Graphs for reference-free analysis of sequencing data

Rayan Chikhi

CNRS
Bonsai team, CRIStAL/INRIA, Univ. Lille 1

Colib'read workshop, November 2016, Paris

Motivation

Reference-free sequencing data analysis:
no genome sequence, or choosing not to use it

Some applications:

1. Read error correction
2. DNA/RNA/metaDNA assembly
3. Alternative splicing detection
4. DNA variants detection
5. Transcript quantification

Presentation will be application-agnostic. We'll talk about the overlap graph, string graph, de Bruijn graph, A-Bruijn graph, superstring graph

This talk

.. will be technical!

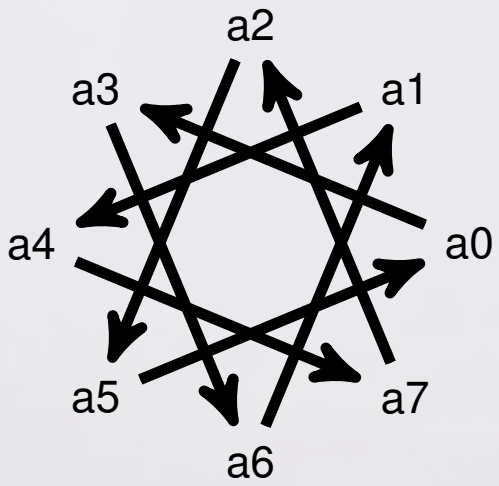
Computer Scientists: sit back and relax, it gets challenging in the second part

(as Pierre would say,) *Biologists*:
the first part will possibly be helpful to understand other talks

Vocabulary

A graph is:

- a set of nodes, and
- a set of edges (directed or not)



k-mers are strings of length *k*

Graphs for sequencing data

Graphs represent overlaps between sequences in reads.

Two widely used families of graphs for sequencing data:

- de Bruijn graphs — generally for Illumina data
- string graphs — generally for Sanger/PacBio/Nanopore data

Overlap graphs

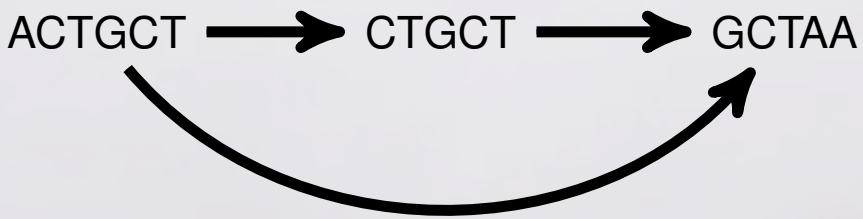
First, agree on some definition of "what is an overlap".

1. **Nodes** = reads.
2. **Edges** = overlap between two reads.

In this example, let's say that an overlap needs to be:

- exact
- and over at least 3 characters.

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



String 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 $a \to c$)

Example:

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

ACTGCT → GCTAA

Let's have inexact overlaps here

```
ACTGCT
 CTACT
   GCTAA
```

ACTGCT → CTACT → GCTAA

String graph / de Bruijn graph (4)

So, which is better?

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

string graphs have been mostly used for long reads and de Bruijn graphs mostly for short reads.

String graphs are also known as the **Overlap Layout Consensus** (OLC) method.

de Bruijn graphs

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

1. **Nodes** = all *k-mers* (substrings of length k) in the reads.
2. There is an **edge** between x and y if the $(k-1)$-mer prefix of y matches exactly the $(k-1)$-mer suffix of x.

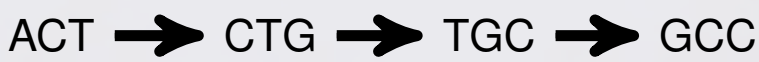
Example for $k = 3$ and a single read:

`ACTG`

de Bruijn graphs

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

```
ACTG
 CTGC
  TGCC
```



de Bruijn graphs: redundancy

What happens if we add redundancy?

```
ACTG
ACTG
 CTGC
 CTGC
 CTGC
  TGCC
  TGCC
```

dBG, $k = 3$:

de Bruijn graphs: errors

How is a sequencing error (at the end of a read) impacting the de Bruijn graph?

```
ACTG
 CTGC
 CTGA
  TGCC
```

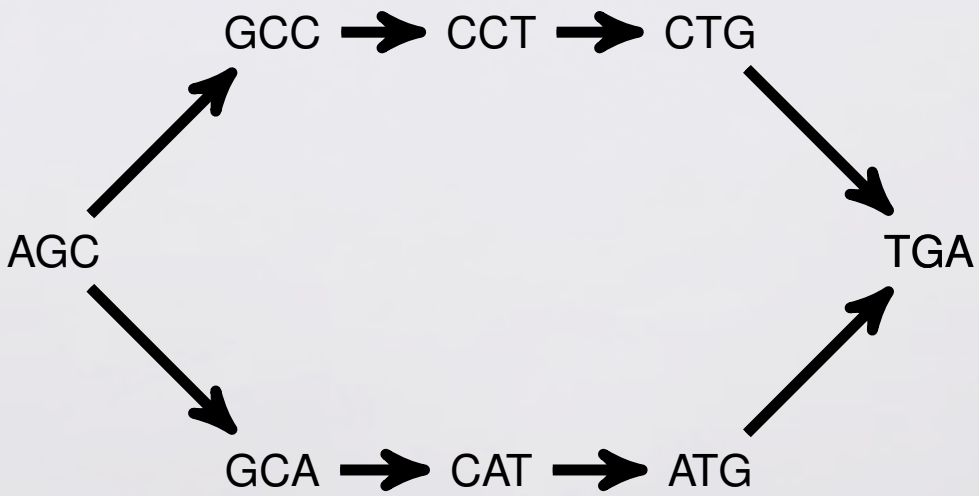
dBG, $k = 3$:

de Bruijn graphs: SNPs

What is the effect of a SNP (or a sequencing error inside a read) on the graph?

AGCCTGA
AGCATGA

dBG, $k = 3$:

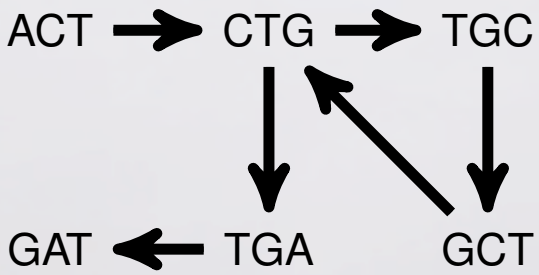


de Bruijn graphs: repeats

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

```
ACTG
 CTGC
  TGCT
   GCTG
    CTGA
     TGAT
```

dBG, $k = 3$:

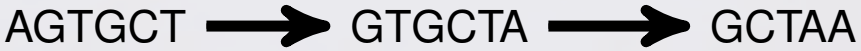


Comparison string graph / de Bruijn graph

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

```
AGTGCT
 GTGCTA
   GCTAA
```

String graph, overlap threshold of 3:



de Bruijn graph, $k = 3$:

Short note on reverse complements

Because sequencing is generally not strand-specific:

> *We always consider that reads (and k-mers) are equal to their reverse complements.*

E.g:
AAA = TTT
ATG = CAT

Short note on dBG definitions

Node-centric dBG:

what we saw.
nodes = k-mers
edges = $(k-1)$-overlaps

Edge-centric dBG:

nodes = k-mers
edges = $(k+1)$-mers

Also called dBG^- and dBG^+ in the south of France.

They're related but not strictly equivalent.

Space needed to represent the dBG

With a hash table: $\geq 2k$ bits / node

Memory-efficient dBG data structures:

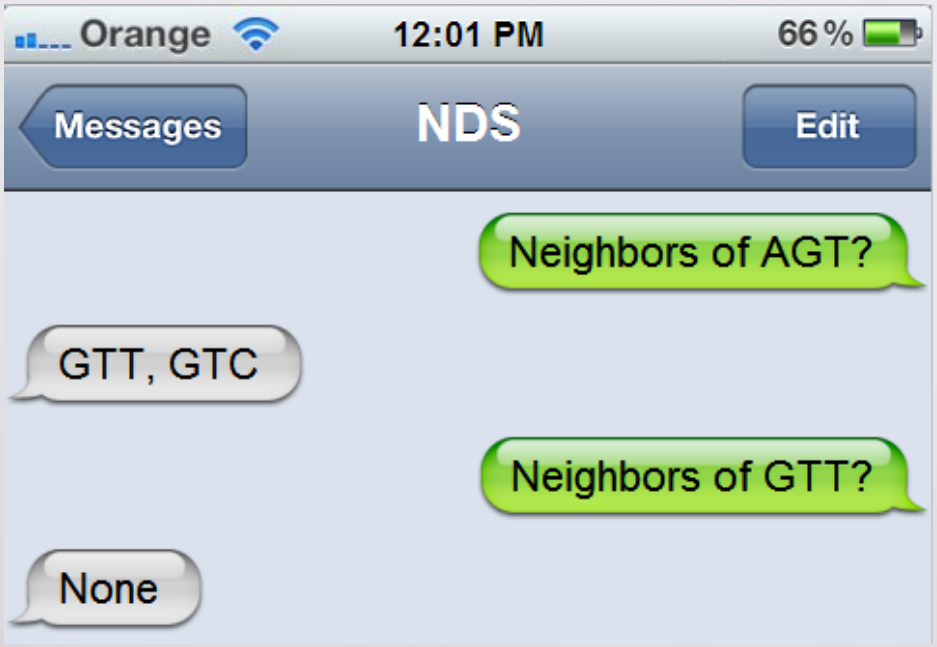
khmer	Bloom filter	[Pell et al. 11]
Minia	BF \ false positives	[Chikhi, Rizk 12], [Salikhov et al. 13]
SDBG	XBW + rank/select	[Bowe et al. 12]

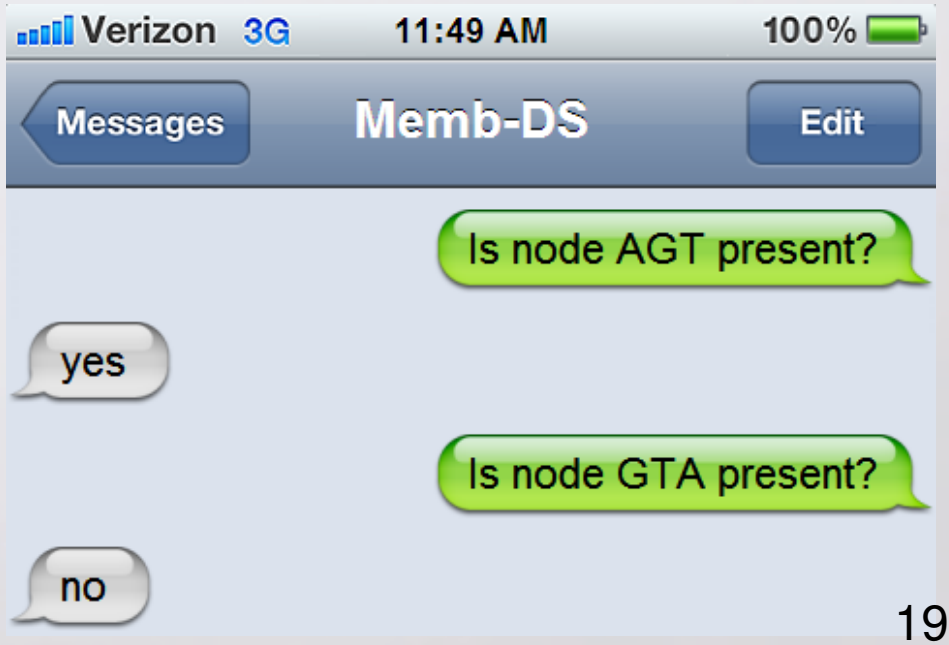
Why are they so efficient? $\rightarrow$ not all operations are supported

Navigational data structures

	NDS	Membership (e.g. hash table)
Traverse dBG from known nodes	✓	✓
Query **membership** of arbitrary nodes	*x*	✓
Enumerate nodes	*x*	✓

NDS has undefined behavior if query node not present.





Software

To just construct the graph:

String graph

Illumina: SGA[1] or SlideSort[2]
PacBio: Minimap + Miniasm[3]

de Bruijn graph

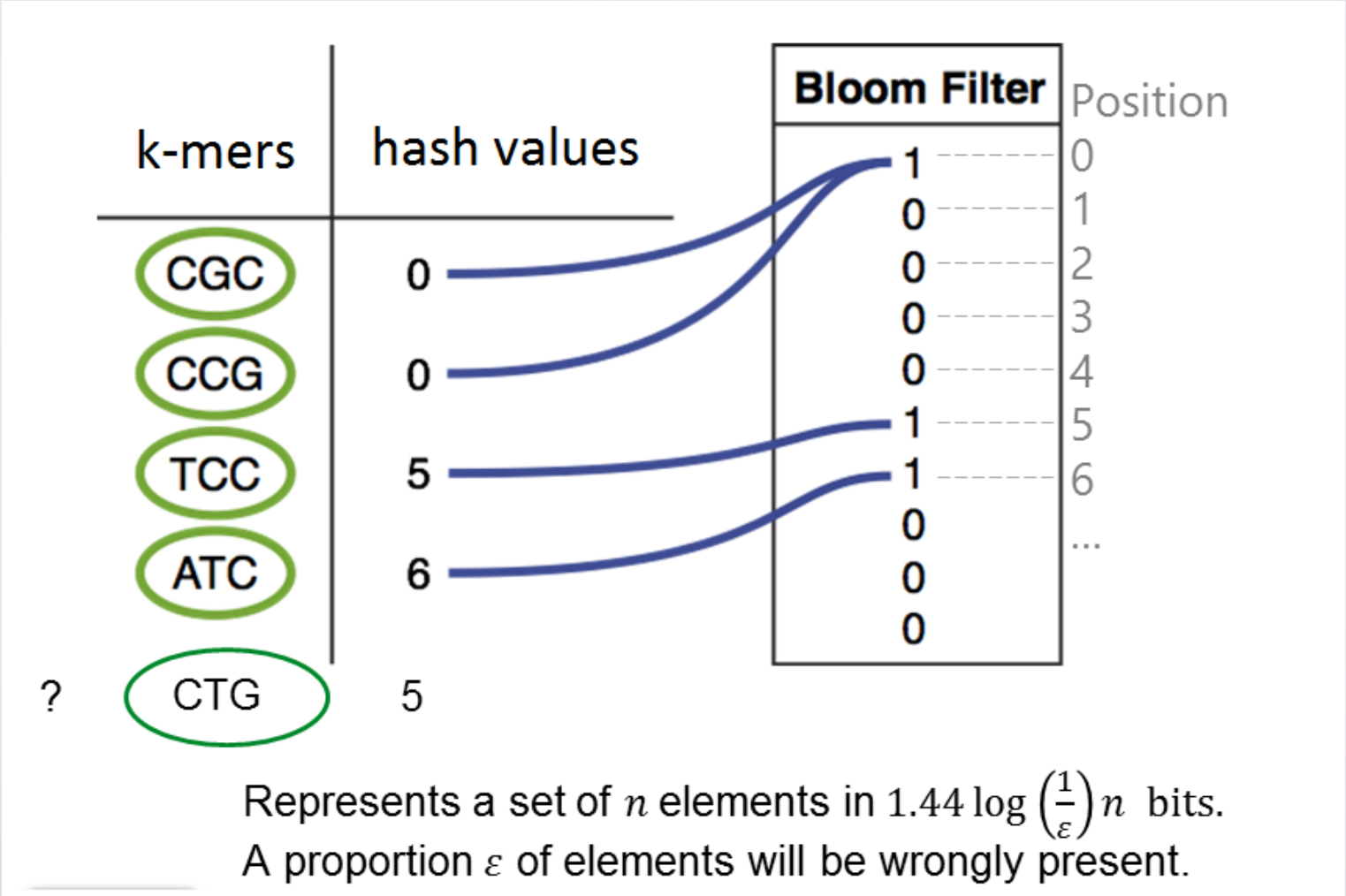
GATB-Core[4] (for developers, navigational index),
BCALM 2[5] (stand-alone, graph on disk),
ABySS[6]

BCALM 2: 3 GB RAM / 2 wall-clock hours for mammalian genome, outputs unitigs.

---

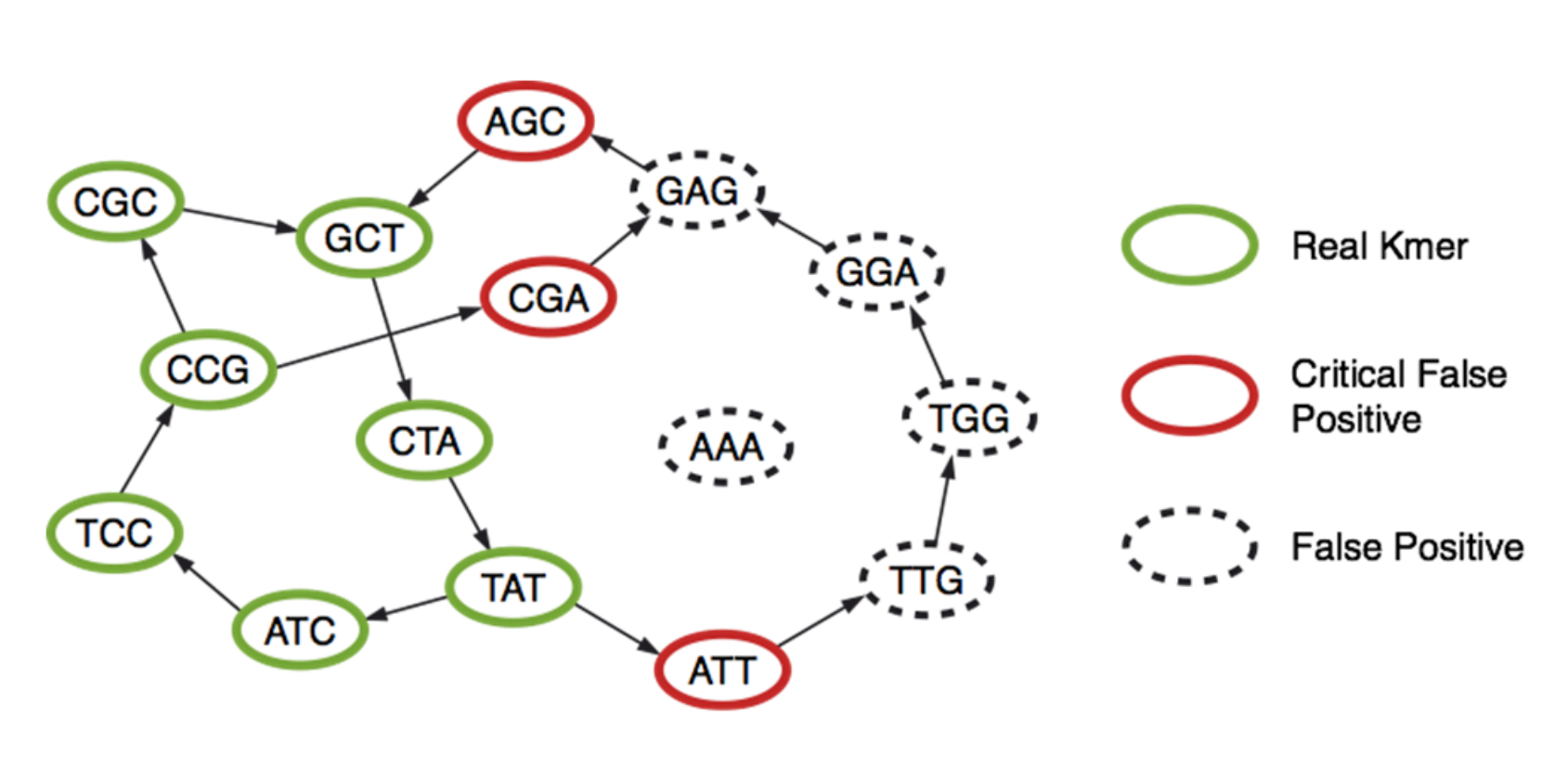
[1] github.com/jts/sga
[2] github.com/iskana/SlideSort
[3] github.com/lh3/miniasm
[4] github.com/GATB/gatb-core
[5] github.com/GATB/bcalm
[6] github.com/bcgsc/abyss

Focus on the Bloom Filter dBG representation (prelude to GATB)



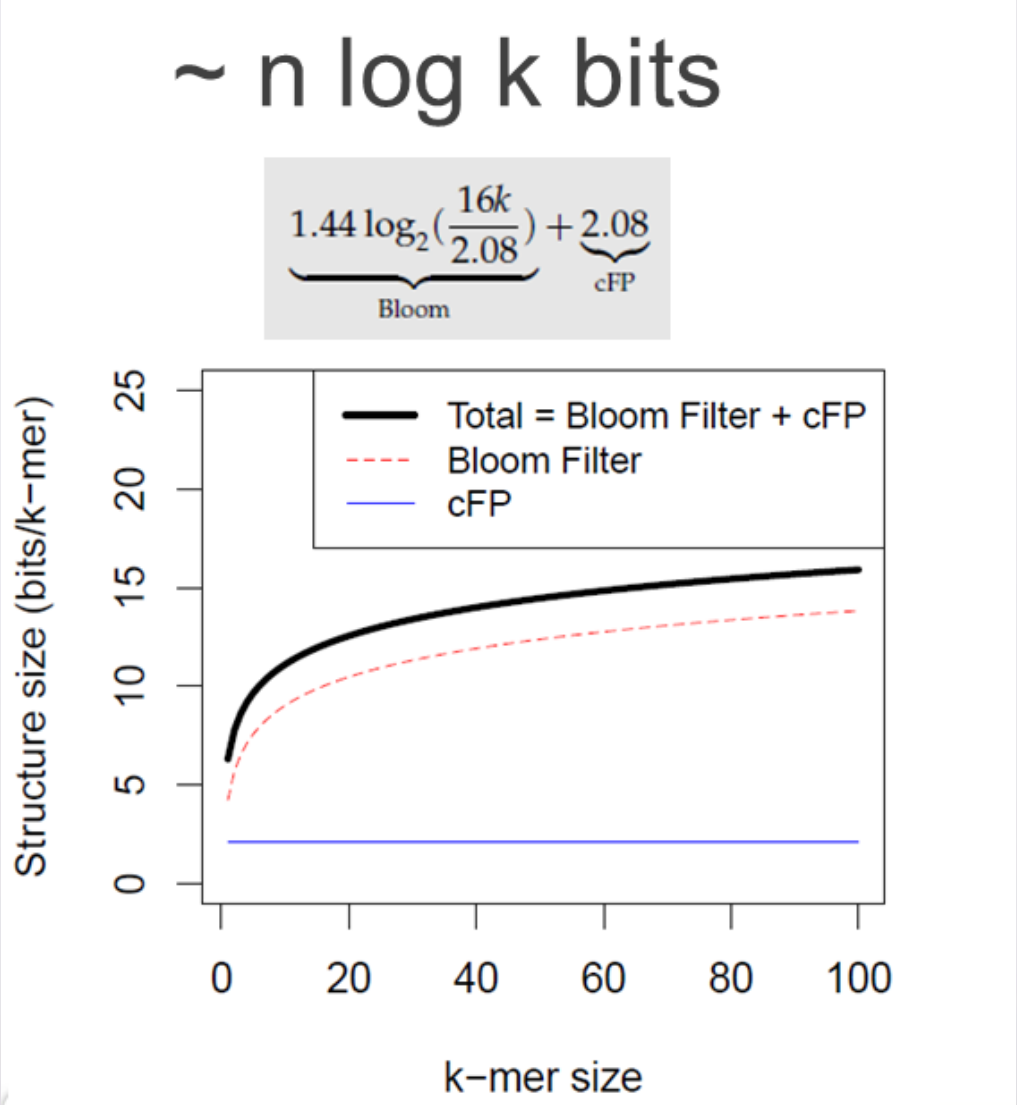
Adapted from slides by G. Collet

Focus on the Bloom Filter dBG representation (prelude to GATB)



Adapted from slides by G. Collet

Focus on the Bloom Filter dBG representation (prelude to GATB)



Adapted from slides by G. Collet

Compacted de Bruijn graph

Compacted de Bruijn graph:

Each non-branching path becomes a single node (*unitig*).

- no loss of information
- less space

Can be constructed using BCALM.
Some assemblers use it as intermediate representation.

A-Bruijn graphs

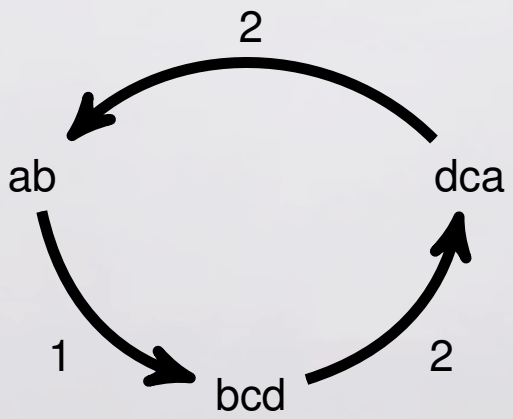
Introduced by [Pevzner 2004]. Generalize de Bruijn graphs.
Recently used in ABruijn assembler (PacBio) [Lin 2016].

Principle: **nodes** = an arbitrary set of words V.
To construct **edges**, given a set of strings (the reads):

- For each string r, find **substrings** that are elements of V.
- Add $v_1 \rightarrow v_2$ when v_1 is *just before* v_2 in r.
- Label edge with $(pos(v_2, r) - pos(v_1, r))$.

Observe that $dBG(Reads) = ABruijn(V = \Sigma^k, E = Reads)$

Ex: $ABruijn(V = \{ab, dca, bcd\}, abcdcab)$:

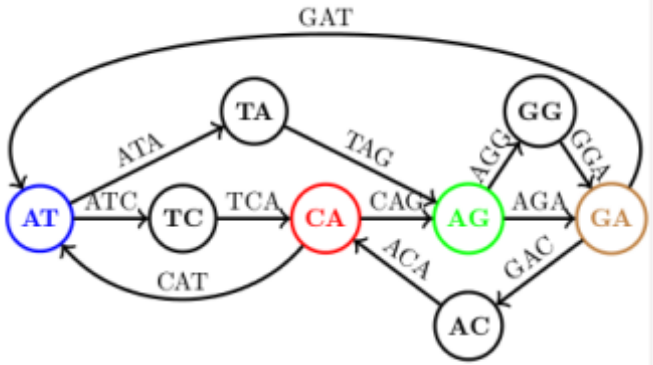


de Bruijn vs A-Bruijn

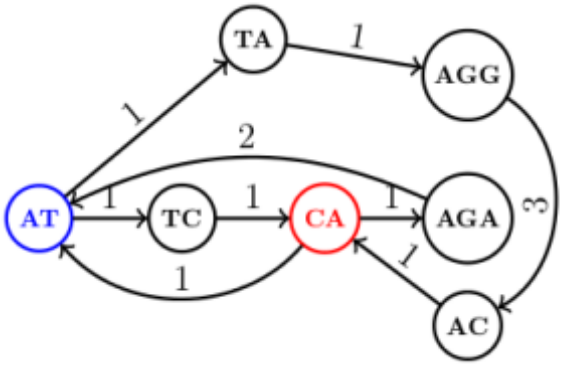
String=CATCAGATAGGGA.
For dBG: $k = 2$,
edge-centric.

For A-Bruijn, V = {CA, AT, TC, AGA, TA, AGG, AC} (V is arbitrary here)

$DB(String, 3)$



$AB(String, V)$



A-Bruijn usages

- Original paper — Sanger, handling repeats in EULER [Pevzner 2004]
- SPAdes (early vers) — Illumina DNA/metaDNA assembler [Bankevich,. . . ,Pevzner 2012]
- ABruijn — Pacbio assembler [Lin,. . . ,Pevzner 2016]

Superstring graph

[Cazaux, Sacomoto, Rivals 2016]

Consider a set of reads **P** (none is contained in another).
Consider all maximal overlaps between these reads, **Ov**(**P**).

Given a set of strings **F** (also none contained), remove all overlaps that are substrings of F. Call this set $\mathbf{Ov}^*(\mathbf{P}, \mathbf{F})$.

E.g.

$P = \{abcc, ccd, cde\}$,

$Ov(P) = \{cc, cd, c\}$.

Now if $F = \{cdef\}$,

$Ov^*(P, F) = \{cc\}$.

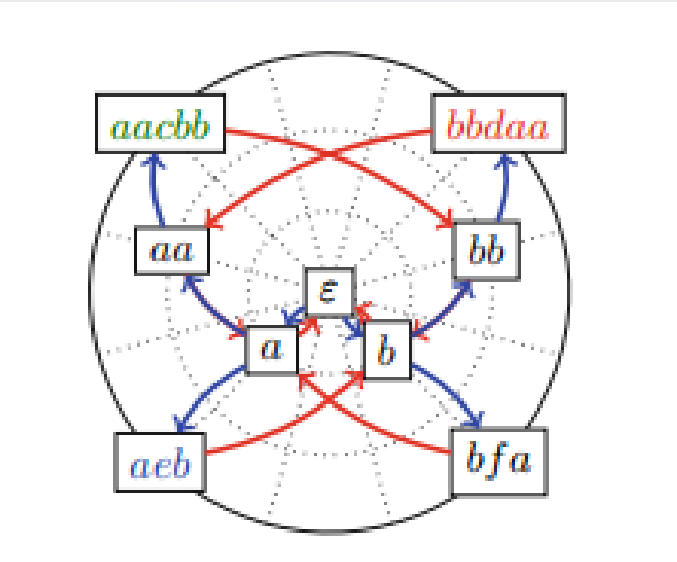
Superstring graph (2)

Recall, $Ov^*(P, F)$ are maximal overlaps between words in P, except substrings in F. Typically $F = \Sigma^{k_{min}-1}$.

The Truncated Hierarchical Overlap Graph, **THOG**(**P**, **F**) is a graph, where:

- nodes = P and $Ov^*(P, F)$
- edges = whenever a node is the longest suffix or the longest prefix of another node

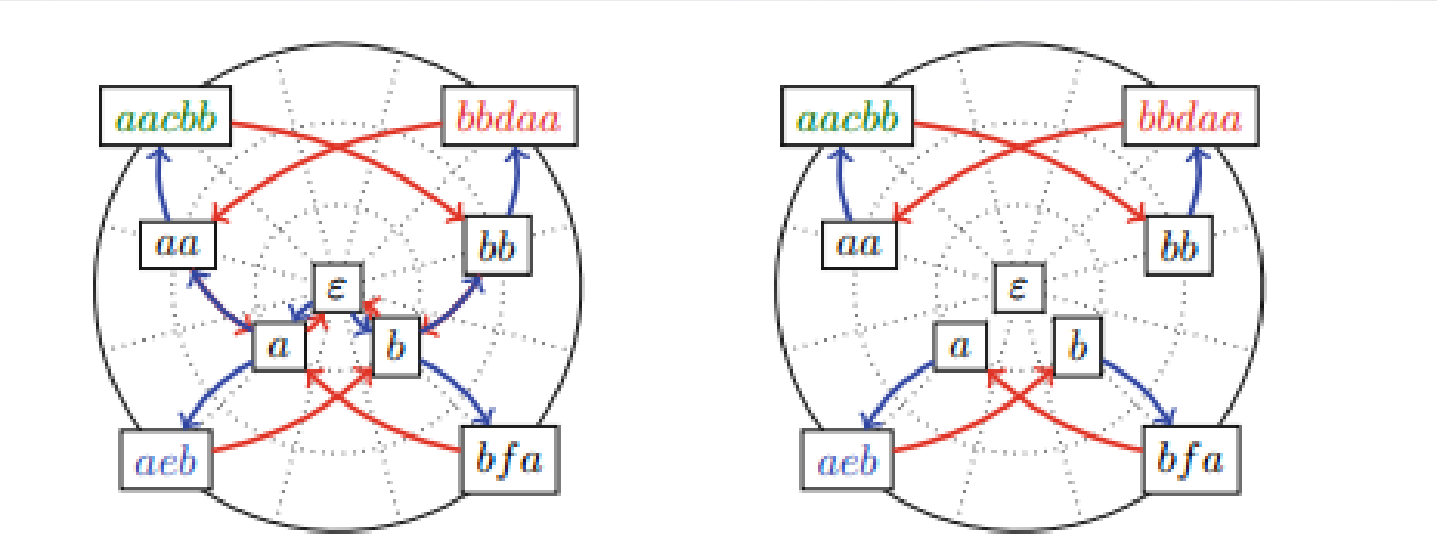
E.g. $THOG(\{aacbb, bbdaa, aeb, bfa\}, \emptyset)$



Superstring graph (3)

The **superstring graph** is defined as a subgraph of *THOG*.

It consists of a set of paths in *THOG* that correspond to a greedy solution of a computational problem (Constrained Shortest Cyclic Cover of Strings).



Applications:

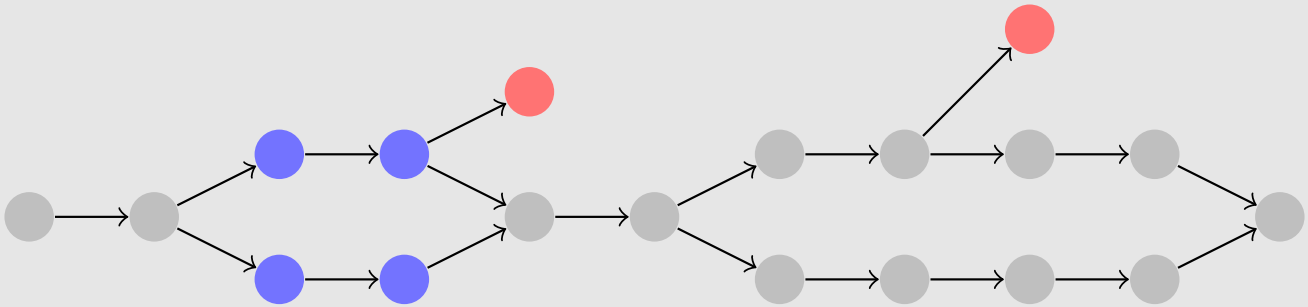
- superstring problems (CompSci)
- multi-k assembly

How an assembler works

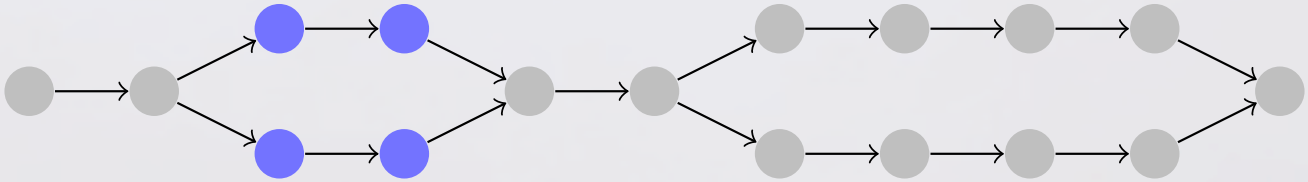
[SPAdes, Velvet, ABySS, SOAPdenovo, SGA, Megahit, Minia, .., HGAP, FALCON]

1) Maybe correct the reads. (SPAdes, HGAP, SGA, FALCON)
2) Construct a graph from the reads.

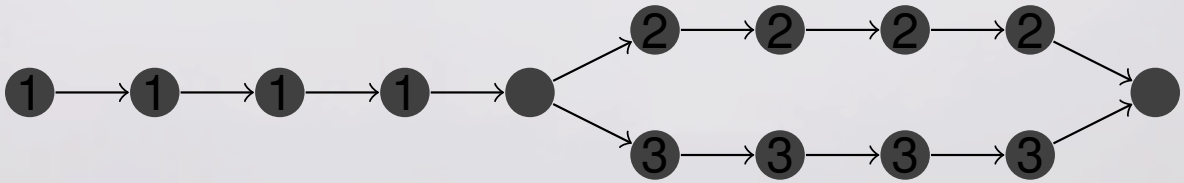
Assembly graph with variants & errors



3) Likely sequencing errors are removed. (not in FALCON)



3) Known biological events are removed. (not in FALCON)
4) Finally, **simple paths** (i.e. contigs) are returned.



The choice of k

Choice of k is critical in dBG applications:

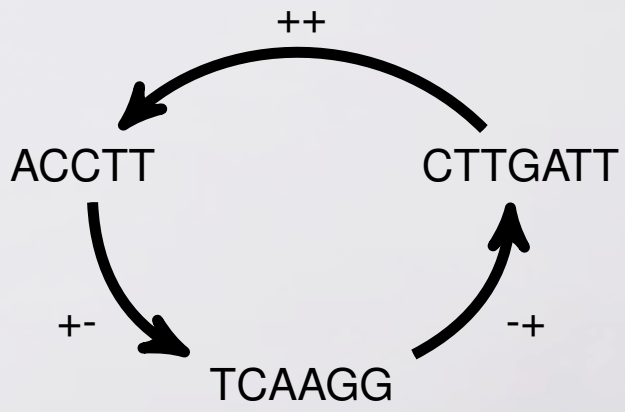
- k-mers with sequencing errors are noise
- only *non-erroneous* k-mers matter
- $k < \log_4(|\text{genome}|)$: nearly complete graph, uninformative
- small k: collapses repeats, more non-erroneous k-mers
- large k: less repeat collapsing, less non-erroneous k-mers (due to error and shortness of reads)

Generally, $k \geq 20$.
(Compare 4^k to the genome size.)
Higher sequencing coverage means larger k values can be used.

Graph formats

- FASTG
- **GFA**
- GFA2

```
H    VN:Z:1.0
S    11   ACCTT
S    12   TCAAGG
S    13   CTTGATT
L    11   +    12   -    4M
L    12   -    13   +    5M
L    11   +    13   +    3M
P    14   11+,12-,13+ 4M,5M
```



Conclusion

We saw many graphs

- Overlap graph
- String graph
- De Bruijn graph
- A-Bruijn graph
- Superstring graph

Those slides will be available at `http://rayan.chikhi.name`

Enjoy the workshop!