Examples of algorithms used in genomics

Michel Koskas

January 10-th 2018

1. Mapping problem
2. De Novo Sequencing (exact match)
   - The de Brujin graph
3. Usual technologies
   - Seed. . .
   - . . . & Grow!
4. Imputing maze lines
5. The Problem
6. Classical Resolution (Beagle)
7. Markov Chains
8. Imputing
9. Performances
10. An R package
11. Conclusion

1 Mapping problem
2 De Novo Sequencing (exact match)
- The de Brujin graph
3 Usual technologies
- Seed. . .
- . . . & Grow!
4 Imputing maze lines
5 The Problem
6 Classical Resolution (Beagle)
7 Markov Chains
8 Imputing
9 Performances
10 An R package
11 Conclusion

Mapping. From:

A reference Genome

$$
\begin{array}{l}
\quad 0 \qquad\qquad\qquad 1 \qquad\qquad\qquad 2 \qquad\qquad\qquad 3 \qquad\qquad\qquad 4 \qquad\qquad\qquad 5 \qquad\qquad\qquad 6 \qquad\qquad\qquad 7 \\
\quad 0123456789012345678901234567890123456789012345678901234567890123456789012345 \\
G{=}TGGCTTGTTGGTCATGGGGGTCGACTAGGAGACCATTATTTAGTGCCTTCACAATATCAACAGGCCCTCCATTTCT
\end{array}
$$

and a set of Reads:

$$
R = \begin{array}{l}
GGGGTCGACT \\
GAGACGTACTGT \\
CCGAGTCCCGGT \\
GCAGAAACCAG \\
CAAAGTCC \\
GGAAAGCTTGAC \\
GATTTGCCTCAG \\
TGCTTCCAAAA \\
GCAAGTTCTCTA \\
AATCACCCATA \\
ATCTTCTAAGGA \\
GGCTCACAAGTA \\
ACGAATTTAA \\
ATGGTTTTCACT \\
GTCACCCAATCA \\
GGTAACGGTGA
\end{array}
$$

Exact Match:

```
   0         1         2         3         4         5         6         7
   0123456789012345678901234567890123456789012345678901234567890123456789012345
G=TGGCTTGTTGGTCATGGGGGTCGACTAGGAGACCATTATTTAGTGCCTTCACAATATCAACAGGCCCTCCATTTCT
```

$R =$

GGGGTCGACT
GAGACGTACTGT
CCGAGTCCCGGT
GCAGAAACCAG
CAAAGTCC
GGAAAGCTTGAC
GATTTGCCTCAG
TGCTTCCAAAA
GCAAGTTCTCTA
AATCACCCATA
ATCTTCTAAGGA
GGCTCACAAGTA
ACGAATTTAA
ATGGTTTTCACT
GTCACCCAATCA
GGTAACGGTGA

Mismatch:

```
0         1         2         3         4         5         6         7
0123456789012345678901234567890123456789012345678901234567890123456789012345
```

G=TGGCTTGTTGGTCATGGGGGCCGACTAGGAGACCATTATTTAGTGCCTTCACAATATCAACAGGCCCTCCATTTCT

R =
GGGGTCGACT
GAGACGTACTGT
CCGAGTCCCGGT
GCAGAAACCAG
CAAAGTCC
GGAAAGCTTGAC
GATTTGCCTCAG
TGCTTCCAAAA
GCAAGTTCTCTA
AATCACCCATA
ATCTTCTAAGGA
GGCTCACAAGTA
ACGAATTTAA
ATGGTTTTCACT
GTCACCCAATCA
GGTAACGGTGA

InDel:

0 1 2 3 4 5 6 7
0123456789012345678901234567890123456789012345678901234567890123456789012345

G=TGGCTTGTTGGTCATGGGGG–CGACTAGGAGACCATTATTTAGTGCCTTCACAATATCAACAGGCCCTCCATTTCT

$$R = \begin{array}{l} GGGGTCGACT \\ GAGACGTACTGT \\ CCGAGTCCCGGT \\ GCAGAAACCAG \\ CAAAGTCC \\ GGAAAGCTTGAC \\ GATTTGCCTCAG \\ TGCTTCCAAAA \\ GCAAGTTCTCTA \\ AATCACCCATA \\ ATCTTCTAAGGA \\ GGCTCACAAGTA \\ ACGAATTTAA \\ ATGGTTTTCACT \\ GTCACCCAATCA \\ GGTAACGGTGA \end{array}$$

InDel:

```
   0         1         2         3         4         5         6         7
   0123456789012345678901234567890123456789012345678901234567890123456789012345
G=TGGCTTGTTGGTCATGGGGGTCGACTAGGAGACCATTATTTAGTGCCTTCACAATATCAACAGGCCCTCCATTTCT
```

$$R = \begin{array}{l} GGGG-CGACT \\ GAGACGTACTGT \\ CCGAGTCCCGGT \\ GCAGAAACCAG \\ CAAAGTCC \\ GGAAAGCTTGAC \\ GATTTGCCTCAG \\ TGCTTCCAAAA \\ GCAAGTTCTCTA \\ AATCACCCATA \\ ATCTTCTAAGGA \\ GGCTCACAAGTA \\ ACGAATTTAA \\ ATGGTTTTCACT \\ GTCACCCAATCA \\ GGTAACGGTGA \end{array}$$

The problem is to find all the occurrences of each read for a given maximum number of mismatches or In/Dels.

```
    0                   1                   2                   3                   4                   5                   6                   7
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
G=TGGCTTGTTGGTCATGGGGGCCGACTAGGGGAC-ACTTTTAGGGGGATCGACCATGGGGTCGACTCCTCCATTT
                  GGGGTCGACT  GGGGTCGACT      GGGG-TCGACT   GGGGTCGACT
```

$$R = \begin{array}{l} GGGGTCGACT \\ GAGACGTACTGT \\ CCGAGTCCCGGT \\ GCAGAAACCAG \\ CAAAGTCC \\ GGAAAGCTTGAC \\ GATTTGCCTCAG \\ TGCTTCCAAAA \\ GCAAGTTCTCTA \\ AATCACCCATA \\ ATCTTCTAAGGA \\ GGCTCACAAGTA \\ ACGAATTTAA \\ ATGGTTTTCACT \\ GTCACCCAATCA \\ GGTAACGGTGA \end{array}$$

1. Mapping problem
2. De Novo Sequencing (exact match)
   - The de Brujin graph
3. Usual technologies
   - Seed. . .
   - . . . & Grow!
4. Imputing maze lines
5. The Problem
6. Classical Resolution (Beagle)
7. Markov Chains
8. Imputing
9. Performances
10. An R package
11. Conclusion

A puzzle

A NGS machine produces billions of pieces of DNA.
These pieces have to be put together to find DNA back.

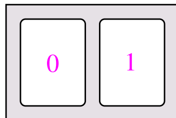
For instance how should we assemble the following *reads*?
BRAC, *CADA*, *ACAD*, *ABRA*, *ABRA*, *ADAB* ?

A puzzle

A NGS machine produces billions of pieces of DNA.
These pieces have to be put together to find DNA back.

For instance how should we assemble the following *reads*?
BRAC, *CADA*, *ACAD*, *ABRA*, *ABRA*, *ADAB* ?

The de Brujin Graphs:
An erudite burglar.

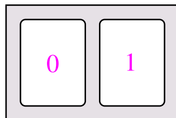


A digital lock has a three digits code.
When trying 0100, one tries 010 and 100.

Question: is it possible to try all the possibilities in a minimal time (minimal number of keys pressed)

(Such a sequence should be of length $3 + 2^3 - 1 = 10$)

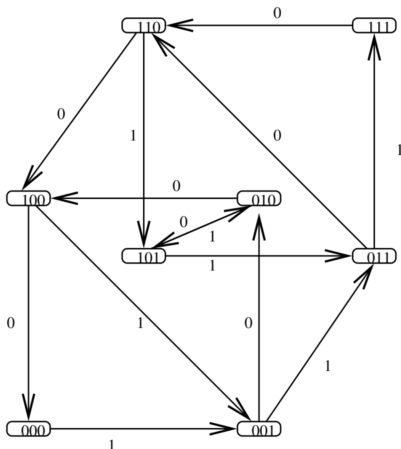
The de Brujin Graphs:
An erudite burglar.



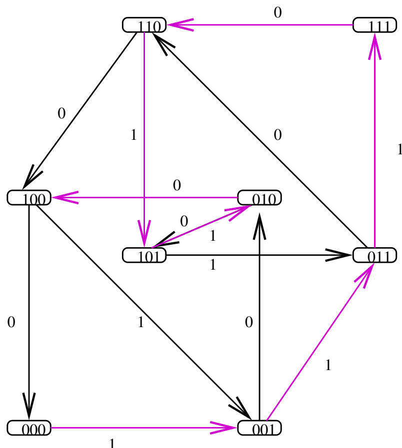
A digital lock has a three digits code.
When trying 0100, one tries 010 and 100.

Question: is it possible to try all the possibilities in a minimal time (minimal number of keys pressed)
(Such a sequence should be of length $3 + 2^3 - 1 = 10$)

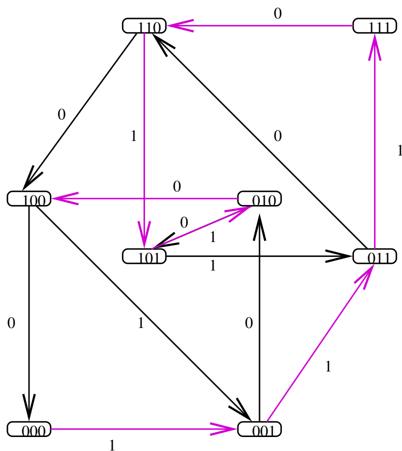
The question is to find a hamiltonian path among this graph:



The question is to find a hamiltonian path among this graph:



The question is to find a hamiltonian path among this graph:



0001110100

Now the first goal is to build the de brujin graph of the reads
This is the problem we address here.
ABRA, ACAD, ABRA, ADAB, BRAC, CADA

	ABRA	ACAD	ABRA	ADAB	BRAC	CADA
ABRA	ABRA	ABRACAD	ABRA	ABRADAB	ABRAC	ABRA_CADA
ACAD	ACAD_ABRA	ACAD	ACAD_ABRA	ACADAB	ACAD_BRAC	ACADA
ABRA	ABRA	ABRACAD	ABRA	ABRADAB	ABRAC	ABRA_CADA
ADAB	ADABRA	ADAB_ACAD	ADABRA	ADAB	ADABRAC	ADAB_CADA
BRAC	BRAC_ABRA	BRACAD	BRAC_ABRA	BRAC_ADAB	BRAC	BRACADA
CADA	CADABRA	CADACAD	CADABRA	CADAB	CADA_BRAC	CADA

Now the first goal is to build the de brujin graph of the reads
This is the problem we address here.
ABRA, ACAD, ABRA, ADAB, BRAC, CADA
(Cleaning up . . .)

	ABRA	ACAD	ABRA	ADAB	BRAC	CADA
ABRA		ABRACAD	ABRA	ABRADAB	ABRAC	ABRA_CADA
ACAD	ACAD_ABRA		ACAD_ABRA	ACADAB	ACAD_BRAC	ACADA
ABRA	ABRA	ABRACAD		ABRADAB	ABRAC	ABRA_CADA
ADAB	ADABRA	ADAB_ACAD	ADABRA		ADABRAC	ADAB_CADA
BRAC	BRAC_ABRA	BRACAD	BRAC_ABRA	BRAC_ADAB		BRACADA
CADA	CADABRA	CADACAD	CADABRA	CADAB	CADA_BRAC	
BRAC	BRAC_ABRA	BRACAD	BRAC_ABRA	BRAC_ADAB	BRAC	BRACADA

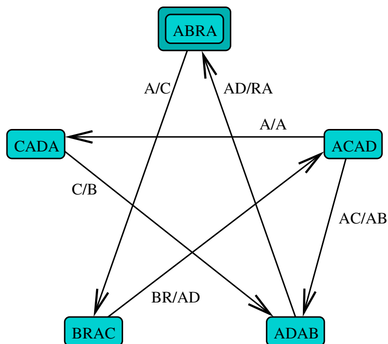
Now the first goal is to build the de brujin graph of the reads
This is the problem we address here.
ABRA, ACAD, ABRA, ADAB, BRAC, CADA
(Cleaning up . . .)

	ABRA	ACAD	ABRA	ADAB	BRAC	CADA
ABRA		ABRACAD	ABRA	ABRADAB	ABRAC	
ACAD				ACADAB		ACADA
ABRA	ABRA	ABRACAD		ABRADAB	ABRAC	
ADAB	ADABRA		ADABRA		ADABRAC	
BRAC		BRACAD				BRACADA
CADA	CADABRA	CADACAD	CADABRA	CADAB		

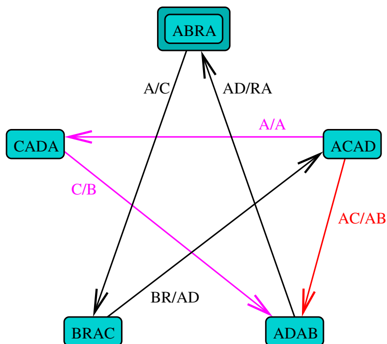
Now the first goal is to build the de brujin graph of the reads
This is the problem we address here.
ABRA, ACAD, ABRA, ADAB, BRAC, CADA
(Cleaning up . . .)

	ABRA	ACAD	ABRA	ADAB	BRAC	CADA
ABRA			ABRA		ABRAC	
ACAD				ACADAB		ACADA
ABRA	ABRA				ABRAC	
ADAB	ADABRA		ADABRA			
BRAC		BRACAD				
CADA				CADAB		

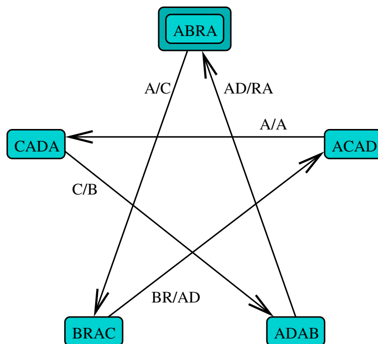
Then one reconstitutes the de Brujin graph:



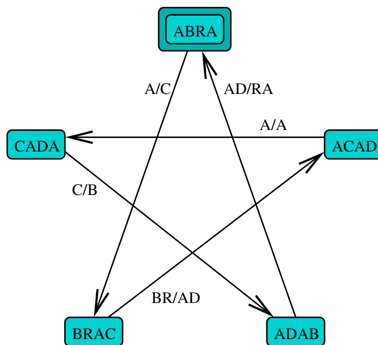
Then one reconstitutes the de Brujin graph:



Then one reconstitutes the de Brujin graph:

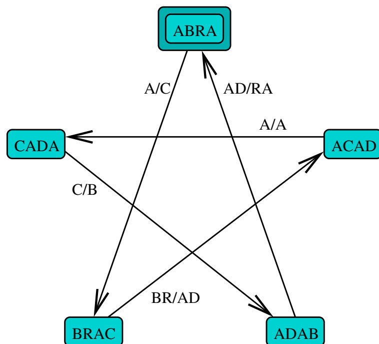


Then one reconstitutes the de Brujin graph:



The frequencies allow us to reconstitute the sequence
ABRACADABRA

Then one reconstitutes the de Brujin graph:



The frequencies allow us to reconstitute the sequence ABRACADABRA
Repetitions in the full sequence is a very serious issue. . . ABRACADABRA (Or was it ABRABRABRABRABRABRACADABRA ?)

de Novo Sequencing (exact matching)
The problem:
Two sets of reads, Left and Right

$$Left = \begin{matrix} CT \ldots AGT \\ AA \ldots CGA \\ \vdots \vdots \ddots \vdots \vdots \vdots \\ GT \ldots CGT \\ CC \ldots GAT \end{matrix} \quad Right = \begin{matrix} GT \ldots AGT \\ AT \ldots CGT \\ \vdots \vdots \ddots \vdots \vdots \vdots \\ GAT \ldots GA \\ CC \ldots GAT \end{matrix}$$

(Actually, when the reads are stored for instance in a FATSA or FASTQ file we have more information that is disregarded for the sake of clarity)

de Novo Sequencing (exact matching)

The problem:

and one looks for exact matching overlaps between the Left and Right sets (prefix of a Left side read matches a suffix of a right side read) ...

$$Left = \begin{matrix} CT\ldots AGT \\ AA\ldots CGA \\ \vdots \ \vdots \ \ddots \ \vdots \ \vdots \ \vdots \\ GT\ldots CGT \\ CC\ldots GAT \end{matrix} \quad Right = \begin{matrix} GT\ldots A\,GT \\ AT\ldots C\,GT \\ \vdots \ \vdots \ \ddots \ \vdots \ \vdots \ \vdots \\ GA\,T\ldots GA \\ CC\ldots G\,AT \end{matrix}$$

de Novo Sequencing (exact matching)

The problem:

...for any length

$$Left = \begin{matrix} CT\ldots AGT \\ AA\ldots CGA \\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots \\ GT\ldots CGT \\ CC\ldots {\color{red}GAT} \end{matrix} \quad Right = \begin{matrix} GT\ldots A\,GT \\ AT\ldots C\,GT \\ \vdots\ \vdots\ \ddots\ \vdots\ \vdots\ \vdots \\ {\color{red}GA\,T}\ldots GA \\ CC\ldots G\,AT \end{matrix}$$

Remark: *Left* and *Right* may be the same set.

Remark: Of course reads might have different lengths, but for the sake of clarity we suppose they have the same length. Otherwise it is easy to adapt the description to the general case.

$$Left = \begin{matrix} C & T & \dots & A & G & T \\ A & A & \dots & C & G & A \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ G & T & \dots & C & G & T \\ C & C & \dots & \color{red}{G} & \color{red}{A} & \color{red}{T} \end{matrix} \quad Right = \begin{matrix} G & T & \dots & A & G & T \\ A & T & \dots & C & G & T \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \color{red}{G} & \color{red}{-} & \color{red}{T} & \dots & G & A \\ C & C & \dots & G & A & T \end{matrix}$$

Of course there might be mismatches and/or indels in the overlaps...

1 Mapping problem
2 De Novo Sequencing (exact match)
- The de Brujin graph
3 Usual technologies
- Seed. . .
- . . . & Grow!
4 Imputing maze lines
5 The Problem
6 Classical Resolution (Beagle)
7 Markov Chains
8 Imputing
9 Performances
10 An R package
11 Conclusion

Usual technologies use “Seed & Grow” techniques:
Find an exact match for a $k$mer of a read (the seed)
Then try to extend the seed, accepting a maximum number of mismatches and/or InDel.

Usual technologies use “Seed & Grow” techniques:
Find an exact match for a $k$mer of a read (the seed)
Then try to extend the seed, accepting a maximum number of mismatches and/or InDel.

Hash Tables

A Hash table is an efficient in-memory storage of key-value couples enabling one

- To add an element in a constant average time
- find the value for a given key in a constant time.

Hash the k-mers of the reference genome (key: hashed k-mer, value: indexes of occurrences)

```
   0                   1                   2                   3                   4                   5                   6                   7
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
G= T G G C T T G T T G G T C A T G G G G C C G A C T A G G A G A C C A T T A T T T A G T G C C T T C A C A A T A T C A A C A G G C C C T C C A T T T C T
```

G=TGGCTTGTTGGTCATGGGGCCGACTAGGAGACCATTATTTAGTGCCTTCACAATATCAACAGGCCCTCCATTTCT

Key	*Value*
TGG	0, 8, 14

Hash the k-mers of the reference genome (key: hashed k-mer, value: indexes of occurrences)

```
  0                   1                   2                   3                   4                   5                   6                   7
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
G=T G G C T T G T T G G T C A T G G G G G C C G A C T A G G A G A C C A T T A T T T A G T G C C T T C A C A A T A T C A A C A G G C C C T C C A T T T C T
```

Key	Value
AAA	87
AAC	58
AAT	52, 88
ACA	50, 59
ACC	31
ACT	23
AGA	29
AGG	26, 61, 84
AGT	41
ATA	53
ATC	55
ATG	13
ATT	34, 37, 70

Key	Value
CAA	51, 57
CAC	49
CAG	60
CAT	12, 33, 69
CCA	32, 68
CCC	64
CCG	20
CCT	45, 65, 80
CGA	21
CGC	78
CTA	24
CTC	67, 75, 77
CTT	3, 46, 81

Key	Value
GAA	86
GAC	22, 30
GAG	28
GCC	19, 44, 63, 79
GCT	2
GGA	27, 85
GGC	1, 18, 62
GGG	15, 16, 17
GGT	9
GTC	10
GTG	42
GTT	6

Key	Value
TAG	25, 40, 83
TAT	36, 54
TCA	11, 48, 56
TCC	67
TCG	77
TCT	73, 75
TGC	43
TGG	0, 8, 14
TGT	5
TTA	35, 39, 82
TTC	47, 72
TTG	4, 7
TTT	38, 71

The hash table *requires* to fit into memory (under penalty of complete loss of efficiency).
When a key-value is grabbed we have no idea of which will next be asked for.
These couples are stored in a hash table. But ...

SeedSize	*TableSize*	*Storagespace*(*GB*)
32	18,446,744,073,709,551,616	137,438,953,472
28	72,057,594,037,927,936	536,870,912
24	281,474,976,710,656	2,097,152
20	1,099,511,627,776	8,192
18	68,719,476,736	512
16	4,294,967,296	32
12	16,777,216	128*MB*

(The storage space is the *maximal* storage space, each number being stored on 8 bytes.)

Long seeds will demand more space,

while short seeds will require more computation time. . .

SeedSize	*TableSize*	*Storagespace*(*GB*)
32	18,446,744,073,709,551,616	137,438,953,472
28	72,057,594,037,927,936	536,870,912
24	281,474,976,710,656	2,097,152
20	1,099,511,627,776	8,192
18	68,719,476,736	512
16	4,294,967,296	32
12	16,777,216	128*MB*

(The storage space is the *maximal* storage space, each number being stored on 8 bytes.)
Long seeds will demand more space,
while short seeds will require more computation time...

SeedSize	*TableSize*	*Storagespace(GB)*
32	18,446,744,073,709,551,616	137,438,953,472
28	72,057,594,037,927,936	536,870,912
24	281,474,976,710,656	2,097,152
20	1,099,511,627,776	8,192
18	68,719,476,736	512
16	4,294,967,296	32
12	16,777,216	128*MB*

(The storage space is the *maximal* storage space, each number being stored on 8 bytes.)
Long seeds will demand more space,
while short seeds will require more computation time…

Suffix Arrays

0 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

G=$TGGCTTGTTGGTCATGGGGG$

Build a suffix array consists in

1. Add a letter \$ at the end of the word (lexicographically smaller than any letter)
   0 1 2
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
   G=$TGGCTTGTTGGTCATGGGGG$\$
2. compute its conjugates
3. Sort them

Suffix Arrays

```
0                   1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
```

G=TGGCTTGTTGGTCATGGGGG

Build a suffix array consists in

1. Add a letter $ at the end of the word (lexicographically smaller than any letter)

```
    0                   1                   2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
G=TGGCTTGTTGGTCATGGGGG$
```

2. compute its conjugates

```
    0                   1                   2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
G=TGGCTTGTTGGTCATGGGGG$    (0)
  GGCTTGTTGGTCATGGGGG$T    (1)
  GCTTGTTGGTCATGGGGG$TG    (2)
  CTTGTTGGTCATGGGGG$TGG    (3)
  TTGTTGGTCATGGGGG$TGGC    (4)
  TGTTGGTCATGGGGG$TGGCT    (5)
  GTTGGTCATGGGGG$TGGCTT    (6)
  TTGGTCATGGGGG$TGGCTTG    (7)
  TGGTCATGGGGG$TGGCTTGT    (8)
  GGTCATGGGGG$TGGCTTGTT    (9)
  GTCATGGGGG$TGGCTTGTTG    (10)
  TCATGGGGG$TGGCTTGTTGG    (11)
  CATGGGGG$TGGCTTGTTGGT    (12)
  ATGGGGG$TGGCTTGTTGGTC    (13)
  TGGGGG$TGGCTTGTTGGTCA    (14)
  GGGGG$TGGCTTGTTGGTCAT    (15)
  GGGG$TGGCTTGTTGGTCATG    (16)
  GGG$TGGCTTGTTGGTCATGG    (17)
  GG$TGGCTTGTTGGTCATGGG    (18)
  G$TGGCTTGTTGGTCATGGGG    (19)
```

Suffix Arrays

```
0                   1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
```

G=TGGCTTGTTGGTCATGGGGG

Build a suffix array consists in

1. Add a letter $ at the end of the word (lexicographically smaller than any letter)

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
```

G=TGGCTTGTTGGTCATGGGGG$

2. compute its conjugates
3. Sort them

```
  0                   1                   2
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
G=$TGGCTTGTTGGTCATGGGGG  (20)
  ATGGGGG$TGGCTTGTTGGTC  (13)
  CATGGGGG$TGGCTTGTTGGT  (12)
  CTTGTTGGTCATGGGGG$TGG  (3)
  GCTTGTTGGTCATGGGGG$TG  (2)
  G$TGGCTTGTTGGTCATGGGG  (19)
  GG$TGGCTTGTTGGTCATGGG  (18)
  GGCTTGTTGGTCATGGGGG$T  (1)
  GGG$TGGCTTGTTGGTCATGG  (17)
  GGGG$TGGCTTGTTGGTCATG  (16)
  GGGGG$TGGCTTGTTGGTCAT  (15)
  GGTCATGGGGG$TGGCTTGTT  (9)
  GTCATGGGGG$TGGCTTGTTG  (10)
  GTTGGTCATGGGGG$TGGCTT  (6)
  TCATGGGGG$TGGCTTGTTGG  (11)
  TGGCTTGTTGGTCATGGGGG$  (0)
  TGGGGG$TGGCTTGTTGGTCA  (14)
  TGGTCATGGGGG$TGGCTTGT  (8)
```

looking for the seed GGG

Suffix Arrays

```
    0                   1
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
G=T G G C T T G T T G G T C A T G G G G G
```

Build a suffix array consists in

1. Add a letter $ at the end of the word (lexicographically smaller than any letter)

```
    0                   1                   2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
G=T G G C T T G T T G G T C A T G G G G G $
```

2. compute its conjugates
3. Sort them

```
    0                   1                   2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
G=$TGGCTTGTTGGTCATGGGGG   (20)
  ATGGGGG$TGGCTTGTTGGTC   (13)
  CATGGGGG$TGGCTTGTTGGT   (12)
  CTTGTTGGTCATGGGGG$TGG   (3)
  GCTTGTTGGTCATGGGGG$TG   (2)
  G$TGGCTTGTTGGTCATGGGG   (19)
  GG$TGGCTTGTTGGTCATGGG   (18)
  GGCTTGTTGGTCATGGGGG$T   (1)
  GGG$TGGCTTGTTGGTCATGG   (17)   looking for the seed GGG
  GGGG$TGGCTTGTTGGTCATG   (16)
  GGGGG$TGGCTTGTTGGTCAT   (15)
  GGTCATGGGGG$TGGCTTGTT   (9)
  GTCATGGGGG$TGGCTTGTTG   (10)
  GTTGGTCATGGGGG$TGGCTT   (6)
  TCATGGGGG$TGGCTTGTTGG   (11)
  TGGCTTGTTGGTCATGGGGG$   (0)
  TGGGGG$TGGCTTGTTGGTCA   (14)
  TGGTCATGGGGG$TGGCTTGT   (8)
```

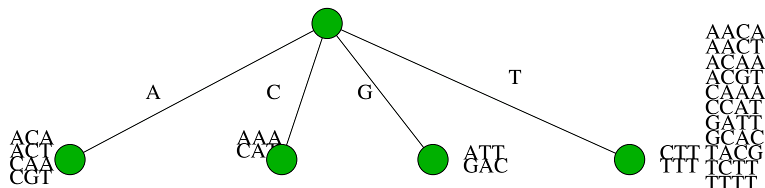
The Suffix Tree may be seen as an attempt to save space...

AACA
AACT
ACAA
ACGT
CAAA
CCAT
GATT
GCAC
TACG
TCTT
TTTT

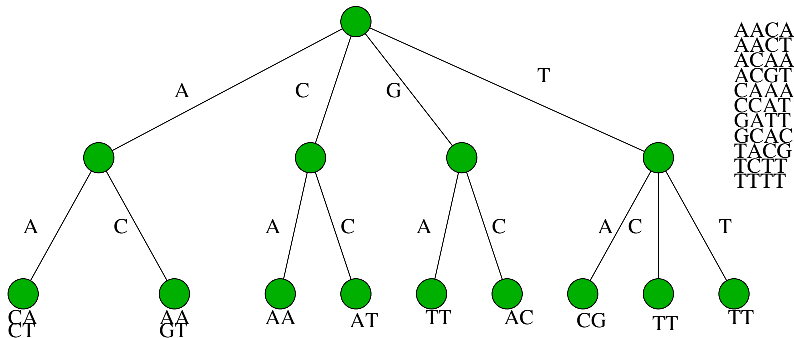
The Suffix Tree may be seen as an attempt to save space...

AACA
AACT
ACAA
ACGT
CAAA
CCAT
GATT
GCAC
TACG
TCTT
TTTT

The Suffix Tree may be seen as an attempt to save space...

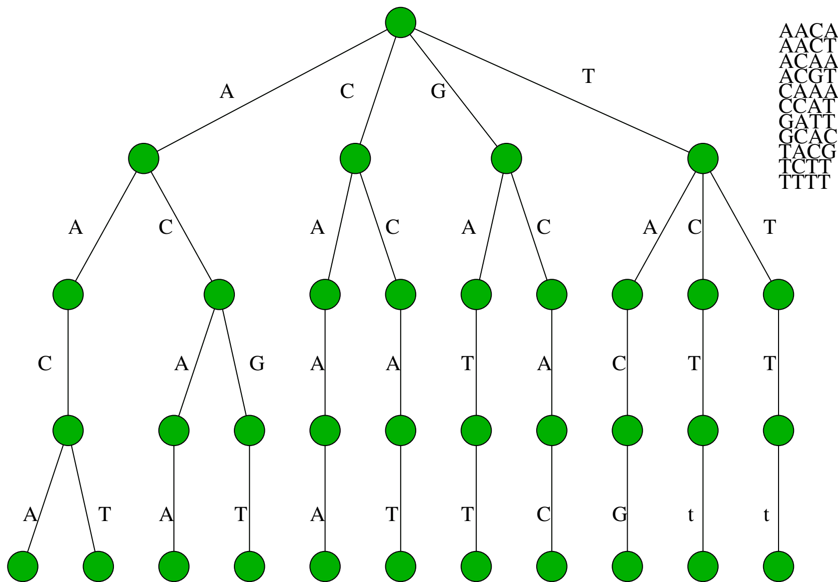


The Suffix Tree may be seen as an attempt to save space. . .



The Suffix Tree may be seen as an attempt to save space. . .

The Suffix Tree may be seen as an attempt to save space. . .



The Burrows-Wheeler Transform consists in keeping only the last column of the preceding array:

```
    0                   1                   2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

G=TGGCTTGTTGGTCATGGGGG$
    0                   1                   2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

G=$TGGCTTGTTGGTCATGGGGG  (20)
  ATGGGGG$TGGCTTGTTGGTC  (13)
  CATGGGGG$TGGCTTGTTGGT  (12)
  CTTGTTGGTCATGGGGG$TGG  (3)
  GCTTGTTGGTCATGGGGG$TG  (2)
  G$TGGCTTGTTGGTCATGGGG  (19)
  GG$TGGCTTGTTGGTCATGGG  (18)
  GGCTTGTTGGTCATGGGGG$T  (1)
  GGG$TGGCTTGTTGGTCATGG  (17)
  GGGG$TGGCTTGTTGGTCATG  (16)
  GGGGG$TGGCTTGTTGGTCAT  (15)
  GGTCATGGGGG$TGGCTTGTT  (9)
  GTCATGGGGG$TGGCTTGTTG  (10)
  GTTGGTCATGGGGG$TGGCTT  (6)
  TCATGGGGG$TGGCTTGTTGG  (11)
  TGGCTTGTTGGTCATGGGGG$  (0)
  TGGGGG$TGGCTTGTTGGTCA  (14)
  TGGTCATGGGGG$TGGCTTGT  (8)
  TGTTGGTCATGGGGG$TGGCT  (5)
  TTGGTCATGGGGG$TGGCTTG  (7)
  TTGTTGGTCATGGGGG$TGGC  (4)
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
GCTGGGGTGGTTGTG$ATTGC
```

The Burrows-Wheeler Transform consists in keeping only the last column of the preceding array:

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
```

G=*TGGCTTGTTGGTCATGGGGG*$

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
```

GCTGGGGTGGTTGTG$*ATTGC*

It is easy to reconstitute the whole array with this single line.

Seeds with FM-Indexes:

One builds two arrays
Relative Indexes & move from relative to Absolute indexes:

	0 0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	$	*A*	*T*	*T*	*G*	*C*
$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7
	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

Seeds with FM-Indexes:

One builds two arrays
Relative Indexes & move from relative to Absolute indexes:

	0										1										2
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	$	*A*	*T*	*T*	*G*	*C*
$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7
	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

And notices that $[L(aP), R(aP)] = Abs(a) + [Rel_a(L(P) - 1), Rel_a(R(P)) - 1]$.

$\varepsilon \rightarrow [0, 20]$.

Hence $G \rightarrow 4 + [0, 10 - 1] = [4, 13]$

Hence $GG \rightarrow 4 + [2, 8 - 1] = [6, 11]$

Hence $GGG \rightarrow 4 + [4, 7 - 1] = [8, 10]$

Hence GGG appears at indexes $\{17, 16, 15\}$.

Seeds with FM-Indexes:

One builds two arrays
Relative Indexes & move from relative to Absolute indexes:

	0 0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	$	*A*	*T*	*T*	*G*	*C*
$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7

	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

And notices that $[L(aP), R(aP)] = Abs(a) + [Rel_a(L(P) - 1), Rel_a(R(P)) - 1]$.
$\varepsilon \rightarrow [0, 20]$.
Hence $G \rightarrow 4 + [0, 10 - 1] = [4, 13]$
Hence $GG \rightarrow 4 + [2, 8 - 1] = [6, 11]$
Hence $GGG \rightarrow 4 + [4, 7 - 1] = [8, 10]$
Hence GGG appears at indexes $\{17, 16, 15\}$.

Seeds with FM-Indexes:

One builds two arrays

Relative Indexes & move from relative to Absolute indexes:

	0										1										2
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	$	*A*	*T*	*T*	*G*	*C*
$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7
	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

And notices that $[L(aP), R(aP)] = Abs(a) + [Rel_a(L(P) - 1), Rel_a(R(P)) - 1]$.

$\varepsilon \rightarrow [0, 20]$.

Hence $G \rightarrow 4 + [0, 10 - 1] = [4, 13]$

Hence $GG \rightarrow 4 + [2, 8 - 1] = [6, 11]$

Hence $GGG \rightarrow 4 + [4, 7 - 1] = [8, 10]$

Hence GGG appears at indexes $\{17, 16, 15\}$.

Seeds with FM-Indexes:

One builds two arrays
Relative Indexes & move from relative to Absolute indexes:

	0										1										2
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	\$	*A*	*T*	*T*	*G*	*C*
\$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7
	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	\$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

And notices that $[L(aP), R(aP)] = Abs(a) + [Rel_a(L(P) - 1), Rel_a(R(P)) - 1]$.

$\varepsilon \to [0, 20]$.

Hence $G \to 4 + [0, 10 - 1] = [4, 13]$

Hence $GG \to 4 + [2, 8 - 1] = [6, 11]$

Hence $GGG \to 4 + [4, 7 - 1] = [8, 10]$

Hence GGG appears at indexes $\{17, 16, 15\}$.

Seeds with FM-Indexes:

One builds two arrays
Relative Indexes & move from relative to Absolute indexes:

	0 0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	$	*A*	*T*	*T*	*G*	*C*
$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7
	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

And notices that $[L(aP), R(aP)] = Abs(a) + [Rel_a(L(P) - 1), Rel_a(R(P)) - 1]$.
$\varepsilon \to [0, 20]$.
Hence $G \to 4 + [0, 10 - 1] = [4, 13]$
Hence $GG \to 4 + [2, 8 - 1] = [6, 11]$
Hence $GGG \to 4 + [4, 7 - 1] = [8, 10]$
Hence GGG appears at indexes $\{17, 16, 15\}$.

Seeds with FM-Indexes:

One builds two arrays
Relative Indexes & move from relative to Absolute indexes:

	0 0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	$	*A*	*T*	*T*	*G*	*C*
$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7
	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

And notices that $[L(aP), R(aP)] = Abs(a) + [Rel_a(L(P) - 1), Rel_a(R(P)) - 1]$.
$\varepsilon \rightarrow [0, 20]$.
Hence $G \rightarrow 4 + [0, 10 - 1] = [4, 13]$
Hence $GG \rightarrow 4 + [2, 8 - 1] = [6, 11]$
Hence $GGG \rightarrow 4 + [4, 7 - 1] = [8, 10]$
Hence GGG appears at indexes $\{17, 16, 15\}$.

Seeds with FM-Indexes:

One builds two arrays

Relative Indexes & move from relative to Absolute indexes:

	0										1										2
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
	G	*C*	*T*	*G*	*G*	*G*	*G*	*T*	*G*	*G*	*T*	*T*	*G*	*T*	*G*	$	*A*	*T*	*T*	*G*	*C*
$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
G	1	1	1	2	3	4	5	5	6	7	7	7	8	8	9	9	9	9	9	10	10
T	0	0	1	1	1	1	1	2	2	2	3	4	4	5	5	5	5	6	7	7	7
	(20)	(13)	(12)	(3)	(2)	(19)	(18)	(1)	(17)	(16)	(15)	(9)	(10)	(6)	(11)	(0)	(14)	(8)	(5)	(7)	(4)

c	$	*A*	*C*	*G*	*T*
First c Ind	0	1	2	4	14

(Each integer should be stored on at least 4 bytes so a naive storage of this array would take at least 17 $\times$ $|Ref|$)

The Smith-Waterman Algorithm
Now that seeds are found, let's grow them! (dynamic programming)
At Index 1 we find:
GGCTT
How does it match *GGACT*?
For instance we suppose that
An exact match is $+3$, A Mismatch is -3, Insertion or Deletion is -2
Let's find the best matching...

		G	G	C	T	T
	0	0	0	0	0	0
G	0					
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0 →	−3				
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	−3				
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3				
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3				
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3	0			
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3	−3			
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3	3			
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3	3			
G	0					
A	0					
C	0					
T	0					

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3	3	0	−3	−3
G	0	3	6	3	0	−3
A	0	0	3	3	0	−3
C	0	−3	0	6	3	0
T	0	−3	−6	3	9	6

		G	G	C	T	T
	0	0	0	0	0	0
G	0	3	3	0	−3	−3
G	0	3	6	3	0	−3
A	0	0	3	3	0	−3
C	0	−3	0	6	3	0
T	0	−3	−6	3	9	6

$$
\begin{matrix} G & G & - & C & T & T \\ G & G & A & C & T & - \end{matrix}
$$

1 Mapping problem
2 De Novo Sequencing (exact match)
- The de Brujin graph
3 Usual technologies
- Seed…
- …& Grow!
4 Imputing maze lines
5 The Problem
6 Classical Resolution (Beagle)
7 Markov Chains
8 Imputing
9 Performances
10 An R package
11 Conclusion

WORK IN PROGRESS

1. Mapping problem
2. De Novo Sequencing (exact match)
   - The de Brujin graph
3. Usual technologies
   - Seed…
   - …& Grow!
4. Imputing maze lines
5. The Problem
6. Classical Resolution (Beagle)
7. Markov Chains
8. Imputing
9. Performances
10. An R package
11. Conclusion

We have maize lines

Ind	M/A	I_1	I_2	I_3	$\ldots$	I_n
L_1	A/C	AA	CC	CC	$\ldots$	AA
L_2	T/A	TT	TT	AA	$\ldots$	TT
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
L_L	G/C	GG	CC	GG	$\ldots$	GG

We have maize lines

		l_1	l_2	l_3	$\dots$	l_n
L_1	AC	A	C	C	$\dots$	A
L_2	TA	T	T	A	$\dots$	T
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
L_L	GC	G	C	G	$\dots$	G

We have maize lines

		l_1	l_2	l_3	$\dots$	L_n
L_1	AC	1	0	0	$\dots$	1
L_2	TA	1	1	0	$\dots$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
L_L	GC	1	0	1	$\dots$	0

BUT some of the contents are unknown

		I_1	I_2	I_3	$\ldots$	I_n
L_1	AC	1	?	0	$\ldots$	1
L_2	TA	?	?	0	$\ldots$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
L_L	GC	1	0	?	$\ldots$	?

and our mission (if we accept it) is to fill missing data by using statistical methods.
(this file format is a very simplified VCF file format)

BUT some of the contents are unknown

		I_1	I_2	I_3	$\dots$	I_n
L_1	AC	1	?	0	$\dots$	1
L_2	TA	?	?	0	$\dots$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
L_L	GC	1	0	?	$\dots$	?

and our mission (if we accept it) is to fill missing data by using statistical methods.
(this file format is a very simplified VCF file format)

1. Mapping problem
2. De Novo Sequencing (exact match)
   - The de Brujin graph
3. Usual technologies
   - Seed…
   - …& Grow!
4. Imputing maze lines
5. The Problem
6. Classical Resolution (Beagle)
7. Markov Chains
8. Imputing
9. Performances
10. An R package
11. Conclusion

A classical resolution consists in:

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

A classical resolution consists in:

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

Identifying haplotypes blocks

A classical resolution consists in:

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	0	0	1	1	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

Then apply majority among blocks *having the same known data*

This is a “natural” way in the sense that it respects biology and its accuracy is very high. But

1. the computation is very long
2. It predicts a few heterozygous loci
3. when the missing data rate is very high, the accuracy drops down

Can we approach this accuracy with much less computation time?

This is a “natural” way in the sense that it respects biology and its accuracy is very high. But

1. the computation is very long
2. It predicts a few heterozygous loci
3. when the missing data rate is very high, the accuracy drops down

Can we approach this accuracy with much less computation time?

This is a “natural” way in the sense that it respects biology and its accuracy is very high. But

1. the computation is very long
2. It predicts a few heterozygous loci
3. when the missing data rate is very high, the accuracy drops down

Can we approach this accuracy with much less computation time?

This is a “natural” way in the sense that it respects biology and its accuracy is very high. But

1. the computation is very long
2. It predicts a few heterozygous loci
3. when the missing data rate is very high, the accuracy drops down

Can we approach this accuracy with much less computation time?

This is a “natural” way in the sense that it respects biology and its accuracy is very high. But

1. the computation is very long
2. It predicts a few heterozygous loci
3. when the missing data rate is very high, the accuracy drops down

Can we approach this accuracy with much less computation time?

1. Mapping problem
2. De Novo Sequencing (exact match)
   - The de Brujin graph
3. Usual technologies
   - Seed...
   - ...& Grow!
4. Imputing maze lines
5. The Problem
6. Classical Resolution (Beagle)
7. **Markov Chains**
8. Imputing
9. Performances
10. An R package
11. Conclusion

This is a purely computer-oriented approach, blind to biology

We first choose a number m (the “Markov Chain Order”).

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

m may be interpreted as a “recall length”
(In this example, $m = 3$).
Then for each and every locus:

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

then at this given locus one numbers the occurrences of consecutive given letters
(number the 000 followed by 0, 000 by 1, 001 by 0, 001 by 1 . . .)

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

. . . this for each Ind

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

For instance here 011 $\rightarrow$ 0 has been seen +1 time here (same locus, Ind varies)

But what if there is missing data?

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	0	?	1	1	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

But what if all data is not seen? ?0? $\rightarrow$ 1 is turned into

000 $\rightarrow$ 1 : +0.25

001 $\rightarrow$ 1 : +0.25

100 $\rightarrow$ 1 : +0.25

101 $\rightarrow$ 1 : +0.25

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	?	?	?	?	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

But what if all data is not seen? ??? $\rightarrow$? could cause increments by $\frac{1}{2^{m+1}}$ which seems useless (won't change relative orders).

	L32	L33	L34	L35	L36	L37	L38	L39	L40	L41	L42	L43	L44	L45
I27	?	0	1	?	0	0	1	1	0	?	0	1	1	0
I28	1	?	?	1	?	?	0	0	?	1	1	?	?	0
I29	?	0	1	1	0	0	0	?	1	?	?	0	0	?
I30	1	?	0	?	1	?	1	0	0	0	?	1	?	0
I31	?	?	?	?	?	0	?	1	1	0	0	?	1	1
I32	?	1	0	0	1	0	0	?	1	?	1	0	?	1
I33	?	?	1	0	1	?	0	1	?	0	?	1	1	?
I34	1	1	1	0	0	1	0	1	1	0	0	1	?	1
I35	?	0	0	1	1	0	?	0	?	1	?	1	0	1
I36	1	1	?	0	?	1	1	0	0	0	1	0	0	0

But what if all data is not seen? ??? $\rightarrow$? could cause increments by $\frac{1}{2^{m+1}}$ which seems useless (won't change relative orders).

We chose to do nothing (no increment)

These matrices of transitions are a markov chain of order m.

How should we choose m? First it has a lot to do with the necessary memory for the process.

If the number of loci is L, the necessary amount of memory is $(L - m)2^{m+3}$ bytes.

That is to say that if $L = 300,000$ and $m = 10$ you will need 2.5 GB of memory.

The amount of necessary memory will *double* if m is *incremented by 1!*

usually $m \leq 4$.

But in our case we will make it run with bigger values of m.

These matrices of transitions are a markov chain of order m.
How should we choose m? First it has a lot to do with the necessary memory for the process.
If the number of loci is L, the necessary amount of memory is $(L - m)2^{m+3}$ bytes.

That is to say that if $L = 300,000$ and $m = 10$ you will need 2.5 GB of memory.

The amount of necessary memory will *double* if m is *incremented by 1!*

usually $m \leq 4$.

But in our case we will make it run with bigger values of m.

These matrices of transitions are a markov chain of order m.
How should we choose m? First it has a lot to do with the necessary memory for the process.
If the number of loci is L, the necessary amount of memory is $(L-m)2^{m+3}$ bytes.

That is to say that if $L = 300,000$ and $m = 10$ you will need 2.5 GB of memory.

The amount of necessary memory will *double* if m is *incremented by 1!*

usually $m \leq 4$.

But in our case we will make it run with bigger values of m.

These matrices of transitions are a markov chain of order m.
How should we choose m? First it has a lot to do with the necessary memory for the process.
If the number of loci is L, the necessary amount of memory is $(L-m)2^{m+3}$ bytes.

That is to say that if $L = 300,000$ and $m = 10$ you will need 2.5 GB of memory.

The amount of necessary memory will *double* if m is *incremented by 1!*

usually $m \leq 4$.

But in our case we will make it run with bigger values of m.

1. Mapping problem
2. De Novo Sequencing (exact match)
   - The de Brujin graph
3. Usual technologies
   - Seed…
   - …& Grow!
4. Imputing maze lines
5. The Problem
6. Classical Resolution (Beagle)
7. Markov Chains
8. **Imputing**
9. Performances
10. An R package
11. Conclusion

At this point we may consider that for a given Ind, for every locus we have the probability that m consecutive letters are followed by a 0 or a 1.

This probability depends on the locus. It is a local property.

These probabilities spread through the whole sequence.

We compute $P(I, L) = 1$ for each locus of a given Ind, taking into account the spreading of the probabilities

Given a threshold, one can now impute the data itself.

For instance if threshold = 0.9, if $P(I, L) > 0.9$ we impute with a 1. If $P(I, L) < 1 - 0.9 = 0.1$, we impute with 0.

In between we do not impute and the unknown char remains unknown.

1. Mapping problem
2. De Novo Sequencing (exact match)
   - The de Brujin graph
3. Usual technologies
   - Seed…
   - …& Grow!
4. Imputing maze lines
5. The Problem
6. Classical Resolution (Beagle)
7. Markov Chains
8. Imputing
9. **Performances**
10. An R package
11. Conclusion

Known & Full VCF files consisted in 381 lines and 81,678 loci (chromosome 1).
From these files were created VCF files with missing data at rates (10%, 20%, . . . , 90%).
Then we imputed these files and compared results with Beagle.
We had several ways to remove known data:

1. choose Ind and locus at random
2. (coming soon): choose a number of untouched Ind, then choose some loci at random and erase all data for these loci for all Ind (except the first ones)

Time computation: (computations have been performed on average computers)

depending on the missing data rate, Beagle averagely needed between 2 and 3 hours to impute a chromosome

The same work was made by LineImputer in a time < 20” (Forward backward and Viterbi imputations both included).

Beware: these times are a rough guide only: softwares did not run on the same machines.

BUT:

Time computation: (computations have been performed on average computers)

depending on the missing data rate, Beagle averagely needed between 2 and 3 hours to impute a chromosome

The same work was made by LineImputer in a time < 20” (Forward backward and Viterbi imputations both included).

Beware: these times are a rough guide only: softwares did not run on the same machines.

BUT:

Time computation: (computations have been performed on average computers)

depending on the missing data rate, Beagle averagely needed between 2 and 3 hours to impute a chromosome

The same work was made by LineImputer in a time < 20” (Forward backward and Viterbi imputations both included).

Beware: these times are a rough guide only: softwares did not run on the same machines.

BUT:

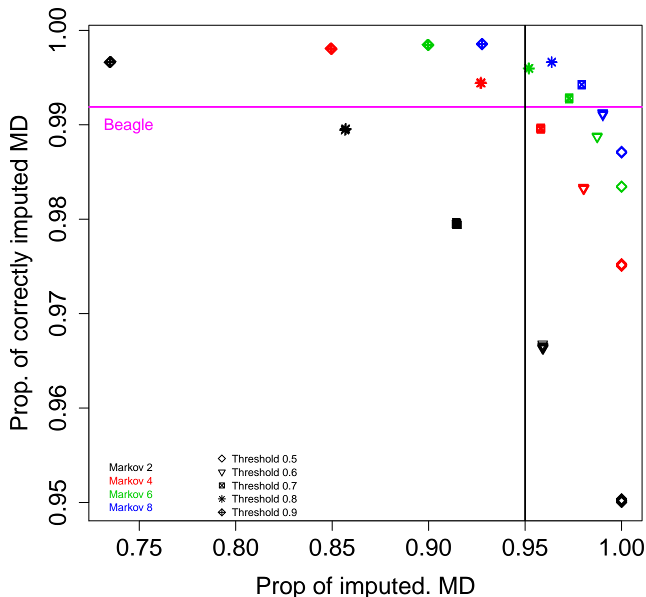
Time computation: (computations have been performed on average computers)

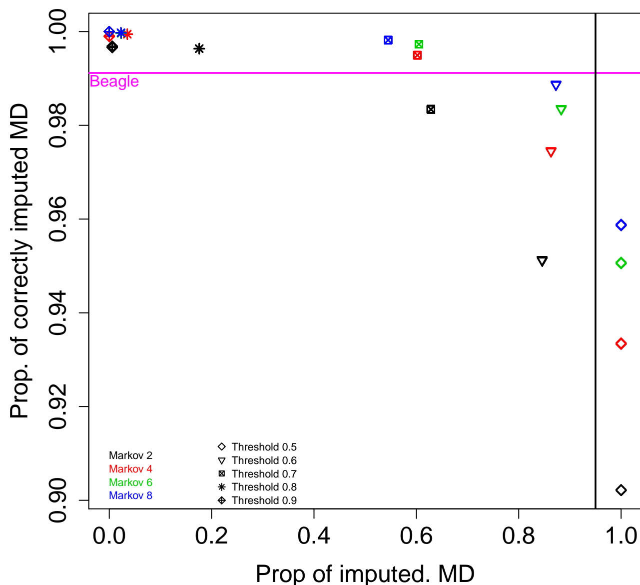
depending on the missing data rate, Beagle averagely needed between 2 and 3 hours to impute a chromosome

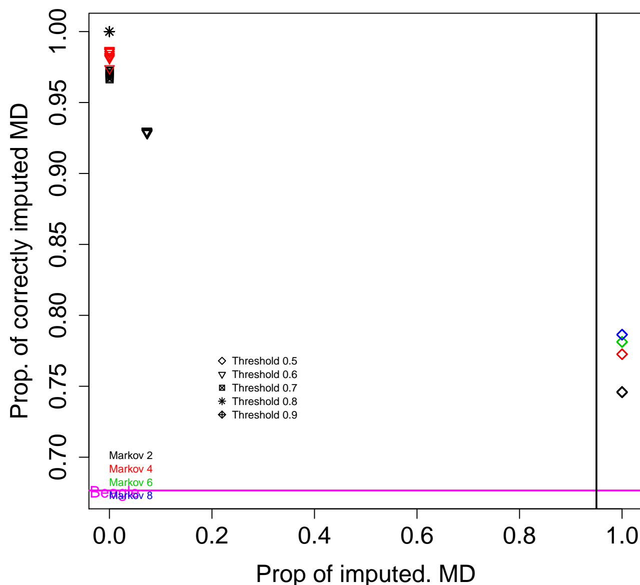
The same work was made by LineImputer in a time < 20” (Forward backward and Viterbi imputations both included).

Beware: these times are a rough guide only: softwares did not run on the same machines.

BUT:


Missing rate: 0.1
Prop. of correctly imputed MD
Prop of imputed. MD
Beagle
1.00
0.99
0.98
0.97
0.96
0.95
0.75
0.80
0.85
0.90
0.95
1.00
Markov 2
Markov 4
Markov 6
Markov 8
Threshold 0.5
Threshold 0.6
Threshold 0.7
Threshold 0.8
Threshold 0.9


Missing rate: 0.5
Beagle
Prop. of correctly imputed MD
Prop of imputed. MD
0.90
0.92
0.94
0.96
0.98
1.00
0.0
0.2
0.4
0.6
0.8
1.0
Markov 2
Markov 4
Markov 6
Markov 8
Threshold 0.5
Threshold 0.6
Threshold 0.7
Threshold 0.8
Threshold 0.9


Missing rate: 0.9
Prop. of correctly imputed MD
Prop of imputed. MD
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.0
0.2
0.4
0.6
0.8
1.0
Threshold 0.5
Threshold 0.6
Threshold 0.7
Threshold 0.8
Threshold 0.9
Markov 2
Markov 4
Markov 6
Markov 8
Beagle

1 Mapping problem
2 De Novo Sequencing (exact match)
   - The de Brujin graph
3 Usual technologies
   - Seed...
   - ...& Grow!
4 Imputing maze lines
5 The Problem
6 Classical Resolution (Beagle)
7 Markov Chains
8 Imputing
9 Performances
10 An R package
11 Conclusion

(Coming soon): an *R* package allowing one to

1. merge VCF files (linear time and space for 2 files)
2. Impute a VCF file. One chooses $2 \leq m \leq 12$ (the Markov chain order) and the Threshold. It produces a file containing probabilities and imputed files using forward/backward and/or Viterbi imputed files (linear time ans space complexity)

(Coming soon): an *R* package allowing one to

1. merge VCF files (linear time and space for 2 files)
2. Impute a VCF file. One chooses $2 \leq m \leq 12$ (the Markov chain order) and the Threshold. It produces a file containing probabilities and imputed files using forward/backward and/or Viterbi imputed files (linear time ans space complexity)

1 Mapping problem

2 De Novo Sequencing (exact match)
- The de Brujin graph

3 Usual technologies
- Seed. . .
- . . . & Grow!

4 Imputing maze lines

5 The Problem

6 Classical Resolution (Beagle)

7 Markov Chains

8 Imputing

9 Performances

10 An R package

11 Conclusion

LineImputer is (will soon be actually) an R package based on a Markov chain approach.

It allows one to merge VCF files.

It imputes provided the user specifies *m* and the threshold

It is very fast

It's accuracy may be good (depending on the choices of the Markov Chain Order and the Threshold) without outbesting Beagle, except for very high missing data rates

The quality of imputation decreases (possibly fast) when the threshold decreases towards 0.5, depending on the choice of *m*.

LineImputer is (will soon be actually) an R package based on a Markov chain approach.
It allows one to merge VCF files.
It imputes provided the user specifies m and the threshold
It is very fast
It's accuracy may be good (depending on the choices of the Markov Chain Order and the Threshold) without outbesting Beagle, except for very high missing data rates
The quality of imputation decreases (possibly fast) when the threshold decreases towards 0.5, depending on the choice of m.

LineImputer is (will soon be actually) an R package based on a Markov chain approach.
It allows one to merge VCF files.
It imputes provided the user specifies m and the threshold
It is very fast
It's accuracy may be good (depending on the choices of the Markov Chain Order and the Threshold) without outbesting Beagle, except for very high missing data rates
The quality of imputation decreases (possibly fast) when the threshold decreases towards 0.5, depending on the choice of m.

LineImputer is (will soon be actually) an R package based on a Markov chain approach.

It allows one to merge VCF files.

It imputes provided the user specifies m and the threshold

It is very fast

It's accuracy may be good (depending on the choices of the Markov Chain Order and the Threshold) without outbesting Beagle, except for very high missing data rates

The quality of imputation decreases (possibly fast) when the threshold decreases towards 0.5, depending on the choice of m.

LineImputer is (will soon be actually) an R package based on a Markov chain approach.

It allows one to merge VCF files.

It imputes provided the user specifies m and the threshold

It is very fast

It's accuracy may be good (depending on the choices of the Markov Chain Order and the Threshold) without outbesting Beagle, except for very high missing data rates

The quality of imputation decreases (possibly fast) when the threshold decreases towards 0.5, depending on the choice of m.

- LineImputer is (will soon be actually) an R package based on a Markov chain approach.
- It allows one to merge VCF files.
- It imputes provided the user specifies m and the threshold
- It is very fast
- It's accuracy may be good (depending on the choices of the Markov Chain Order and the Threshold) without outbesting Beagle, except for very high missing data rates
- The quality of imputation decreases (possibly fast) when the threshold decreases towards 0.5, depending on the choice of m.

Future work: try to improve accuracy, for instance by:

1. choose a value of m (quite big) and a big threshold
2. impute (this will necessitate a very smart management of hardware for big values of m, in particular the available memory)
3. use the output file as an input file
4. decrease m (do not touch the threshold)
5. if a lot remains to impute go back to step 2
6. else end the process

The goal would be to get comparable performances to the ones of Beagle but in much less time

Future work: try to improve accuracy, for instance by:

1. choose a value of m (quite big) and a big threshold
2. impute (this will necessitate a very smart management of hardware for big values of m, in particular the available memory)
3. use the output file as an input file
4. decrease m (do not touch the threshold)
5. if a lot remains to impute go back to step 2
6. else end the process

The goal would be to get comparable performances to the ones of Beagle but in much less time

Future work: try to improve accuracy, for instance by:

1. choose a value of m (quite big) and a big threshold
2. impute (this will necessitate a very smart management of hardware for big values of m, in particular the available memory)
3. use the output file as an input file
4. decrease m (do not touch the threshold)
5. if a lot remains to impute go back to step 2
6. else end the process

The goal would be to get comparable performances to the ones of Beagle but in much less time

Future work: try to improve accuracy, for instance by:

1. choose a value of m (quite big) and a big threshold
2. impute (this will necessitate a very smart management of hardware for big values of m, in particular the available memory)
3. use the output file as an input file
4. decrease m (do not touch the threshold)
5. if a lot remains to impute go back to step 2
6. else end the process

The goal would be to get comparable performances to the ones of Beagle but in much less time

Future work: try to improve accuracy, for instance by:

1. choose a value of m (quite big) and a big threshold
2. impute (this will necessitate a very smart management of hardware for big values of m, in particular the available memory)
3. use the output file as an input file
4. decrease m (do not touch the threshold)
5. if a lot remains to impute go back to step 2
6. else end the process

The goal would be to get comparable performances to the ones of Beagle but in much less time

Future work: try to improve accuracy, for instance by:

use Line Imputer to impute a vast majority of Data

and then use Beagle to impute the unimputed data

Future work: try to improve accuracy, for instance by:

simply use bigger values for m (take care of the use of memory in this case)

Thank you for your attention.
Questions?