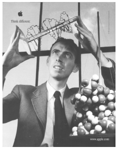
Computational Methods for *de novo* Assembly of Next-Generation Genome Sequencing Data

Rayan Chikhi
ENS Cachan Brittany / IRISA (Genscale team)
Advisor : Dominique Lavenier

Introduction, Year 2000 : Human Genome Project

"It's a giant resource that will change mankind, like the printing press."

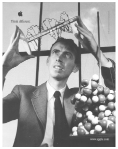
Dr James Watson, co-discoverer of DNA structure



INTRODUCTION, YEAR 2000 : HUMAN GENOME PROJECT

"It's a giant resource that will change mankind, like the printing press."

Dr James Watson, co-discoverer of DNA structure

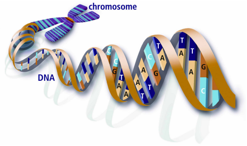


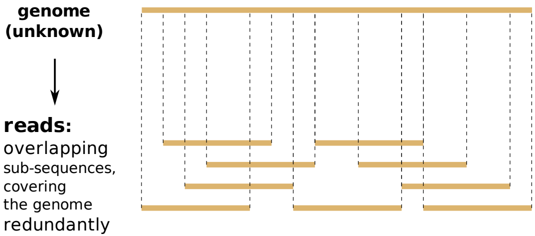
First achievement : human **sequencing**

- the **only way to read DNA** is **through small fragments** (called *reads*)

Sequencing process :

1) Obtain many **copies** of the genome
2) Cut them into **millions** of **short fragments**
3) Output the **sequences** of these fragments




genome (unknown)
reads: overlapping sub-sequences, covering the genome redundantly

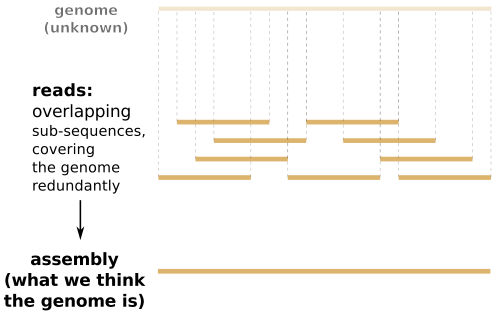
Introduction, Year 2000 : Human Genome Project

Second achievement :

Introduction, Year 2000 : Human Genome Project

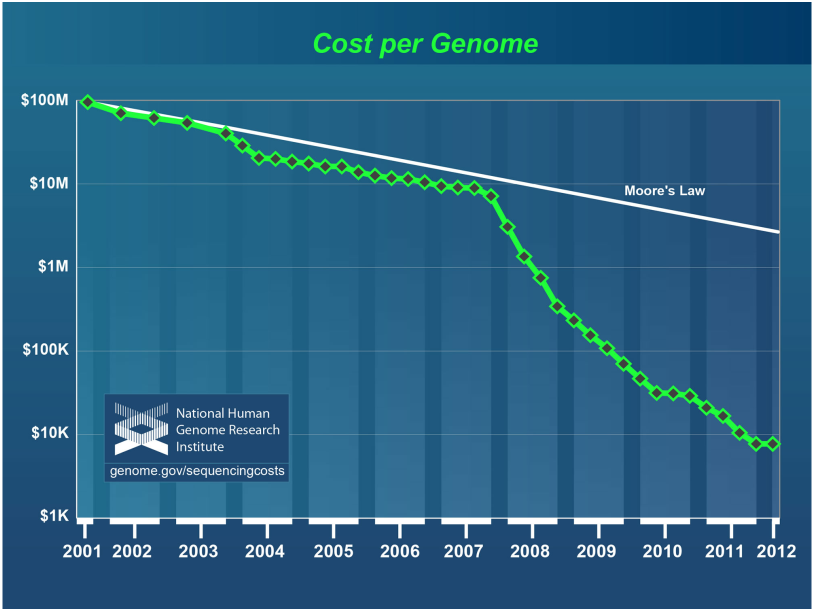
Second achievement : human *de novo* **assembly** (thesis topic)

- from **millions** of **small fragments of DNA** to **a single sequence**
- purely computational process
- required a supercomputer with 64 GB memory
- result was actually not perfect : assembly was fragmented



Context, Year 2012 : still difficult to sequence today ?

Context, Year 2012 : still difficult to sequence today ?



Next-Generation Sequencing Technologies

- NGS = massively parallel sequencing

3 main NGS technologies

HGP technology

Sanger

SOLiD

454

Illumina

Proton, PacBio, Oxford

Next-Generation Sequencing Technologies

- What everyone uses today :

$$\underbrace{\text{HGP technology}}\quad\underbrace{\text{3 main NGS technologies}}$$

HGP technology

3 main NGS technologies

Illumina

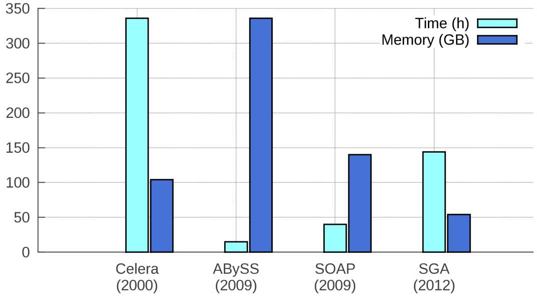
***90 percent** of the **world's sequencing output** is produced on Illumina instruments.*

GenomeWeb, February 14, 2012 ; verified with `http://omicsmaps.com/stats`

read length	$\approx$ 100 nt, i.e. 0.000003% of the human genome
throughput	equivalent to 1 human genome per day

HOW COMPUTATIONALLY HARD IS *assembly* TODAY ?

Tentative comparison of some software methods :



≈ 20 *de novo* assemblers omitted.

Datasets : whole human genome, Illumina reads (except for Celera : Sanger reads)

- We focus on computational difficulty
- *Quality* of results : newer assemblies (≥ 2009) are much more fragmented, because of shorter reads

Outline

Genome assembly

Informal problem

Given a set of sequenced reads, retrieve the genome.

In computational terms

Find an algorithm such that :
Input : a set of reads that are **sub-strings** of the genome
Output : the genome

Toy example

Input : {`GAT`, `ATT`, `TTA`, `TAC`, `ACA`, `CAT`, `CAA`}
Output : `GATTACATCAA`

Genome assembly

Informal problem

Given a set of sequenced reads, retrieve the genome.

In computational terms

Find an algorithm such that :
Input : a set of reads that are **sub-strings** of the genome
Output : the genome

Toy example

Input : {GAT, ATT, TTA, TAC, ACA, CAT, CAA}
Output : GATTACATCAA

Immediate questions

Q : Is there a single possible output ?
A : no, $s =$ GATTACATTACAA is another possible output
Q : Then, how to choose ?
A : need to formulate an optimization problem[a]

[a] **optimization problem :** problem of finding the best solution from all feasible solutions

Shortest common super-string problem

Shortest common super-string (*SCS*) problem

Given a set S of strings,
construct a string of **minimal length**
which contains all strings of S as **sub-strings**.
(there can be many solutions)

Toy example

$S = \{\texttt{GAT}, \texttt{ATT}, \texttt{TTA}\}$
Trivial super-string : $\{\texttt{GATATTTTA}\}$
Super-strings of length 3 :

Shortest common super-string problem

Shortest common super-string (*SCS*) problem

Given a set S of strings,
construct a string of **minimal length**
which contains all strings of S as **sub-strings**.
(there can be many solutions)

Toy example

$S = \{\texttt{GAT}, \texttt{ATT}, \texttt{TTA}\}$
Trivial super-string : $\{\texttt{GATATTTTA}\}$
Super-strings of length 3 : none
Super-strings of length 4 :

Shortest common super-string problem

Shortest common super-string (*SCS*) problem

Given a set S of strings,
construct a string of **minimal length**
which contains all strings of S as **sub-strings**.
(there can be many solutions)

Toy example

$S = \{\texttt{GAT}, \texttt{ATT}, \texttt{TTA}\}$
Trivial super-string : $\{\texttt{GATATTTTA}\}$
Super-strings of length 3 : none
Super-strings of length 4 : none
Super-strings of length 5 :

Shortest common super-string problem

Shortest common super-string (*SCS*) problem

Given a set S of strings,
construct a string of **minimal length**
which contains all strings of S as **sub-strings**.
(there can be many solutions)

Toy example

$S = \{$`GAT`, `ATT`, `TTA`$\}$
Trivial super-string : {`GATATTTTA`}
Super-strings of length 3 : none
Super-strings of length 4 : none
Super-strings of length 5 : {`GATTA`} $\leftarrow$ solution

Problem with SCS-based assembly

The genome is not a SCS.
Genomes contain long repetitions, e.g. `GATTACATTACAA` (length = **13**).
Sequencing yields reads : {`GAT`, `ATT`, `TTA`, `TAC`, `ACA`, `CAT`, `CAA`}
A shortest common super-string is : `GATTACATCAA` (length = **11**).

A better problem formulation

Overlap graph (simplified definition) [Myers 95]

Directed graph,

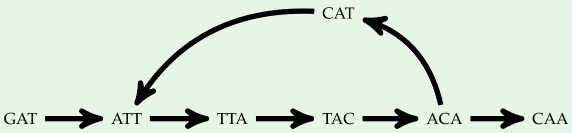
- **vertices = reads**
- **edge** $r_1 \to r_2$ if r_1 and r_2 exactly **overlap** over $\geq$ **k** characters.

String graph

Remove **transitively inferable overlaps** from the overlap graph.

Toy string graph

$$S = \{\texttt{GAT}, \texttt{ATT}, \texttt{TTA}, \texttt{TAC}, \texttt{ACA}, \texttt{CAT}, \texttt{CAA}\} \qquad k = 2$$


$$\begin{tikzcd} & & & \text{CAT} \arrow[dll, bend right] & & \\ \text{GAT} \arrow[r] & \text{ATT} \arrow[r] & \text{TTA} \arrow[r] & \text{TAC} \arrow[r] & \text{ACA} \arrow[r] \arrow[ul, bend right] & \text{CAA} \end{tikzcd}$$

```
GAT
 ATT
```

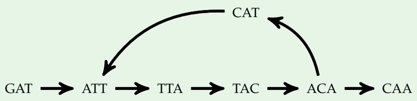
Assembly using an string graph

Assembly in theory [Nagarajan 09]

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

Illustration

For the previous example,



The only solution is `GATTACATTACAA`.
(Recall that SCS was `GATTACATCAA`)
→ **Graphs provide a good framework for assembly.**

Assembly using an string graph

Example of ambiguities

$$\begin{tikzcd}
& \text{GAG} \arrow[r] & \text{AGT} \arrow[r] & \text{GTG} \arrow[dl] & \\
\text{ACT} \arrow[r] & \text{CTG} \arrow[r] & \text{TGA} \arrow[ul] \arrow[r] \arrow[dl] & \text{GAC} \arrow[r] & \text{ACC} \\
& \text{GAA} \arrow[r] & \text{AAT} \arrow[r] & \text{ATG} \arrow[ul] &
\end{tikzcd}$$

Assembly in practice

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

Solution of the example above

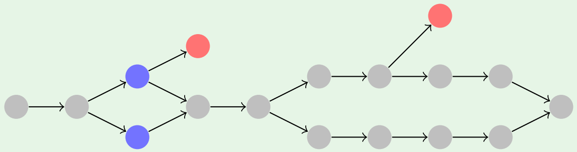
The assembly is the following set of paths :

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

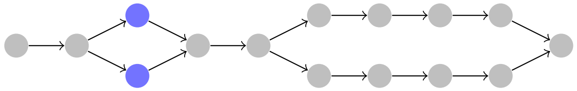
ALMOST EVERY ASSEMBLY ALGORITHM

[Zerbino, Birney 08 ; Li et al. 09 ; Simpson et al. 12 ; ..]

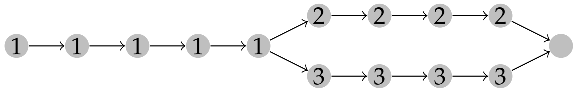
Assembly graph with variants & errors



1) The graph is completely constructed.
2) Likely sequencing errors are removed.



3) Known biological events are removed.
4) Finally, **simple paths** are returned.



WHOLE-GENOME GRAPHS ARE UNNECESSARY

Practically

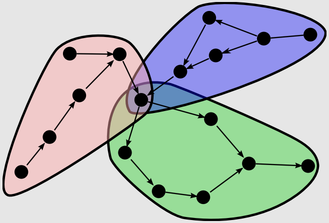
Genome graphs are a better framework than SCS, but they

- are monolithic, **hard to parallelize**, and [Simpson et al. 09]
- **require a lot of memory** (human : 150+ GB). [Li et al. 09]

Contribution 1 : localized assembly

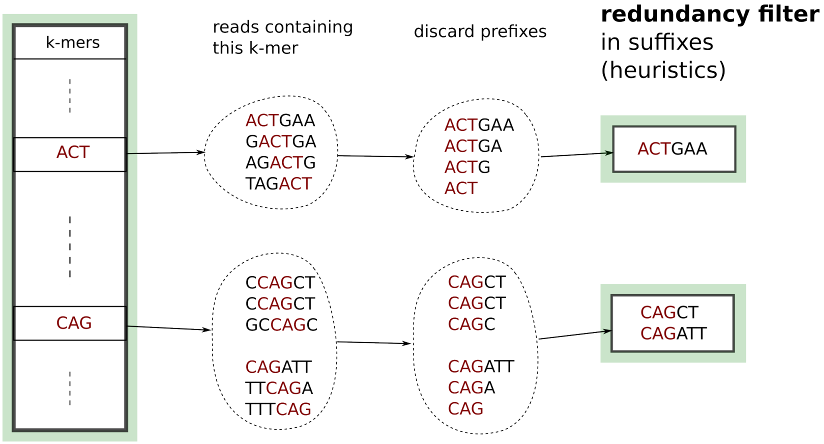
Proposed approach :

- Store reads in a **redundancy-filtered index**
- **Locally** construct portions of the graph at a time



CONTRIBUTION 1.1 : REDUNDANCY-FILTERED READ INDEX

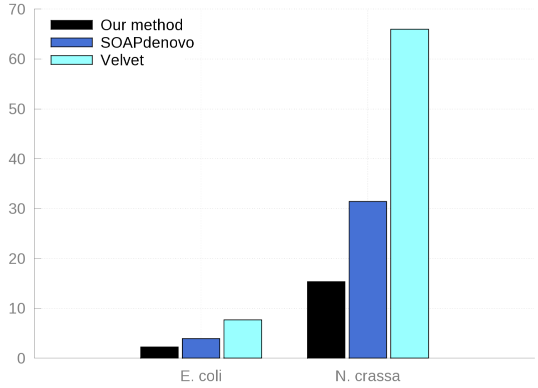
- Store reads in a **redundancy-filtered index** [GC, RC, DL 11]
- Locally construct portions of the graph



Redundancy-filtered read index : benchmark

- Store reads in a redundancy-filtered index
- Locally construct portions of the graph

Memory usage (GB) of indexes



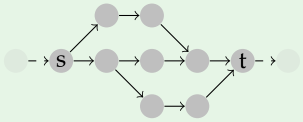
Construction time

SOAP : 41 mins
us : 64 mins

Contribution 1.2 : localized traversal

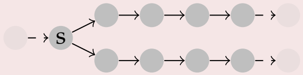
- Store reads in a redundancy-filtered index
- Locally construct portions of the graph, according to these rules :

Will traverse : *variant sub-graphs*



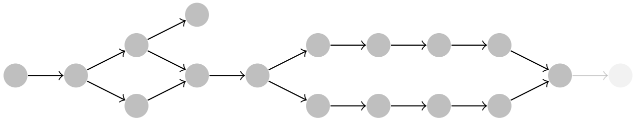
(max breadth b, max depth d)

Won't traverse : long branches



(min depth d)

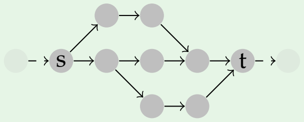
Example : Whole graph



CONTRIBUTION 1.2 : LOCALIZED TRAVERSAL

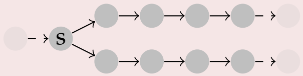
- Store reads in a redundancy-filtered index
- Locally construct portions of the graph, according to these rules :

Will traverse : *variant sub-graphs*



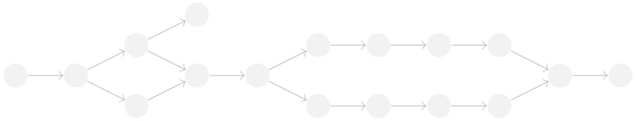
(max breadth b, max depth d)

Won't traverse : long branches



(min depth d)

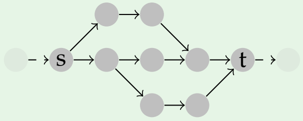
Example : Start with an empty graph



Contribution 1.2 : localized traversal

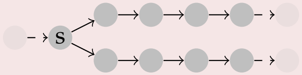
- Store reads in a redundancy-filtered index
- Locally construct portions of the graph, according to these rules :

Will traverse : *variant sub-graphs*



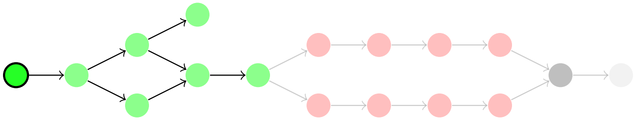
(max breadth b, max depth d)

Won't traverse : long branches



(min depth d)

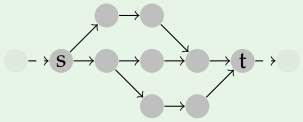
Example : Construct the first portion



CONTRIBUTION 1.2 : LOCALIZED TRAVERSAL

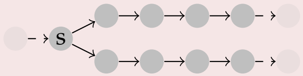
- Store reads in a redundancy-filtered index
- Locally construct portions of the graph, according to these rules :

Will traverse : *variant sub-graphs*



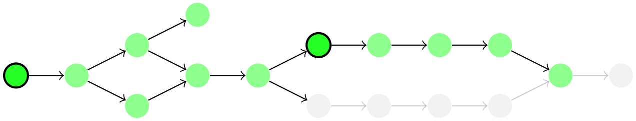
(max breadth b, max depth d)

Won't traverse : long branches



(min depth d)

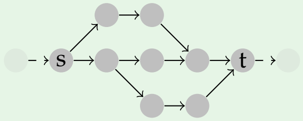
Example : Construct the second portion



CONTRIBUTION 1.2 : LOCALIZED TRAVERSAL

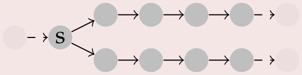
- Store reads in a redundancy-filtered index
- Locally construct portions of the graph, according to these rules :

Will traverse : *variant sub-graphs*



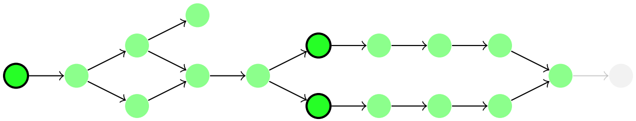
(max breadth *b*, max depth *d*)

Won't traverse : long branches



(min depth *d*)

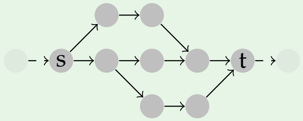
Example : Construct the third portion



Contribution 1.2 : localized traversal

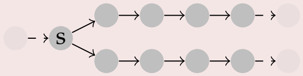
- Store reads in a redundancy-filtered index
- Locally construct portions of the graph, according to these rules :

Will traverse : *variant sub-graphs*



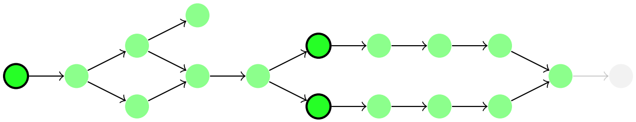
(max breadth b, max depth d)

Won't traverse : long branches



(min depth d)

Example : Construct the third portion

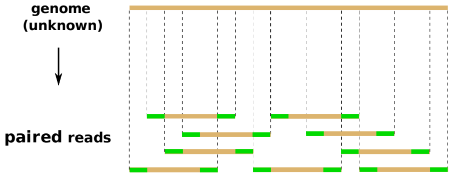


Summary of Contribution 1 :
greedy, localized assembly ≡ **whole-genome graph** assembly

OUTLINE

Pairing information

A vision of sequencing closer to reality is :



Sequencing a toy genome with paired reads of length 4 nt (with gaps of length 2).

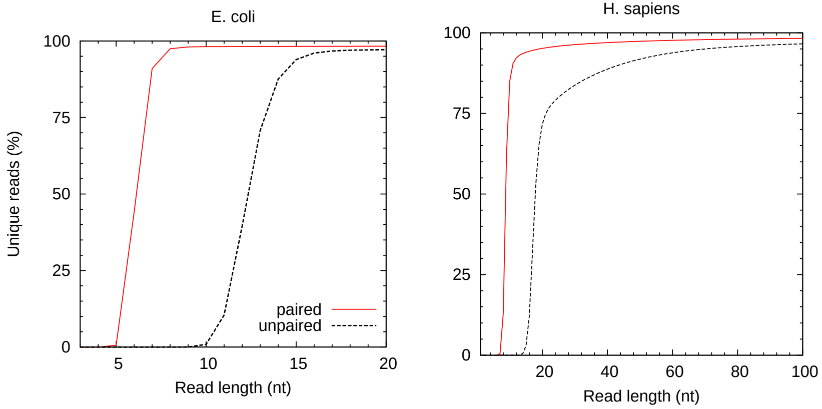
```
Genome ??????????????
       ACTA--GATA
          AGAG--ACCT
        CTAG--ATAC
         TAGA--TACC
```

In practice :

- read length ≈ 100 nt
- depending on seq. method, gaps are **0**, **300**, **2000** or **10000** nt.

Contribution 2 : Studying the impact of pairing information

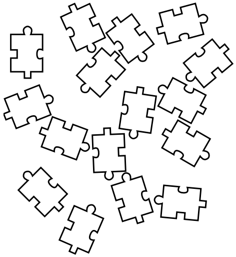
Reads that belong to **multiple genome locations** complicate analysis.
Pairing information contributes to **uniquely** localize reads.

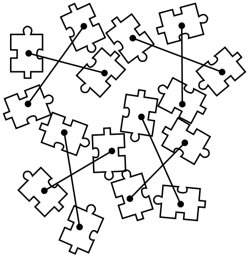


In this figure, paired reads are separated by ($\mathbf{300} - \mathbf{2} \cdot$ [read length]) nt.

Contribution 2 : Incorporating pairing information in assembly

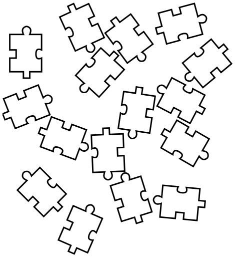
You are asked to solve one of these two jigsaws. Which one looks easier ?

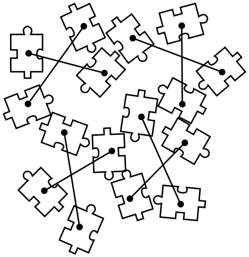




Contribution 2 : Incorporating pairing information in assembly

You are asked to solve one of these two jigsaws. Which one looks easier ?





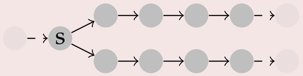
Both are equally hard (NP-hard). [Demaine 07], [RC, DL 11]

We **defined** the following problems, and showed their **NP-hardness** :

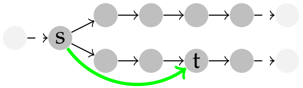
- SCS over paired strings
- paired Hamiltonian path
- super-walk in a de Bruijn graph over paired strings
- paired Assembly Problem (introducing paired string graphs)

CONTRIBUTION 2 : PAIRED STRING GRAPHS

Recall that long branches cannot be traversed

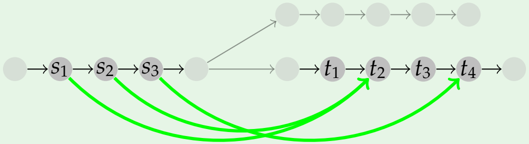


Now, add pairing information to the graph (**paired string graph**) :

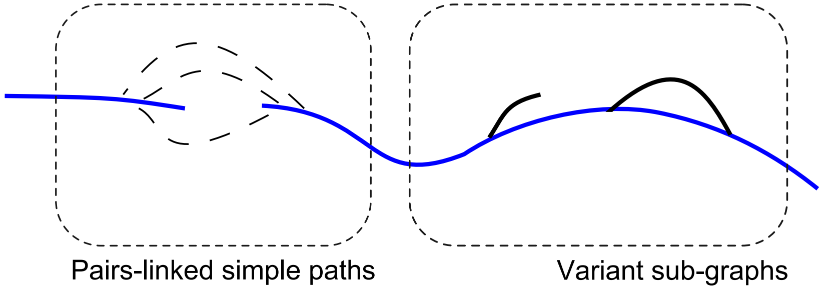


In actual data, pairing is incomplete, with varying distance between mates.

Will traverse : *pairs-linked simple paths* *(heuristics)*



Implementation : Monument assembler



- ***de novo* genome assembly software** for Illumina reads
- 8,000 lines of Python + 5,000 lines of C code
- proof of concept of the two previous contributions
- unreleased, used in-house

RESULTS : ASSEMBLATHON 1 & 2

Assemblathon 1

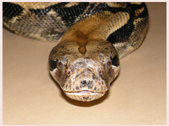
[Earl et al. (incl. RC, DL, DN, GC, NM) 11]

- International competition
- **Research teams are given a set of reads to assemble**
- No knowledge of the solution, no preliminary ranking
- Synthetic genome, 100 Mb (1/30-th of the human genome)

Assemblathon 2

Unknown animal genomes, ≈ 1-2 Gb (half of the human genome)

Maylandia zebra


Red tailed boa constrictor

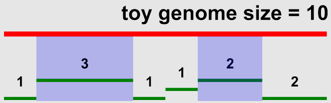
common pet parakeet

QUALITY OF AN ASSEMBLY

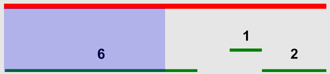
- **contigs** : **gap-less** assembled sequences
- **scaffolds** : contigs separated by **gaps**

Fragmentation

NG50 : length l at which **half** of the genome is covered by sequences of length $\geq l$



NG50 = 2



NG50 = 6

- **accuracy** (many ways) and **coverage** (% of the genome covered)

Results : Assemblathon 1

Assemblathon 1
Contiguity of sequences (kbp) :

Method	*contig NG50* (rank)		*scaffold NG50* (rank)	
Meraculous	16	(10)	9073	(1)
Allpaths	219	(2)	8396	(2)
..	..		..	
Monument	**7**	(13)	**1421**	(7)
..	..		..	
Cortex	3	(16)	9.3	(16)

Performance (reported by participants) (wall h, GB) :

Method	*Memory* (rank)		*Time* (rank)	
Monument	**6.3**	(3)	**2**	(1)
Meraculous	4	(1)	6	(2)
..	..		..	
Allpaths	≈100		12	
..	..		..	
Celera	100		120	

Results : Assemblathon 2

For Assemblathon 1, we used :

- Prototype of Monument (without variants traversal)
- Single **finishing** step : scaffolding (SSPACE) [Boetzer 11]

What we changed for Assemblathon 2 :

- Variant sub-graph traversal [RC, DL 11]
- More elaborate finishing steps :
  - scaffolding (SuperScaffolder) [RC, DN @ Jobim 12]
  - **gap-filling** (SOAP) [Li et al. 09]

Assemblathon 2 (preliminary)

Snake (N50, kbp) :

Method	*ctg.* (rank)		*scaf.* (rank)	
SGA	29	(4)	4505	(1)
Phusion	73	(1)	4066	(2)
..	..		..	
Monument	**65**	(2)	**1149**	(6)
..	..		..	
CLC	8	(11)	19	(11)
PRICE	6	(12)	6	(12)

Fish (N50, kbp) :

Method	*ctg.* (rank)		*scaf.* (rank)	
Bayor	31	(1)	4966	(1)
Allpaths	20	(4)	4014	(2)
..	..		..	
Monument	**31**	(2)	**1241**	(6)
..	..		..	
SGA	8	(8)	110	(10)
Ray	9	(12)	47	(12)

Outline

Recent improvement : lower-memory structure

This is not in the manuscript.

de Bruijn graph [Idury, Waterman 95]

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

$$\begin{tikzcd} \text{GAT} \arrow[r] & \text{ATT} \arrow[r] & \text{TTA} \arrow[r] & \text{TAC} \arrow[r] & \text{ACA} \arrow[r] & \text{CAA} \end{tikzcd}$$

Structurally similar to string graphs.

How to encode de Bruijn graphs using as little space as possible ?

Memory usage (illustration for human, $k = 25$)

- Explicit list of nodes : $2k \cdot n$ bits — 50 bits per node
- Self-information of n nodes :

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

20 bits per node.

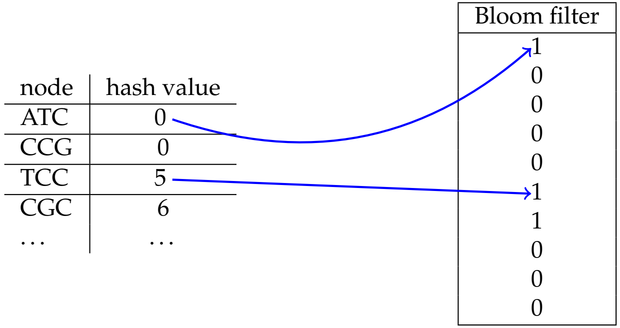
Recent improvement : lower-memory structure (2)

Bloom filter

Bit array to describe any set with a “precision” of ϵ.

- a proportion of ϵ elements will be wrongly included in the set.

First step : stores nodes in a Bloom filter.

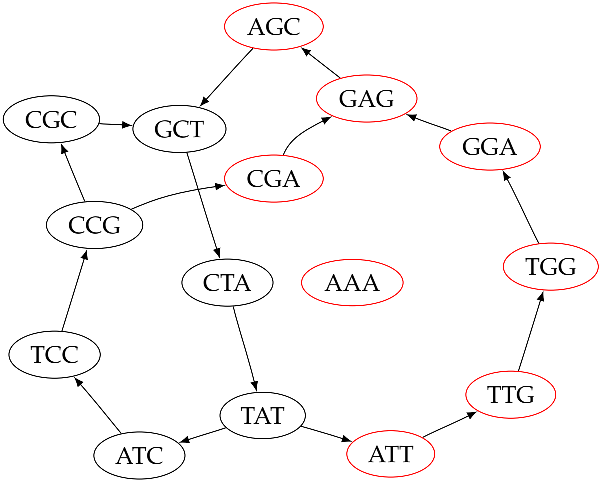


node	hash value
ATC	0
CCG	0
TCC	5
CGC	6
...	...

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

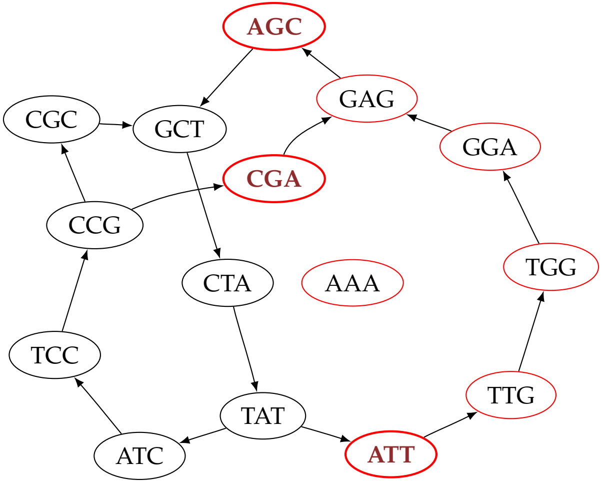
Recent improvement : lower-memory structure (3)

Actual set of **nodes** : {TAT, ATC, CGC, CTA, CCG, TCC, GCT}
Graph as stored in the previous Bloom filter :



Recent improvement : lower-memory structure (4)

Insight : using localized traversal from **black** nodes, only small a fraction of the red false positives are **troublesome**.

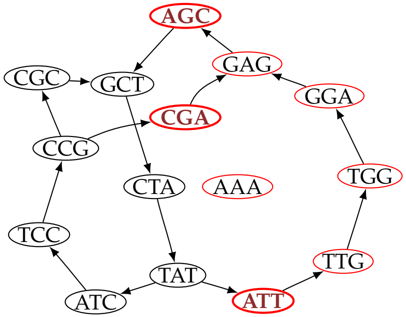


RECENT IMPROVEMENT : LOWER-MEMORY STRUCTURE (4)

Proposed method [RC, GR 11]

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

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



Nodes self-information :

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

Our structure size :

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

RECENT IMPROVEMENT : LOWER-MEMORY STRUCTURE (5)

Result statement

The de Bruijn graph can be encoded using

$$1.44 \log_2\left(\frac{16k}{2.08}\right) + 2.08$$

bits of memory per node.

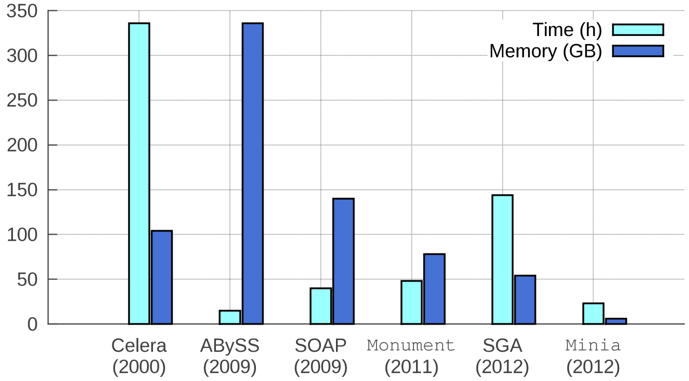
human, $k = 25$: **13** bits per node.

- Effectively below the self-information (20 bits/node)
- Not magic : it's an **over-approximation** made **exact** where it matters

Is it possible to perform assembly with this **immutable** structure ?
$\rightarrow$ **Yes, with localized traversal (Contribution 1)**.

Human genome assembly	**Minia**	C. & B.	ABySS	SOAPdenovo
Contig N50 (bp)	**1156**	250	870	886
$> 95\%$ 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**

YEAR 2012 : HOW COMPUTATIONNALLY HARD IS *assembly* TODAY ?



Summary of contributions

Contribution 1 :

- Redundancy-filtered reads **index**
- **Localized** assembly technique

Contribution 2 :

- Incorporation of **pairing information** in assembly models

Contribution 3 :

- **Space**-efficient **de Bruijn graph** representation

Contributions in the manuscript :

- Analysis of **re-sequencing** feasibility with exact paired reads
- **Index-free** targeted assembly (Mapsembler)

Perspectives

Applications

Why assemble a human genome *again* ?

- To exhibit novel **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.

→ paired short-read assembly will remain a hot topic for at least a few years.

Perspectives

Extension of localized assembly :

- **Graph**-based **gap-filling** (Monument, with T. Derrien, C. Lemaitre, & F. Legeai)

Extension of paired assembly theory :

- **Global scaffolding** (SuperScaffolding, with D. Naquin)
  common sub-paths that appear in all solutions of a Chinese Postman instance

Applications of Minia codebase :

- **Huge metagenomic** assemblies (with O. Jaillon, JM. Aury)
- **Transcriptome** assembly (Inchworm replacement)
- **Alternative splicing** detection (KisSplice module replacement)
- **SNP** detection (KisSnp 2, with R. Uricaru & P. Peterlongo)
- Read **compression** (with G. Rizk & D. Lavenier)
- Constant-memory **k-mer counting** (with G. Rizk)

Software contributions

Released software :

- Mapsembler[1]
- KisSplice[2]
- Minia[3]

On my github[4] :

- Paired repetitions analysis package
- Light-weight, explicit de Bruijn graph construction

Internal software :

- Monument
- SuperScaffolder

[1] http://alcovna.genouest.org/mapsembler
[2] http://alcovna.genouest.org/kissplice
[3] http://minia.genouest.org
[4] http://github.com/rchikhi

PUBLICATIONS

WABI 2011	*RC, DL*
PBC 2011	*GC, RC, DL*
BMC Bioinformatics 2011	*PP, RC*
Genome Research 2011	*Earl et al. (RC, DL, DN, GC, NM)*
RECOMB-Seq 2012	*Sacomoto et al. (RC, RU, PP)*
WABI 2012	*RC, GR*

Extended abstracts, posters :

BMC Bioinformatics, ISCB-SC 2009	*RC, DL*
Jobim 2012	*RC, DN*

Acknowledgments

- Dominique Lavenier
- Everyone at Symbiose, GenScale, GenOuest, Dyliss
- Pierre asked for a special dedicace
- M-F. Sagot, S. Gnerre, O. Jaillon, E. Rivals, B. Schmidt
- My family, D.

Thank you all for coming !