Space-efficient and exact de Bruijn graph representation based on a Bloom filter

Rayan Chikhi, Guillaume Rizk

ENS Cachan Brittany / IRISA, France
AlgoRizk, France

WABI 2012

UMR IRISA

ENS CACHAN BRETAGNE

Transition from the previous talk

The **previous talk** dealed with a **succinct de Bruijn graph representation**.
This talk covers exactly the **same topic**, with some differences :

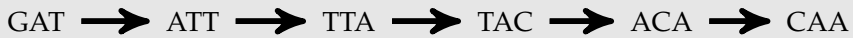
- (Hopefully) simpler data structure
- Less succinct
- Implemented in an assembly program

Outline

Presentation of the data structure

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]

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

20 bits per node.

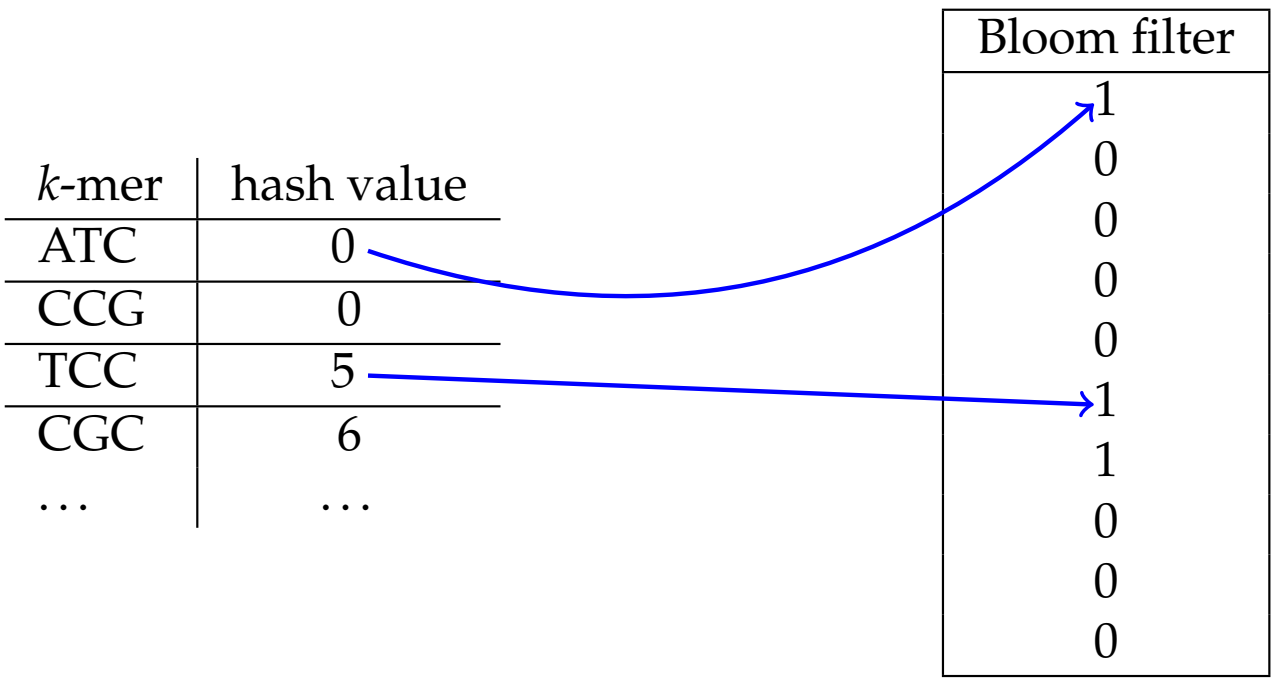
Bloom filter

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 :



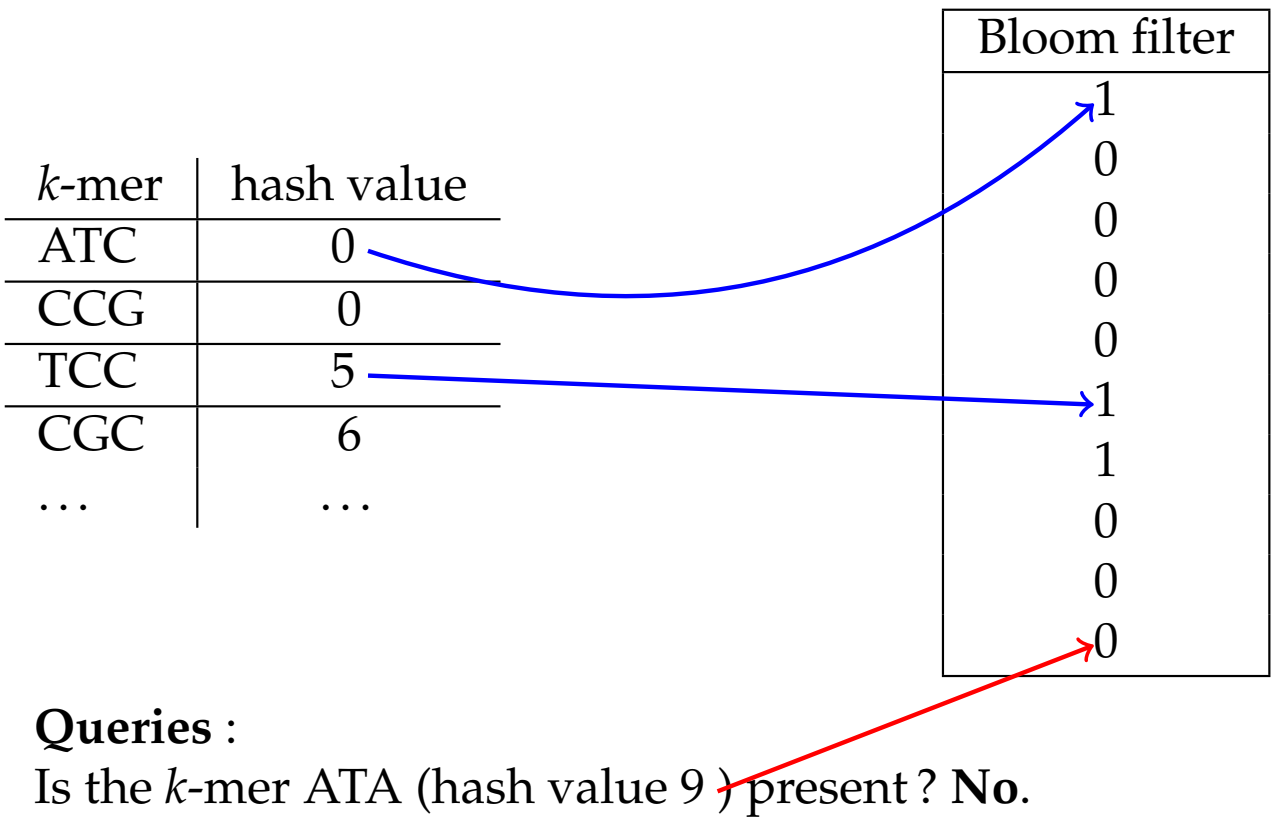
Bloom filter

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 :



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

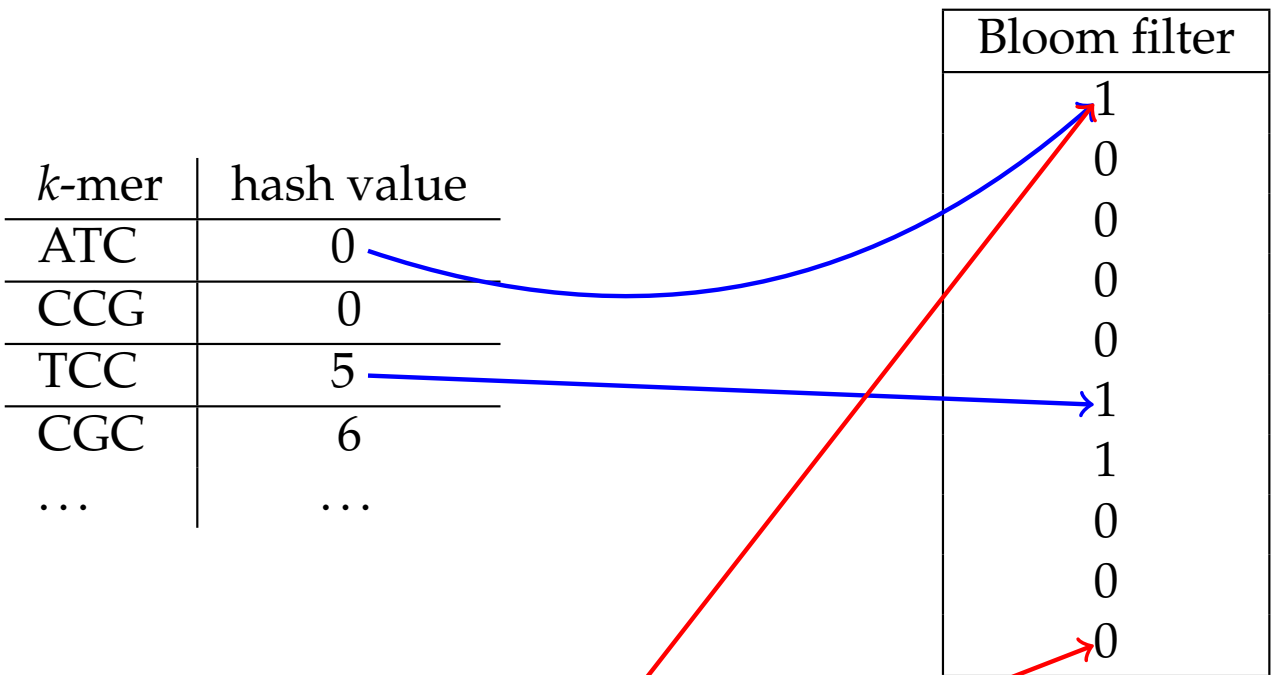
Bloom filter

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 :

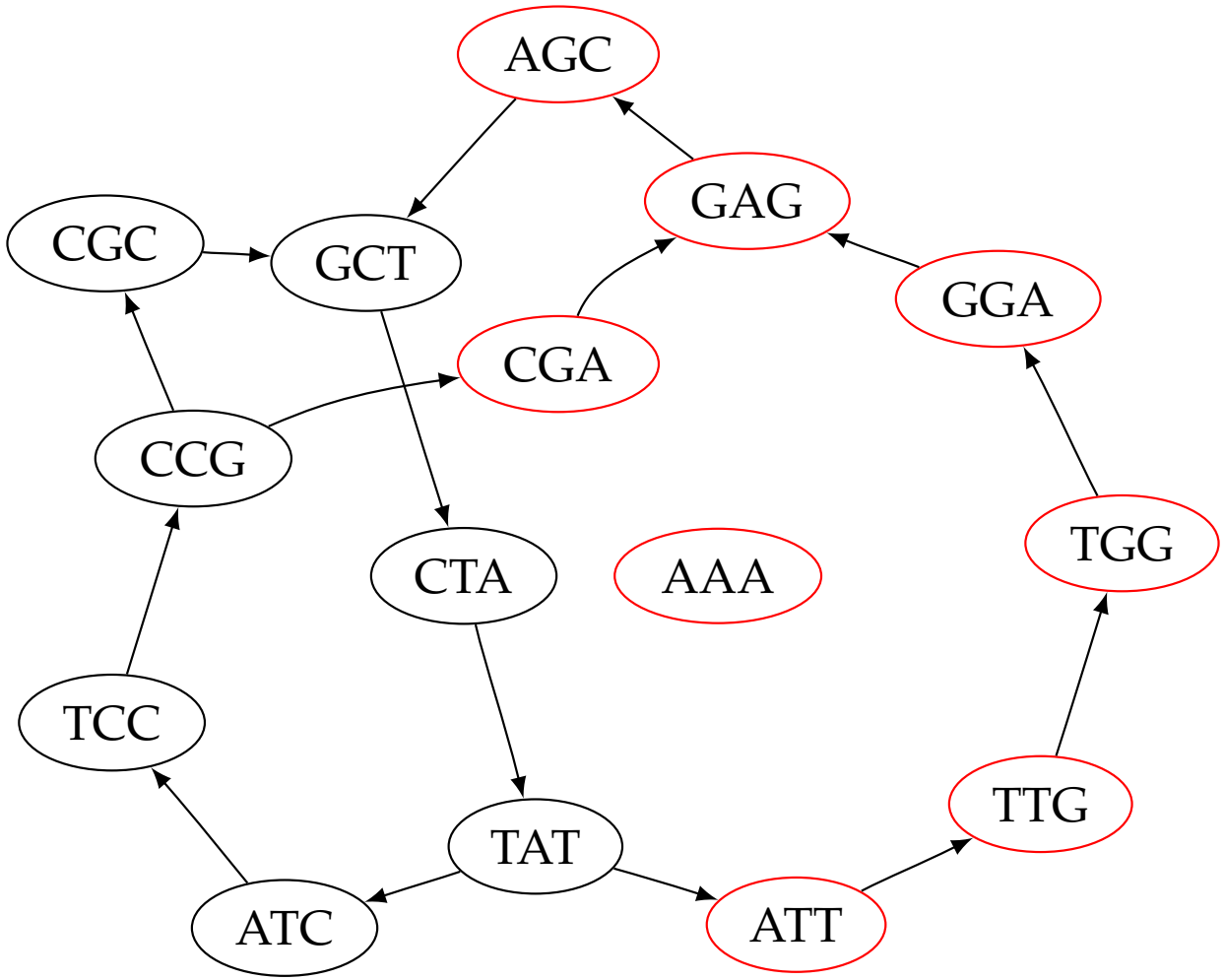


k-mer	hash value
ATC	0
CCG	0
TCC	5
CGC	6
...	...

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

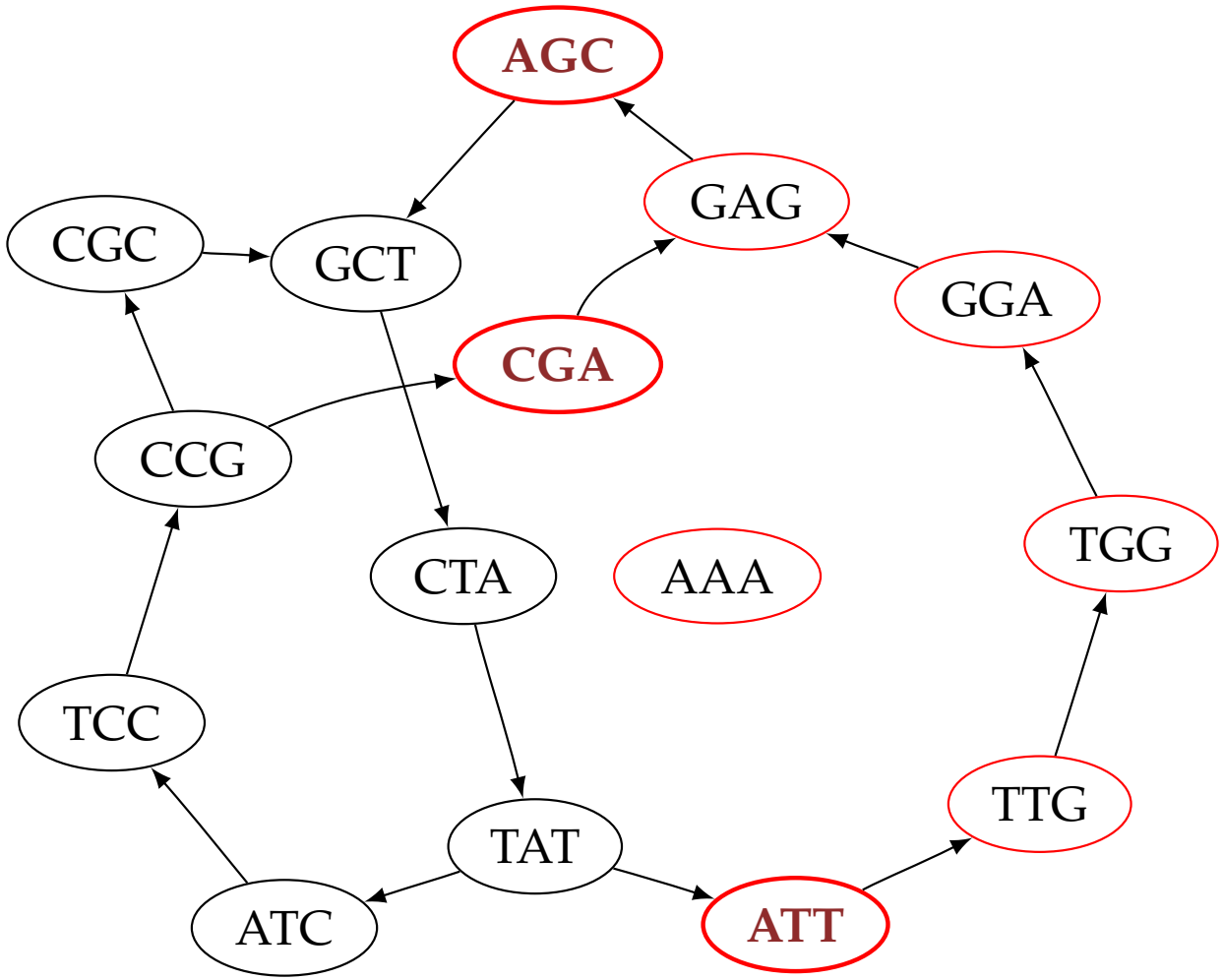
AAA (hash value 0) present ? **Yes**, maybe : either a true or a false positive.

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

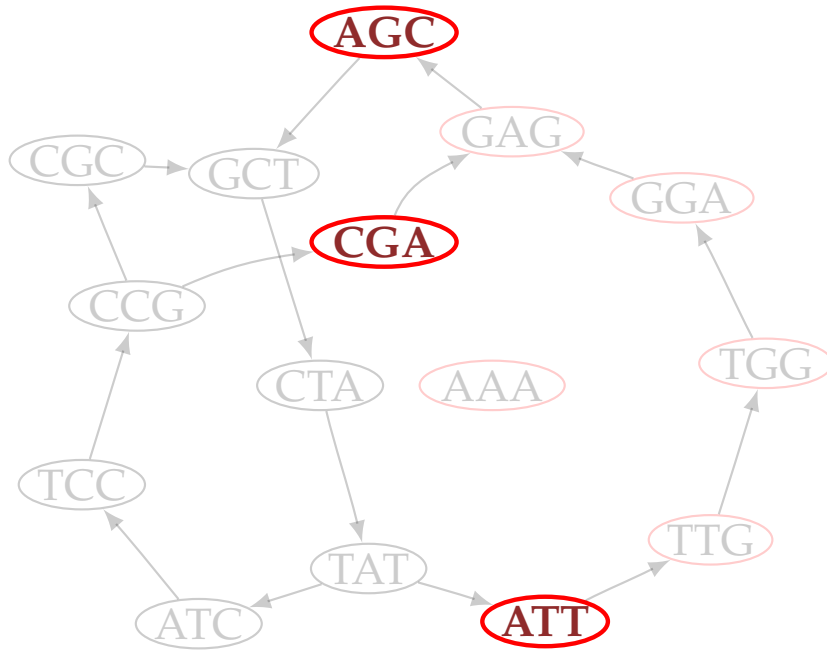
Insight : to **traverse** the graph from **true positive** nodes, only a **small fraction of the false positives** need to be **avoided** (*critical false positives*, *CFP*).



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 :

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

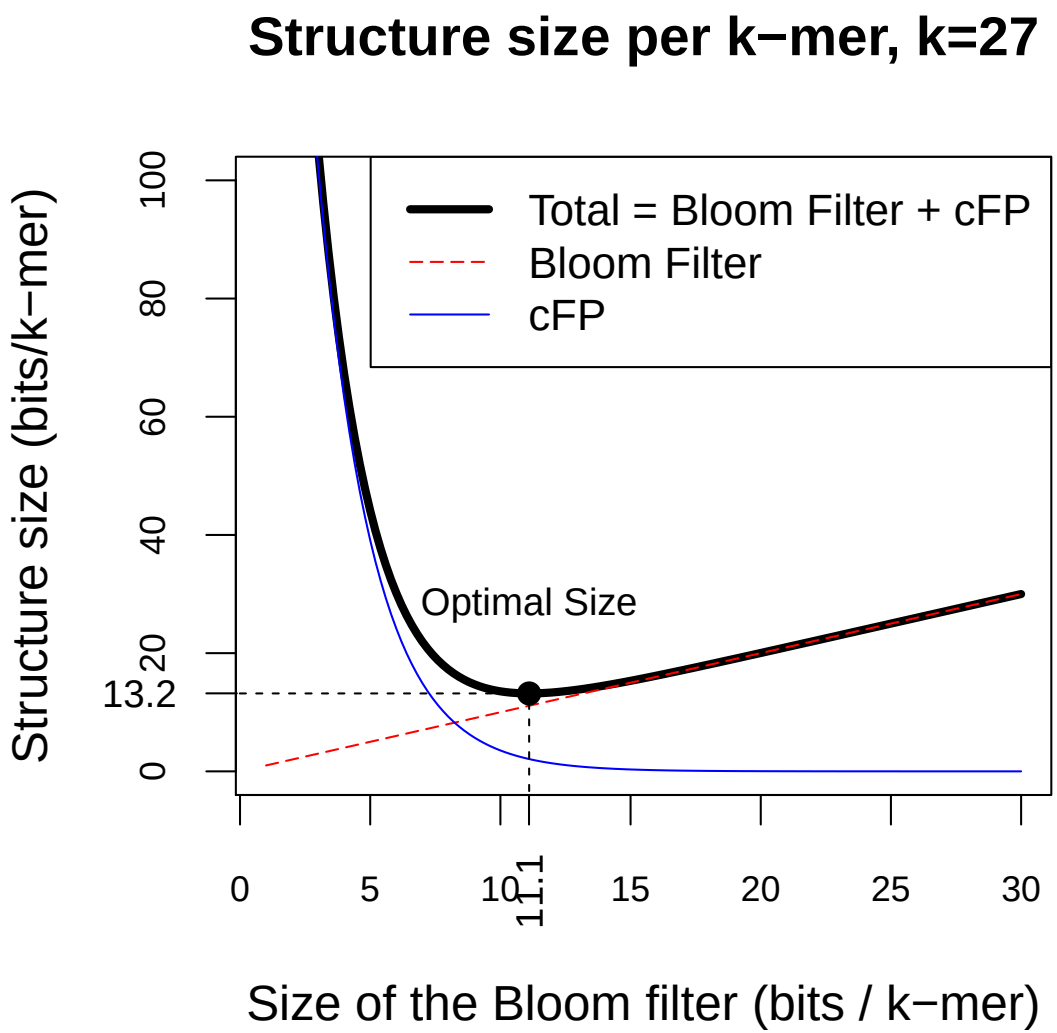
Analysis

Construction time (for n nodes)

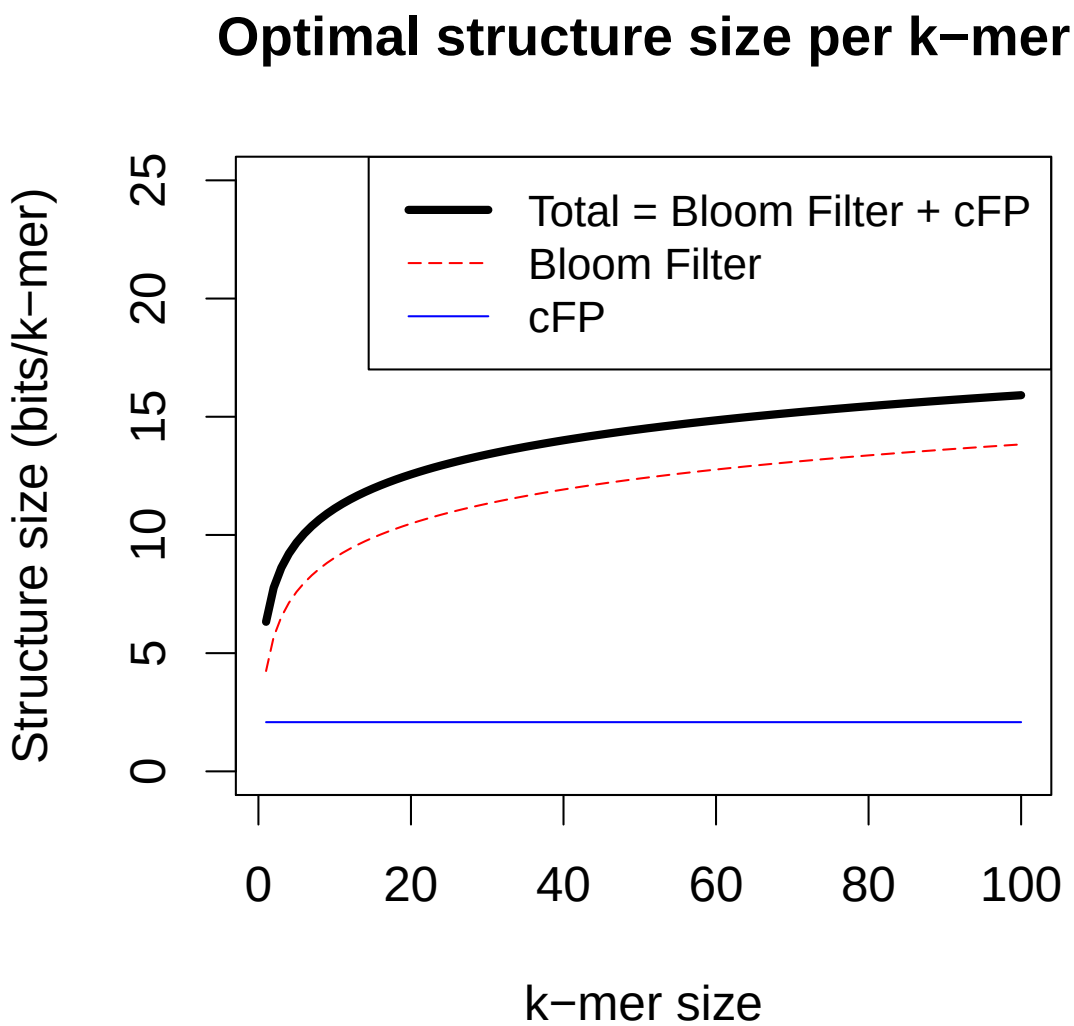
Assume that k-mer arithmetic takes constant time.

- 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

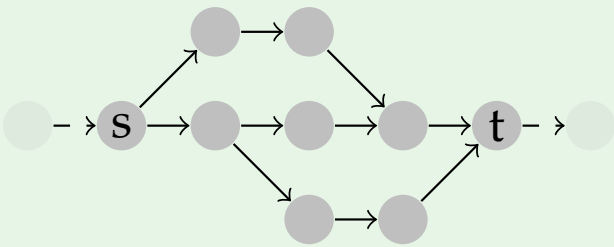
Graph-based assemblers typically **modify the graph** to remove artifacts (variants, errors).
Is it possible to perform *de novo* assembly with this (immutable) structure ?
$\rightarrow$ **Yes, using localized traversal**. [RC DL, WABI 11]

Assembly aspects

Localized traversal

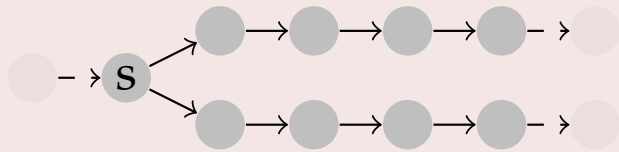
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



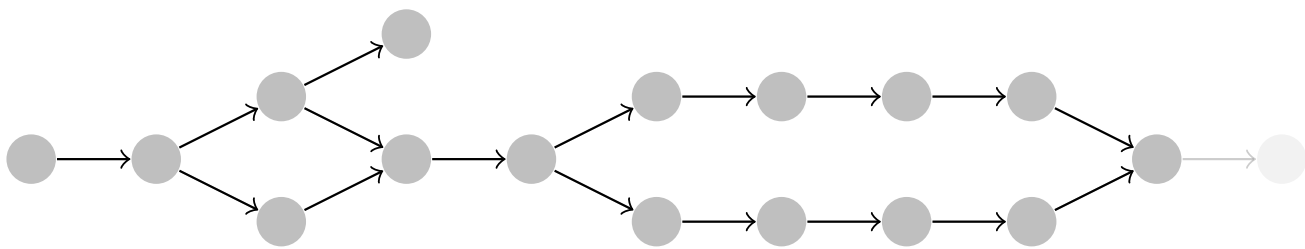
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



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

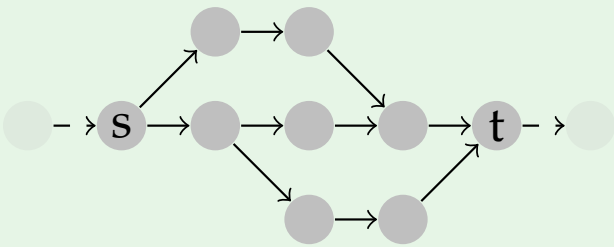
Example : Whole graph



Localized traversal

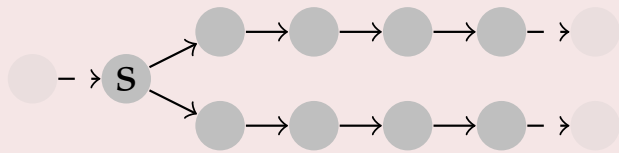
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



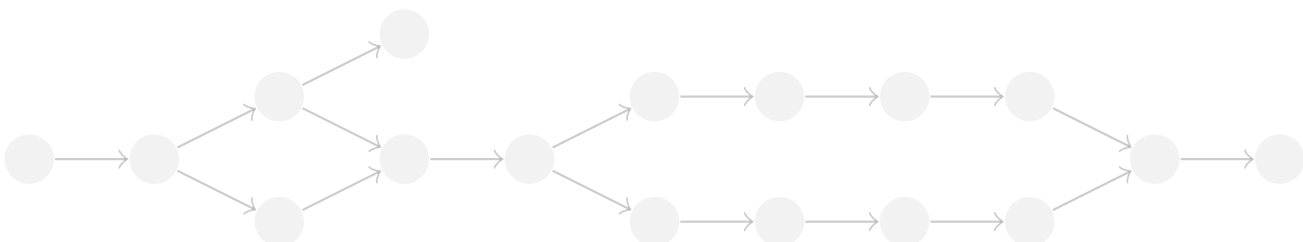
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



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

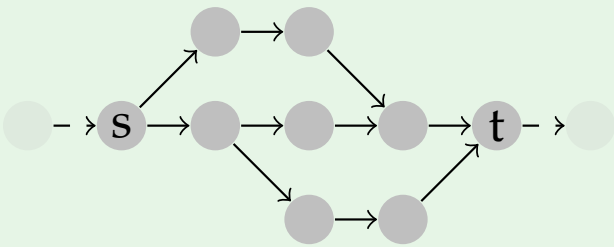
Example : Start with an empty graph



Localized traversal

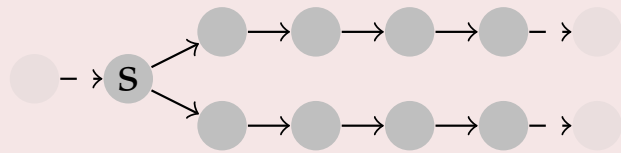
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



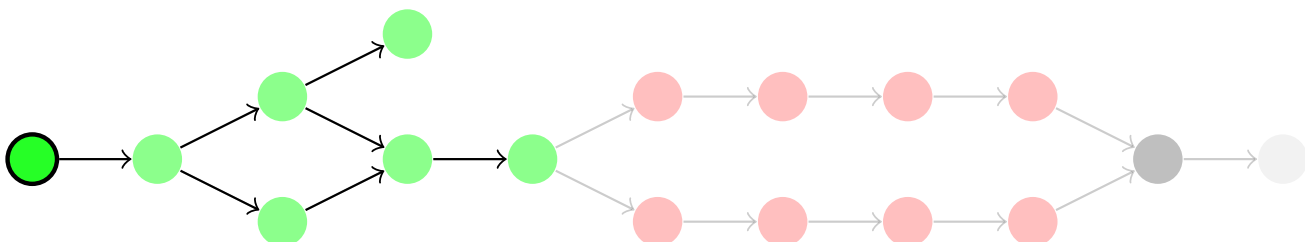
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



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

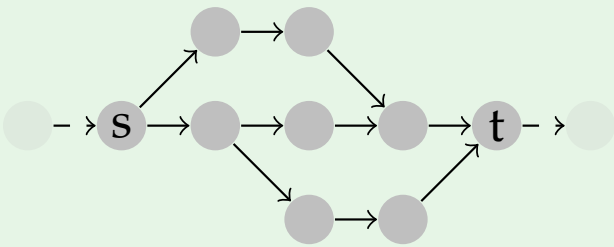
Example : Pick a new node, construct the first portion



Localized traversal

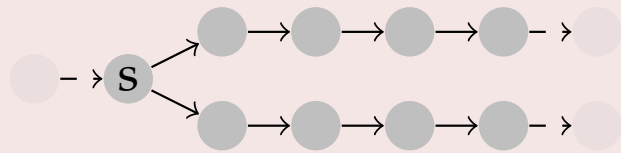
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



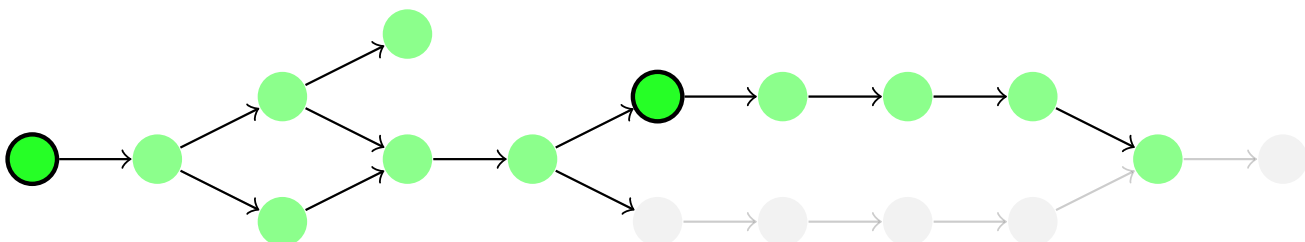
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

Won't traverse : long branches



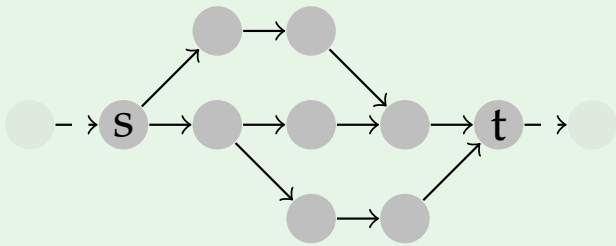
BFS from s, breadth remains > 1 for depths $1..d$

Example : Construct the second portion



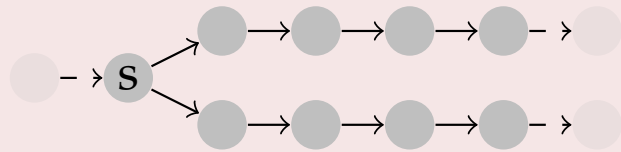
Traverse the graph greedily, according to these rules :

Will traverse : *variant sub-graphs*



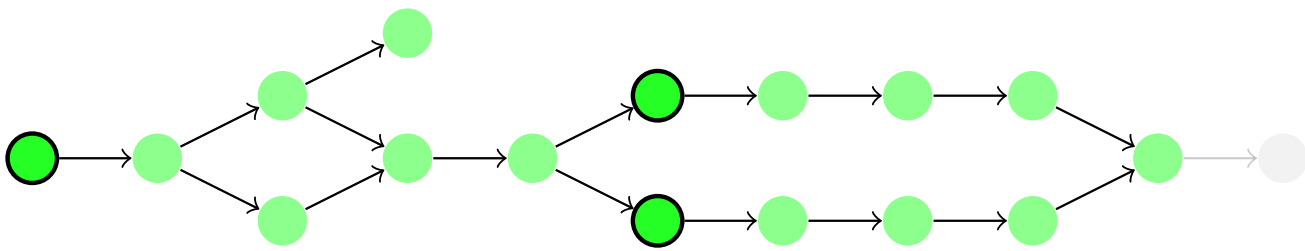
BFS from s until a depth of breadth 1 is reached, keeping breadth $< b$ and depth $< d$

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

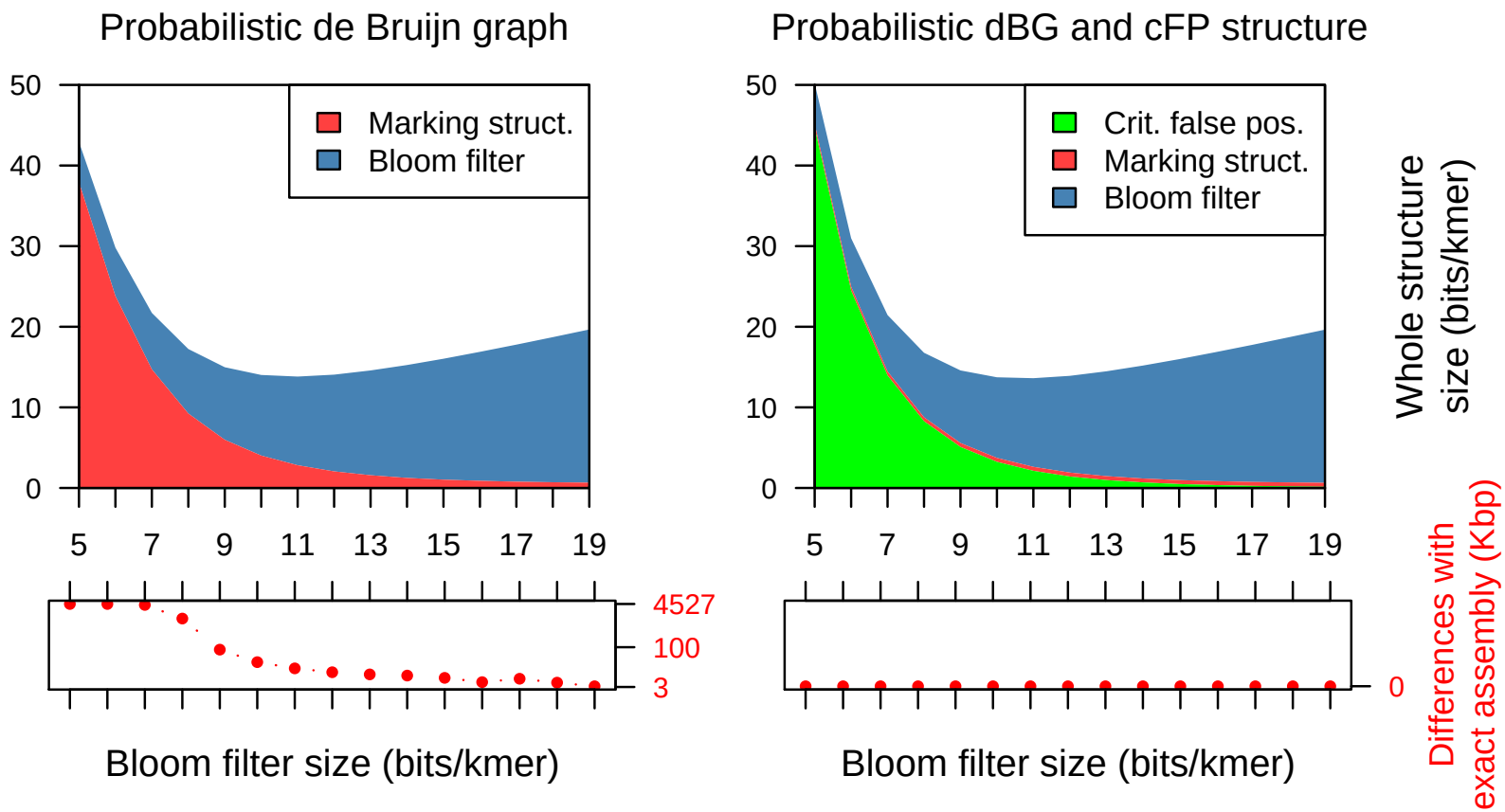
- Nodes which have already been traversed need to be **marked**
- No extra information can be stored in our structure
- We used a **separate** hash table to remember if **branching** or **dead-end** nodes have already been visited.

Contigs construction

Consensus from each path obtained by localized traversal

Results

Usefulness of cFP structure



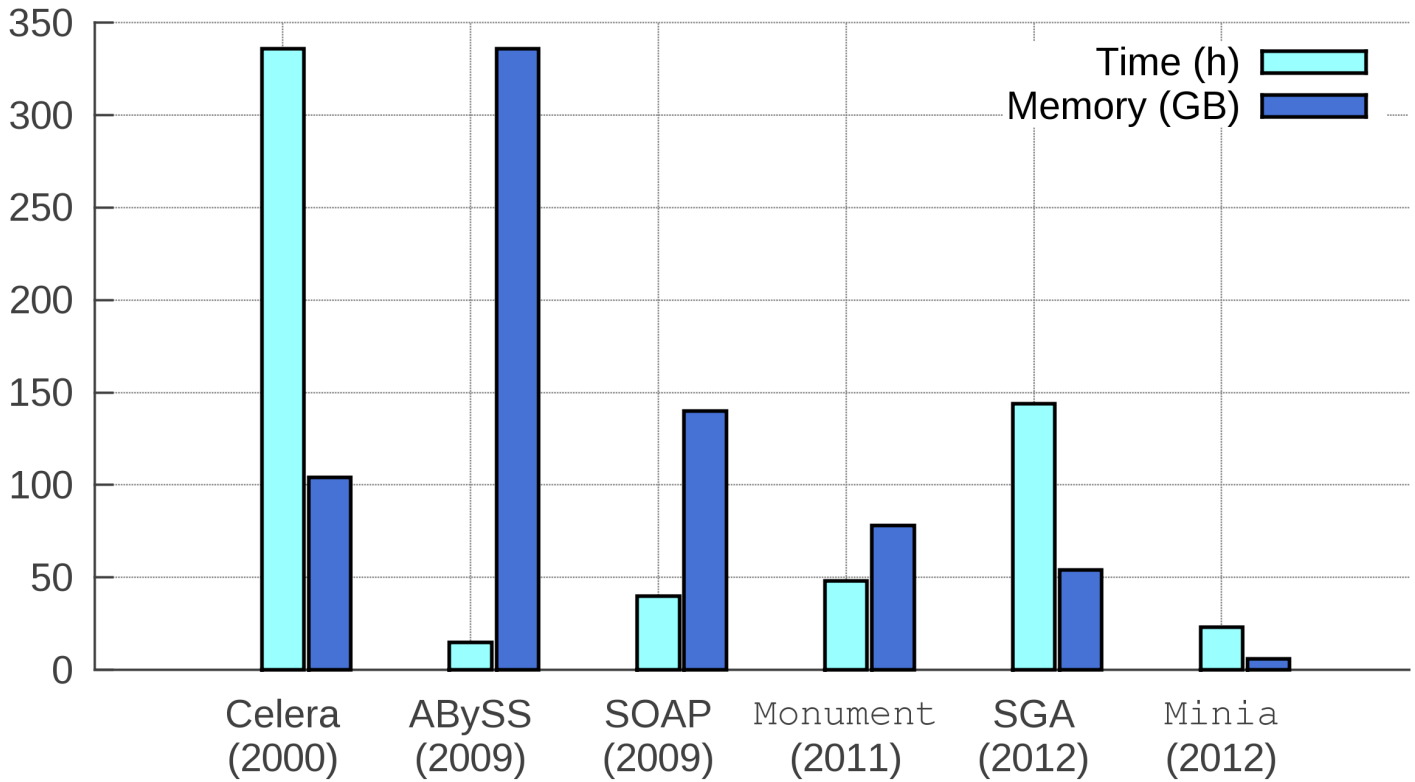
Assembly : E. coli, $k = 23$

COMPLETE *de novo* HUMAN GENOME ASSEMBLY

N50 : length **l** at which half of the assembly contains sequences of length $\geq l$

Human genome assembly	**Minia**	C. & B.	ABySS	SOAPdenovo
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**

Rough performance comparison with other human genome assemblies



Perspectives

Perspectives

Applications

Why assemble a human genome *again* ?

- To exhibit novel structural **variations** [Iqbal 11]
- As a **benchmark**, for the immense number of (meta)genomes that will be sequenced next

Future of sequencing

Predictions :

DNA assembly Relevant until 10-100 kbp high-accuracy read lengths

RNA assembly, **metagenomics** and **metatranscriptomics** No announced technology other than **Illumina** permits high depth of sampling.

→ My opinion is that **short-read assembly** (with paired reads) will remain a hot topic for a few years.

Perspectives

Potential applications of Minia codebase :

- **Huge metagenomic** assemblies (with Genoscope)
- **Transcriptome** assembly (Inchworm replacement)
- **Alternative splicing** detection (KisSplice module replacement)
- **SNP** detection (KisSnp 2, with R. Uricaru & P. Peterlongo)
- **Structural variants** detection
- **Gap-filling** of scaffolds
- Read **compression**

AVAILABILITY

Manuscript available at **`minia.genouest.org`**.
To obtain the source code of Minia (pending license) :

Now Email me

Next month Website above

Acknowledgements : Dominique Lavenier, GenScale team (IRISA, France)

Thank you ! Any questions ?