de novo assembly

Rayan Chikhi

Pennsylvania State University

Workshop On Genomics - Cesky Krumlov - January 2013

Course structure

Topics I want to convey through this lecture :

- Short intro
- Basic definitions : what an assembly is.
- Fundamentals : what should I know to run an assembler ?
- Metrics : what a satisfactory assembly is.
- Software : how to choose an assembler in 2013.

Your instructor is..

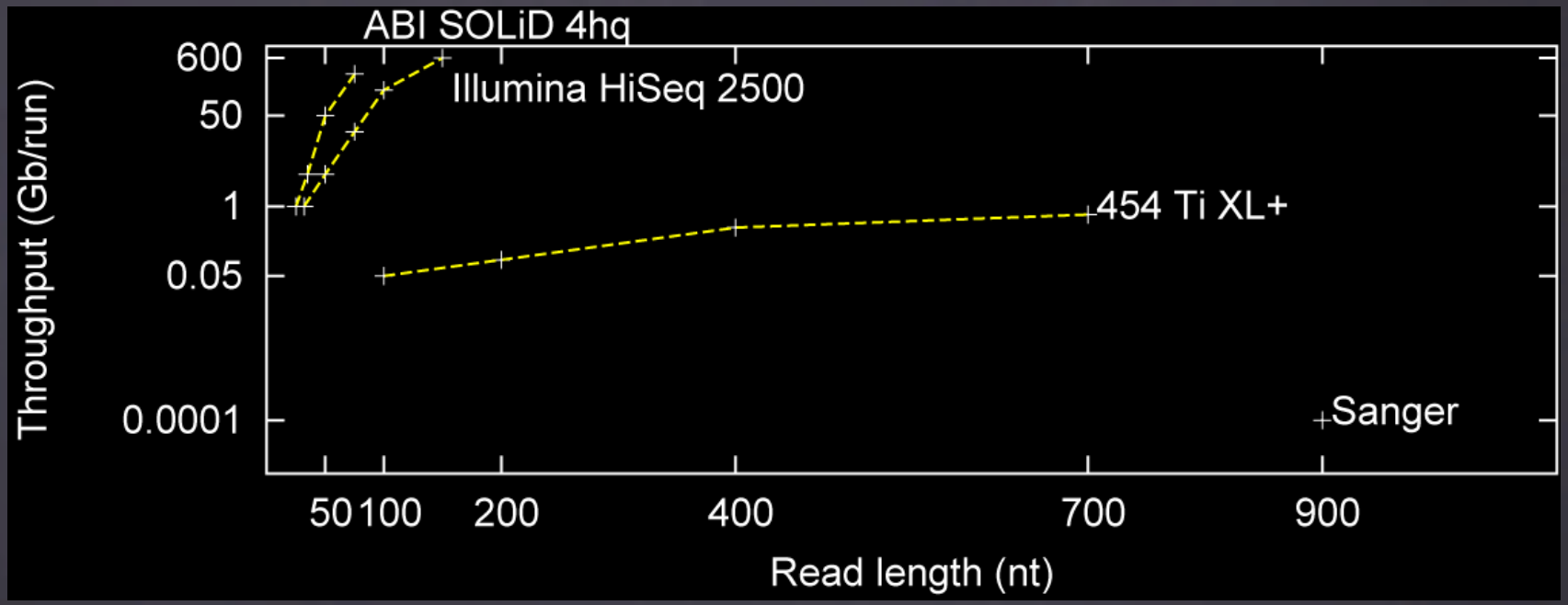
- PhD at INRIA / ENS Cachan, France
- Postdoc at Penn State, USA

Research :

- Paired string graphs
- Targeted assembly
- Ultra-low memory assembly
- Constant-memory k-mer counting

@RayanChikhi on Twitter
"Rayan Chikhi" on Google for my web page

NGS


ABI SOLiD 4hq
Illumina HiSeq 2500
454 Ti XL+
Sanger
Throughput (Gb/run)
600
50
1
0.05
0.0001
50
100
200
400
700
900
Read length (nt)

NGS FUTURE

PacBio	Longer reads (5 kbp), low throughput, accuracy not a problem anymore. Great for gap-filling today.
Nanopore	No data yet. Possibly very long reads (10 kbp), very low throughput. Won't replace Illumina for all applications
Illumina	Will remain medium-sized reads. Currently the only player for large genomes, RNA-seq, metagenomics.

Assembly difficulty

DNA assembly is still a difficult problem in 2013.

1. High computational resources

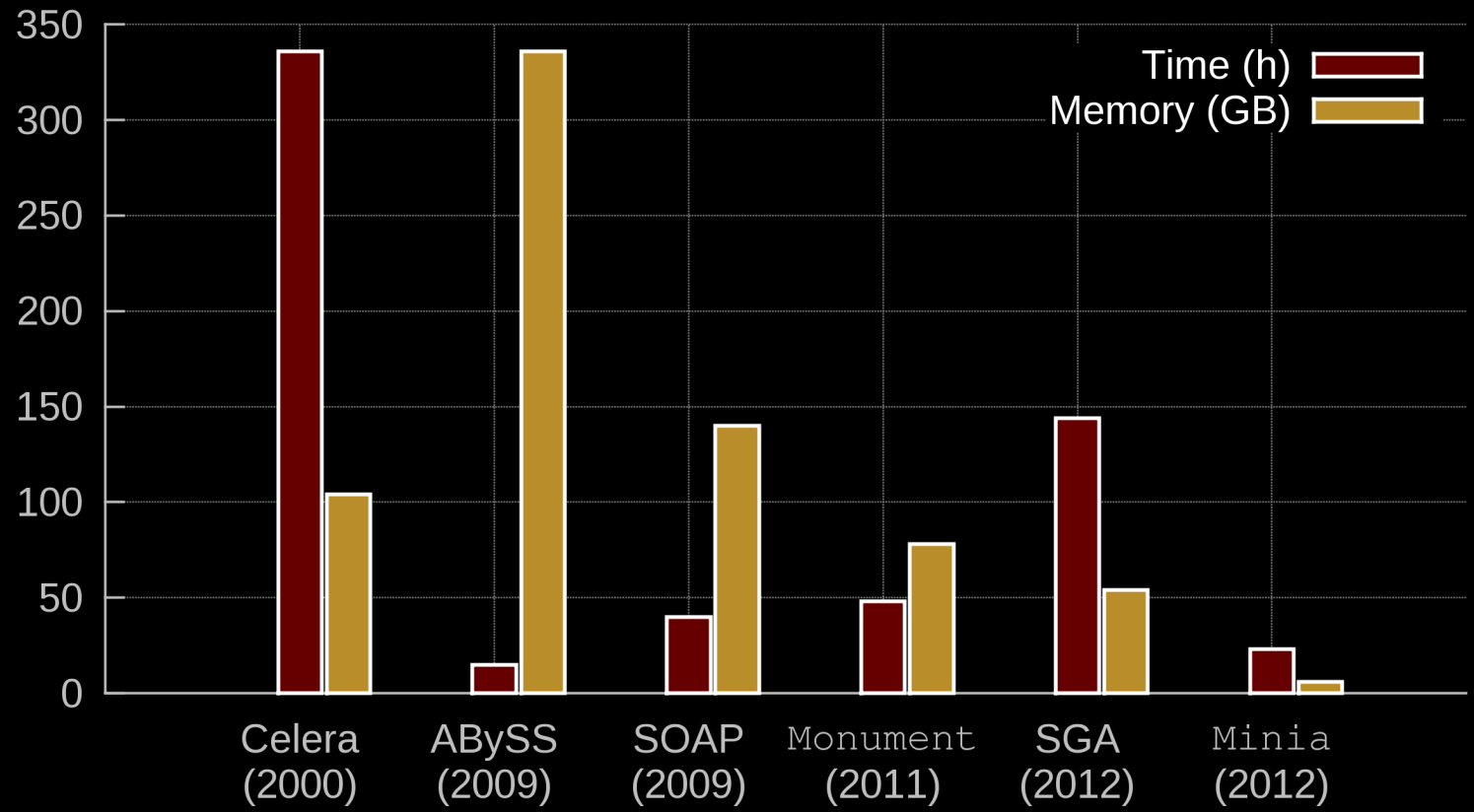
2. Hard to find an optimal solution

Conclusions of the GAGE benchmark (2012) : in terms of assembly quality, there is no single best de novo assembler

Assembly difficulty

DNA assembly is still a difficult problem in 2013.

1. Efficiency : still an area of active research. We're making progress..



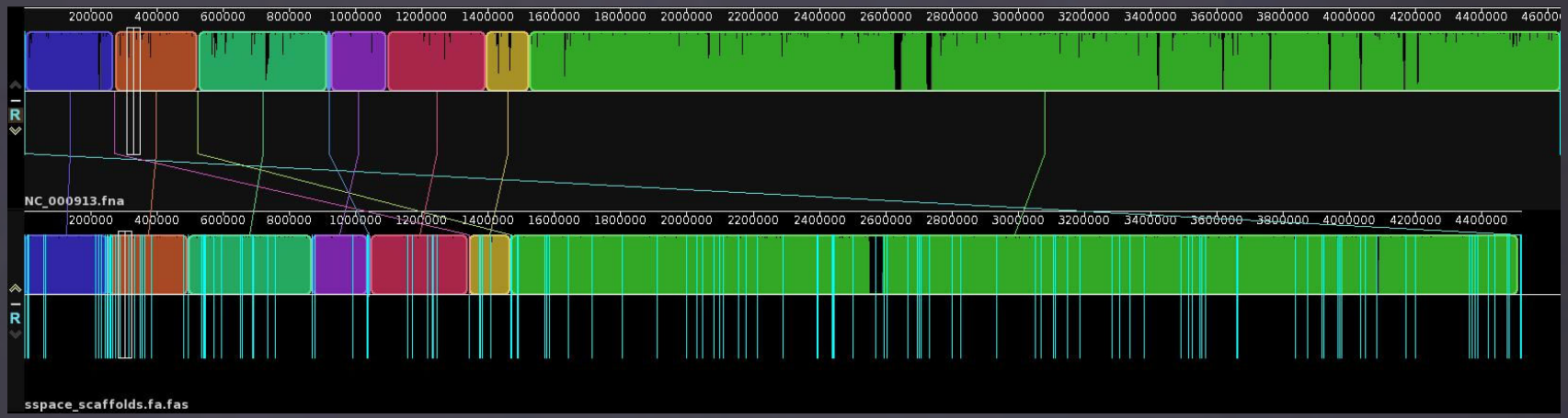
2. Quality : making progress empirically (see SOAPdenovo2 [2013])..

Plan

Definition of an **assembly** (a trickier question than it seems)

Set of sequences which best approximate the original sequenced material.

Example of a reference genome (top),
and an assembly aligned to it (bottom, sequences separated by blue lines).



Simple facts, the aligned assembly is :

- smaller than the reference,
- fragmented

Some vocabulary :

Read Any sequence that comes out of the sequencer

Paired read $read_1$, gap $\leq$ 500 bp, $read_2$

Mate-pair $read_1$, gap $\geq$ 1 kbp, $read_2$

Single read Unpaired read

***k*-mer** Any sequence of length k

Contig gap-less assembled sequence

Scaffold sequence which may contain gaps (N)

Exercice

Here is a set of reads :

```
TACAGT
  CAGTC
   AGTCA
      CAGA
```

1. How many k-mers are in these reads (including duplicates), for $k = 3$?
2. How many *distinct* k-mers are in these reads ?
   - (i) for $k = 2$
   - (ii) for $k = 3$
   - (iii) for $k = 5$
3. It appears that these reads come from the (toy) genome TACAGTCAGA. What is the largest k such that the set of k-mers in the genome is exactly the set of k-mers in these reads ?
4. For any value of k, what is a mathematical relation between N, the number of k-mers (incl. duplicates)in a sequence, and L, the length of that sequence ?

Exercice (solution)

Here is a set of reads :

```
TACAGT
  CAGTC
   AGTCA
      CAGA
```

1. How many k-mers are in these reads (including duplicates), for $k = 3$? 12
2. How many *distinct* k-mers are in these reads ?
   - (i) for $k = 2$, 7
   - (ii) for $k = 3$, 7
   - (iii) for $k = 5$, 4
3. It appears that these reads come from the (toy) genome TACAGTCAGA. What is the largest k such that the set of k-mers in the genome is exactly the set of k-mers in these reads ? 3
4. For any value of k, what is a mathematical relation between N, the number of k-mers (incl. duplicates)in a sequence, and L, the length of that sequence ? $N = L - k + 1$

Plan

Graphs

A graph is a set a nodes and a set of edges (directed or not).

Graphs for sequencing data

Overlaps between reads is the fundamental information used to assemble.
Graphs permit to represent these overlaps.

Two different types of graphs for sequencing data are known :

- de Bruijn graphs — Generally used with Illumina data
- string graphs — Generally used with 454 data

de Bruijn graphs

A **de Bruijn** graph for a fixed integer k :

1. **Nodes** = all *k-mers* (k-length sub-strings) present in the reads.
2. For each $(k+1)$-mer x present in the reads, there is an **edge** between the k-mer prefix of x and the k-mer suffix of x.

Exemple for $k = 3$ and a single read :

`ACTG`

de Bruijn graphs

Example for many reads and still $k = 3$.

```
ACTG
 CTGC
  TGCT
```

de Bruijn graphs : redundancy

What happens if we add redundancy ?

```
ACTG
ACTG
 CTGC
 CTGC
 CTGC
  TGCT
  TGCT
```

dBG, $k = 3$:

de Bruijn graphs : errors

How is a sequencing error impacting the de Bruijn graph ?

```
ACTG
 CTGC
 CTGA
  TGCT
```

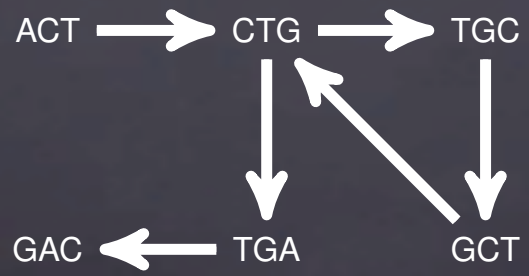
dBG, $k = 3$:

de Bruijn graphs : repeats

What is the effect of a small repeat on the graph ?

```
ACTG
 CTGC
  TGCT
   GCTG
    CTGA
     TGAC
```

dBG, $k = 3$:



String graphs : overlap graphs

Definition of an **overlap graph**. It is *almost* a string graph.

1. **Nodes** = reads.
2. Two nodes are linked by an **edge** if both reads overlap[1].

Example for $k = 3$ and a single read :

`ACTG`

ACTG

1. The definition of overlap is voluntarily fuzzy, there are many possible definitions.

Overlap graphs

Given $k > 0$, we say that r and r' **overlap** if a suffix of r of length $l > k$ is *exactly* a prefix of r' of similar length.

Overlap graph for $k = 3$,

`ACTGCT`
`CTGCT` (overlap of length 5)
`GCTAA` (overlap of length 3)

String graphs : overlap graphs

A **string graph** is obtained from an overlap graph by removing redundancy :

- redundant reads (those fully contained in another read)
- transitively redundant edges(if $a \to c$ and $a \to b \to c$, then remove $a \to c$)

From Overlap graphs to String graphs

Overlap graph for $k = 3$,

String graph for $k = 3$,

The read CTGCT is contained in ACTGCT, so it is redundant

Comparison string graph / de Bruijn graph

On the same example, compare the de Bruijn graph with the string graph :

```
ACTGCT
 CTGCTA
   GCTAA
```

String graph, $k = 3$:

de Bruijn graph, $k = 3$:

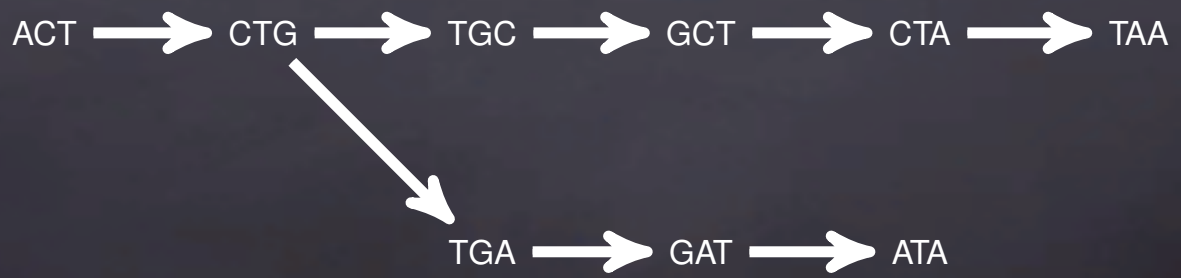
String graph / de Bruijn graph (2)

Let's add an error :

```
ACTGCT
 CTGATA
   GCTAA
```

String graph, $k = 3$:

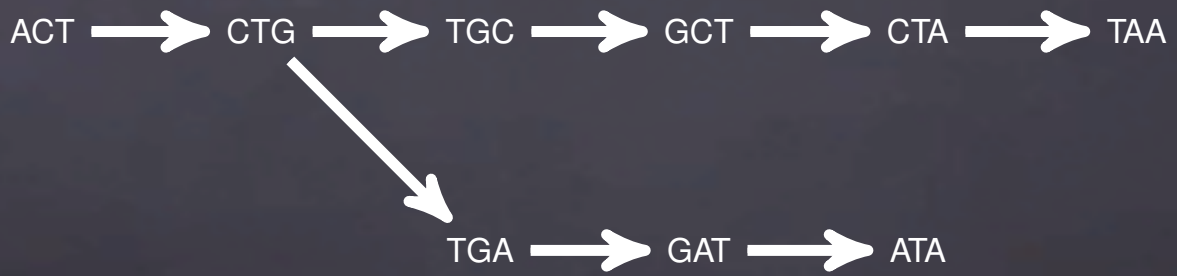
de Bruijn graph, $k = 3$:



String graph / de Bruijn graph (2)

How to "fix" the string graph ?
$\rightarrow$ use a relaxed definition of overlaps.
String graph where overlaps may ignore 1 error, $k = 3$:

de Bruijn graph, $k = 3$:



String graph / de Bruijn graph (3)

So, which is better ?

- String graphs capture whole read information
- de Bruijn graphs are conceptually simpler :
  - single node length
  - single overlap definition

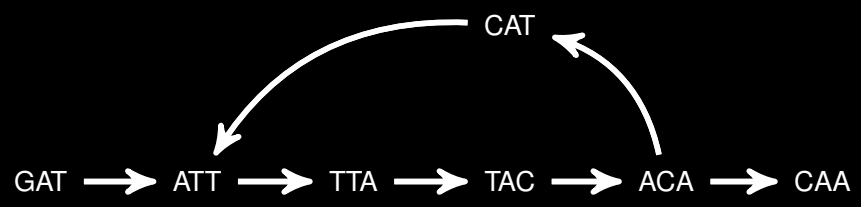
Historically, string graphs were used for long reads and de Bruijn graphs for short reads.

How does one assemble using a graph ?

Assembly in theory [Nagarajan 09]

Return a path of *minimal length* that traverses **each node at least once**.

Illustration

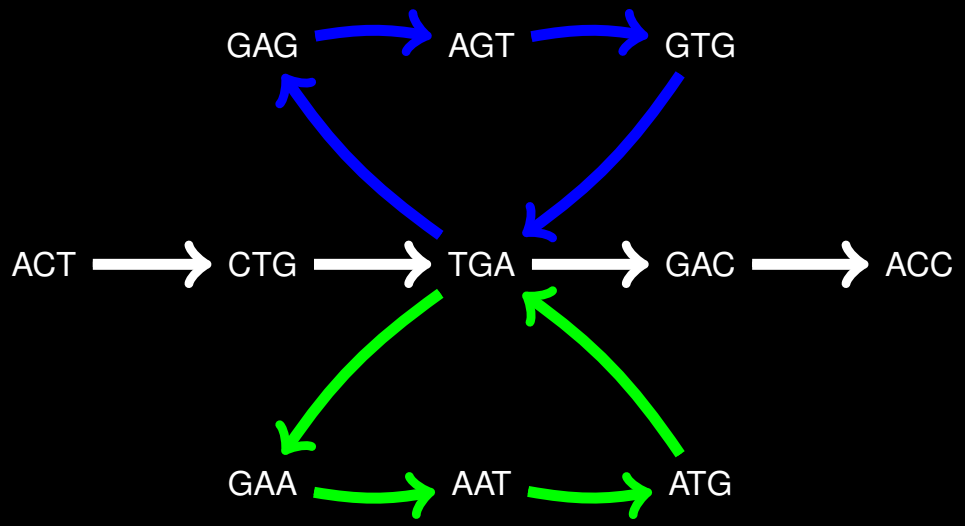


The only solution is GATTACATTACAA.

Assembly in practice

Because of ambiguities and low-coverage region, a single path is almost never found is theory, and is really never found in practice.

Example of ambiguities



Assembly in practice

Return a **set of paths** covering the graph, such that *all possible assemblies* contain these paths.

Solution of the example above

The assembly is the following set of paths :

$$\{\texttt{ACTGA}, \texttt{TGACC}, \texttt{TGAGTGA}, \texttt{TGAATGA}\}$$

CONTIGS CONSTRUCTION

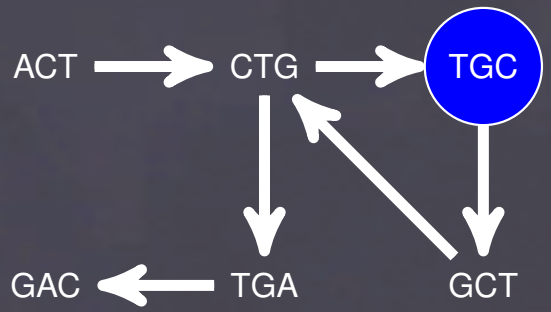
Contigs construction from a graph (de Bruijn or string).

The naive way is to enumerate all *node-disjoint* **simple paths**.

Node-disjoint means that two different paths cannot share a node.
(Edge-disjoint simple paths also work).

CONTIGS CONSTRUCTION EXAMPLE

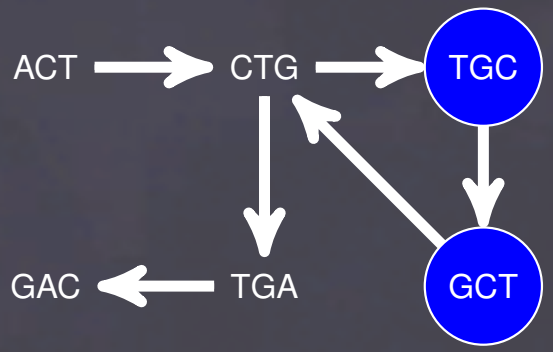
dBG, $k = 3$:



Contigs :

CONTIGS CONSTRUCTION EXAMPLE

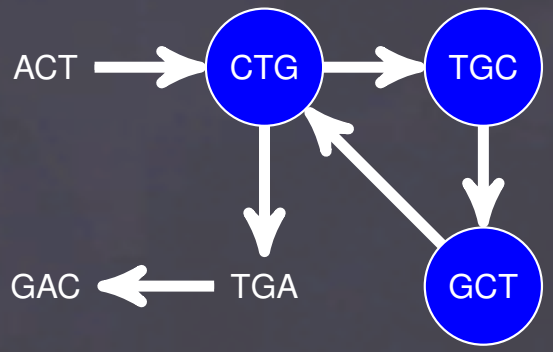
dBG, $k = 3$:



Contigs :

CONTIGS CONSTRUCTION EXAMPLE

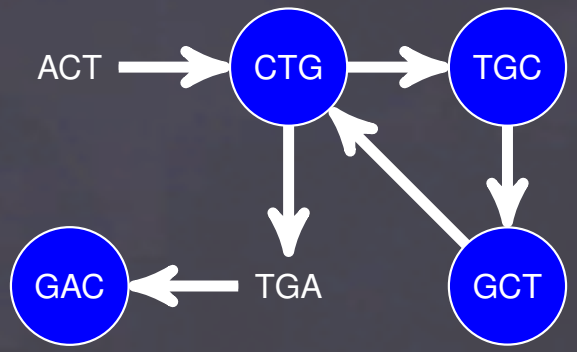
dBG, $k = 3$:



Contigs :
CTGCT

CONTIGS CONSTRUCTION EXAMPLE

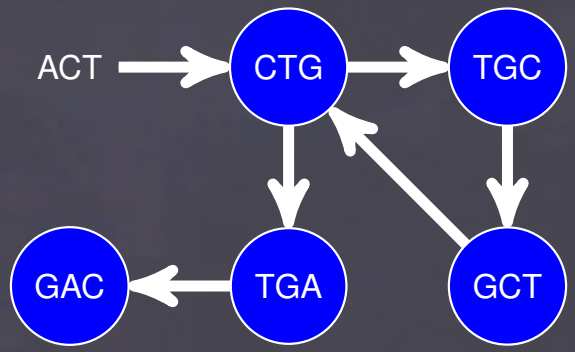
dBG, $k = 3$:



Contigs :
CTGCT

CONTIGS CONSTRUCTION EXAMPLE

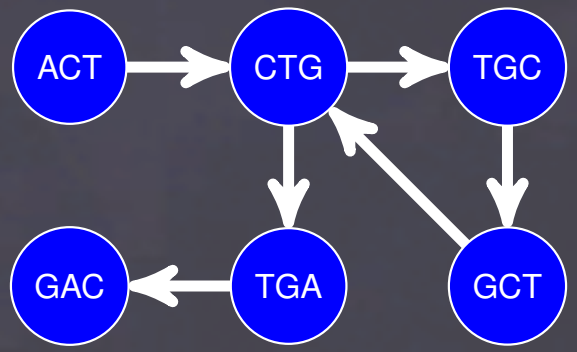
dBG, $k = 3$:



Contigs :
CTGCT
TGAC

Contigs construction example

dBG, $k = 3$:



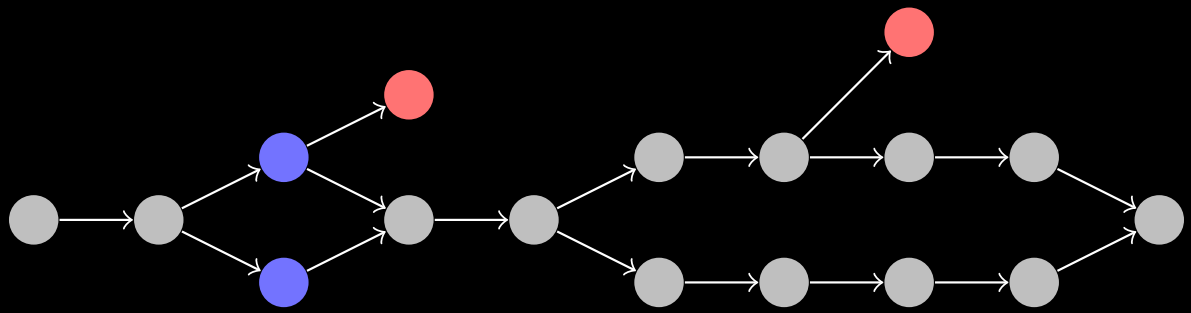
Contigs :
CTGCT
TGAC
ACT

How an assembler works

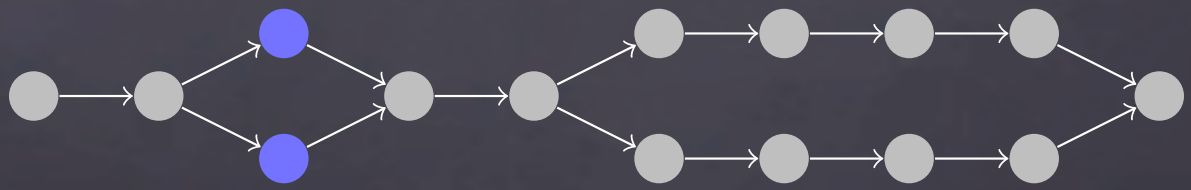
[Velvet, ABySS, SOAPdenovo, SGA ..]

1) Construct a graph from the reads.

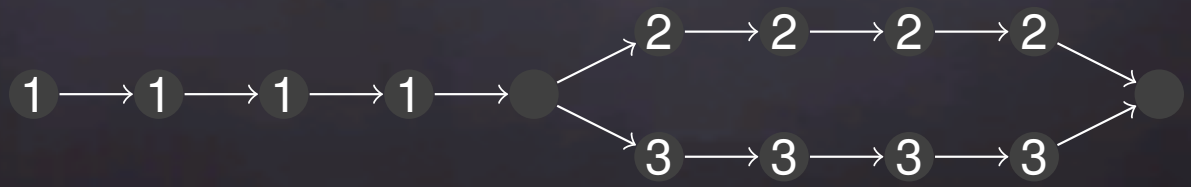
Assembly graph with variants & errors



2) Likely sequencing errors are removed.



3) Known biological events are removed.
4) Finally, **simple paths** (i.e. contigs) are returned.



Short note on reverse complements

Because sequencing isn't strand-directed :

In assembly, we always identify a read with its reverse complement.

E.g : AAA = TTT, ATG = CAT

Exercice

In this exercice, for simplicity, ignore reverse complements.
Reference genome : TACAGTCAGA.
Reads :

```
TACAGT
  CAGTC
   AGTCA
     TCAGA
```

1. Construct the de Bruijn graph for $k = 3$.
   (Reminder : nodes are k-mers and edges correspond to $(k + 1)$-mers)
2. How many contigs can be created ? (stopping at any branching)
3. At which value of k is there a single contig (i.e., no branching) ?
4. (bonus) Find a mathematical relationship between k_a, the smallest value of k for which a genome can be assembled into a single contig, and ℓ_r, the length of the longest exactly repeated substring in that genome.

Exercice (solution)

In this exercice, for simplicity, ignore reverse complements.
Reference genome : TACAGTCAGA.
Reads :

```
TACAGT
  CAGTC
   AGTCA
     TCAGA
```

1. Construct the de Bruijn graph for $k = 3$.
   (Reminder : nodes are k-mers and edges correspond to $(k + 1)$-mers)
2. How many contigs can be created ? (stopping at any branching) 3
3. At which value of k is there a single contig (no branching) ? 4
4. Find a mathematical relationship between k_a, the smallest value of k for which a genome can be assembled into a single contig, and ℓ_r, the length of the longest exactly repeated substring in that genome.
   $k_a = \ell_r + 1$

Plan

Metrics

Preamble : There is no trivial total order (i.e. ranking) between assemblies.

Why ? $>$ 2 independent criteria to optimize (e.g., total length, and average size of assembled sequences)

Example Would you rather have an assembly with good coverage and short contigs, or an assembly with mediocre coverage and long contigs ?

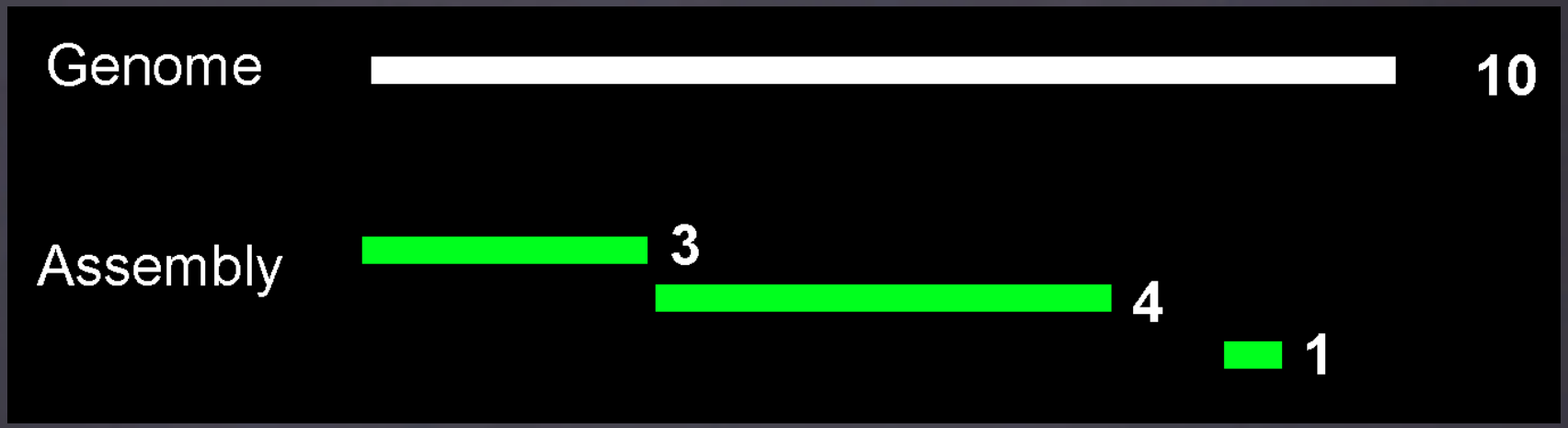
Reference-free metrics

- Number of contigs/scaffolds
- Total length of the assembly
- Length of the largest contig/scaffold
- Percentage of gaps in scaffolds ('N')
- N50 of contigs/scaffolds
- Overlooked but very important : internal consistency
- Number of predicted genes

REFERENCE-FREE METRICS : N50

N50 = Largest contig length at which longer contigs cover 50% of the total **assembly** length

NG50 = Largest contig length at which longer contigs cover 50% of the total **genome** length



If you didn't know N50, write down the definition down, there will be an exercice ;)

A practical way to compute N50 :

- Sort contigs by decreasing lengths
- Take the first contig (the largest) : does it cover 50% of the assembly ?
- If yes, this is the N50 value. Else, try the next one (the second largest), and so on..

Reference-free metrics : Internal consistency

Rarely appears in assemblers articles but extremely useful in *de novo* projects.

Internal consistency : Percentage of paired reads correctly aligned back to the assembly (*happy* pairs).

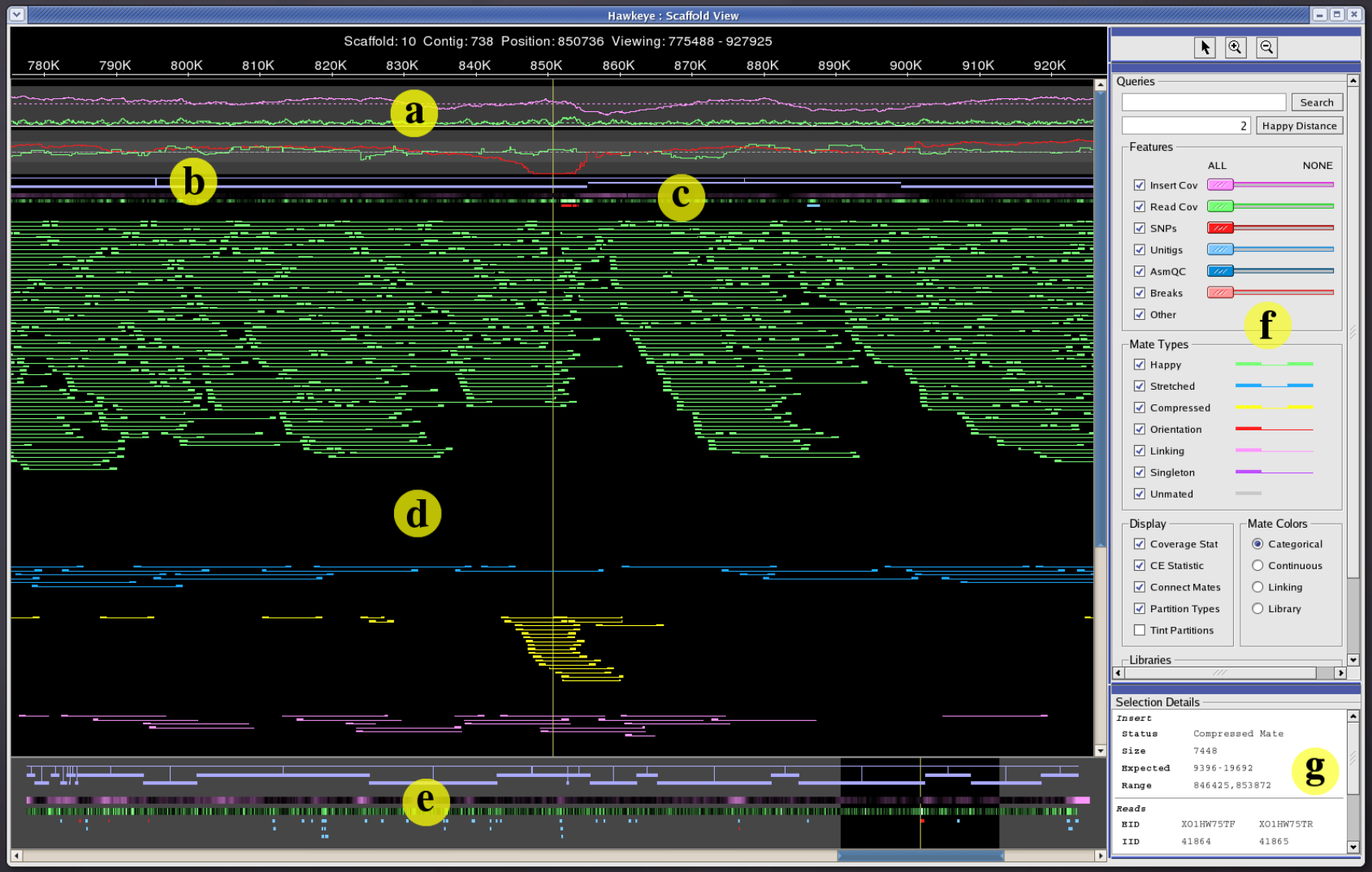
Allows to locate certain types of misassemblies (mis-joins).
Recent tools enable to compute this metric :

- REAPR [2]
- FRCurve [3]

[F. Vezzi (Plos One) 2013]

2. Google : REAPR assembly
3. Google : FRCurve

INTERNAL CONSISTENCY : EXAMPLE

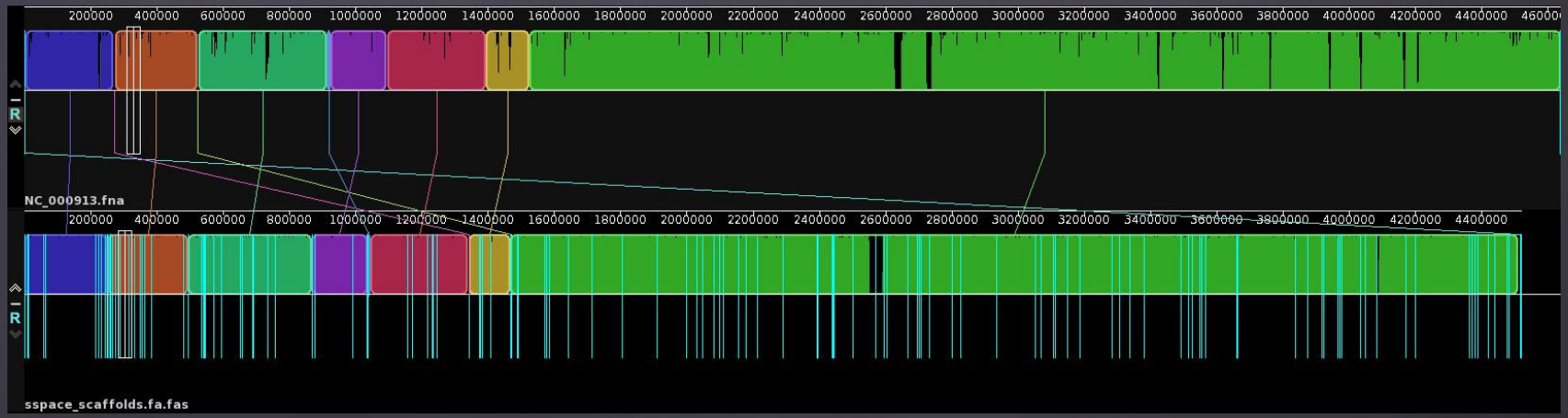


Hawkeye software

METRICS USING A REFERENCE : COVERAGE

Given an assembly aligned to a reference.

Coverage Percentage of bases in the reference which are covered by the alignment.



METRICS USING A REFERENCE : ASSEMBLY ERRORS

Also requires that the assembly is aligned to a reference.

- Number of substitutions.
- Number of small indels
- Number *misjoins*, i.e. splitted contigs or scaffolds

Assembly errors (2)

Is there a “global“ accuracy metric ?

Allpaths : % of blocks ($<$ 10kbp) aligning with $>$ 90% identity.

Assemblathon 1 : Number of structural errors (indels, misjoins) in the adjacency graph [Paten 11].

QUAST : Number of splitted alignments.

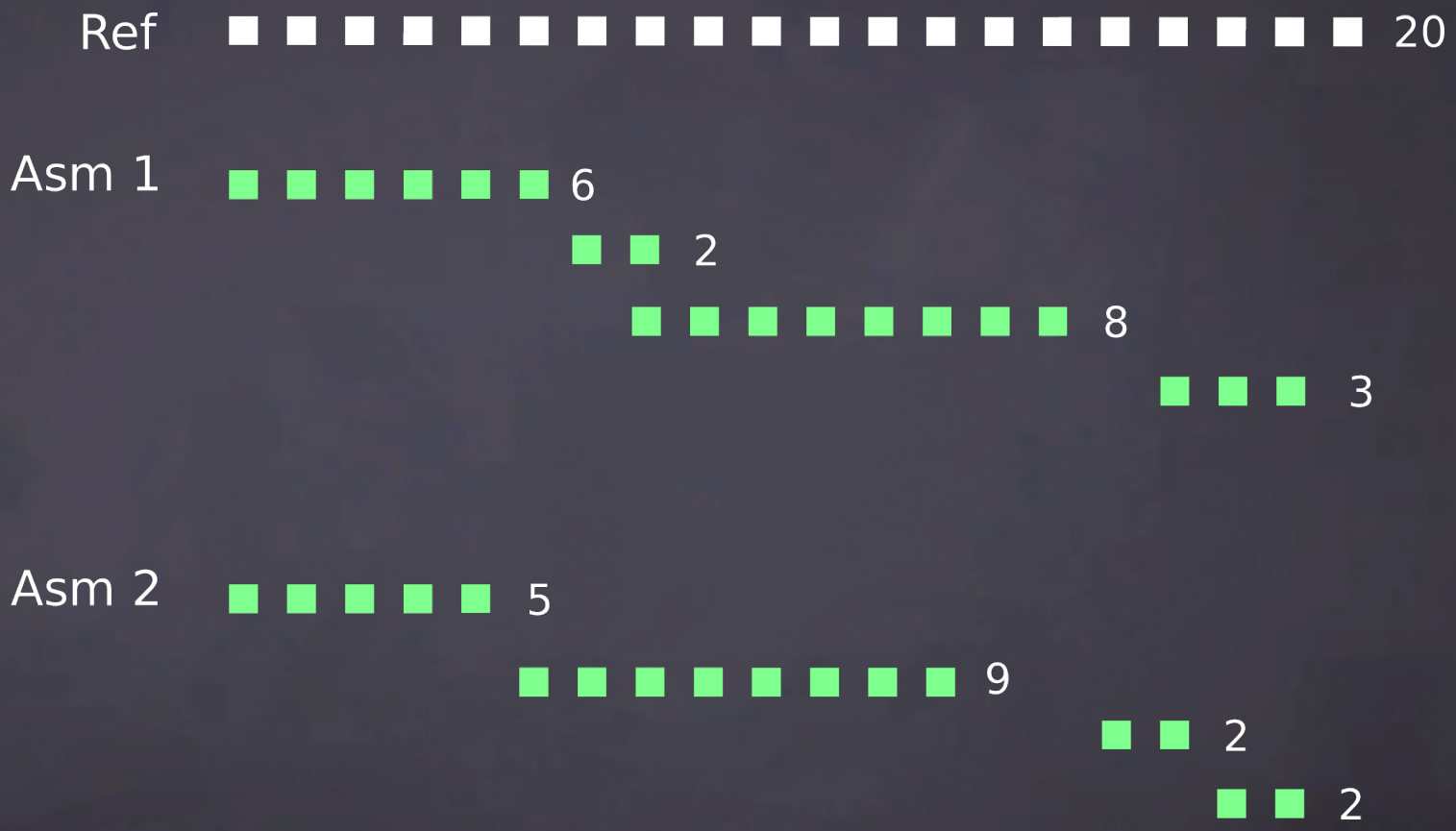
Assembly quality software

In order of preference :

1. With or without a reference genome, the QUAST software is highly recommended.
2. Assemblathon and GAGE evaluation scripts
3. Many perl/python scripts can compute basic reference-free metrics (N50).

EXERCICE

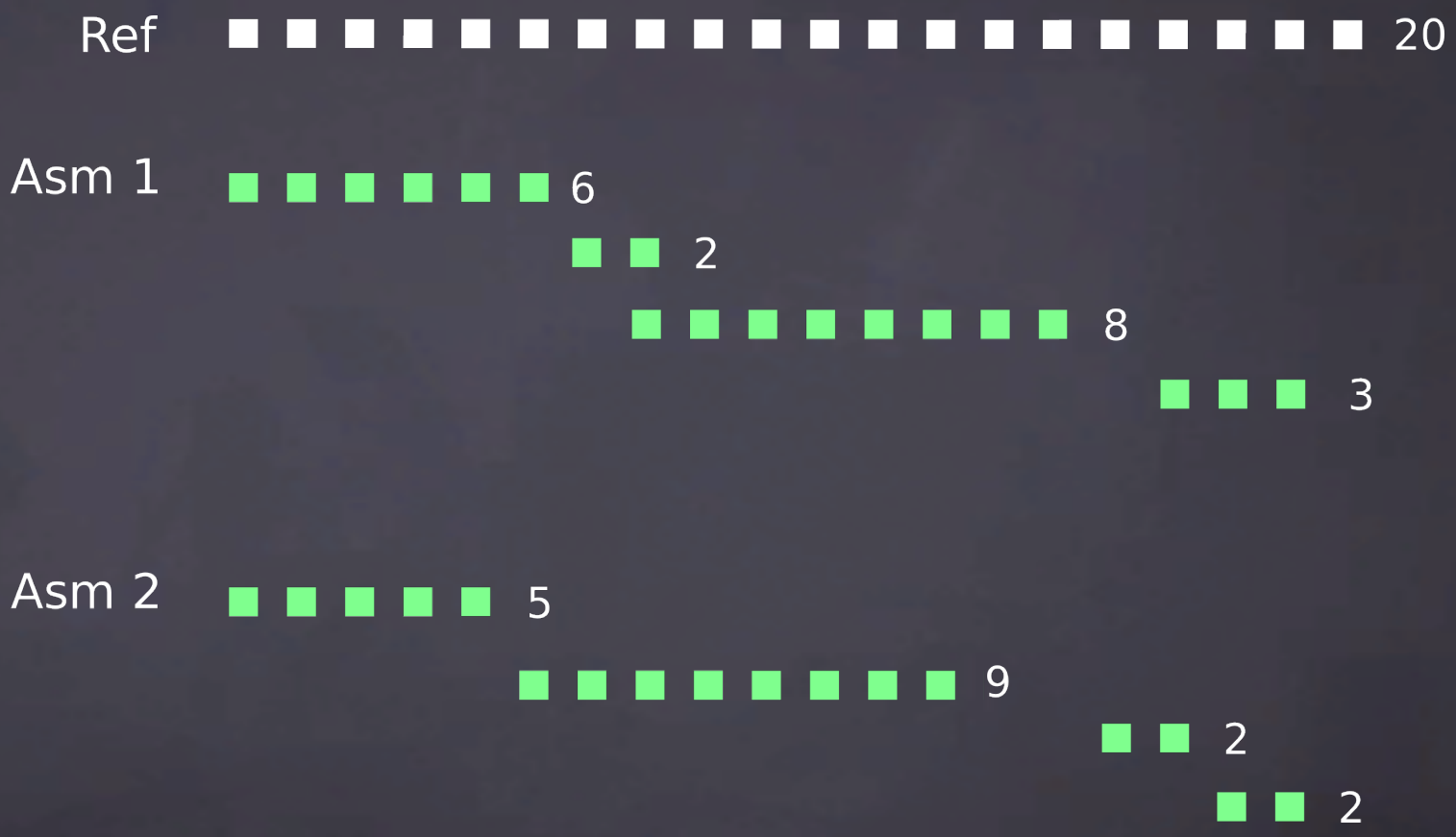
Here are two assemblies, aligned to the same reference :



- For each, compute the following metrics :
  - Total size of the assembly, N50, NG50 (bp)
  - Coverage (%)
- Which one is better than the other ?

EXERCICE (SOLUTION)

Here are two assemblies, aligned to the same reference :



- For each, compute the following metrics :
  - Total size of the assembly (19 bp, 18 bp), N50 (6 bp, 9 bp), NG50 (6 bp, 5 bp)
  - Coverage (%) (90, 90)
- Which one is better than the other ? (I would say first one)

Plan

Landscape

- Before Illumina Hi-Seq : Newbler for 454 (reads $>$ 200 bp), any de Bruijn graph assembler for Illumina (reads $<$ 100 bp).
- Now and later : **150** bp reads, **high** coverage, mate pairs : grey area for assembly techniques.

Short-Read Assemblers

Assembler	Method	Error Corr.	Remarks
Euler	de Bruijn	pre-assembly	Pioneer
Velvet	de Bruijn	in-assembly	(still) Popular
ABySS, CLC-bio, Meraculous, SOAPdenovo	de Bruijn	in-assembly	Parallel, large genomes
Allpaths LG	de Bruijn	pre-assembly	Needs short/long inserts
IDBA	de Bruijn	pre-assembly	Multi-k
Newbler, Celera	String	in-assembly	Long reads
Ray	de Bruijn	in-assembly	Parallel short/long reads
SGA, Fermi	String	pre-assembly	Compressed, promising
Minia	de Bruijn	in-assembly	ultra-low memory

DE NOVO METAGENOMIC/RNA ASSEMBLERS

de novo metagenomic assemblers :

Genovo : Pioneer. Assembles up to 10^5 454 reads [4].

MetaVelvet : based on Velvet [5]

Meta-Idba : based on IDBA..

RayMéta : based on Ray..

Too early to tell a preferred method.

de novo RNA assemblers :

Oases : Pioneer. A post-processing step for Velvet.

Trinity : *de facto* reference method [6]

Trans-Abyss : based on ABySS

SOAP-Trans : based on SOAPdenovo

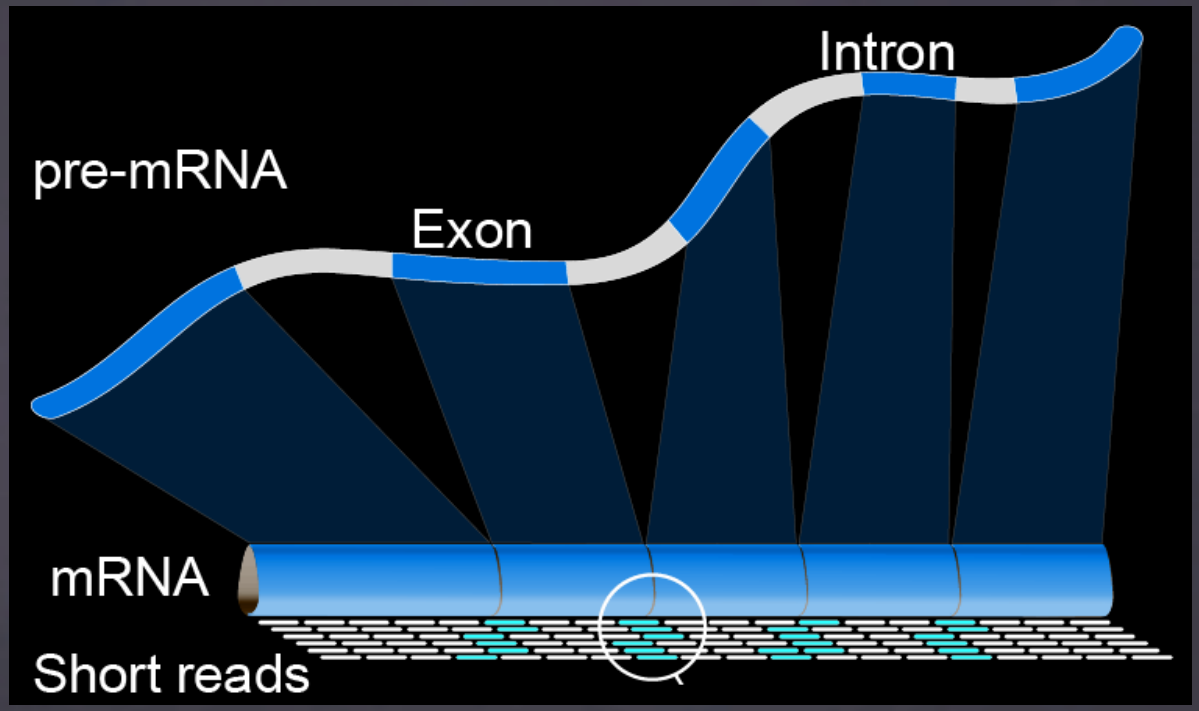
4. Recomb 2010, `http://cs.stanford.edu/genovo/`
5. `http://metavelvet.dna.bio.keio.ac.jp/`
6. `http://trinityrnaseq.sourceforge.net/`

Personal experience (for Illumina assembly)

If I had to choose one..

Your data follows the Broad recipe	Allpaths-LG
General purpose	SOAPdenovo2
If not enough memory	Minia
454	Newbler
RNA-Seq	Trinity
Metagenome	RayMéta (?)

RNA-SEQ AND ASSEMBLY



Goal : reconstruct mRNA sequences

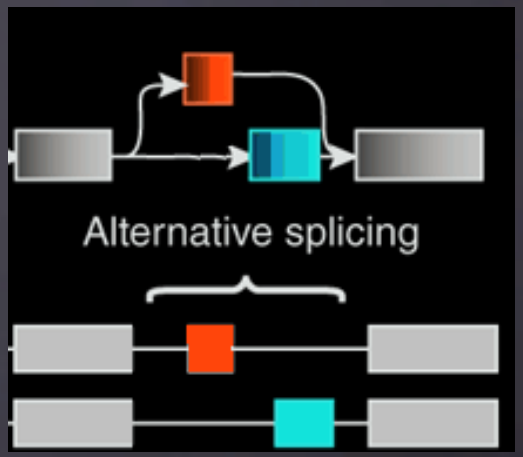
RNA-Seq Assembly

- Short contigs
- Uneven coverage
- Contigs are re-used

average mRNA length : 2 kbp

varying expression levels

alternative splicing



RNA-Seq Assembly

Despite these differences, DNA-seq assembly methods apply :

- Construct a de Bruijn graph (same as DNA)
- Output contigs (same as DNA)
- Allow to re-use the same contig in many different transcripts (new part)

RNA-Seq Assembly : Trinity

Quick overview of Trinity steps :

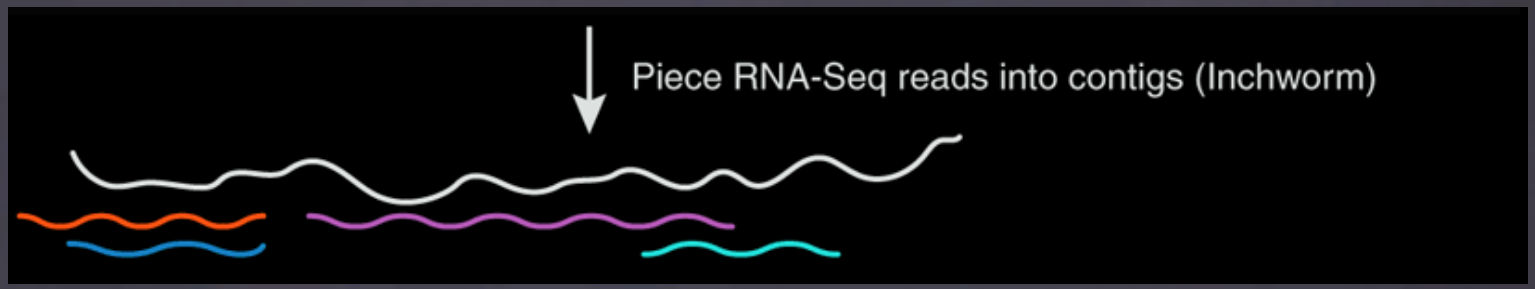
- Inchworm
- Chrysalis
- Butterfly

RNA-Seq Assembly : Trinity

- ~~Inchworm~~ de Bruijn graph construction, part 1
- ~~Chrysalis~~ de Bruijn graph construction, part 2, then partitioning
- ~~Butterfly~~ Graph traversal using reads, isoforms enumeration

RNA-Seq Assembly : Trinity - 1

- **Inchworm** - de Bruijn graph construction, part 1

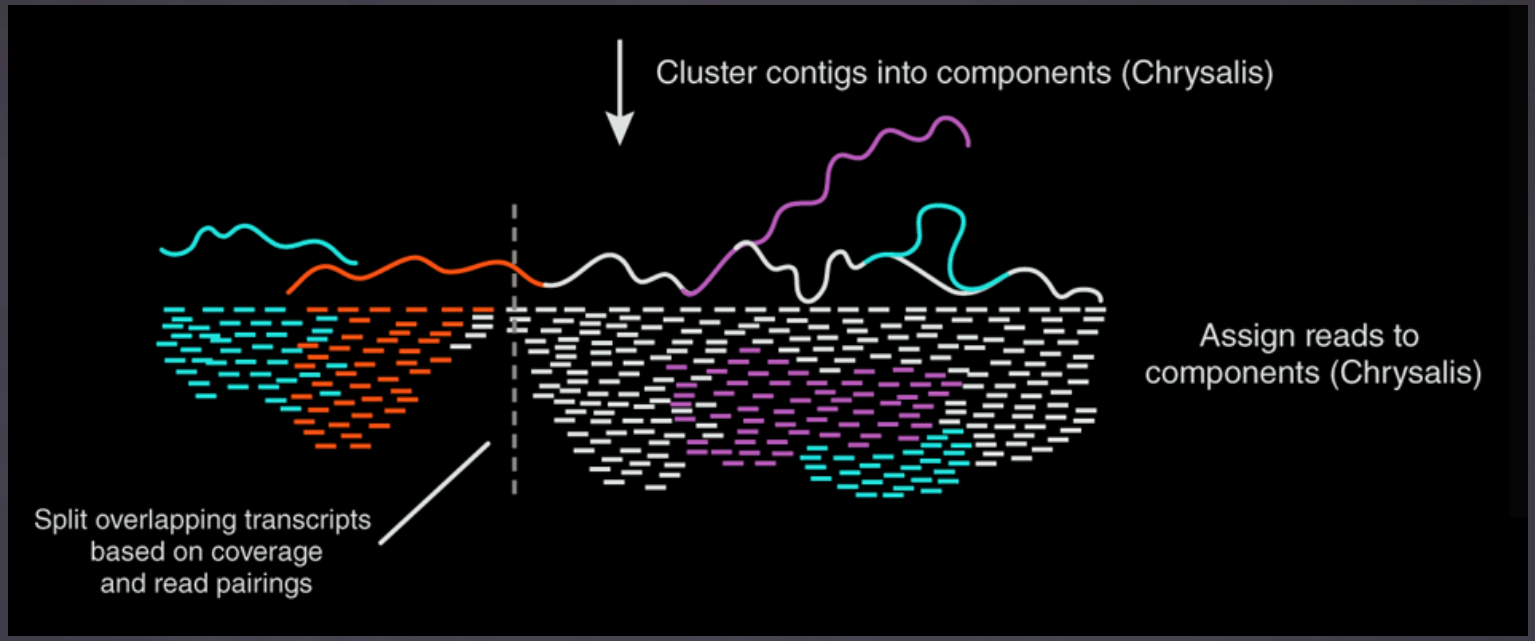


Using k-mers, construct contigs carelessly.

Contigs might correspond to the most abundant isoform, but no guarantee.

RNA-Seq Assembly : Trinity - 2

- **Chrysalis** - de Bruijn graph construction, part 2, then reads partitioning



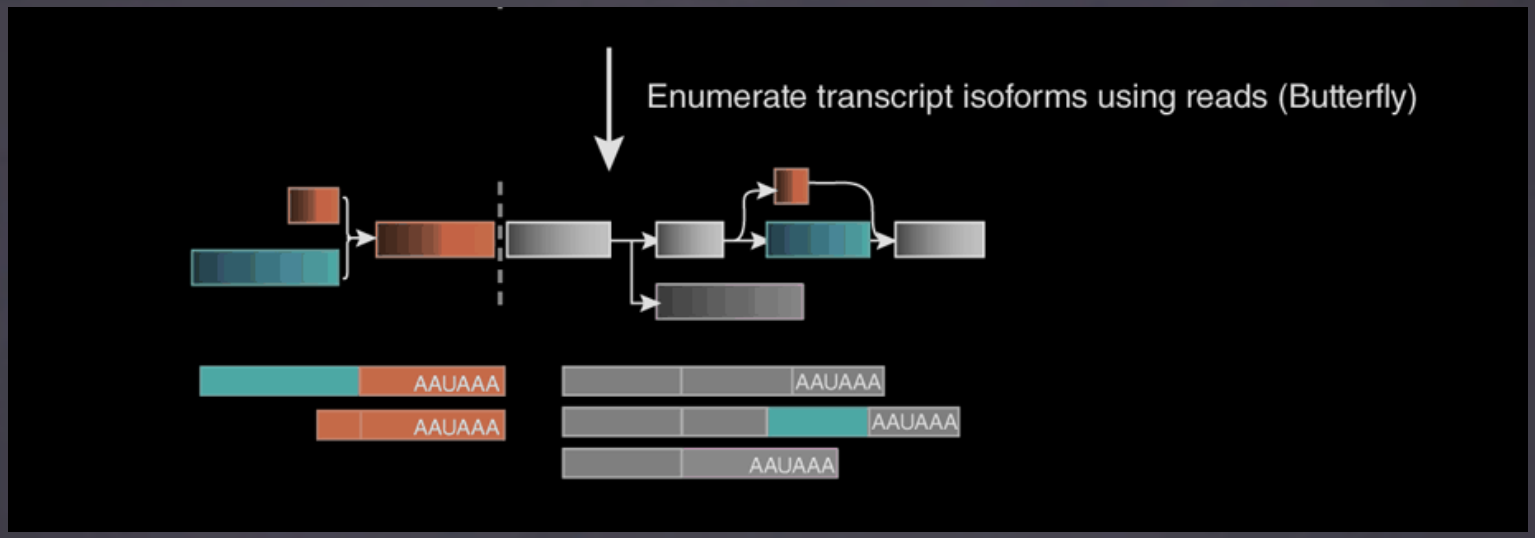
By overlapping Inchworm contigs, construct the true de Bruijn graph.

Then,

Partition the graph and output the reads aligning to each partition.

RNA-Seq Assembly : Trinity - 3

- **Butterfly** - Graph traversal using reads, isoforms enumeration



Traverse each de Bruijn graph partition to output isoforms

Difference with DNA-seq assembly : isoforms are, by definition, not *k*-mer-disjoint.

Tips

General assembly advice follows

The k parameter

There is no optimal k-mer size, it varies with each dataset.

A few things to keep in mind :

- Low limit : For common genomes sizes (10 Mbp - 1 Gbp), there is a high chance that any $\approx$ 12-mers will be repeated in many locations ($4^{12} = 16 \cdot 10^6$).
- High limit : with very good error-correction, the Broad typically uses $k = |readlen| - 1$..
- Ideally, k should be the highest value such that each error-free k-mer is present ≥ 2 times in the reads.
- If you have time, re-assemble with many different k values.

UPDATE (8/11/2013) : Google our new tool : "KmerGenie".

Error correction

Except if you have excellent coverage, error-correction may help getting better assemblies.

- Allpaths-LG stand-alone error corrector (highly recommended)
- Quake
- SOAPdenovo stand-alone corrector

A good assembly is typically done with several pre-correction stages :

- low-quality reads removal
- trimming
- overlapping paired reads merged into single reads

Scaffolding

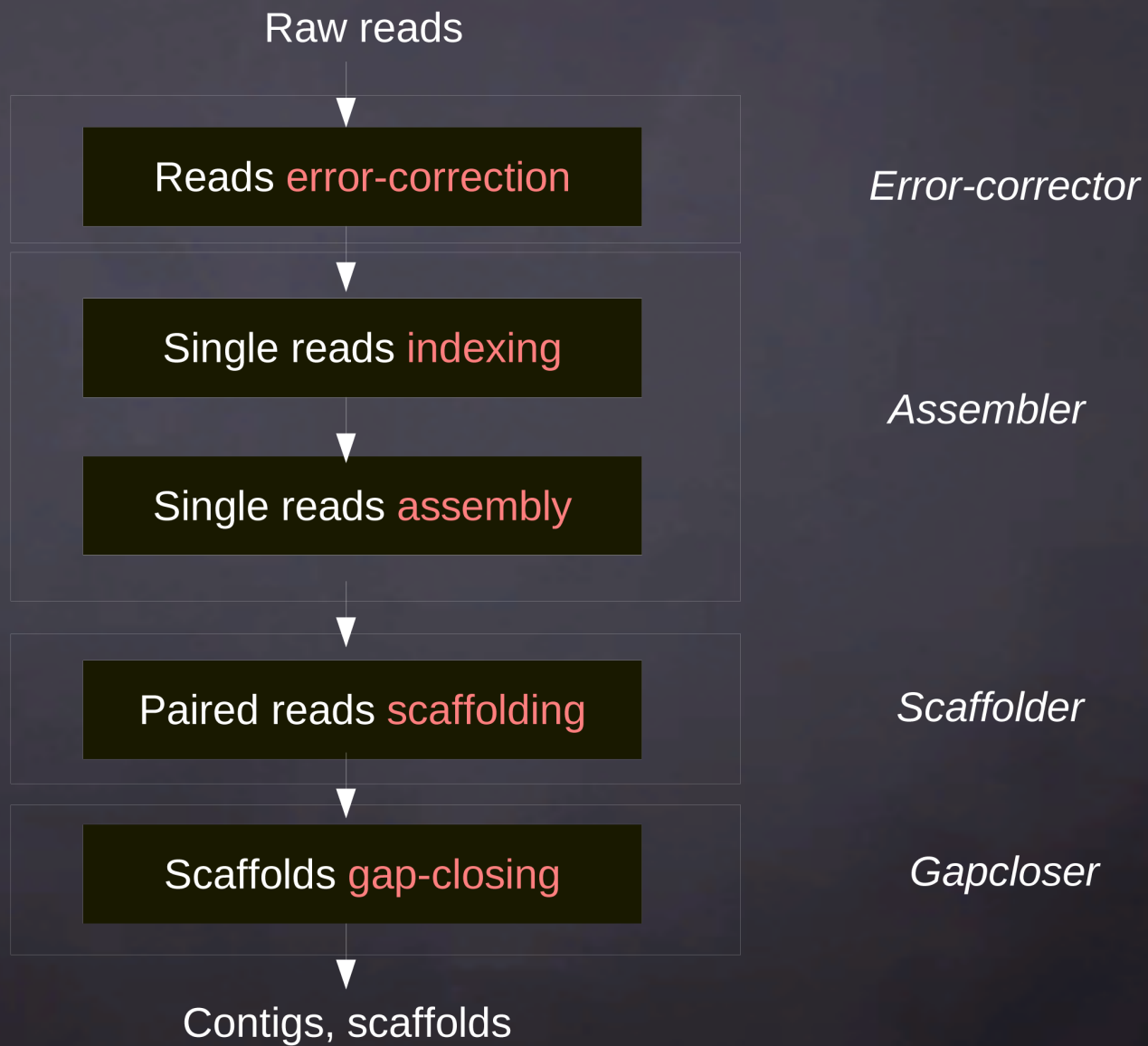
Scaffolding is the step that maps paired reads to contigs in order to create scaffolds.

If an assembly software returns scaffolds, that means that it includes its own scaffolder (SOAP, SGA, ABySS, Velvet..).

Several stand-alone scaffolders are also developped, and some give good practical results.

E.g. : SSPACE (generally outperforms Bambus 2, Opera, etc..)
I haven't tried it yet, but SOAPdenovo2 scaffolder looks promising.

A typical assembly **pipeline**



Last Exercice

Reads :

1. `AGTC`
2. `TCAA`
3. `AATT`
4. `GTCT`
5. `TATT`
6. `TCTA`

1. Assemble these reads
2. What was special about this genome ?

Last Exercice (solution)

1. `AGTCAATT`
   `AGTCTATT`
2. "diploid genome", 1 SNP

Plan

Minia

How the assembler Minia works : Warning : slides taken from a computer science talk

1. Storing the de Bruijn graph in memory
2. Actual contigs construction procedure

de Bruijn graph [Idury, Waterman 95]

Nodes are k-mers, edges are $(k-1)$-overlaps between nodes.

Only **nodes** need to be encoded, as **edges** are inferred.

How to encode the de Bruijn graph using as little space as possible ?

Memory usage (illustration for $k = 25$)

- Explicit list : $2k \cdot n$ bits — 50 bits per node
- Self-information of n nodes : [Conway, Bromage 11]

$$\log_2 \left(\left(\binom{4^k}{n} \right) \right) \text{ bits}$$

20 bits per node.

Bloom filter

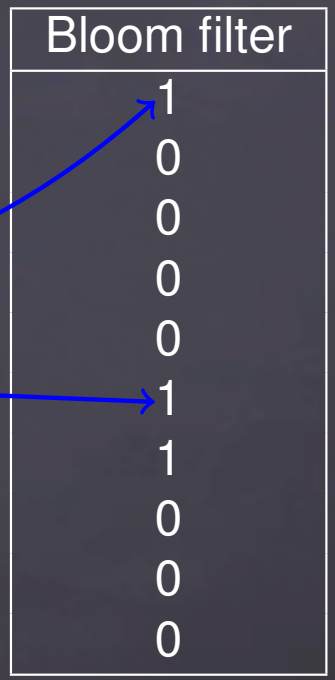
Bit array to represent any set with a "precision" of ϵ.

- a proportion ϵ of elements will be wrongly included (*false positives*).

To represent a set of n elements, requires $\approx 1.44 \log_2(\frac{1}{\epsilon}) \cdot n$ bits.

Storing k-mers in a Bloom filter :

k-mer	hash value
ATC	0
CCG	0
TCC	5
CGC	6
...	...



Bloom filter

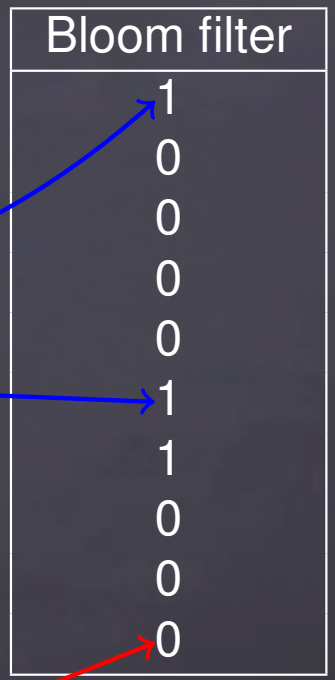
Bit array to represent any set with a "precision" of ϵ.

- a proportion ϵ of elements will be wrongly included (*false positives*).

To represent a set of n elements, requires $\approx 1.44 \log_2(\frac{1}{\epsilon}) \cdot n$ bits.

Storing k-mers in a Bloom filter :

k-mer	hash value
ATC	0
CCG	0
TCC	5
CGC	6
...	...



Queries :
Is the k-mer ATA (hash value 9) present ? **No**.

Bloom filter

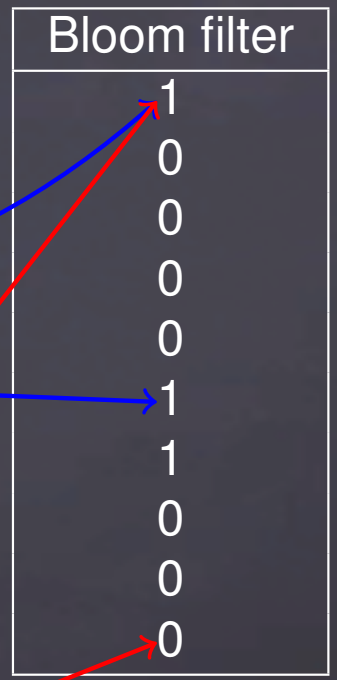
Bit array to represent any set with a "precision" of ϵ.

- a proportion ϵ of elements will be wrongly included (*false positives*).

To represent a set of n elements, requires $\approx 1.44 \log_2(\frac{1}{\epsilon}) \cdot n$ bits.

Storing k-mers in a Bloom filter :

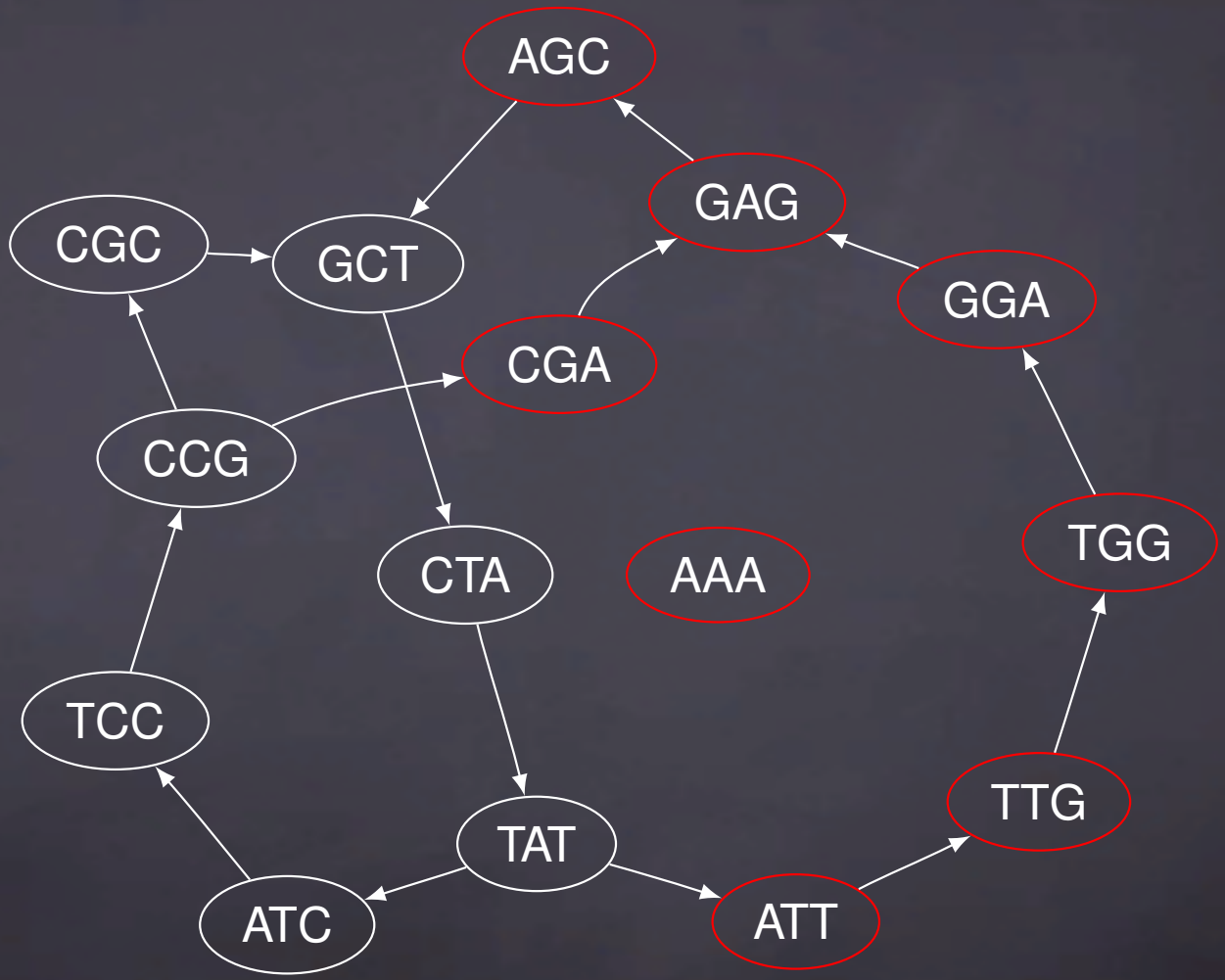
k-mer	hash value
ATC	0
CCG	0
TCC	5
CGC	6
...	...



Queries :
Is the k-mer ATA (hash value 9) present ? **No**.

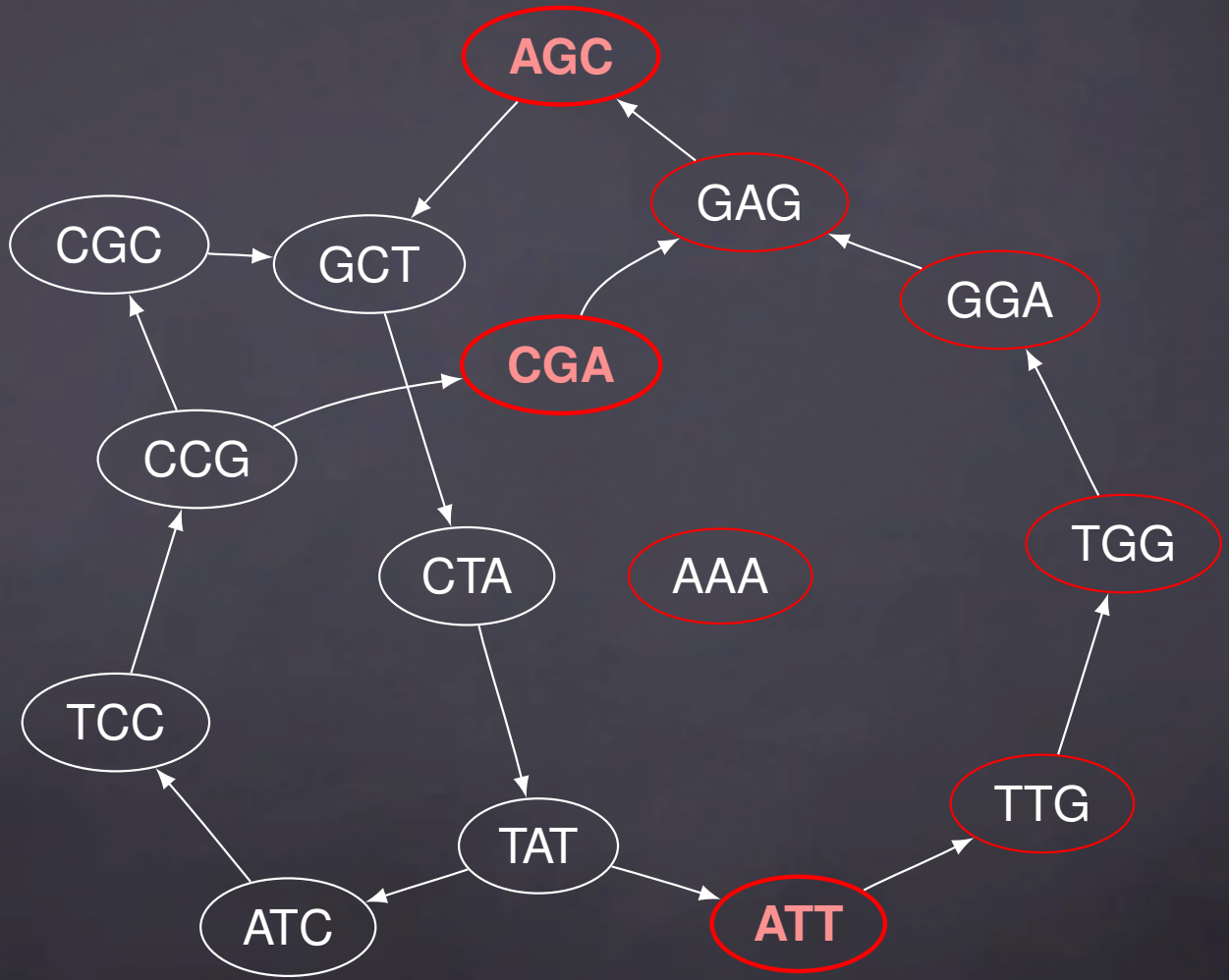
AAA (hash value 0) present ? **Yes**, maybe : either a true or a false positive.

Set of **nodes** : {TAT, ATC, CGC, CTA, CCG, TCC, GCT}
Graph as stored in a Bloom filter : [Pell et al 12]



Black nodes : true positives ; **Red** nodes : false positives

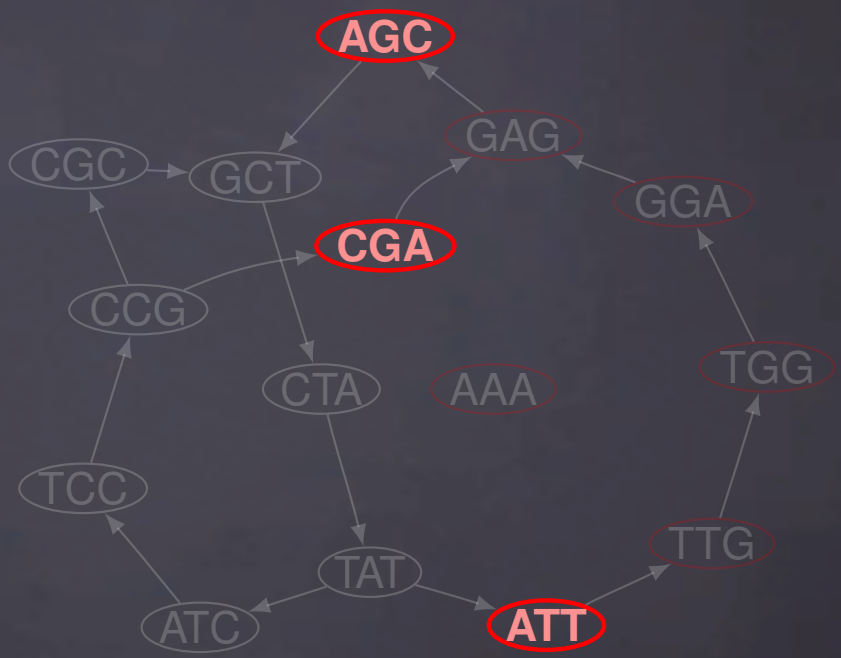
Insight : to **traverse** the graph from **true positive** nodes, only a **small fraction of the false positives** need to be **avoided** (*critical false positives, CFP*).



Proposed method

Store **nodes** on **disk** for sequential enumeration,
and in **memory** store the **Bloom filter** + the critical FPs **explicitly**.

Bloom filter
1
0
0
0
0
1
1
0
0
0



Nodes self-information :

$$\lceil \log_2 \binom{4^3}{7} \rceil = 30 \text{ bits}$$

Our structure size :

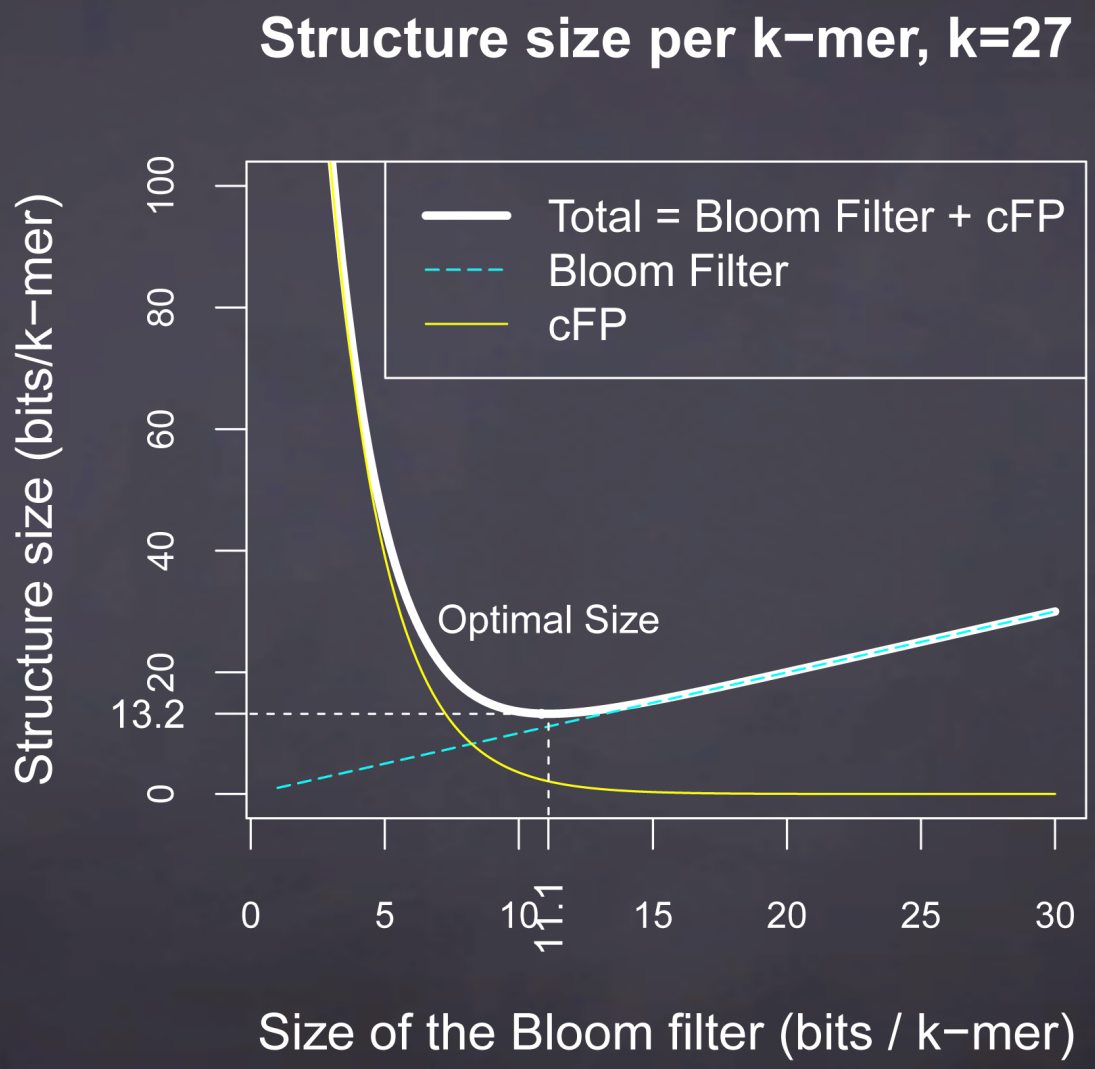
$$\underbrace{10}_{\text{Bloom}} + \underbrace{3 \cdot 6}_{\text{Crit. false pos.}} = 28 \text{ bits}$$

Construction time (for n nodes)

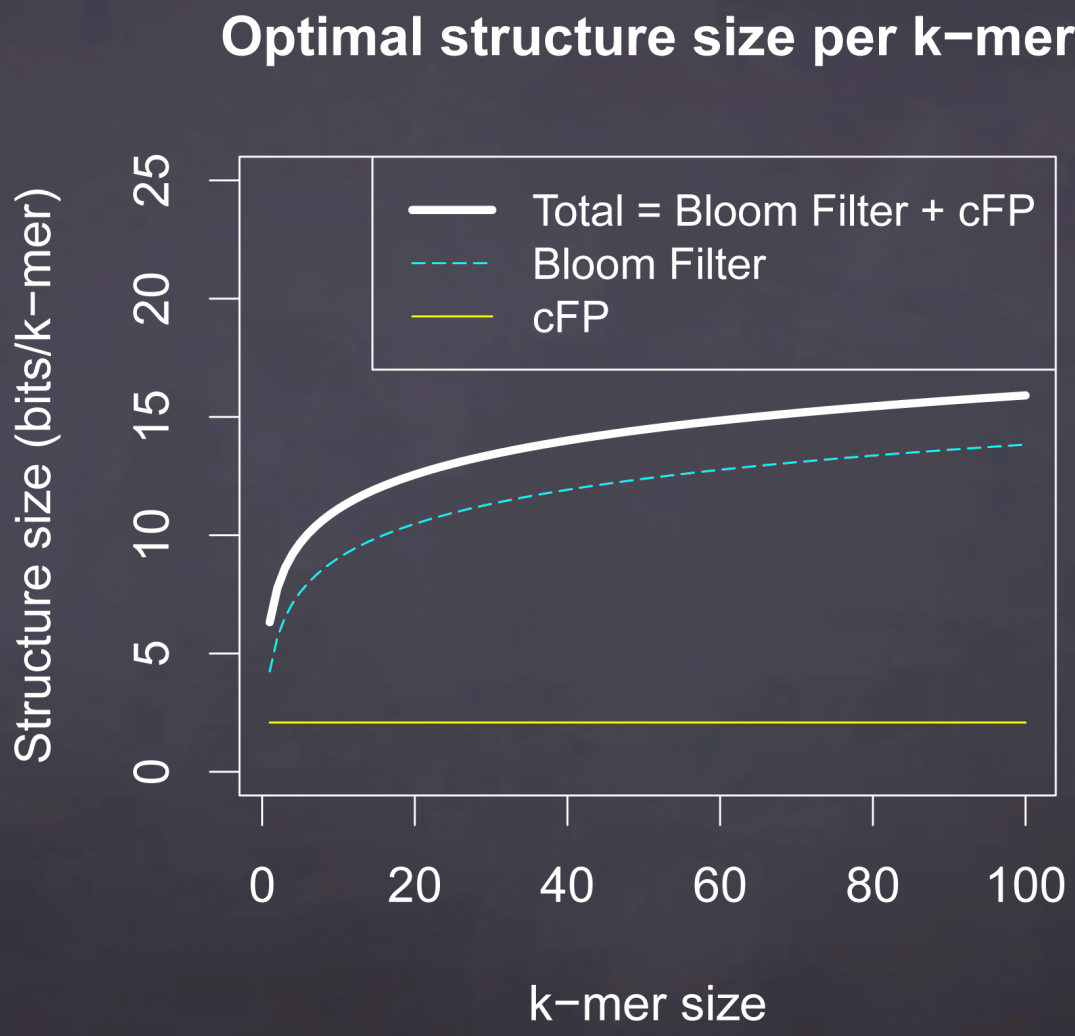
Assume that k-mer arithmetic takes constant time.

- Bloom filter construction : $O(n)$
- cFP construction :
  - Enumeration of neighbors of all graph nodes, keeping only Bloom-positive neighbors : $O(n)$
  - Intersection between Bloom-positive neighbors and nodes, with limited memory usage : $O(\frac{k}{\log(k)} n)$

Optimal Bloom filter size



Dependence on the parameter k



Result statement

The de Bruijn graph can be encoded using

$$\underbrace{1.44 \log_2(\frac{16k}{2.08})}_{\text{Bloom}} + \underbrace{2.08}_{\text{cFP}}$$

bits of memory per node.

$k = 25$: **13** bits per node.

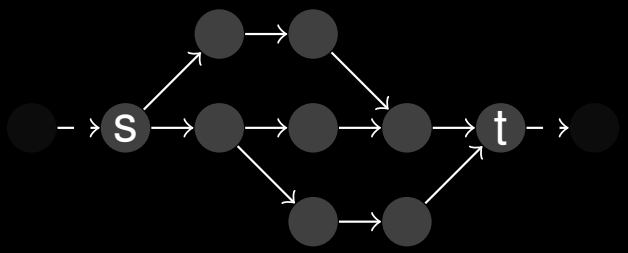
- Below the self-information (20 bits/node for $k = 25$)
- The part stored in memory doesn't support enumeration of nodes, only traversal

Graph-based assemblers typically **modify the graph** to remove artifacts (variants, errors).
Is it possible to perform *de novo* assembly with this (immutable) structure ?
$\rightarrow$ **Yes, using localized traversal**. [RC DL, WABI 11]

Localized traversal

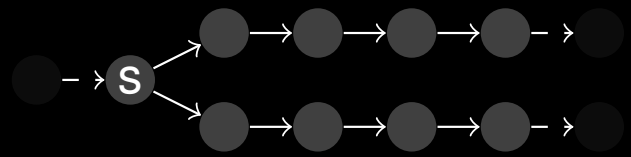
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



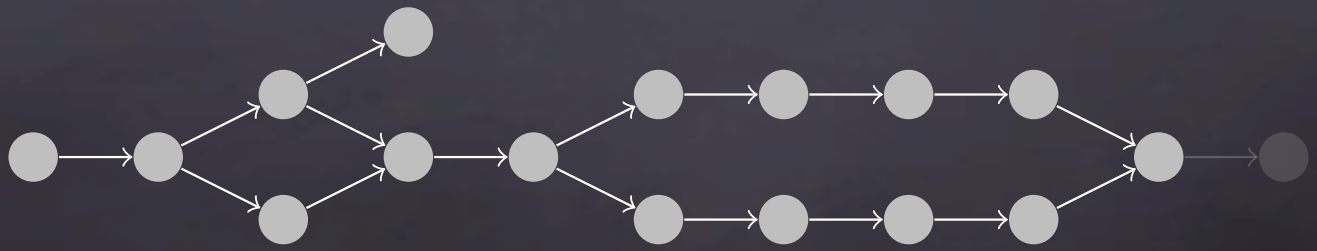
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



BFS from s, breadth remains > 1 for depths $1..d$

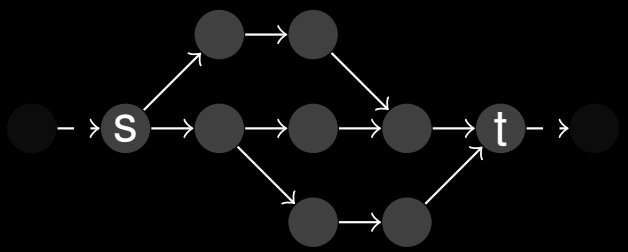
Example : Whole graph



Localized traversal

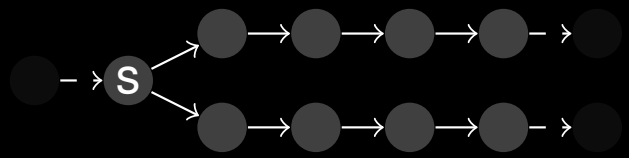
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



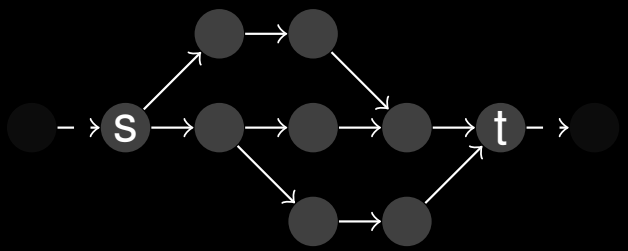
BFS from s, breadth remains > 1 for depths $1..d$

Example : Start with an empty graph

Localized traversal

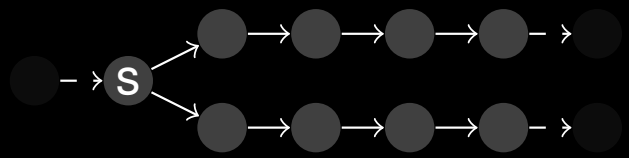
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



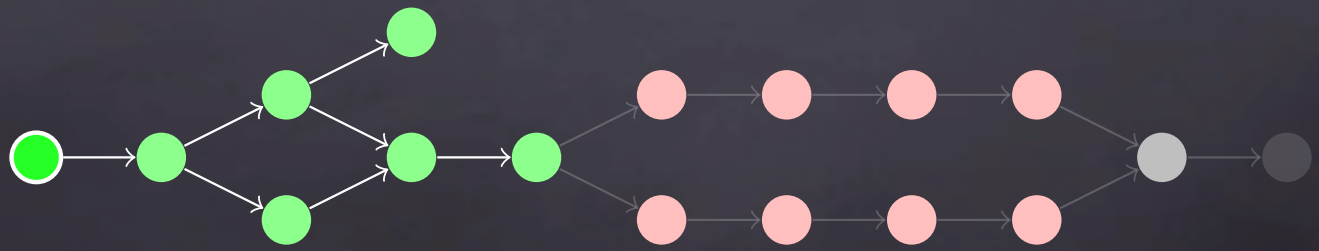
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



BFS from s, breadth remains > 1 for depths $1..d$

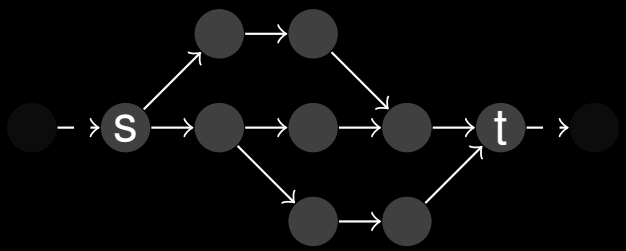
Example : Pick a new node, construct the first portion



Localized traversal

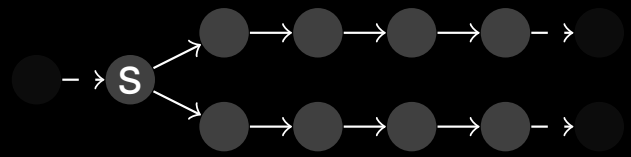
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



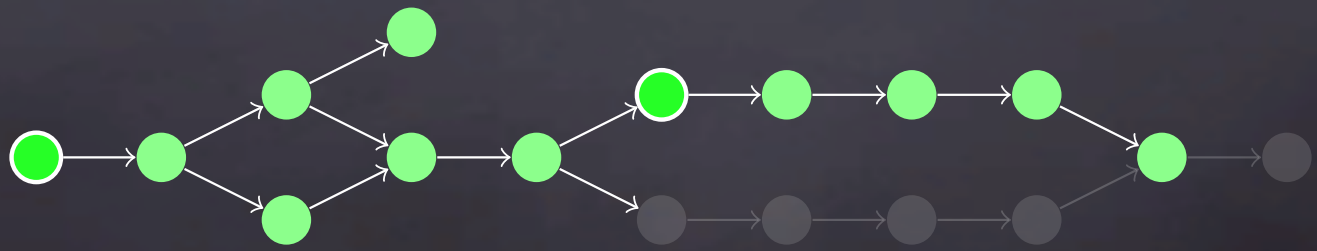
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



BFS from s, breadth remains > 1 for depths $1..d$

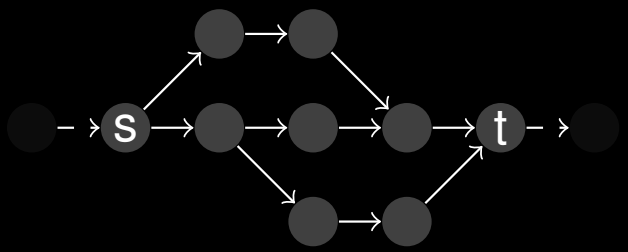
Example : Construct the second portion



Localized traversal

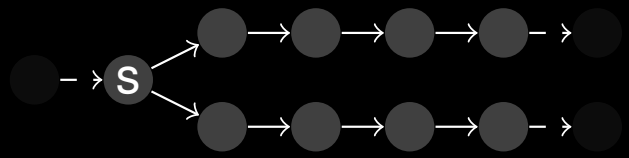
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



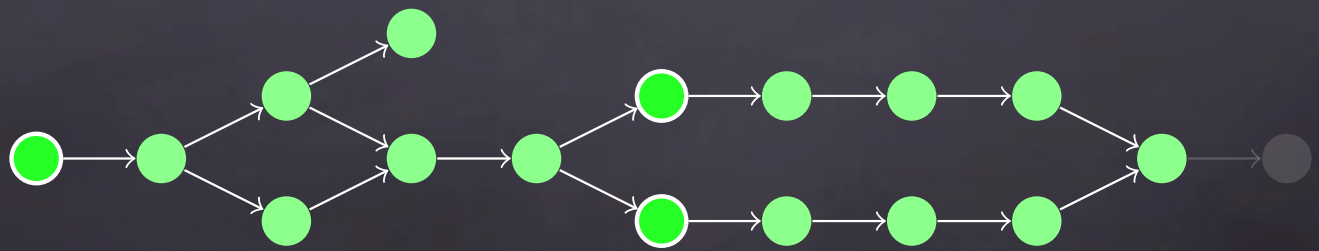
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



BFS from s, breadth remains > 1 for depths $1..d$

Example : Construct the third portion



Assembler implementation

k-mer counting
- Need to determine the set of **solid** nodes (seen $\geq x$ times)
- Current methods (e.g. Jellyfish) require **more memory** than our structure
- We designed a constant-memory k-mer counting procedure

Graph traversal
- Nodes which have already been traversed need to be **marked**
- No extra information can be stored in our structure
- We used a **separate** hash table to remember if **branching** or **dead-end** nodes have already been visited.

Contigs construction
Consensus from each path obtained by localized traversal

Plan

Step 1 : Data preparation

1. Download raw human genome reads from a public FTP server (SRX016231)
2. Decompress them
3. Create a list of all FASTQ files (`HG_reads.txt`)

Assembling a human genome with Minia (2)

Step 2 : Running Minia

Command line :

```
./minia HG_reads.txt 27 5 3000000000 human_assembly
```

Assembling a human genome with Minia (3)

Step 3 : Evaluate results

Human genome assembly	**Minia**	C. & B.	ABySS	SOAPdenovo
Value of *k* chosen	27	27	27	25
Contig N50 (bp)	**1156**	250	870	886
Sum (Gbp)	2.09	1.72	**2.10**	2.08
$>$ 95% Accuracy (%)	**94.6**	-	94.2	-
Nb of nodes/cores	1/1	1/8	21/168	1/40
Time (wall-clock, h)	23	50	15	33
Memory (sum of nodes, GB)	**5.7**	**32**	**336**	**140**

Conclusion, what we have seen

- What is a good assembly ?
  - No total order
  - Main metrics : N50, coverage, accuracy
  - Use QUAST
- How are assemblies made ?
  - Typically, using a de Bruijn graph or a string graph.
  - Errors and small variants are removed from the graph.
  - Contigs are just simple paths from the graph.
- Assembly software
  - Recommended software for Illumina data : SOAPdenovo2, Allpaths-LG
  - Plethora of other software for custom needs : Minia for low-memory, SGA for very accurate assembly, etc..
  - Recommended software for 454 data : Newbler, Celera
- A few tips
  - How to choose k : always try many values
  - Put the assembler inside a pipeline : error correction, scaffolding, gap-filling
- Case study
  - How to assemble a human genome with Minia