UNIVERSITÄT DES SAARLANDES

ZBI ZENTRUM FÜR BIOINFORMATIK

Genome Assembly

Algorithms for Sequence Analysis

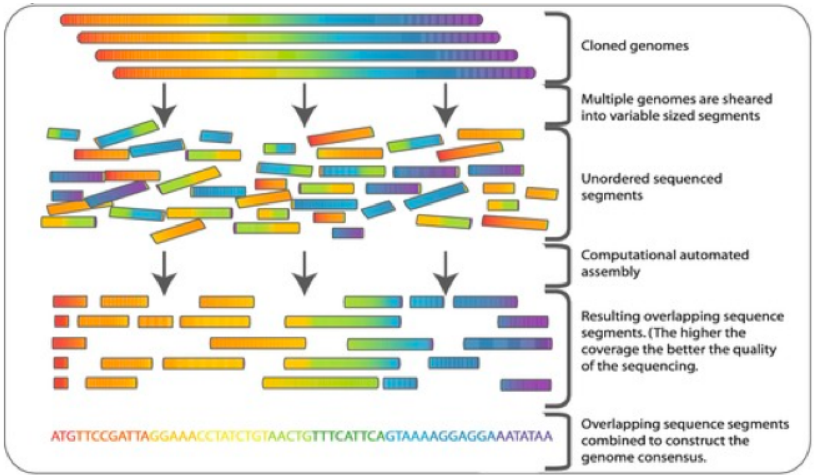
Sven Rahmann

Summer 2021

Overview

- Introduction to genome assembly
- The **de Bruijn graph** (DBG)
  - simplification
  - error correction in the graph
- Traversal of de Bruijn graphs
- Representations for de Bruijn graphs
  - hash table
  - bloom filters (inexact vs. exact)

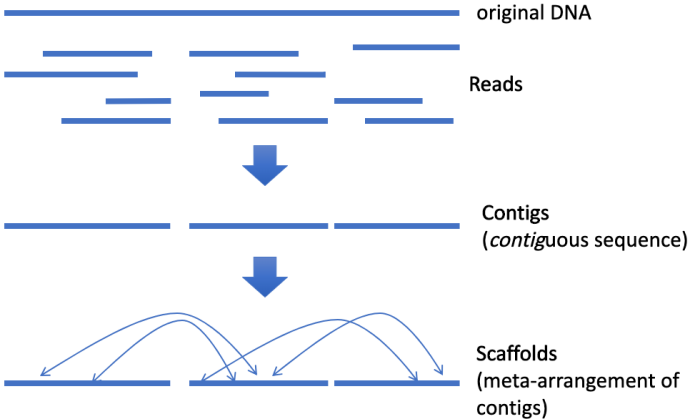
The current approach to genome assembly



Genome Assembly

Definition?

Assembly is reconstruction of (long) DNA fragments from sequencing reads.



Genome Assembly: Challenging to Define

Definition?

Assembly is reconstruction of (long) DNA fragments from sequencing reads.

Possible Criteria

- Reads should be approximate substrings of assembled fragments.

Genome Assembly: Challenging to Define

Definition?

Assembly is reconstruction of (long) DNA fragments from sequencing reads.

Possible Criteria

- Reads should be approximate substrings of assembled fragments.
- Assembly should be “short”, but not “overcompressed”.

Genome Assembly: Challenging to Define

Definition?

Assembly is reconstruction of (long) DNA fragments from sequencing reads.

Possible Criteria

- Reads should be approximate substrings of assembled fragments.
- Assembly should be "short", but not "overcompressed".
- Assembly should consists of few independent pieces.

Genome Assembly: Challenging to Define

Definition?

Assembly is reconstruction of (long) DNA fragments from sequencing reads.

Possible Criteria

- Reads should be approximate substrings of assembled fragments.
- Assembly should be "short", but not "overcompressed".
- Assembly should consists of few independent pieces.
- On the other hand, no arbitrary decisions should be made.

Two Main Approaches

Overlap Graphs

- **Nodes are reads.**
- Edges represent overlapping reads.
- Challenge: Pairwise comparison (overlap detection) of millions of reads

De Bruijn graph

- **Nodes are $k-1$-mers**.
- Edges are k-mers, connecting nodes with exact suffix-prefix overlap.

De Bruijn Graphs

Imagine two sequences:
TAGTCGAGGCTTTAGAGACAG
TAGTCGAGTCCGATAGAGACAG

De Bruijn Graphs

Imagine two sequences:
TAGTCGAGGCTTTAGAGACAG
TAGTCGAGTCCGATAGAGACAG

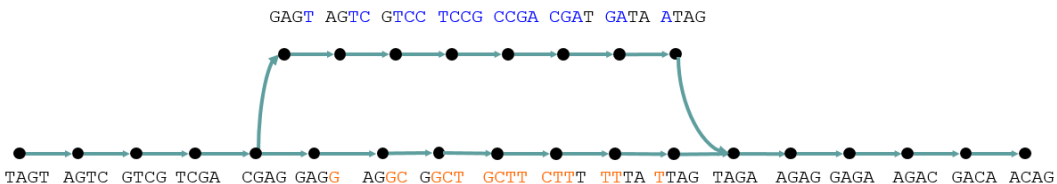
Reads generated from the sequences

AGTCGAG CTTTAGA CGATGAG CTTTAGA GTCGAGG
TTAGATC ATGAGGC GAGACAG GAGGCTC GTCCGAT
AGGCTTT GAGACAG AGTCGAG TAGATCC ATGAGGC
TAGAGAA TAGTCGA CTTTAGA CCGATGA TTAGAGA
CGAGGCT AGATCCG TGAGGCT AGAGACA TAGTCGA
GCTTTAG TCCGATG GCTTTAG TCGATTG GATCCGA
GAGGCTT AGAGACA TAGTCGA TTAGATC GATGAGG
TTTAGAG GTCGAGG TCTAGAT ATGAGGC TAGAGAC
AGGCTTT GTCCGAT AGGCTTT GAGACAG AGTCGAG

Definition: de Bruijn Graph

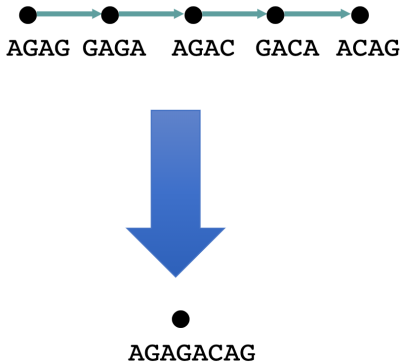
For a set of read s(strings) $R \subseteq \Sigma^* = \{A, C, G, T\}^*$ and a given parameter k,
let $T \subseteq \Sigma^k$ be the set of k-mers present in R as substrings.
The directed **de Bruijn graph** $G = (V, E)$ is defined by

- nodes: $V = T$,
- edges: $(v_i \to v_j) \in E$ iff $v_i[1:] = v_j[:k-1]$ (overlap by $k-1$ characters)



Collapsing the de Bruijn Graph

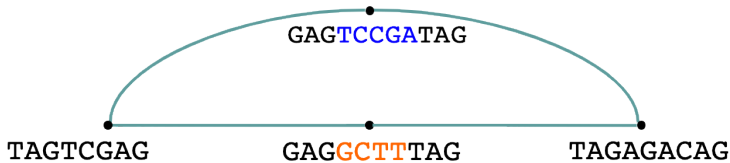
Linear chains of nodes hold redundant information.
For each edge $i \to j$ where node i has outdegree 1 and node j has indegree 1,
we can create a new combined node k and transfer the sequences of i and j to k.



Collapsing the de Bruijn graph

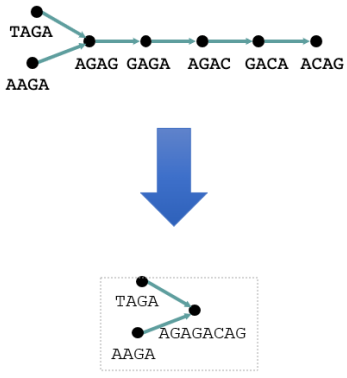
Reads from two sequences:
TAGTCGAGGCTTTAGAGACAG
TAGTCGAGTCCGATAGAGACAG

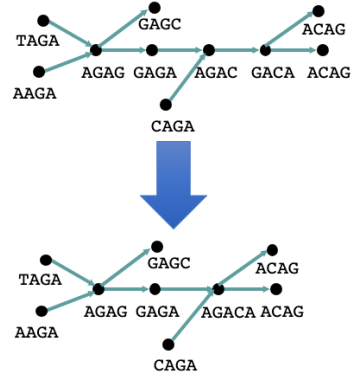
Graph simplified:



Collapsing the de Bruijn graph

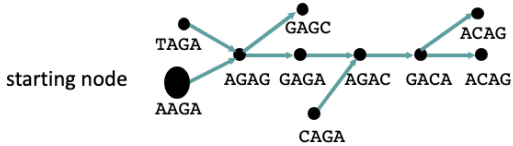
Which of these nodes can be merged?



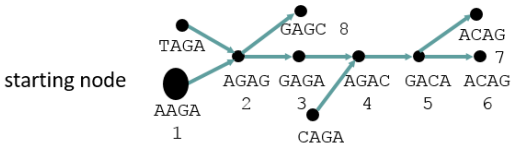


Graph traversals

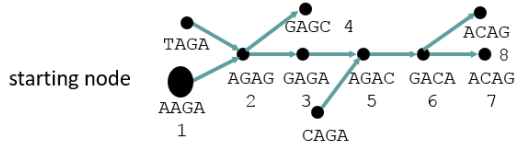
In which order are the nodes visited?



Depth-first search traversal



Breadth-first search traversal



Collapsing Linear Stretches

collapse

Input: Graph $G = (V, E)$
Output: Graph $G' = (V', E')$ with collapsed nodes

1. Identify the set of nodes `Starts` with:

$$\texttt{indegree}(n) = 0 \textbf{ or } \texttt{indegree}(n) > 1$$

$$\textbf{or } \big(\texttt{indegree}(n) = 1 \textbf{ and } \texttt{outdegree}(\texttt{prev}(n)) > 1\big)$$

2. For each node n in `Starts`:
   - **while** $\texttt{outdegree}(n) = 1$ **and** $\texttt{indegree}(\texttt{next}(n)) = 1$
     - $n \leftarrow \texttt{merge}(n, \texttt{next}(n))$

Simple Node-Based Assembly

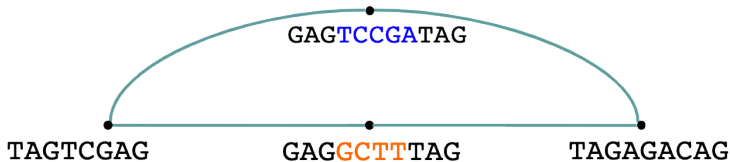
NodeBasedAssembly

Input: reads R, parameter k
Output: set of assembled sequences (unitigs)

1. $G \leftarrow \mathrm{DBG}(R, k)$ (De Bruijn graph)
2. $G' = (V', E') \leftarrow \texttt{collapse}(G)$
3. Return the set of sequences of nodes in V'

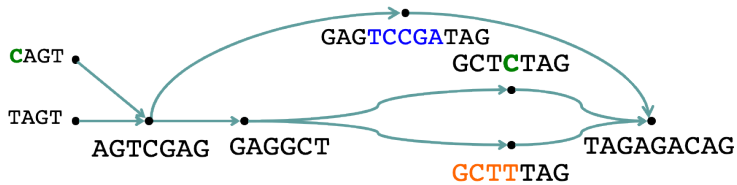
Collapsed de Bruijn Graph

Assembly results on the simplified graph
$S = \{\text{TAGTCGAG, GAGTCCGATAG, GAGGCTTTAG, TAGAGACAG}\}$



Sequencing Errors in the de Bruijn Graph

Graph with sequencing errors



Errors create two types of topologies in the graph:

- tips (CAGT node)
- bubbles (between GCTCTAG and GCTTTAG nodes)

Error Removal in Collapsed de Bruijn Graphs

Definition: Coverage

For a node $n \in V$, let $\text{cov}(n)$, be the number of times the $(k-1)$-mer n appears in R. If n is **simplified node**, then $\text{cov}(n)$ is the average count of all $(k-1)$-mers in n.

Coverage cutoff c

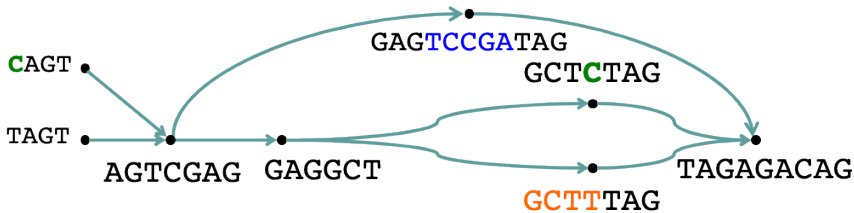
A node $n \in V$ is removed from the graph if $\text{cov}(n) < c$

The rationale is that nodes with such a low coverage are likely errors and as such can be removed to simplify the graph.

Error Removal in Collapsed de Bruijn Graphs

Tip clipping

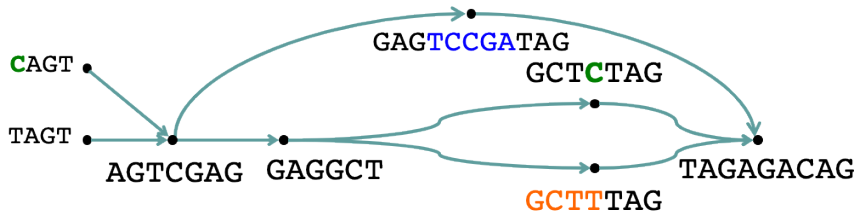
A node $n \in V$ is a **tip** if $\text{indegree}(n) = 0$ or $\text{outdegree}(n) = 0$ and $\text{length}(n) < 2k$.
The tip with smallest **coverage** is removed first.



Error Removal in Collapsed de Bruijn Graphs

Bubble removal

Consider bubbles in increasing order of coverage.
Align the sequences in the nodes of a bubble against each other.
If the sequences are similar, collapse bubble.



Simple Node-Based Assembly with Error Removal

NodeBasedAssembly

Input: reads R, parameter k, coverage cutoff c
Output: set of assembled sequences (contigs)

1. $G \leftarrow$ DBG build from R with parameter k
2. $G = (V, E) \leftarrow \texttt{collapse}(G)$
3. $G = (V, E) \leftarrow \texttt{remove\_tips}(G)$
4. $G = (V, E) \leftarrow \texttt{remove\_bubbles}(G)$
5. $G = (V, E) \leftarrow \texttt{remove\_low\_coverage\_nodes}(G, c)$
6. Return the set of sequences of nodes in V

Strandedness of DNA

- Sequencing removes (DNA strand) **orientation of reads**

antisense strand	TGGACTGAG
sense strand	ACCTGACTC

- Need to include **reverse complement** of each k-mer in a read
- Odd k-mers cannot make palindromes:

TATA	TATAT
ATAT	ATATA
$k = 4$	$k = 5$

- Implementations often store k-mer and its reverse complement as one:
  $\rightarrow$ select one **canonical k-mer**, i.e. the lexicographically smaller one

Representation of de Bruijn Graphs

Explicit data structures

- represent node as an object

Node	
Array of Pointers	next_nodes
String	sequence
Array of Pointers	previous_nodes

- Each node takes $16 + 16$ bytes $+ 2 \cdot (k - 1)$ bits (binary DNA encoding)

Representation of de Bruijn Graphs

Implicit Data Structures

- Any data structure that answers k-mer existence queries can be used
- Example k=3

AAC		CGA
CAC	ACG	CGC
GAC		CGG
TAC		CGT

possible extensions of ACG are 3-mers

Representation of de Bruijn Graphs

Implicit Data Structures

- Any data structure that answers k-mer existence queries can be used
- Example k=3

AAC		CGA
CAC	ACG	CGC
GAC		CGG
TAC		CGT

possible extensions of ACG are 3-mers

Edge traversal with implicit de Bruijn graphs

Idea To find all neighbors of a node, just query all neighboring k-mers for existence.

Implicit Representation with Arrays

Complete bit array

- Store a bit array of size Σ^k
- Example $k = 4$

AAAA 0
AAAC 1
....
TTTG 1
TTTT 0

Σ^k	4^{10}	4^{18}	4^{21}
size in million bits	1.05	68719	4398047

Too large to store higher values of k!

Implicit Representation with Hash Tables

Simple hash table

- Use a hash function f to project k-mers to an array much smaller than Σ^k that records (key, value) pairs.
- The value can be used to store $cov(n)$
- Need to handle collisions

- Still potentially wasting memory if initial guess on size was bad
- Slow access times if many collisions

Bloom Filters

Bloom filter

- Use h hash functions $f_1, ..., f_h$ to project k-mers to a bit array B with m bits, where $m \ll \Sigma^k$
- initially $B_i = 0, \forall i \in \{0, ..m-1\}$
- add a k-mer by setting all positions of the h hash function to 1

after initialization

seq	f_1	f_2	f_3
AAAA	0	3	6
AAAC	4	1	2
....			
TTTG	0	3	6
TTTT	0	1	4

index	B
0	0
1	0
2	0
3	0
4	0
5	0
6	0

Bloom Filters

Bloom filter

- Use h hash functions $f_1, ..., f_h$ to project k-mers to a bit array B with m bits, where $m \ll \Sigma^k$
- initially $B_i = 0, \forall i \in \{0, ..m-1\}$
- add a k-mer by setting all positions of the h hash function to 1

after inserting AAAA

seq	f_1	f_2	f_3
AAAA	0	3	6
AAAC	4	1	2
....			
TTTG	0	3	6
TTTT	0	1	4

index	B
0	1
1	0
2	0
3	1
4	0
5	0
6	1

Bloom Filters

Bloom filter

- Use h hash functions $f_1, ..., f_h$ to project k-mers to a bit array B with m bits, where $m \ll \Sigma^k$
- initially $B_i = 0, \forall i \in \{0, ..m-1\}$
- add a k-mer by setting all positions of the h hash function to 1

after inserting AAAA,AAAC

seq	f_1	f_2	f_3
AAAA	0	3	6
AAAC	4	1	2
....			
TTTG	0	3	6
TTTT	0	1	4

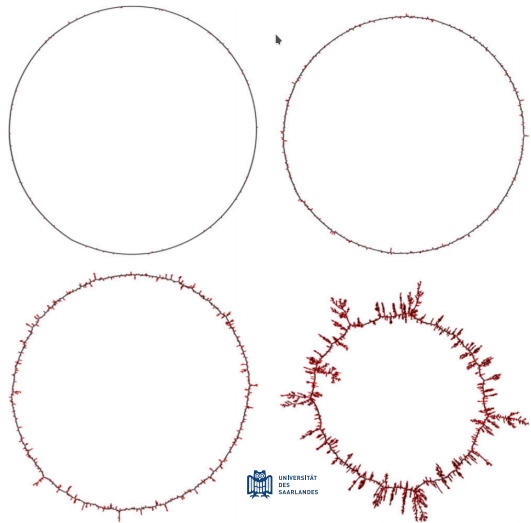
index	B
0	1
1	1
2	1
3	1
4	1
5	0
6	1

Querying Bloom Filter

- Query a k-mer by testing if bits at all h addresses are 1
- If all bits are set, k-mer **mau be present**
  - There can be **false positives**
  - Rate of false positives depends on load and h
- If there is at least one bit that is not set, then k-mer is **definitely not present**.

Effect of False Positives in Bloom Filters

Circle of 1000 random 31-mers, FPRs of 1%, 5%, 10%, 15%

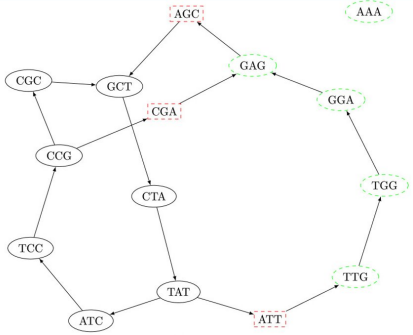


[Pell et al., PNAS, 2012]

Exact Bloom Filters for de Bruijn Graphs

Idea

- Critical false positives are direct neighbors of true positives
- Only the **critical FPs** are problematic in a graph traversal
- Store all critical false positives in extra data structure (e.g. simple set) that are encountered while traversal



- circles: true nodes
- squares: critical FPs
- dashed circles: other FPs

Chikhi and Rizk, WABI, 2012

Comparison of Represenations

	Pros	Cons
Explicit representation	■ node operations easy ■ traversals fast with collapsed nodes	■ memory intensive
Implicit representation	■ memory efficient	■ no collapsed nodes → runtime increases ■ algorithms more complicated

Assembly evaluation

Most Common Metric: N50

Definition

The largest contig length L, such that using contigs of length $\geq L$ accounts for at least 50% of the bases of the assembly.

Example:

1 Mbp genome
Contigs: 250k, 125k, 50k, 30k, 25k, 22k, 14k, 10k, ...
N50 size = 22kbp
(250k + 125k + 50k + 30k + 25k + 22k > 500kbp)

Important

Comparison using N50 values assumes that the base genome has the same size.

Other Metrics for Evaluation of Assembly Quality

- **Number of contigs:** The total number of contigs in the assembly.
- **Largest contig:** The length of the largest contig in the assembly.
- **Total length:** The total number of bases in the assembly.
- **NG50, Genome N50:** The contig length such that using equal or longer length contigs produces 50% of the length of the reference genome, rather than 50% of the assembly length.
- Software **Quast** can be used to compute these metrics for an assembly (http://quast.sourceforge.net/quast) Gurevich et al. Bioinformatics 2013

GAGE – Community Evaluation in 2012

3. Assemblies of Human chromosome 14 (ungapped size 88,289,540).

Assembler	Contigs				Scaffolds			
	Num	N50 (kb)	Errors	N50 corr. (kb)	Num	N50 (kb)	Errors	N50 corr. (kb)
ABySS	51,924	2.0	**704**	2.0	51,301	2.1	**9**	2
Allpaths-LG	4,529	36.5	2,760	21.0	**225**	**81,647**	45	**4,702**
Bambus2	13,592	5.9	11,943	4.3	1,792	324	143	161
CABOG	**3,361**	**45.3**	3,181	**23.7**	479	393	597	26
MSR-CA	30,103	4.9	5,550	4.3	1,425	893	1068	94
SGA	56,939	2.7	981	2.7	30,975	83	19	79
SOAPdenovo	22,689	14.7	6,424	7.4	13,502	455	268	214
Velvet	45,564	2.3	4,910	2.1	3,565	1,190	9156	27

Salzberg et al. 2012 Genome Research

Assembly Performance/Cost in 2020

Genome assembly	Data type (coverage, read N50 (kb))	Assembler	Size (Gb)	No. of contigs	Contig N50 (Mb)	Estimated cost (US$)	Ref.
HGP (2001 draft)	Multitechnology[a]	GigAssembler, PHRAP	2.69	149,821	0.082	300,000,000	72
GRCh38 (hg38)	Multitechnology[a]	Multiple algorithms	3.01	998	57.88	Not determined	160
YH	Illumina (56×, <0.075)	SOAPdenovo	2.91	361,157	0.02	1,600[b]	161
CHM13	PacBio CLR (77×, 17.5)	FALCON	2.88	1,916	29.30	2,700[c]	30
	PacBio HiFi (24×, 10.9)	FALCON	3.00	2,116	31.92	4,100[c]	52
		Canu	3.03	5,206	25.51		
	PacBio CLR (77×, 17.5) and ONT (50×, 70.4)	Canu	2.94	590	72.00	55,000[d]	34
HG002	PacBio HiFi (28×, 13.5)	FALCON	2.91	2,541	28.95	2,700[c]	53
	PacBio HiFi (28×, 13.5)	Canu	3.42	18,006	22.78		
	ONT (47×, 48.7)	Shasta	2.80	1,847	23.34	5,000[e]	36
		Flye	2.82	1,627	31.25		
		Canu	2.90	767	33.06		
NA12878	Illumina (103×, 0.101)	ALLPATHS-LG	2.79	231,194	0.02	2,900[b]	162
	ONT (29×, 10.6; 5×; 99.8)	Flye	2.82	782	18.18	4,000[e]	76
		Canu	2.82	798	10.41		35
NA12878 (phased)	PacBio HiFi (30×, 10.0)	Peregrine	2.97 (H1)	9,334 (H1)	19.6 (H1)	4,100[c]	22
			2.97 (H2)	9,127 (H2)	18.7 (H2)		
HG00733	ONT (73×, 29.6)	Shasta	2.78	2,150	24.43	6,000[d]	36
		Flye	2.81	1,852	28.76		
		Canu	2.90	778	44.76		

Sequence error correction before assembly

Why Sequence Error Correction?

- Errors create tips and bubbles in the graph
- Removing errors before graph construction can save runtime and avoid “tangles”

Correction of Sequencing Errors

- Suppose the genomic location of every read is known
  - Use the consensus from the multiple alignment to correct errors.

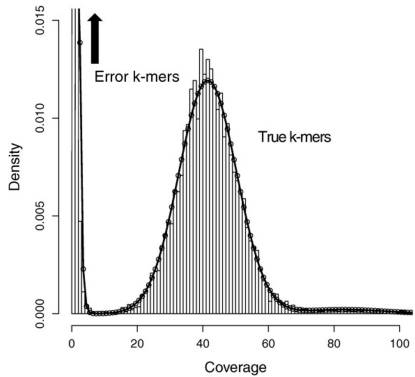
Reads:

ACAATT**C** CTTATTT ATTCCCA GTGGTAC CAATAT CT-AAA
ATACAAT TATCTTA CCATTCCC TGTGG**A**A GCAATAT T-AAA
TACAATT ATC-TAT CATTCCC**T** TGGTACG
ACAATTA A-TTCCA CCCATAT GG**A**ACGC TCCT**C**AAA
ACAATTA TCTTATT CCATTCC TGTGGTA AATATCC
TCTTATTT ATTCCCAT GTGGTACG

Genome:

ATACAATTATATCTTATTTCCATTCCCATATGTGGTACGCAATATCCT-AAA

Error k-mers Occur at Low Frequency



Kelley, Schatz, and Salzberg, Genome Biology, 2010

Density = normalized frequency of k-mers with certain coverage

Error Correction with Spectral Alignment

SpectralAlignment

Input: reads R, cutoff c

Output: corrected set of reads

1. Build hashtable H from R storing (k-mer, count) pairs
2. For each read r from R:
    1. For each index i:
        1. $v \leftarrow r[i, i+k]$
        2. if $H(v) < c$:
            $r[i, i+k] \leftarrow$ `BestHammingNeighbor(v,c,H)`

Choose Suitable Correction Candidate

Consider a set of 5 reads. Of which one contains a sequencing error
$R = \{r_1 = \text{TACCGA}, r_2 = \text{TACCGA}, r_3 = \text{TACCGA}, r_4 = \text{TACCGA}, r_5 = \text{TACTGA}\}$

TACTGA read r_5 contains 3 erroneous k-mers
TAC
ACT
CTG
TGA

k-mer	count
TAC	5
ACC	4
CCG	4
CGA	4
CTG	1
ACT	1
TAG	1

Choose Suitable Correction Candidate

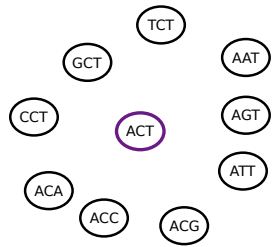
TACTGA read r_5 contains 3 erroneous k-mers

TAC

ACT

CTG

TGA



Possible Hamming neighbors for k-mer ACT

hashtabe H on R

k-mer	count
TAC	5
ACC	4
CCG	4
CGA	4
CTG	1
ACT	1
TAG	1

Choose Suitable Correction Candidate

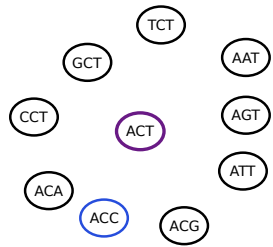
TACTGA read r_5 contains 3 erroneous k-mers

TAC

ACT

CTG

TGA



hashtabe H on R

k-mer	count
TAC	5
ACC	4
CCG	4
CGA	4
CTG	1
ACT	1
TAG	1

ACC is Hamming neighbor with highest count in H

Computing the Best Hamming Neighbor

BestHammingNeighbor

Input: k-mer v, cutoff c, hashtable H

Output: highest count Hamming neighbor of v above c

1. $bestSeq \leftarrow v$
2. $bestCount \leftarrow c - 1$
3. For each index i and each letter a from Σ:
    1. $v' \leftarrow$ replace letter i in v by a
    2. If $H(v') > bestCount$:
        $bestSeq \leftarrow v'$
        $bestCount \leftarrow H(v')$

Literature

- De Bruijn graph error correction
  - Velvet: Algorithms for de novo short read assembly using de Bruijn graphs, Zerbino and Birney 2008
- Bloom filters for de Bruijn graphs
  - Scaling metagenome sequence assembly with probabilistic de Bruijn graphs, Pell et al. 2012
- Exact bloom filter for de Bruijn graphs
  - Space-efficient and exact de Bruijn graph representation based on a Bloom filter, Chikhi and Rizk 2013

Summary

- The de Bruijn graph
  - simplification
  - error correction in the graph
- Representations for de Bruijn graphs
  - hash table
  - bloom filters
    - inexact bloom filters
    - exact bloom filters
- Traversal of de Bruijn graphs

Possible exam questions

- What are the main approaches to genome assembly?
- Define a de Bruijn graph for genome assembly.
- Construct a de Bruijn graph for a given example.
- What is the effect of sequencing errors on a DBG?
- Explain strategies to remove such errors.
- Mention different representations for de Bruijn graphs.
- Which representation of a DBG is the most space-efficient?
- Why can Bloom filters have false positives?
- What is a critical false positive?