TwoPaCo: An efficient algorithm to build the compacted de Bruijn graph from many complete genomes

Ilia Minkin[1], Son Pham[2], Paul Medvedev[1]

Pennsylvania State University[1]
Salk Institute for Biological Studies[2]

8th July 2016

Motivation

- More and more complete genomes
- Pan-genome: analysis within same species
- Mammalian-sized genomes are coming soon

Motivation

- More and more complete genomes
- Pan-genome: analysis within same species
- Mammalian-sized genomes are coming soon

Key question: what is a handy data structure to represent genomes?

Motivation

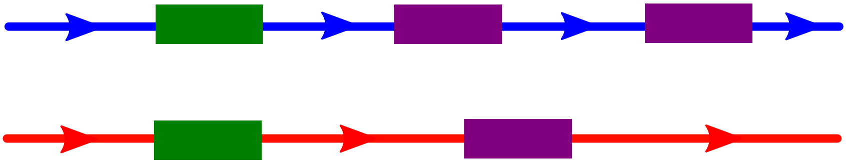
- More and more complete genomes
- Pan-genome: analysis within same species
- Mammalian-sized genomes are coming soon

Key question: what is a handy data structure to represent genomes?

The simplest way: string(s) of characters.

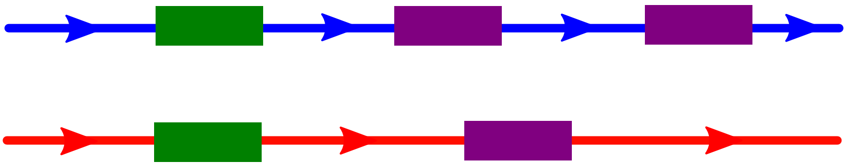
The Linear Representation

Two genomes:



The Linear Representation

Two genomes:

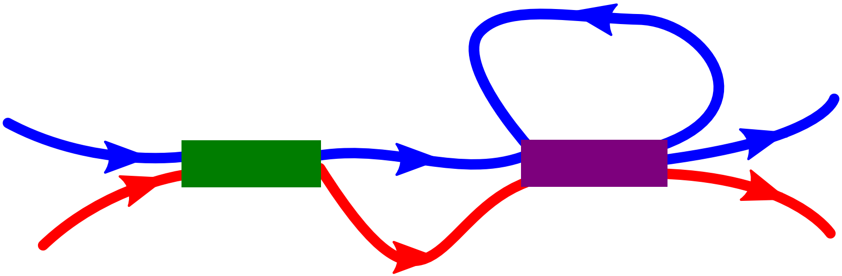


Issues:

- Homology between genomes?
- Duplications?
- Rearrangements?

Solution: a Graph Representation

What we want to see:



Why de Bruijn graph?

A simple object.

Demonstrated utility in:

- Assembly
- Read mapping
- Synteny identification

The de Bruijn Graph

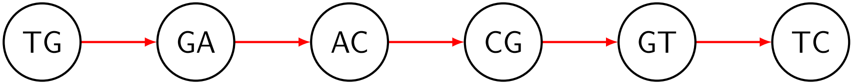
$k = 2$

TGACGTC TGACTTC

The de Bruijn Graph

k = 2

TGACGTC TGACTTC

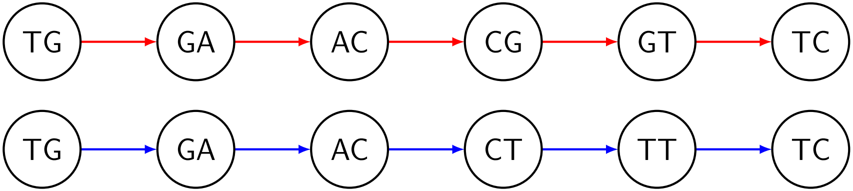

$$\begin{tikzcd} TG \arrow[r] & GA \arrow[r] & AC \arrow[r] & CG \arrow[r] & GT \arrow[r] & TC \end{tikzcd}$$

The de Bruijn Graph

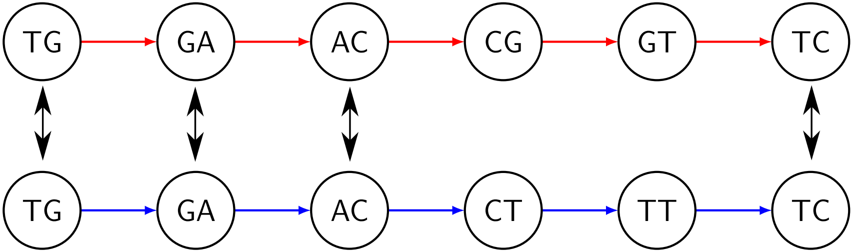
k = 2

TGACGTC

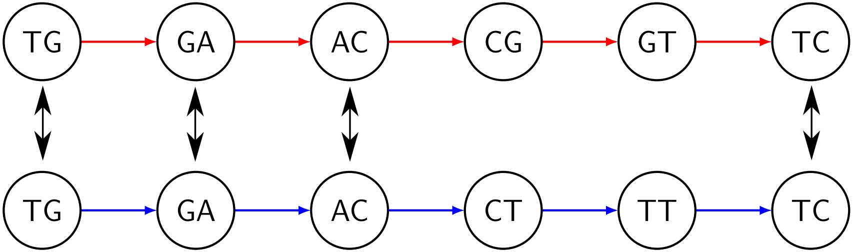
TGACTTC

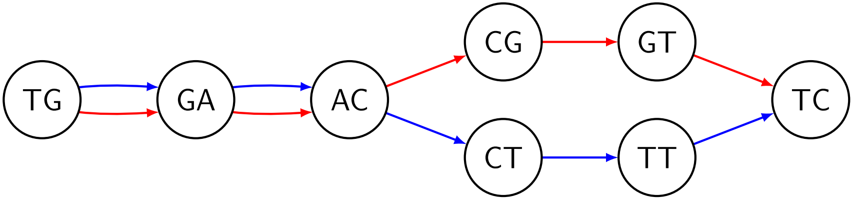


The de Bruijn Graph

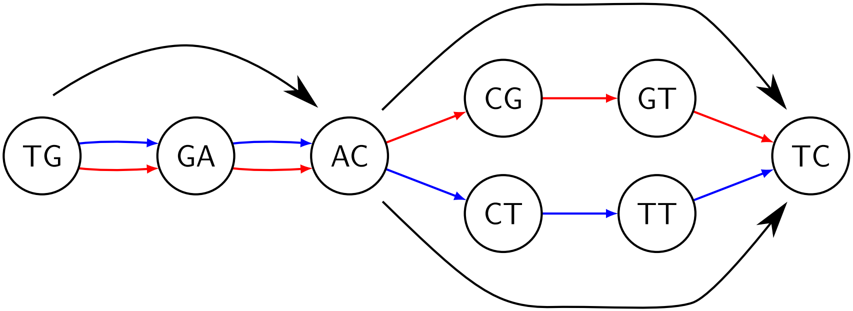


The de Bruijn Graph

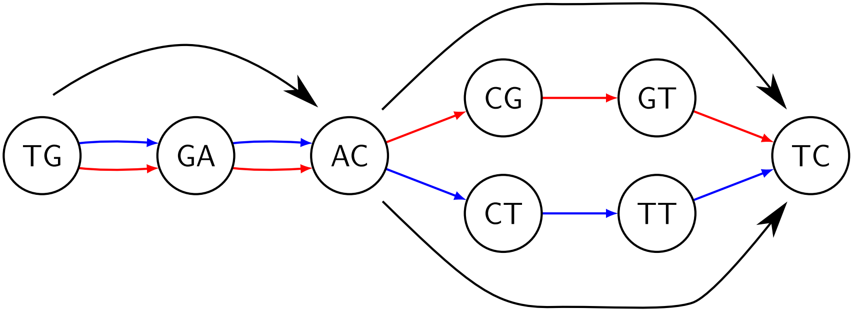




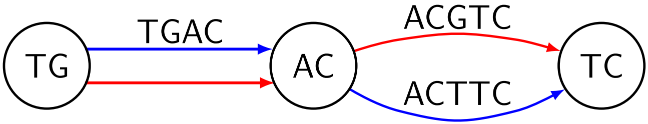
Compaction



Compaction



After compaction:



The Challenge

Construct the compacted graph from many large genomes **bypassing** the ordinary graph traverse.

The Challenge

Construct the compacted graph from many large genomes **bypassing** the ordinary graph traverse.

Earlier work: based on suffix arrays/trees Sibelia & SplitMEM handled > 60 E.Coli genomes.

The Challenge

Construct the compacted graph from many large genomes **bypassing** the ordinary graph traverse.

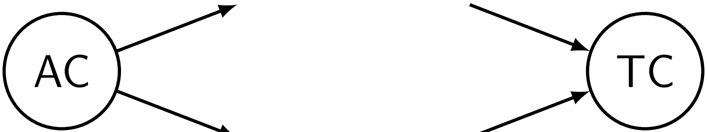
Earlier work: based on suffix arrays/trees Sibelia & SplitMEM handled > 60 E.Coli genomes.

A recent advance: 7 Humans in 15 hours using 100 GB of RAM using a BWT-based algorithm by Baier *et al.*, 2015, Beller *et al.*, 2014.

Junctions

A vertex v is a **junction** if:

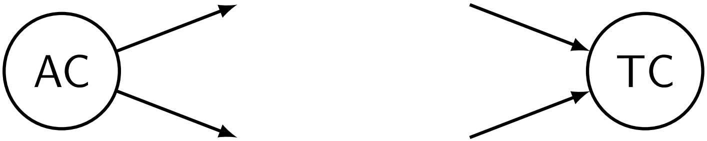
- v has ≥ 2 distinct outgoing or incoming edges:



Junctions

A vertex v is a **junction** if:

- v has ≥ 2 distinct outgoing or incoming edges:

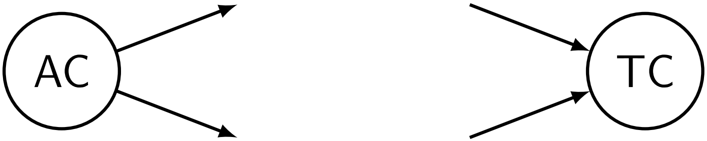


- v is the first or the last k-mer of an input string

Junctions

A vertex v is a **junction** if:

- v has ≥ 2 distinct outgoing or incoming edges:

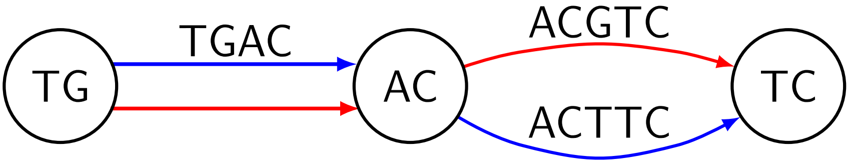


- v is the first or the last k-mer of an input string

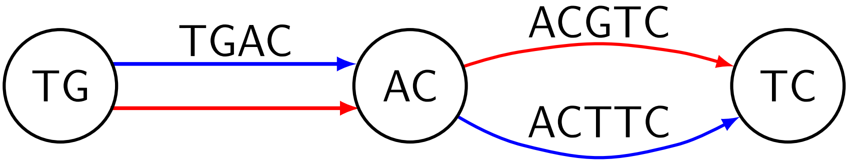
Facts:

- Junctions $=$ vertices of the compacted graph
- Compaction $=$ finding positions of junctions

Observations

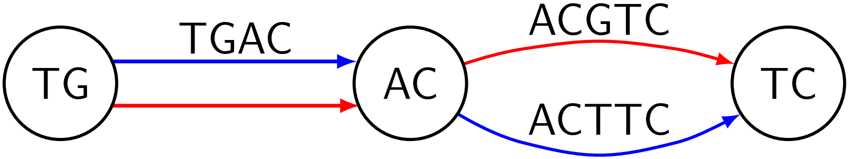


Observations



TG GA AC CG GT TC

Observations



TG GA AC CG GT TC

TG $\rightarrow$ AC $\rightarrow$ TC

The Observation

The observation only works when we have complete genomes.

Once we know junctions, construction of the edges is simple.

We can simply traverse input strings and record junctions in the order they appear.

How to identify junctions?

The Naive Algorithm

A naive way:

- Store all $(k+1)$-mers (edges) in a hash table
- Consider each vertex one by one
- Query all possible edges from the table
- If found > 1 edge, mark vertex as a junction

The Naive Algorithm

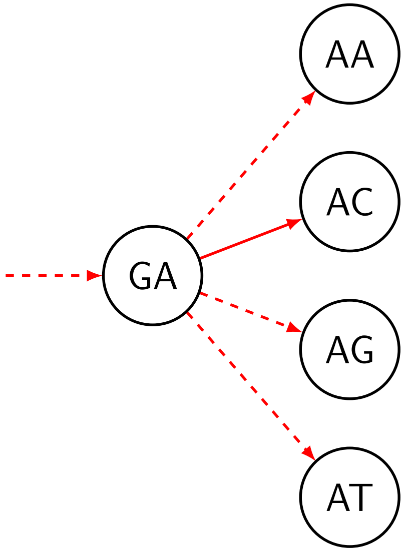
A naive way:

- Store all $(k+1)$-mers (edges) in a hash table
- Consider each vertex one by one
- Query all possible edges from the table
- If found > 1 edge, mark vertex as a junction

Problem: the hash table can be too large.

An Example

Hash table = { GA $\rightarrow$ AC }



What is the Bloom filter

A probabilistic data structure representing a set

Properties:

- Occupies fixed space
- May generate false positives on queries
- False positive rate is low

What is the Bloom filter

A probabilistic data structure representing a set

Properties:

- Occupies fixed space
- May generate false positives on queries
- False positive rate is low

Example: Bloom Filter $= \{$ GA $\rightarrow$ AC $\}$

Is GA $\rightarrow$ AC in the set? Yes.

What is the Bloom filter

A probabilistic data structure representing a set

Properties:

- Occupies fixed space
- May generate false positives on queries
- False positive rate is low

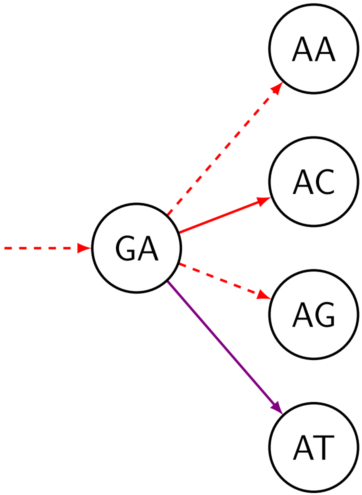
Example: Bloom Filter $=$ { GA $\rightarrow$ AC }

Is GA $\rightarrow$ AC in the set? Yes.

Is GA $\rightarrow$ AT in the set? **Maybe** no.

An Example

Bloom Filter = { GA → AC, GA → AT }



The purple edge is a false positive.

The Two Pass Algorithm

How to eliminate false positives?

The Two Pass Algorithm

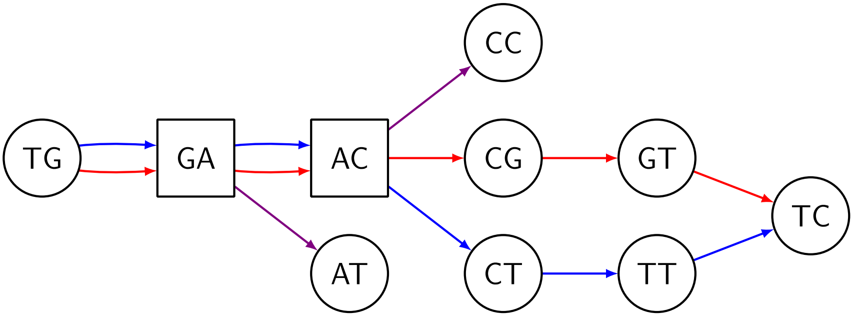
How to eliminate false positives?

Two-pass algorithm:

1. Use the Bloom filter to identify **junction candidates**
2. Use the hash table, but store **only edges that touch candidates**

An Example: the First Step

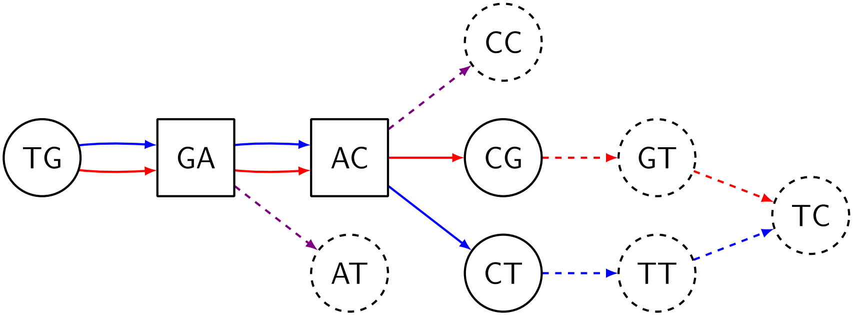
Here edges stored in the Bloom filter, purple ones are false positives:



Junction candidates: GA & AC

An Example: the Second Step

Edges stored in the hash table. We kept only edges touching junction candidates:



Junction: AC

Results

Datasets:

- 7 humans: 5 versions of the reference + 2 haplotypes of NA12878 from 1000 Genomes
- 93 simulated humans (FIGG)
- 8 primates available in UCSC genome browser

Results

Running time (minutes) & memory usage (GBs).

# genomes	BWT-based	TwoPaCo	
	1 thread	1 thread	15 threads
Humans			
7, $k = 25$	867 (100.30)	436 (4.40)	63 (4.84)
7, $k = 100$	807 (46.02)	317 (8.42)	57 (8.75)
43+7, $k = 25$	-	-	705 (69.77)
43+7, $k = 100$	-	-	927 (70.21)
93+7, $k = 25$	-	-	1383 (77.42)
Primates			
8, $k = 25$	-	914 (34.36)	111 (34.36)
8,$k = 100$	-	756 (56.06)	101 (61.68)

Conclusion & Future Work

Advantages of the algorithm:

- Fast
- Small memory footprint
- Can handle large inputs

Drawbacks:

- Less applicable for large k

Conclusion & Future Work

Advantages of the algorithm:

- Fast
- Small memory footprint
- Can handle large inputs

Drawbacks:

- Less applicable for large k

Take home message: it is easy to construct the compacted de Bruijn graph for complete genomes.

Conclusion & Future Work

Can potentially facilitate:

- Visualization
- Synteny mining (Sibelia)
- Structural variations analysis
- ...

Acknowledgments

Personal:

- Son Pham
- Daniel Lemire

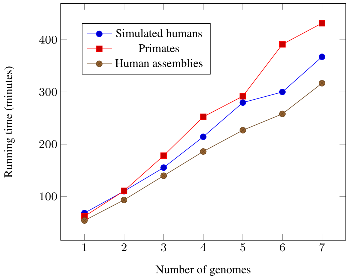
Funding, NSF awards:

- DBI-1356529
- CCF-1439057
- IIS-1453527
- IIS-1421908

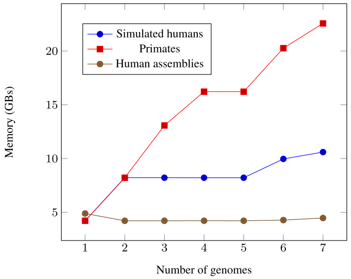
Thank you for your attention!

Input Size vs. Performance

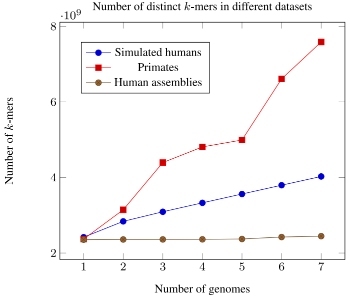
Running time



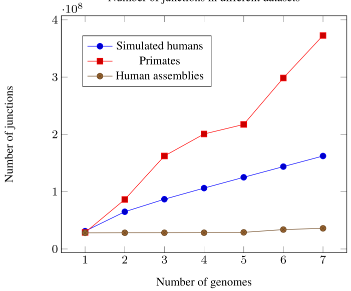
Maximum memory consumption



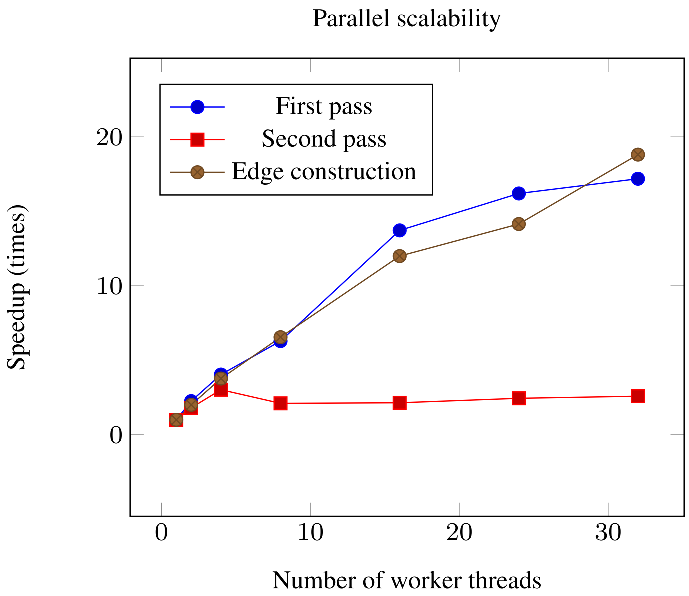
Number of distinct k-mers in different datasets



Number of junctions in different datasets



Parallel Scalability



Splitting

Table 1: The minimal number of rounds it takes to compress the graph without exceeding a given memory threshold.

Memory threshold	Used memory	Bloom filter size	Running time	Rounds
10	8.62	8.59	259	1
8	6.73	4.29	434	3
6	5.98	4.29	539	4
4	3.51	2.14	665	6