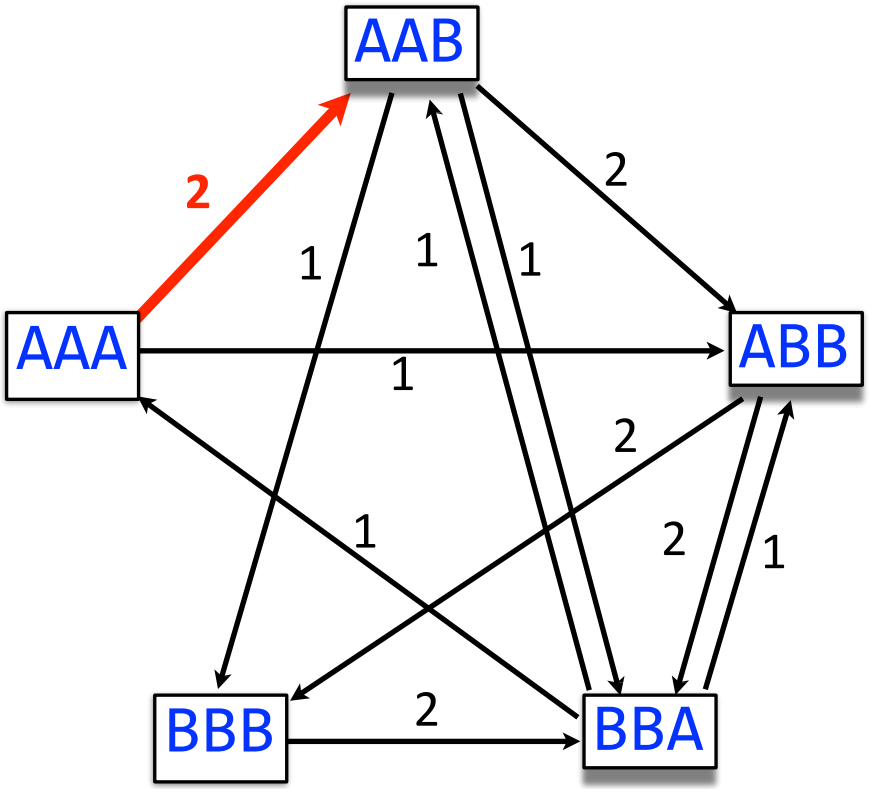
Overlap graph



Greedy shortest common superstring

De novo Assembly

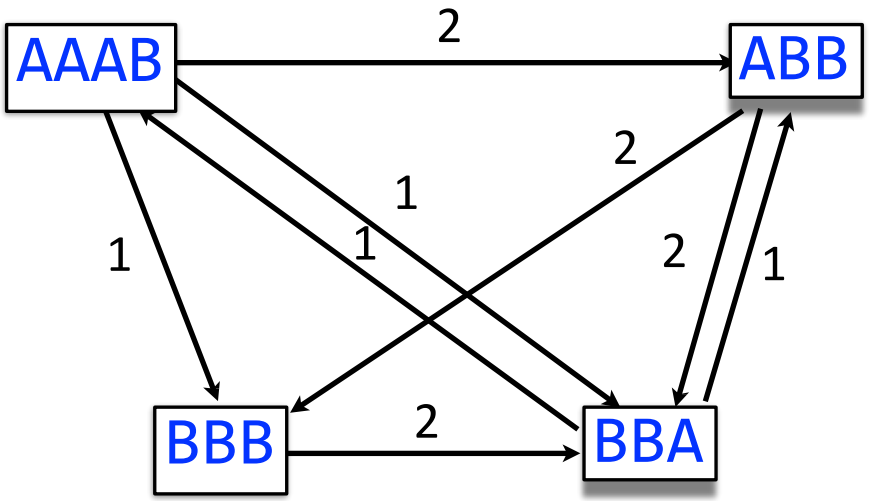
Overlap graph



Greedy shortest common superstring

De novo Assembly

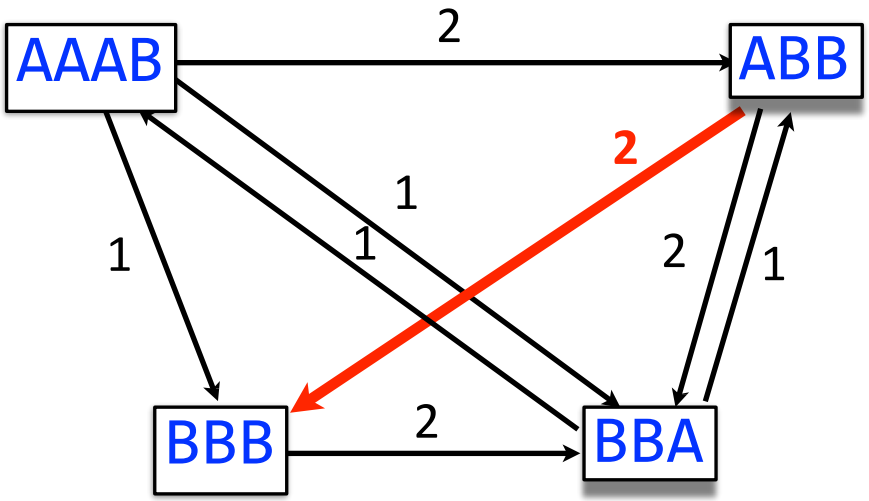
Overlap graph



Greedy shortest common superstring

De novo Assembly

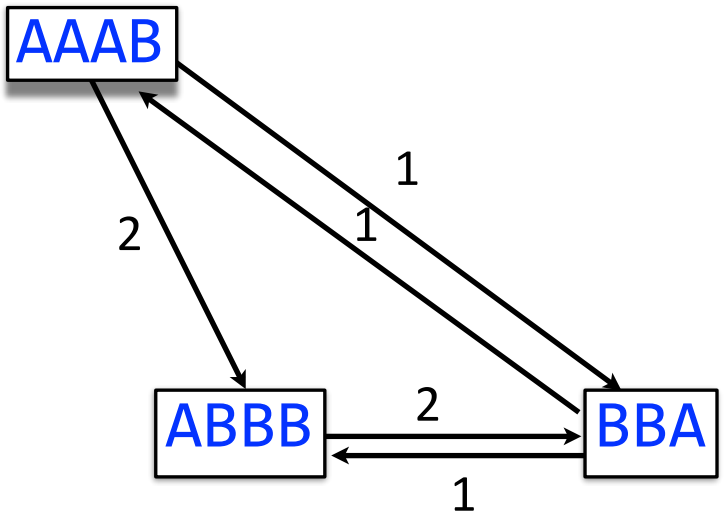
Overlap graph



Greedy shortest common superstring

De novo Assembly

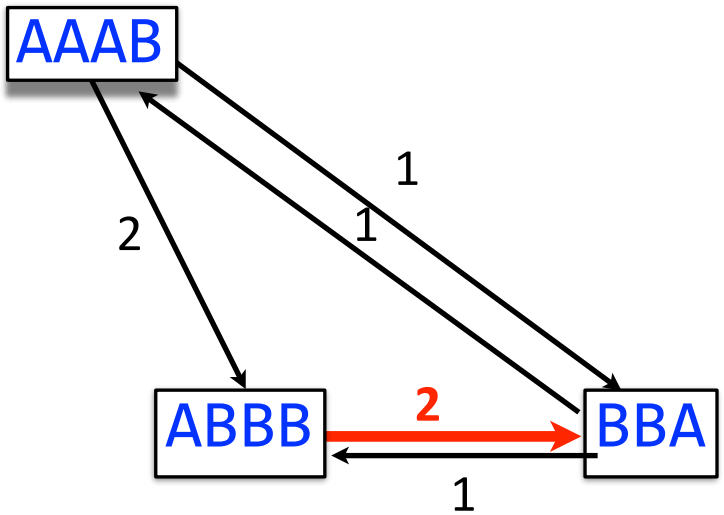
Overlap graph



Greedy shortest common superstring

De novo Assembly

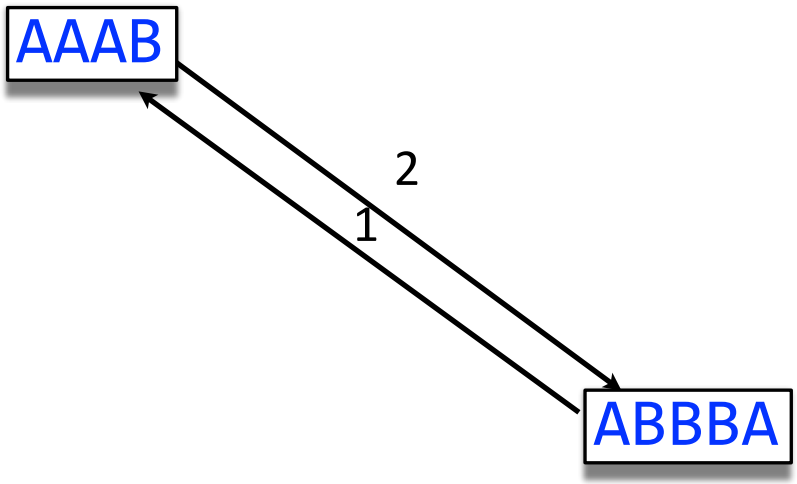
Overlap graph



Greedy shortest common superstring

De novo Assembly

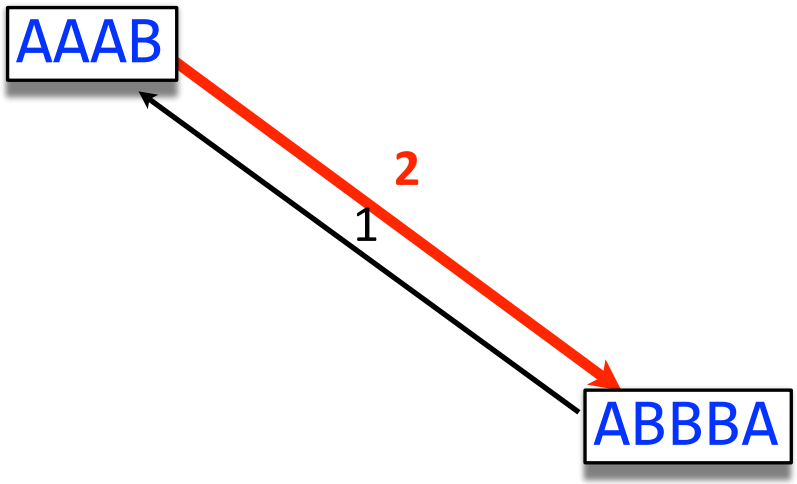
Overlap graph



Greedy shortest common superstring

De novo Assembly

Overlap graph



Greedy shortest common superstring

De novo Assembly

Overlap graph

AAABBBA ← superstring, length=7

De novo Assembly

Overlap graph

AAA AAB ABB BBA BBB

AAAB ABB BBA BBB

AAAB ABBA BBB

AAABBA BBB

AAABBABBB ← superstring, length=9

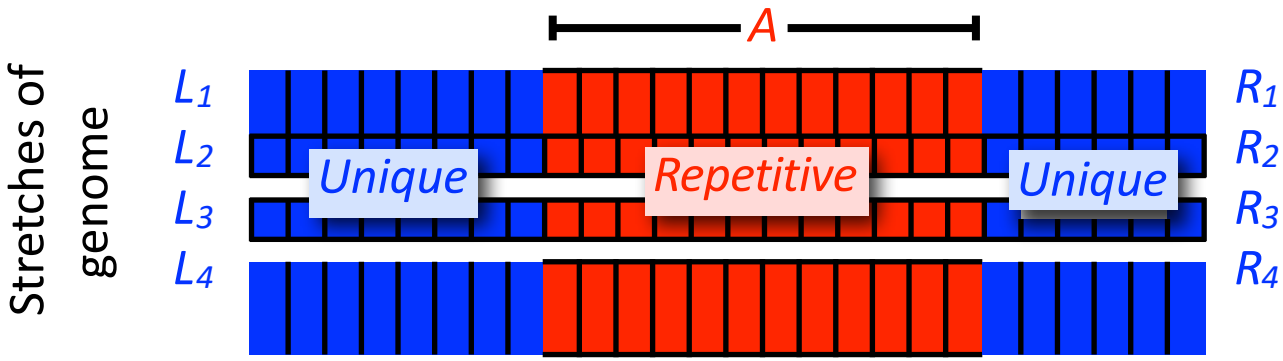
AAABBBA ← superstring, length=7

Greedy answer *isn't necessarily optimal*

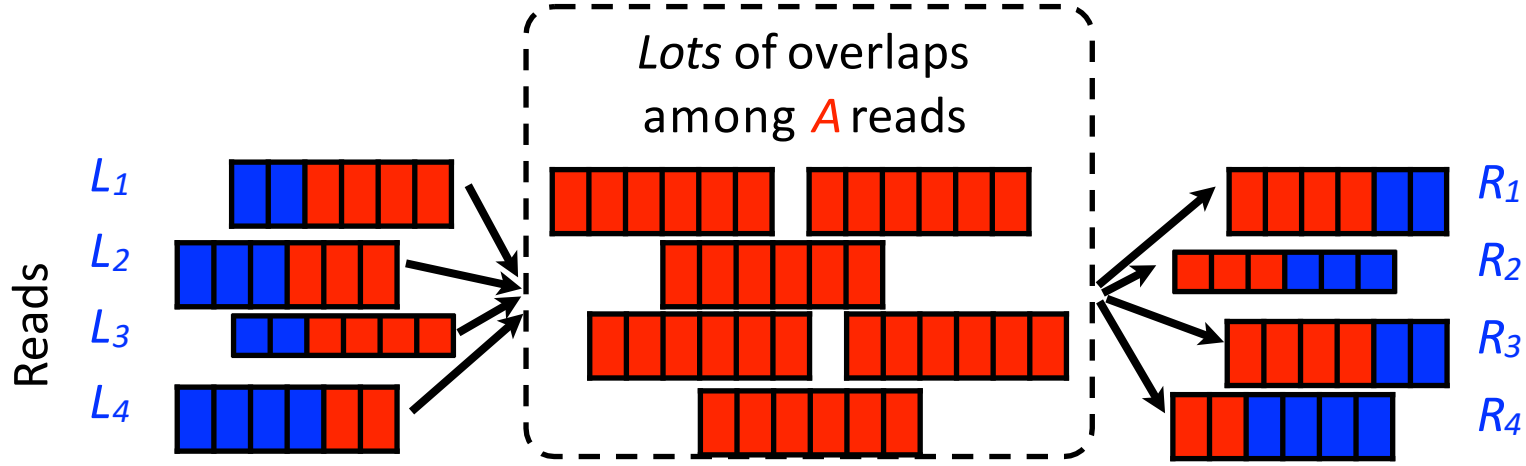
De novo Assembly

Repeats foil assembly

Portion of overlap graph involving repeat family *A*



*A*s are longer than read length



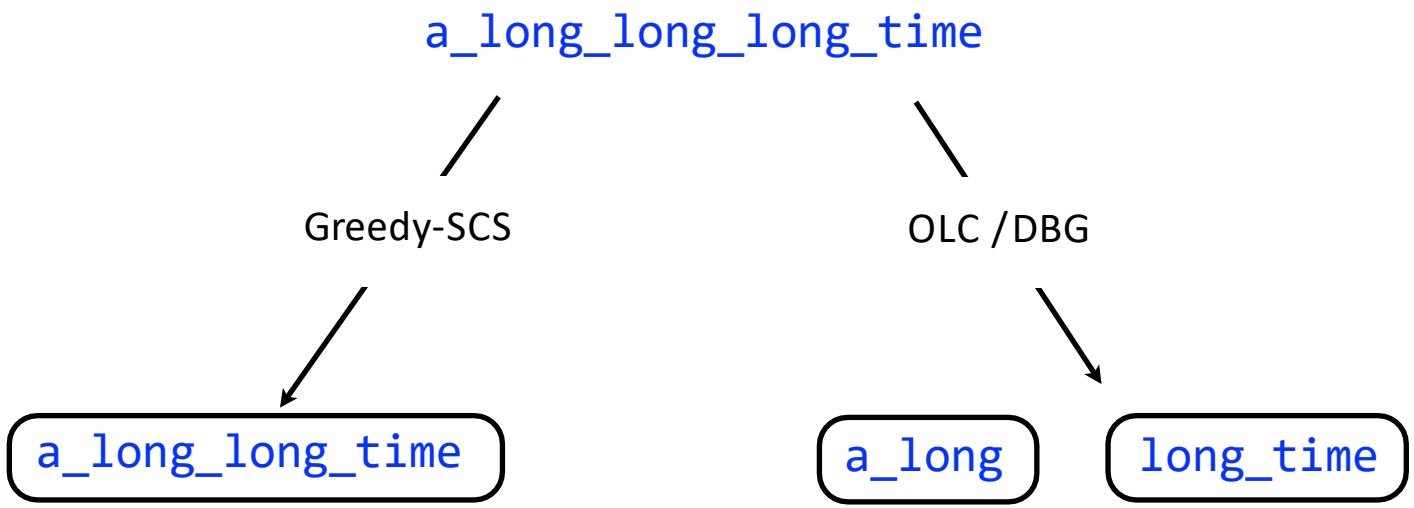
Even if we avoid collapsing copies of *A*, we can't know which paths *in* correspond to which paths *out*

Assembly in the real world

OLC: Overlap-Layout-Consensus assembly

DBG: De Bruijn graph assembly

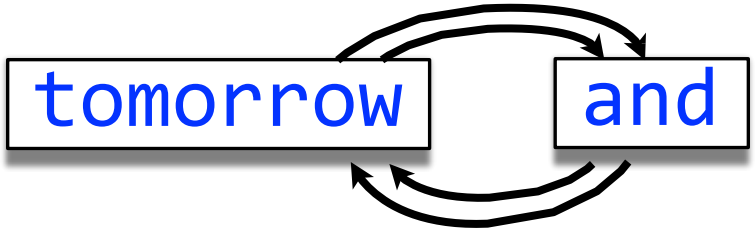
Handle unresolvable repeats by *leaving them out* This breaks the assembly into fragments Fragments called *contigs* (short for *contiguous*)



Different kind of graph

De Bruijn graph

"tomorrow and tomorrow and tomorrow"



An edge represents an ordered pair of adjacent words in the input

Multigraph: there can be more than one edge from node A to node B

k-mer

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

S: GGCGATTCATCG

All 4-mer of *S*

GGCG
GCGA
CGAT
ATTC
TTCA
TCAT
CATC
ATCG

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

All 3-mers of *S:*

GGC
GCG
CGA
GAT
ATT
TTC
TCA
CAT
ATC
TCG

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.

AAB 3-mer

AA AB

L R

AAB's left 2-mer AAB's right 2-mer

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*.

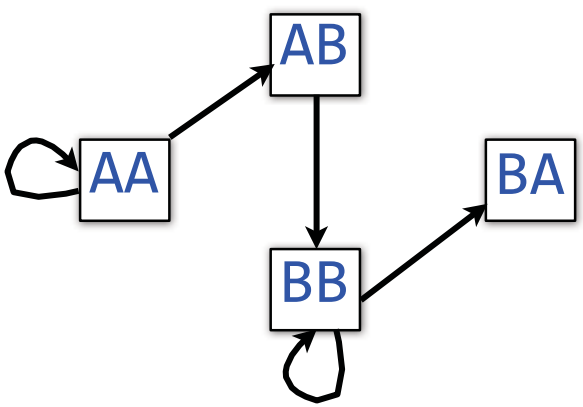
AAABBBA

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.

De Bruijn graph

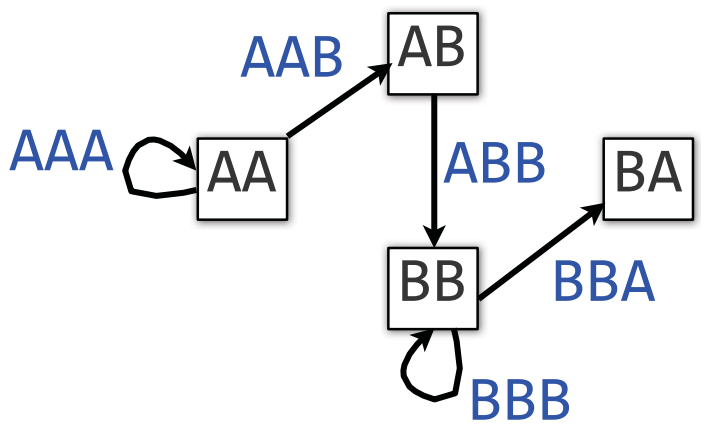
Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the *left* and *right 2-mers*.

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



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.

Eulerian walk definitions and statements

Node is *balanced* if indegree equals outdegree

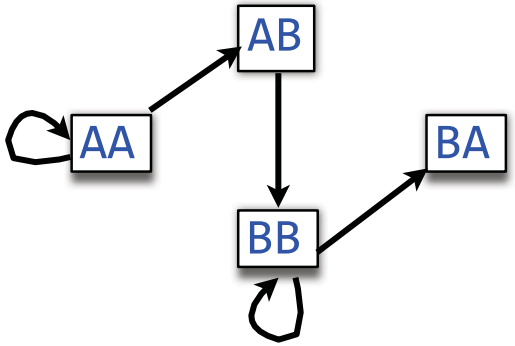
Node is *semi-balanced* if indegree differs from outdegree by 1

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

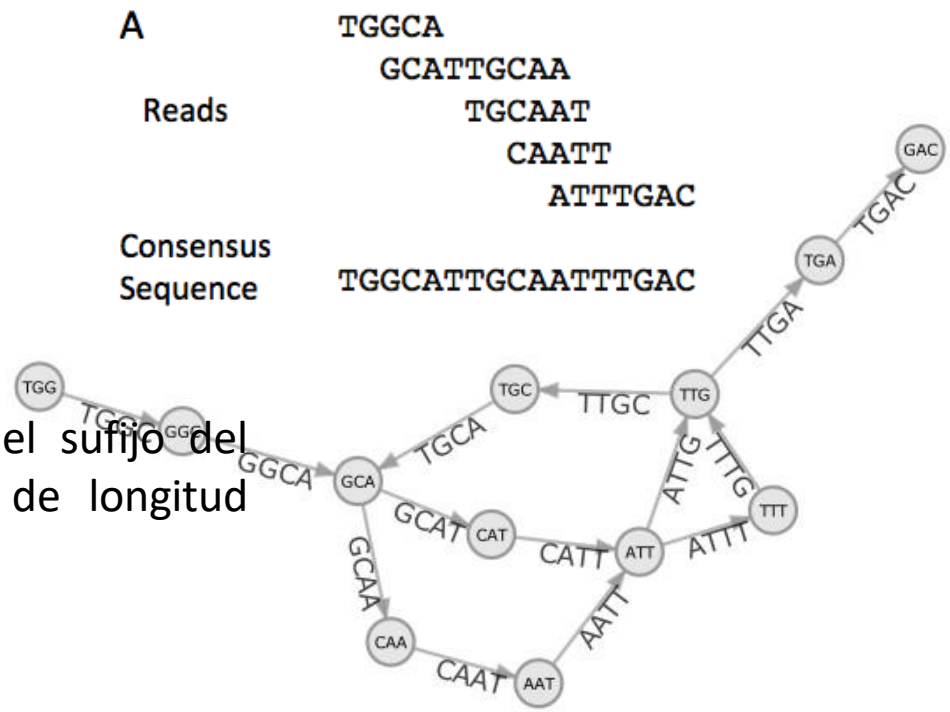


De novo Assembly: de Bruijn graph

Eulerian graph: En este grafo, las aristas son sub-secuencias únicas entre las lecturas mientras que los nodos son superposiciones de secuencias de lecturas de longitud uniforme. De esta manera, el proceso de ensamblado consiste en encontrar en el grafo el camino que visite cada arista al menos una vez.

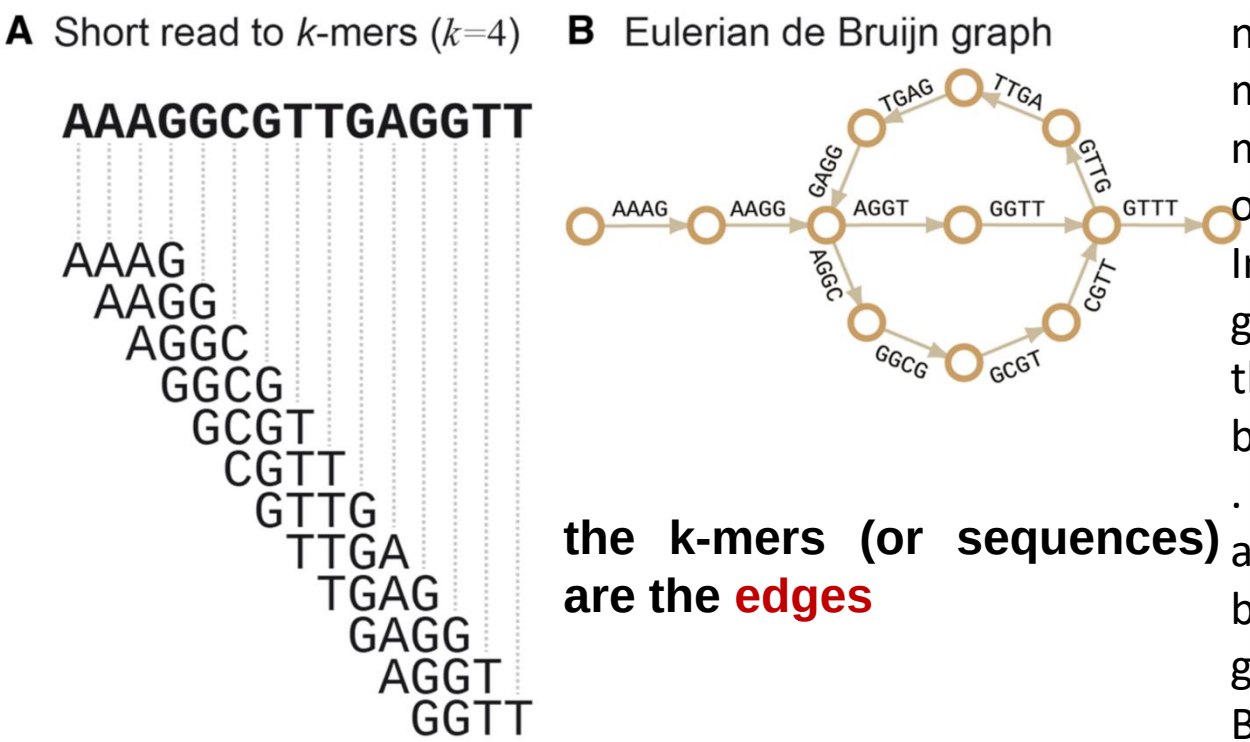
En este método:

- Las aristas son sub-secuencias únicas de las lecturas de longitud k (k-mer)
- Los nodos representan sub-secuencias comunes de longitud k-1
- Así, una arista conecta dos nodos si el sufijo del nodo origen comparte un match exacto de longitud k-2 con el prefijo del nodo destino.



De novo Assembly: de Bruijn graph

the Eulerian de graph approach is able to solve the complicated graph problem by finding the Eulerian paths that traverse all edges, each of which is visited only once without simplification in polynomial time. The *k*-mers are connected to neighbors by overlapping prefix and suffix (*k*-1)-mers.



the k-mers (or sequences) are the edges

Eulerian de Bruijn graph presents nodes and edges in the opposite manner: the sequence of the *k*-mer is an edge and the overlapped (*k*-1)-mer is a node. In contrast, the Eulerian de Bruijn graph approach is able to solve the complicated graph problem by finding the Eulerian paths that . Eulerian de Bruijn graph-based assemblers generally perform better in the assembly of a large genome than the Hamiltonian de Bruijn graph approach in terms of the assembly resultsraverse all edges, each of which is visited only once without simplification in polynomial time

Eulerian cycles

Se representan las secuencias como un grafo de *k-mers*, donde cada ***edge*** es un k-mer, y donde los nodos son prefijos y sufijos de cada k-mer. En este caso hay que buscar un camino que pase por **todos** los *k-mers* (ejes).

Reads:

CGTGCAA

TGCAATG

ATGGCGT

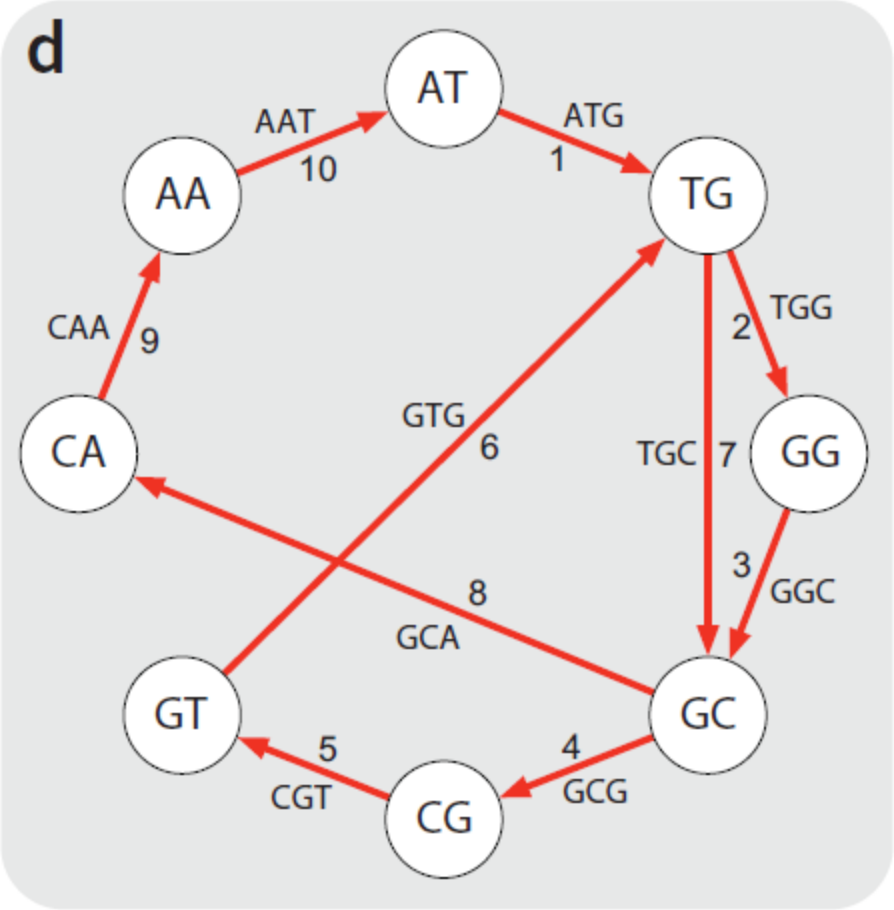
GGCGTGC

CAATGGC

Para k=3, los k-mers son:

CGT, GTG, TGC, GCA, CAA,

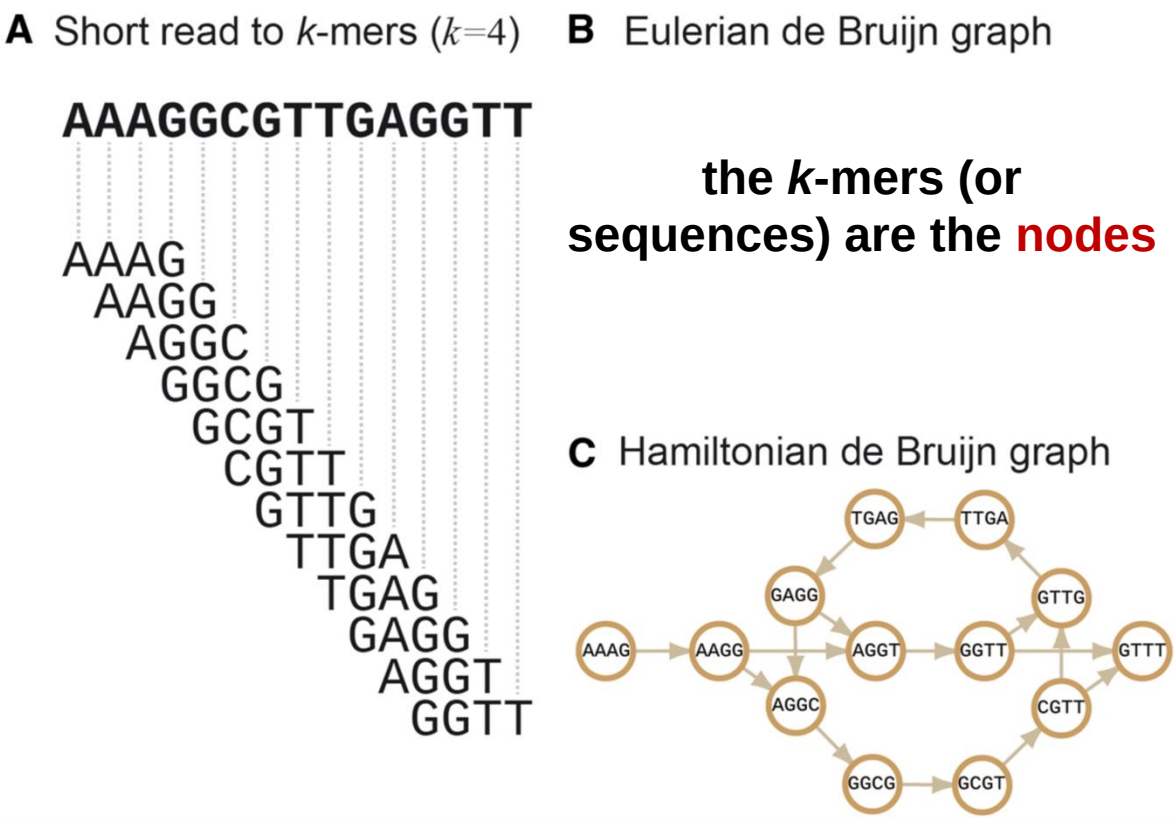
AAT, ATG, TGG, GGC, GCG



Eulerian cycle
Visit each edge once

De novo Assembly: de Bruijn graph

The Hamiltonian graph approach: In these approaches, the sequences are assembled by finding Hamiltonian paths that traverse all nodes, each of which is visited only once. the *k*-mer itself becomes a node, and the (*k*-1)-mer suffix of the *k*-mer that overlapped with the (*k*-1)-mer prefix of the next *k*-mer becomes an edge.



In the Hamiltonian de Bruijn graph, the *k*-mer itself becomes a node, and the (*k*-1)-mer suffix of the *k*-mer that overlapped with the (*k*-1)-mer prefix of the next *k*-mer becomes an edge. In other words, if the prefix of a node is the same (or overlaps) as the suffix of another node, the two nodes are connected. The Hamiltonian graph approach is similar to the OLC approach in that the node is the sequence and the edge is the overlap. In these approaches, the sequences are assembled by finding Hamiltonian paths that traverse all nodes, each of which is visited only once.

Hamiltonian cycles

Se representan las secuencias como un grafo de *k-mers*, donde los *edges* sean alineamientos de a pares, y buscar un camino que pase por **todos** los *k- mers* (nodos).

Reads:

CGTGCAA

TGCAATG

ATGGCGT

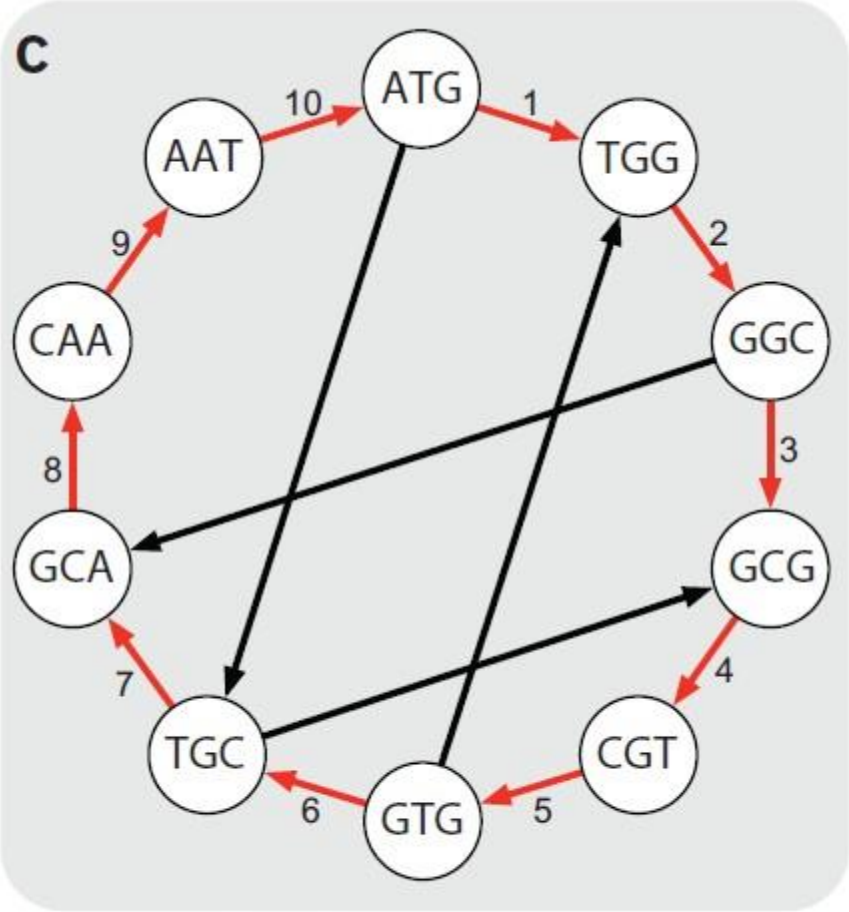
GGCGTGC

CAATGGC

Para k=3, los k-mers son:

CGT, GTG, TGC, GCA, CAA,

AAT, ATG, TGG, GGC, GCG



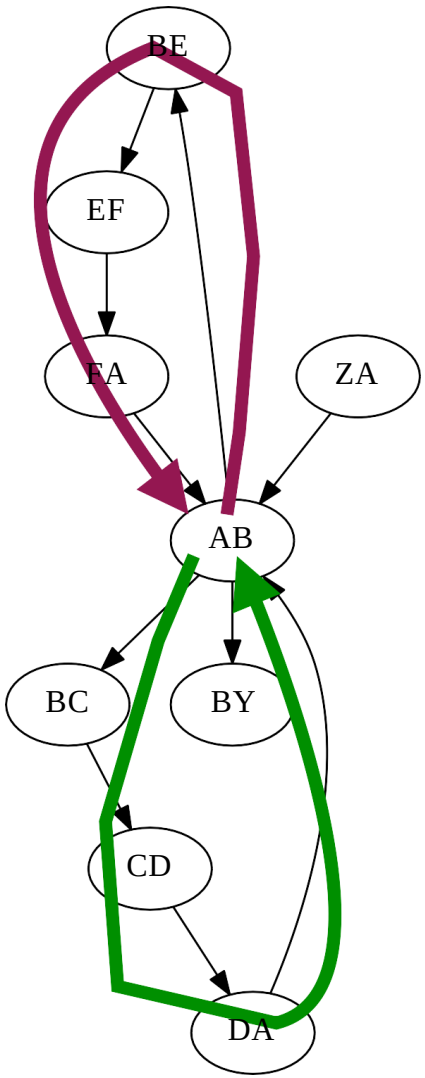
Hamiltonian cycle
Visit each vertex once

De Bruijn graph

Problem 1: Repeats still cause misassembles

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

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



Problem 2:

We've been building DBGs assuming "perfect" sequencing: each *k*-mer reported exactly once, no mistakes. Real datasets aren't like that.

OLC vs de Bruijn assemblies

Ventajas

de Bruijn	OLC
Never explicitly computes pairwise overlaps. Overlap computation is a very time and computationally intensive step that other assembly approaches must take.	Because of the distinct overlap, layout, and consensus stages, OLC algorithms are naturally implmented in a modular algorithmic design. This modular design allows researchers to easily tweak and optimize one portion of assembly for a specific assembly project.
There are more efficient ways to find Eulerian paths than Hamiltonian paths.	Overlaps can vary in length.
Very sensitive to repeats.	

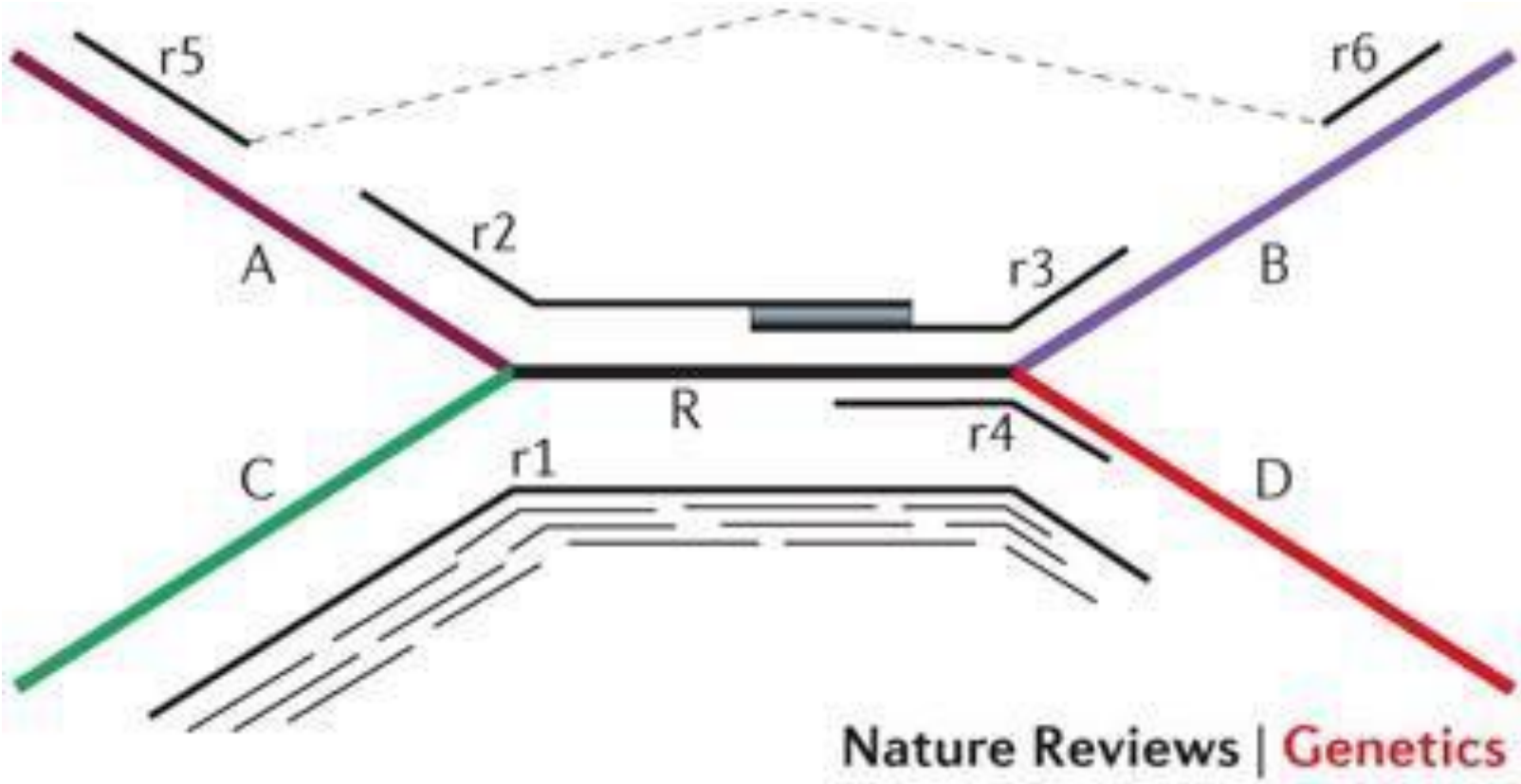
OLC vs de Bruijn assemblies

Desventajas

de Bruijn	OLC
There can be very many Eulerian paths. In order to find the one that represents the actual genome, constraints must be added that make the assembly much more difficult than it is in theory.	The overlap stage is very time consuming and requires a lot of computational power.
Very sensitive to sequencing errors, as errors lead to new k-mers. Since errors complicate the graph, error correction is a crucial step for de Bruijn assemblies.	It is generally more difficult to identify a Hamiltonian paths than Eulerian paths.
Very sensitive to repeats. This sensitivity can introduce additional k-mers, adding to the graph complexity.	
Overlaps are limited to uniform k length sequences.	

El problema de las repeticiones

Interacción entre el largo de lectura, el paradigna de ensamblado y una estructura repetitiva de un genoma siendo ensamblado. El objetivo de un ensamblador es utilizar la información contenida en las lecturas para aproximar y resolver la estructura del grafo de repetición.



Las lecturas cortas son más difíciles de ensamblar

1. Overlap Effect: For same number of sequenced bases, shorter reads require more coverage to achieve comparable N50.

Assembly 1.
9 reads of length = 30bp.
Total sequenced bases = 270.
Assemble with min overlap = 20bp.
Result = 7 contigs.

Same inputs

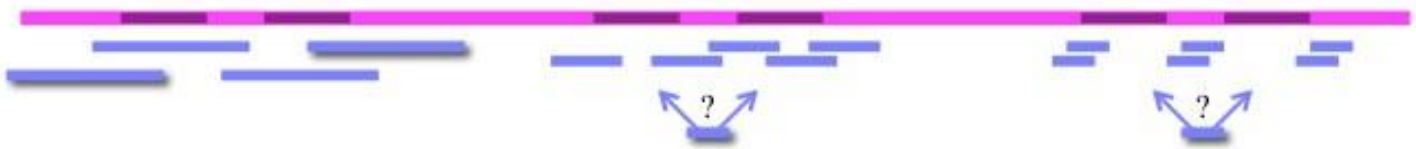
Assembly 2.
3 read length = 90.
Total sequenced bases = 270.
Assemble with min overlap = 20bp.
Result = 1 contig.

2. Repeat Effect: Shorter reads resolve fewer repeats.

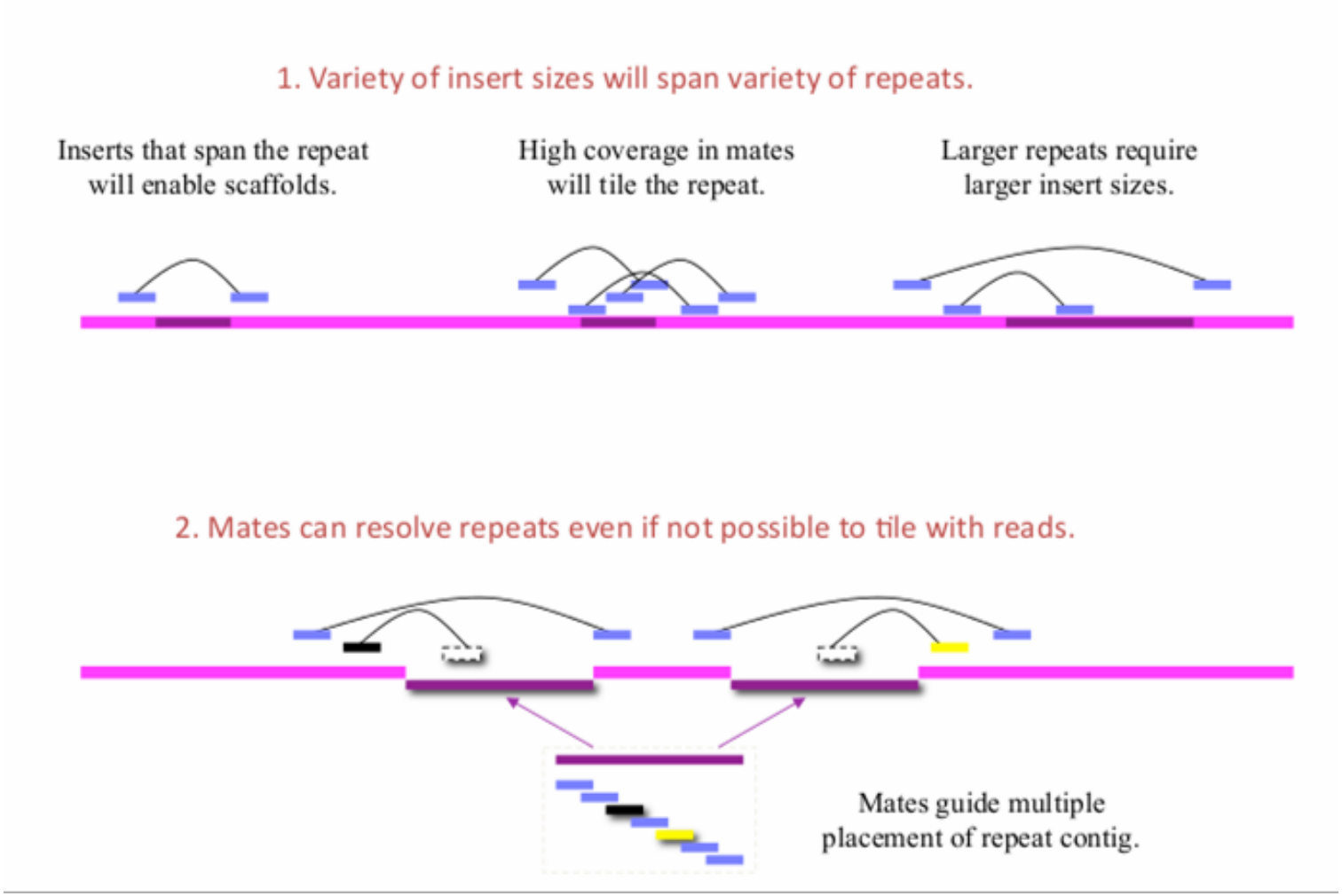
Repeat length = 600bp.
Read length = 800bp (Sanger).
Reads span the repeats.

Repeat length = 600bp.
Read length = 400bp (454).
Reads bridge the repeats.

Repeat length = 600bp.
Read length = 75bp (Solexa).
Repeats not resolved.



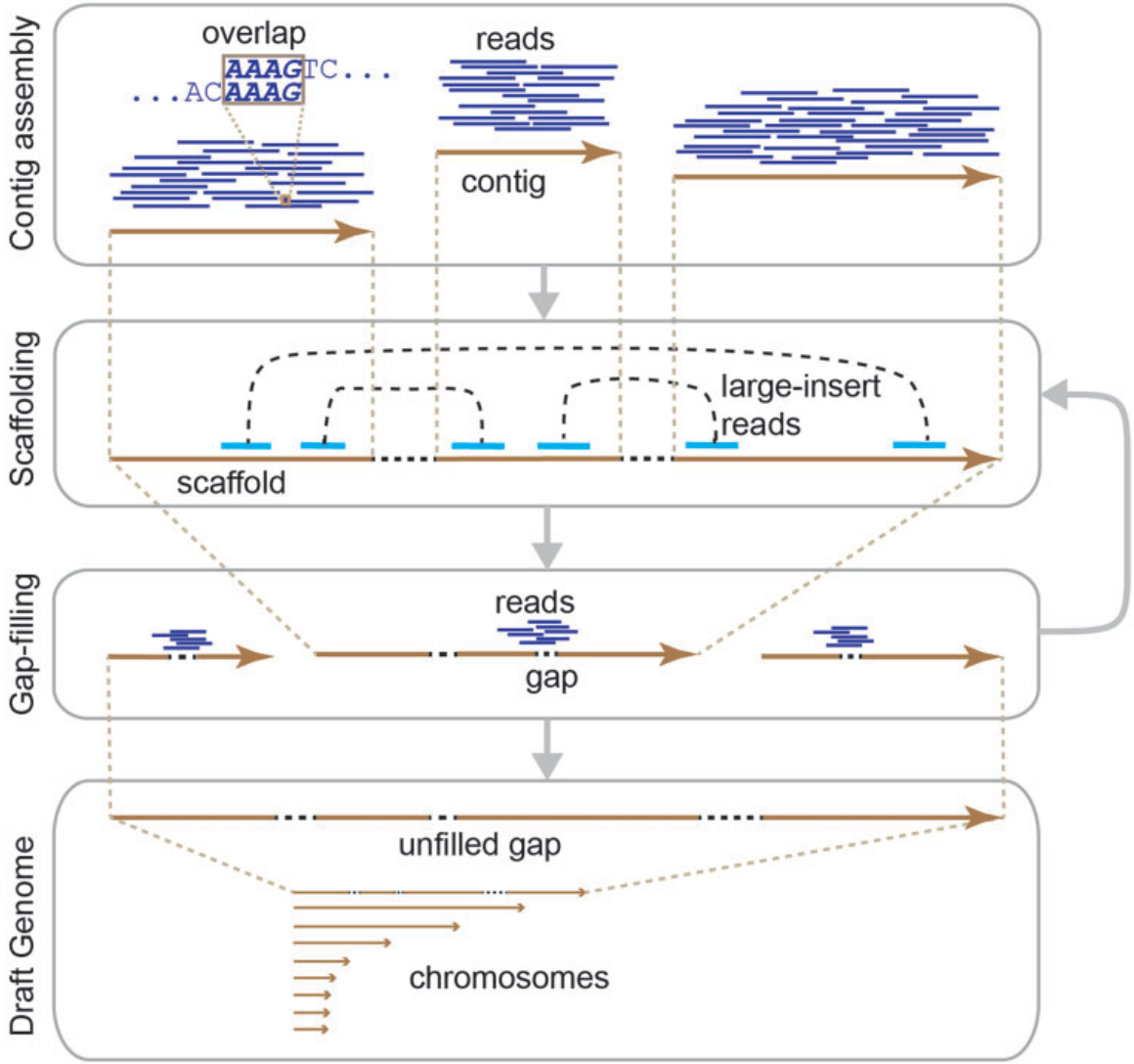
La necesidad de mate-pairs



Consideraciones para un buen ensamblado

1) Lecturas largas para abarcar las repeticiones cortas.
2) Alta cobertura para alineamientos de a pares más largos.
3) Mate-Pairs
   - Lecturas lo suficientemente largas como para colocarse inequívocamente
   - Insertos más largos que las repeticiones largas
   - Densidad de pares suficiente como para atravesar clusters de repeticiones
   - Baja varianza en el tamaño de los insertos.
   - Diversidad de tamaños de inserto (2Kb, 8Kb, 20Kb)
4) Lecturas sin contaminación de vectores.
5) Lecturas con baja contaminación de ADN mitocondrial o cloroplástico
6) Los requisitos dependen en gran medida de la calidad: no es lo mismo un genoma borrador que uno de alta calidad.

Cualquiera de los métodos da lugar a contigs



- **Un “contig” es la secuncia que surje de superponer lecturas No puede tener GAPs**
- **Los “contigs” se generán a partir de los grafos**

Assemblers

Software	Method
ALLPATHS-LG	Eulerian de Bruijn graph
ABySS	Hamiltonian de Bruijn graph
JR-Assembler	greedy algorithm
MaSuRCA	OLC and Eulerian de Bruijn graph
Meraculous	Hamiltonian de Bruijn graph
SGA	Hamiltonian de Bruijn graph + Burrows–Wheeler transform
SOAPdenovo	Hamiltonian de Bruijn graph + sparse k-mer
SPAdes	Eulerian de Bruijn graph
SparseAssembler	Hamiltonian de Bruijn graph + sparse *k*-mer
SparseAssembler	Hamiltonian de Bruijn graph
Velvet	Eulerian de Bruijn graph
Platanus	Hamiltonian de Bruijn graph

And others

Error correction

- **Sequencing error**
- **Complexity reducing**
- **Repeat resolving**
- **Uneven depth**
- **RAM memory**

However low complexity regions or repetitive sequences are still a problem

How to resolve it?

Assemblers

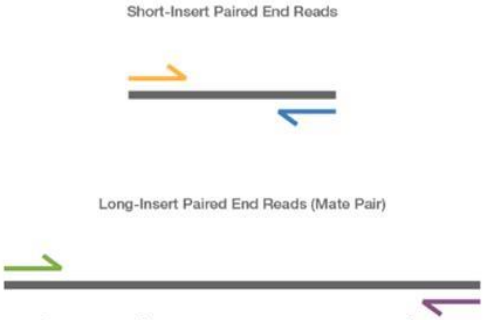
Software	Method
ALLPATHS-LG	Eulerian de Bruijn graph
ABySS	Hamiltonian de Bruijn graph
JR-Assembler	greedy algorithm
MaSuRCA	OLC and Eulerian de Bruijn graph
Meraculous	Hamiltonian de Bruijn graph
SGA	Hamiltonian de Bruijn graph + Burrows–Wheeler transform
SOAPdenovo	Hamiltonian de Bruijn graph + sparse k-mer
SPAdes	Eulerian de Bruijn graph
SparseAssembler	Hamiltonian de Bruijn graph + sparse *k*-mer
SparseAssembler	Hamiltonian de Bruijn graph
Velvet	Eulerian de Bruijn graph
Platanus	Hamiltonian de Bruijn graph

And others

Error correction

- **Sequencing error**
- **Complexity reducing**
- **Repeat resolving**
- **Uneven depth**
- **RAM memory**

PacBio **NanoPore**



Long insert – mate pairs

Assembly validation

- What can we do to evaluate an assembly?

Two approach:

Statistical

- Assembly statistics
- K-mer statistics
- Read alignment statistics and properties
- Comparative alignment

⬇

- ❖ Measures the integrity of the genome
- ❖ Measures the genome size
- ❖ Estimates how good is the assembly

Biological

- Contamination assessment
- Gene space statistics

⬇

- ❖ the presence of contaminants
- ❖ the presence of symbionts
- ❖ Biological sense of the assembly
- ❖ Probability of finding genes

Assembly validation

Statistical common parameters - CONCEPTS

- **N50 is a common statistical measure of sequence length. – The size of the smallest contig in the set of largest contigs that make up 50% of the assembly size.**
- **L50 The number of contigs in the set of largest contigs that make up 50% of assembly size.**
- **NG50 – The size of the smallest contig in the set of largest contigs that make up 50% of the estimated genome size (not assembly).**
- **Cumulative length – Determine the number of contigs needed to cover a reference genome**

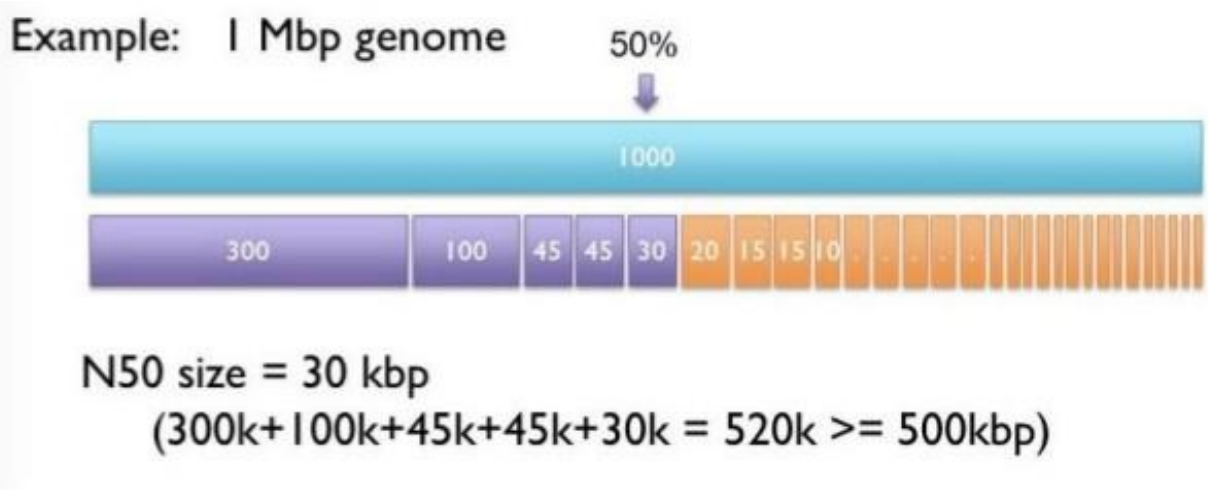
- **Number of contigs**
- **Longest contig**
- **Total size in contigs**

Scripts and softwares that do the work for you

Quast
K-mer Analysis Toolkit
REAPR
LAP

Calidad del ensamblado

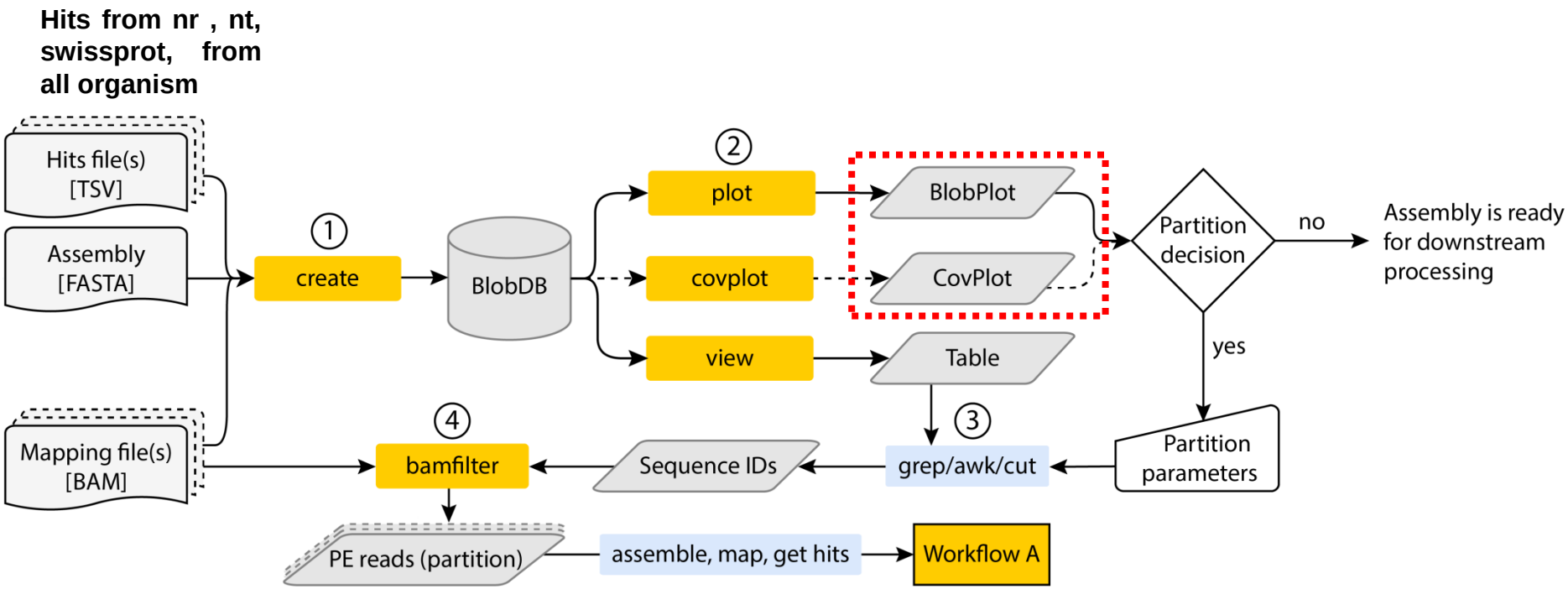
N50: es la longitud del "contig más pequeño" que sumando todos los contigs de mayor longitud a menor, representa al menos el 50% del genoma ensamblado.



Assembly validation

Biological approach

Contamination assessment to remove all contaminant data

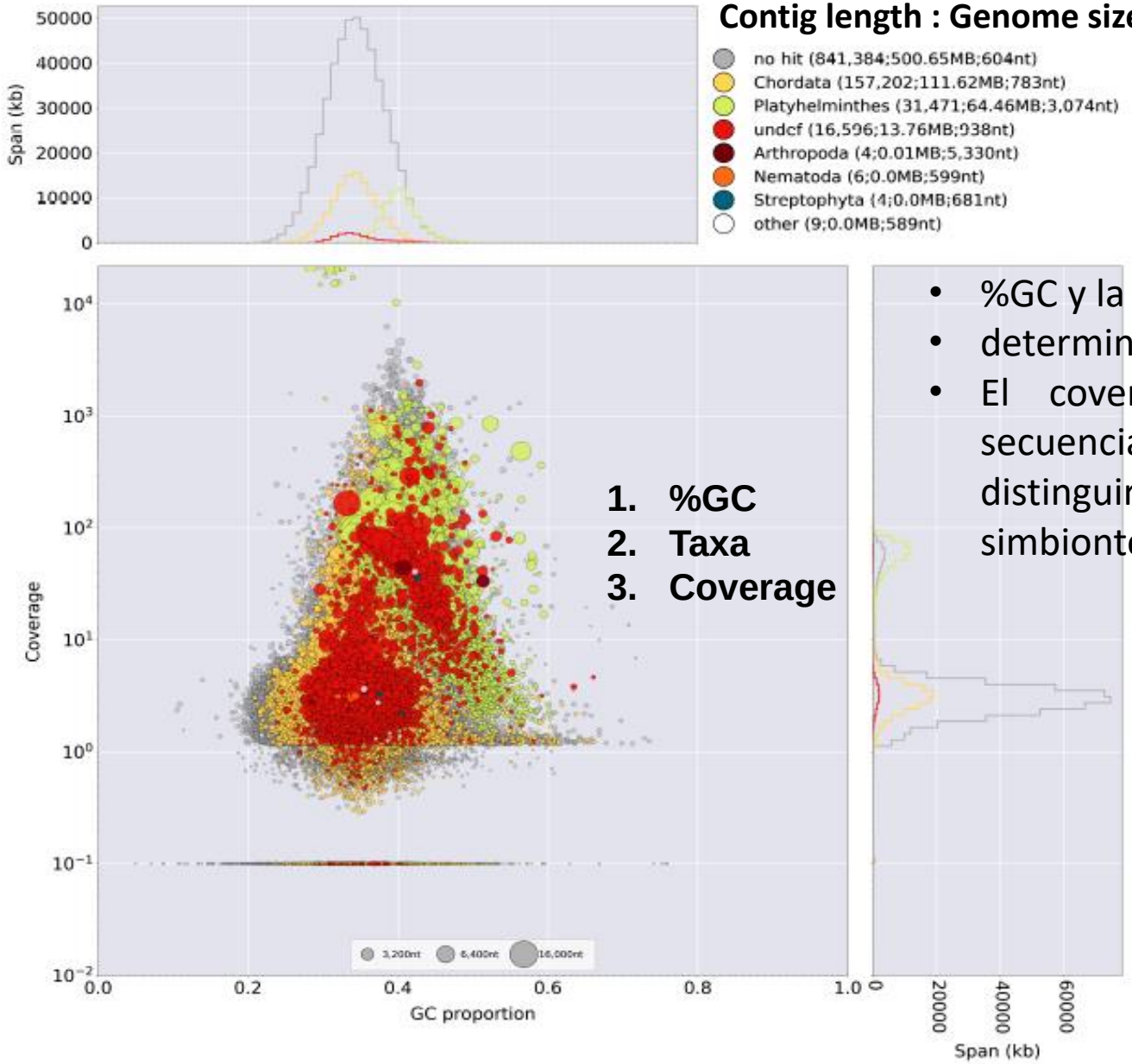


1. Construction a BlobDB data structure based on input files
2. Visualisation of assembly and generation of tabular output
3. Partitioning of sequence IDs based on user-defined parameters informed by the visualisations
4. Partitioning of paired-end reads based on their mapping behaviour to sequence partitions
5. Resulting reads are then assembled by partition and the assemblies can be screened again using the workflow.

Laetsch DR and Blaxter ML, 2017

Assembly validation

(Number of counts; total span (cumulative length);N50 by taxonomic group)



- %GC y la clasificación por taxon
- determina el organismo target
- El coverage permite definir la secuenciación del genoma target y distinguir entre contaminantes y simbiontes

Assembly validation

Biological approach

Completeness of the gene space: How probable is to find genes in the genome - Biological sense of the assembly

- **Core Eukaryotic Genes Mapping Approach (CEGMA) – Parra et al. (2007)**
  - Found 458 genes highly conserved across eukaryotes in the eu**K**aryotic **O**rthologous **G**roups (KOG) database
  - tblastn of CEGs to your genome
  - Refines gene models using HMMs

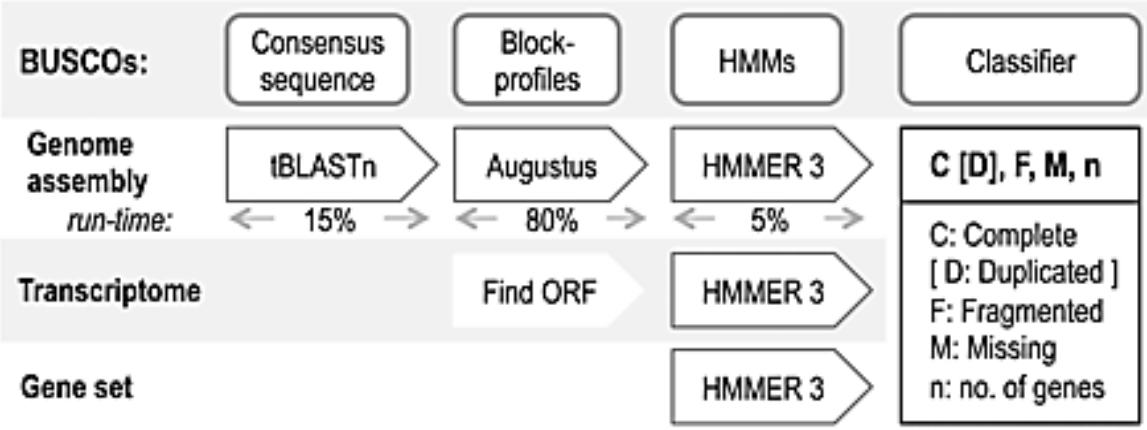
❖Proportion of 248 of the most highly conserved single-copy CEGs can be used to estimated how many genes you have in your assembly

Benchmarking Universal Single-Copy Orthologs

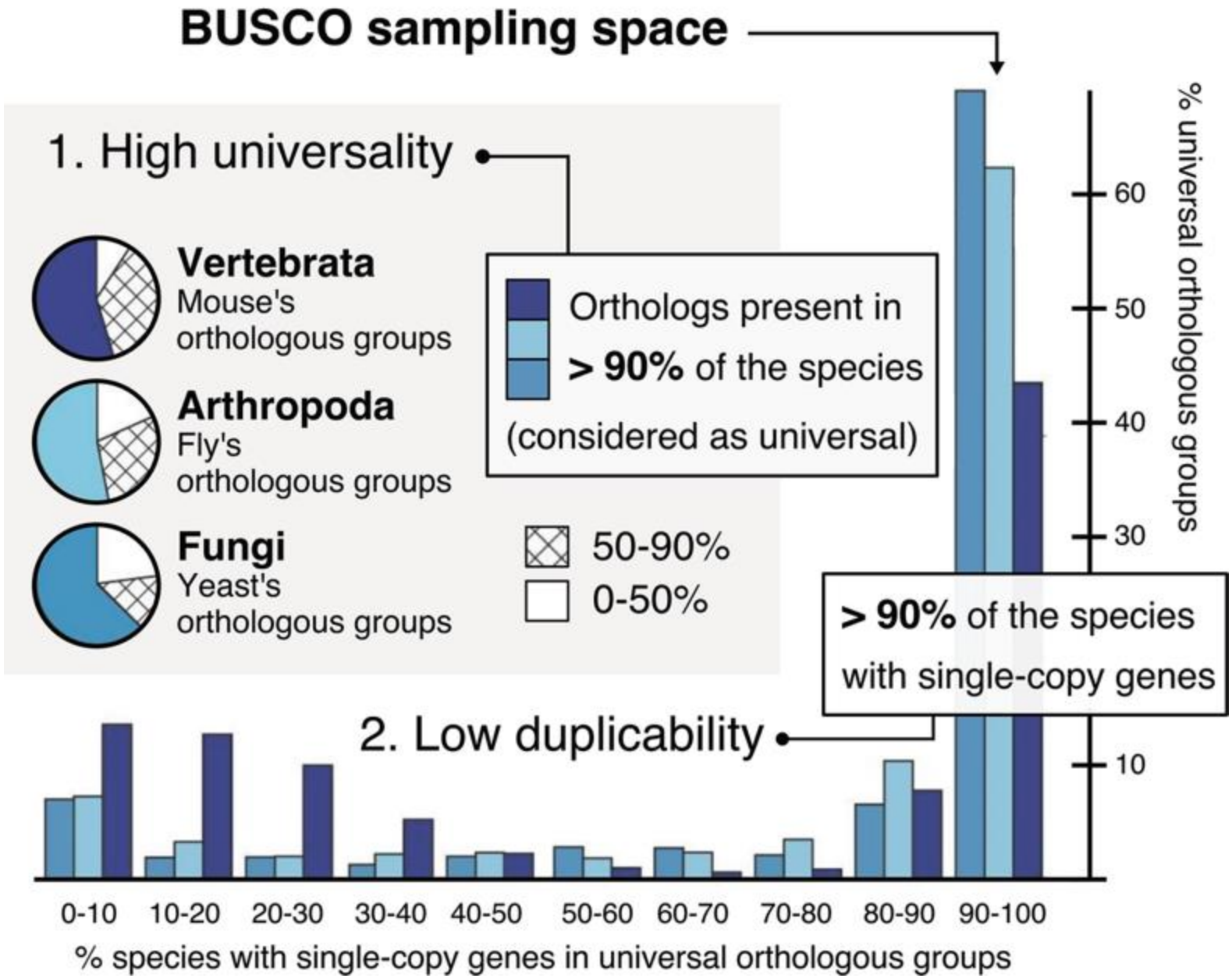
Based on evolutionarily-informed expectations of gene content from near-universal single-copy orthologs selected from OrthoDB *v9*.

- **Based on OrthoDB** instead of outdated KOGs database
- **Clade-specific** conserved single-copy orthologs
  - 3,023 for vertebrates
  - 2,675 for arthropods

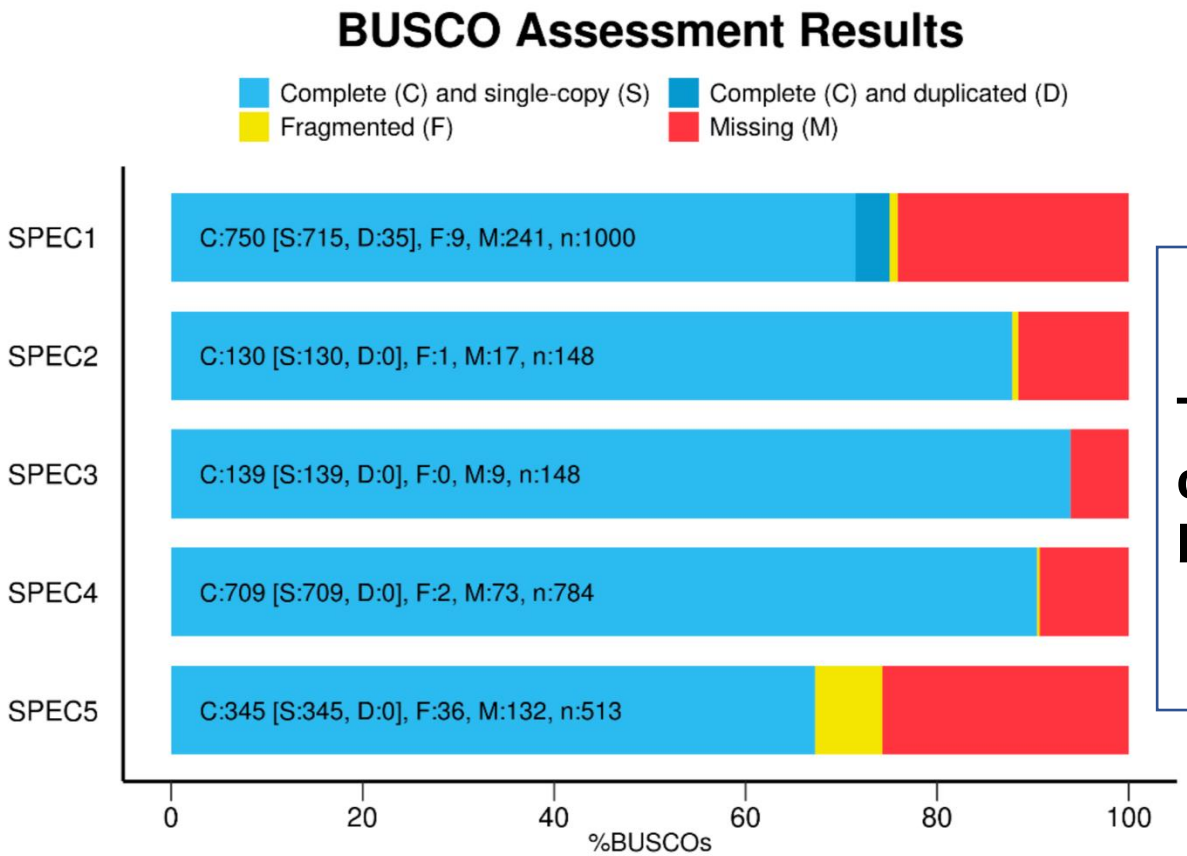
BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. Felipe A. Simão, Robert M. Waterhouse, Panagiotis Ioannidis, Evgenia V. Kriventseva, and Evgeny M. Zdobnov *Bioinformatics*, published online June 9, 2015



1. **tblastn of SCOs to your genome**
2. **Refines gene models using HMMs**



https://busco-data.ezlab.org/v4/data/lineages/



The higher percentage of complete orthologs the better the assembly is

Species	Data Type	BUSCO Benchmarks
Human	Genome	C:89% [D:1.5%], F:6.0%, M:4.5%, n:3023
	Gene set	C:99% [D:1.7%], F:0.0%, M:0.0%, n:3023
Mouse	Genome	C:78% [D:3.0%], F:19%, M:2.5%, n:3023
	Gene set	C:99% [D:2.5%], F:99%, M:0.1%, n:3023
Platypus	Genome	C:55% [D:0.8%], F:25%, M:18%, n:3023
	Gene set	C:72% [D:1.1%], F:19%, M:8.2%, n:3023