UNIVERSITÄT
DES
SAARLANDES

Genome Assembly (Part I and II)

Algorithms for Sequence Analysis

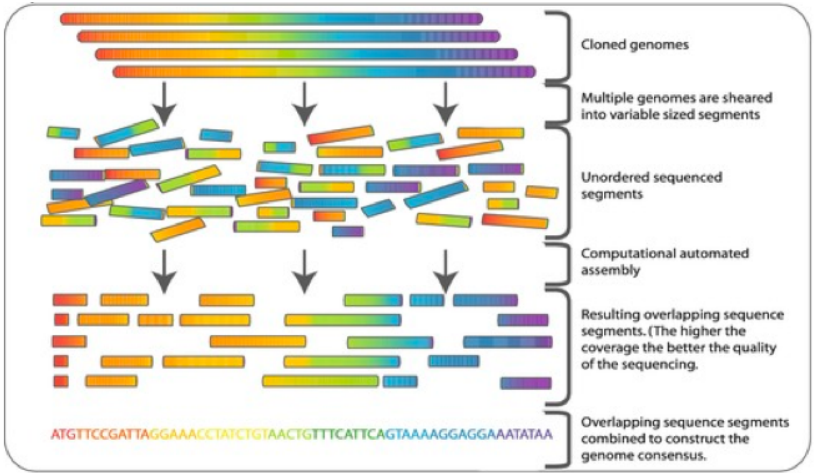
Sven Rahmann

Summer 2021

Overview

- Introduction to genome assembly, repeat problem
- The **overlap-layout-consensus** (OLC) approach
  - definition of overlap graph
  - computation (many pairwise overlaps); reduction
  - overlap: Hamitonian path problem
- The **de Bruijn graph** (DBG) or k-mer approach
  - simplification
  - error correction in the graph
  - traversal of de Bruijn graphs
  - representations of de Bruijn graphs (k-mer sets):
    - hash tables
    - bloom filters (inexact vs. exact)
- Evaluation metrics for assemblies (e.g., N50)
- Error correction before graph construction

Genome Assembly

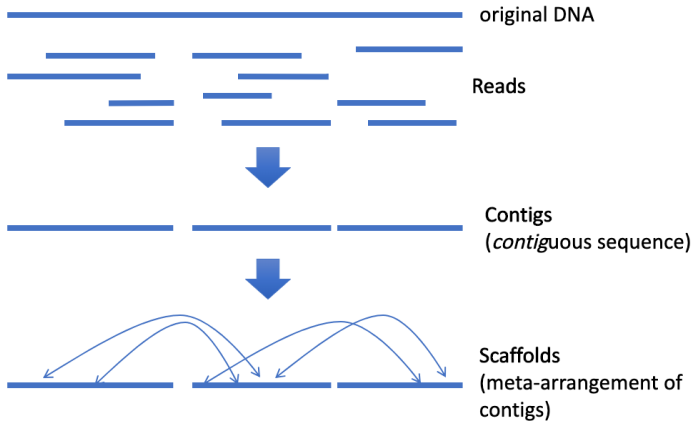


Meyerson et al., Nat Rev Genet. (2010).

Genome Assembly

Definition?

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



Genome Assembly: Challenging to Define the Objectives

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 the Objectives

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 the Objectives

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 consist of few independent pieces.

Genome Assembly: Challenging to Define the Objectives

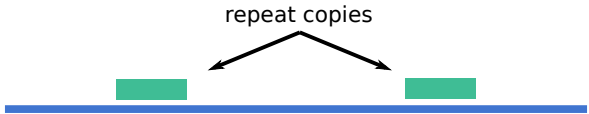
Definition?

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 consist of few independent pieces.
- On the other hand, no arbitrary decisions should be made.

Fundamental Problem for Assembly: Repeats



>50% of the human genome (by position) is some kind of repeat.

Human repeat classes (examples):

- short tandem repeat (ATATATATAT)
- SINE (about 300bp)
- LINE (about 7000bp)
- ribosomal DNA (rDNA): tandem repeat clusters with ∼43kb units
- gene families (duplicated throughout evolution)
- segmental duplications (>1000 bp and >90% identity)

Two Main Approaches

Overlap graphs

- **Nodes are reads.**
- Edges represent long overlaps between reads.
- Challenge: Pairwise comparison (overlap detection) of millions of reads
- Locality sensitive hashing may reduce number of pairs to compare

Two Main Approaches

Overlap graphs

- **Nodes are reads.**
- Edges represent long overlaps between reads.
- Challenge: Pairwise comparison (overlap detection) of millions of reads
- Locality sensitive hashing may reduce number of pairs to compare

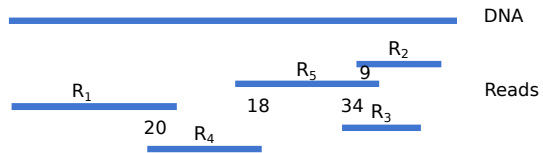
De Bruijn graphs (DBGs)

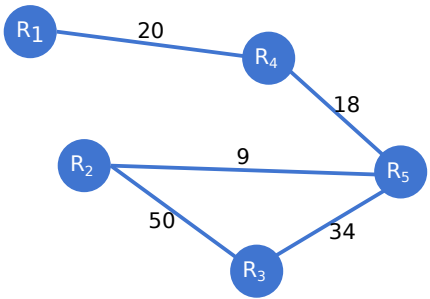
- DBG is a representation of the k-mer set of the reads.
- **Nodes are** $(k-1)$**-mers**.
- Edges are k-mers, connecting nodes with exact suffix-prefix overlap.

Overlap-Layout-Consensus Assembly

Overlap Graphs

nodes V: reads
edges E: overlap between reads
edge weights: length of overlap





Phase

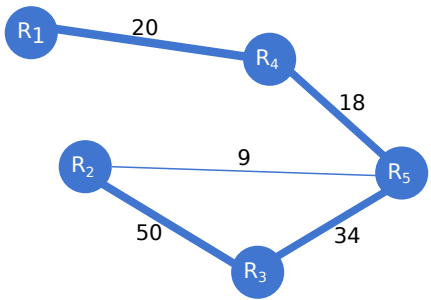
- Overlap
- Layout
- Consensus

Layout: Assembly as Hamiltonian Path Problem

Hamiltonian path

a path that visits each node exactly once

Reconstruct DNA by ordering nodes as a maximum weight Hamiltonian path (similar to traveling salesperson problem or TSP):

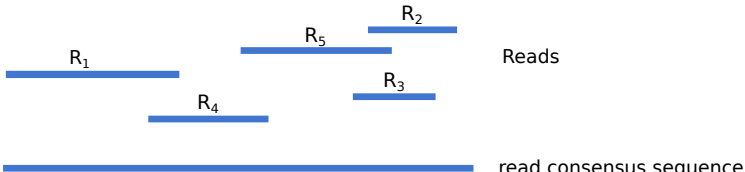


Phase

- Overlap
- Layout
- Consensus

Consensus

From ordered reads, form consensus sequence.



Phase

- Overlap
- Layout
- Consensus

Disadvantage of OLC Approach: Complexity

NP-hardness

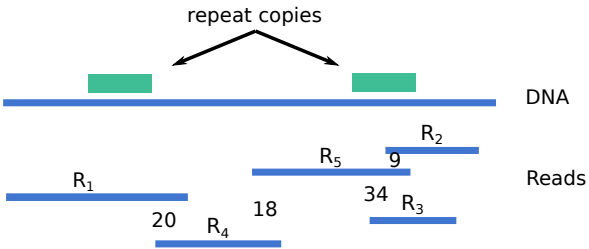
The Hamiltonian path (traveling salesman) optimization problem is NP-hard. Unless P=NP, no polynomial algorithm (in $|V| = n$: number of nodes) exists that guarantees to find an optimal solution.

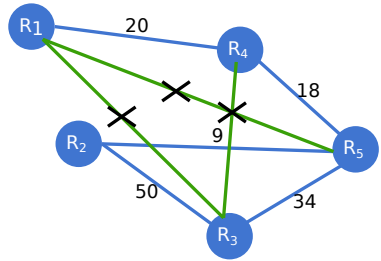
Number of possible paths: up to $(|V| - 1)!/2$.

Heuristic algorithms generate solutions for large instances of reasonable quality

number of reads	6	10	20	50
possible paths	360	181440	6.082×10^{16}	3.041×10^{62}

Effects of Repeats on OLC Approach





- Repeats increase amount of overlaps (i.e., high-weight edges)
- increased complexity of layout
- increased number of high-weight solutions
- many almost equivalent solutions

De Bruijn Graph Assembly

De Bruijn Graphs

Imagine two haplotype sequences with a difference:
TAGTCGAGGCTTTAGAGACAG
TAGTCGAGTCCGATAGAGACAG

De Bruijn Graphs

Imagine two haplotype sequences with a difference:
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 reads (strings) $R \subseteq \Sigma^* = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}^*$ and a given parameter k,
let $T_k \subseteq \Sigma^k$ be the set of k-mers present in R as substrings.
The directed **de Bruijn graph** $G = (V, E)$ is defined by

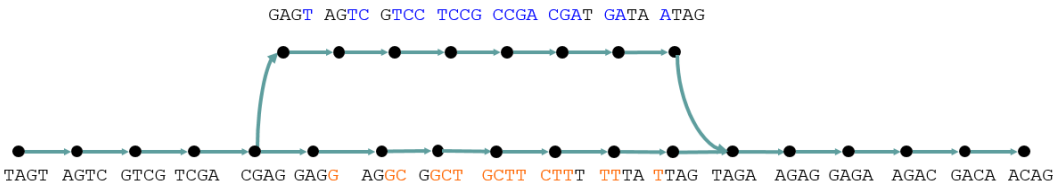
- nodes: $V = T_{k-1}$,
- edges: $E = T_k$,
  $(u \to v) \in E$ iff $u[1:] = v[:k-2]$ (overlap by $k-2$ characters)

Definition: de Bruijn Graph

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

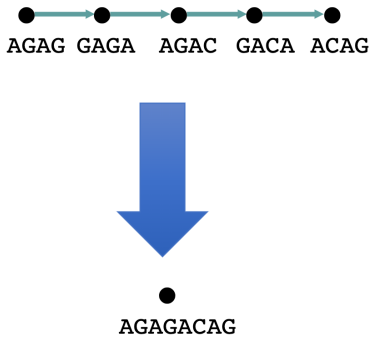
- nodes: $V = T_{k-1}$,
- edges: $E = T_k$,
  $(u \to v) \in E$ iff $u[1:] = v[:k-2]$ (overlap by $k-2$ characters)

Example with $k = 5$:



Collapsing the de Bruijn Graph

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

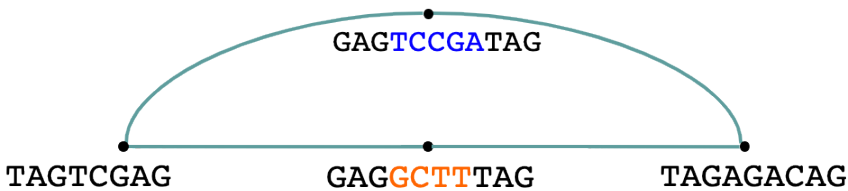


Collapsing the de Bruijn graph

Example: Two reads with a variation

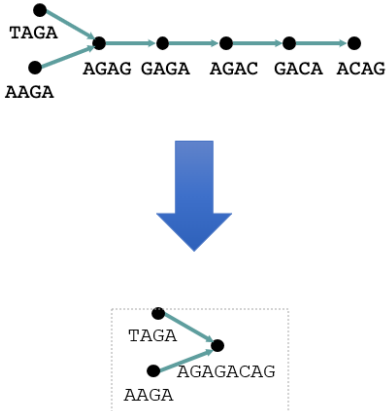
TAGTCGAGGCTTTAGAGACAG
TAGTCGAGTCCGATAGAGACAG

Simplified graph:



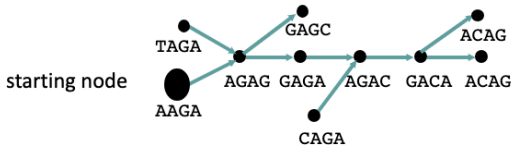
Collapsing the de Bruijn graph

Which of these nodes can be merged?

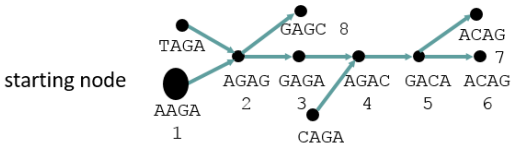


Graph Traversals

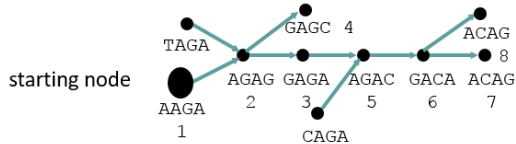
In which order are the nodes visited?



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

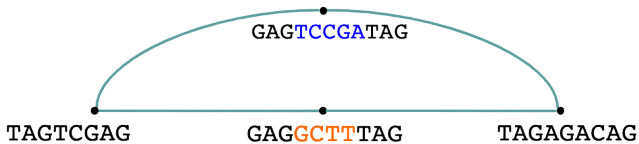
Input: reads R, parameter k

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' (called **unitigs**)

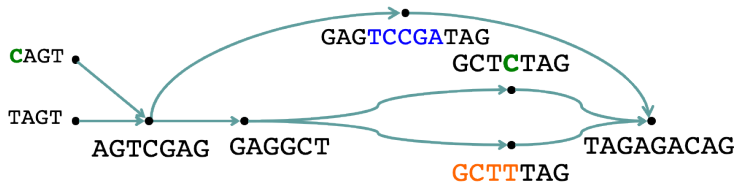
Example

$S = \{\mathrm{TAGTCGAG}, \mathrm{GAGTCCGATAG}, \mathrm{GAGGCTTTAG}, \mathrm{TAGAGACAG}\}$



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

Coverage cutoff c

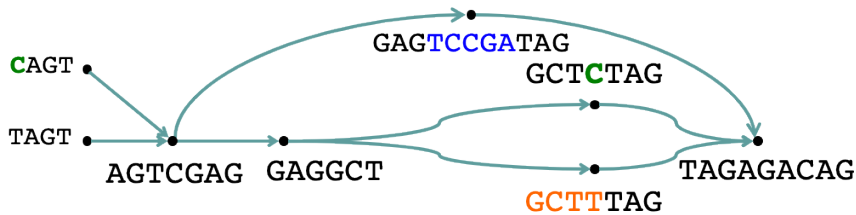
A node $v \in V$ is removed from the graph if $\text{cov}(v) < c$

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

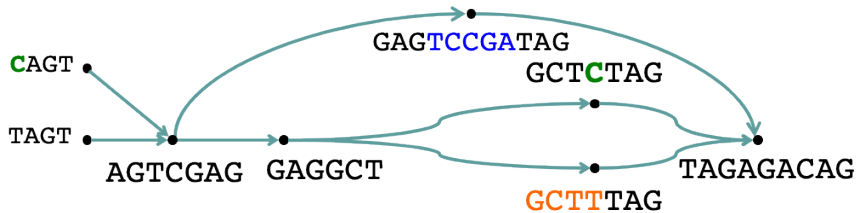
A node $v \in V$ is a **tip** if $\text{indegree}(v) = 0$ or $\text{outdegree}(v) = 0$ and $\text{length}(v) < 2(k-1)$.
The tip with smallest **coverage** is removed first.



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

Simple Node-Based Assembly with Error Removal

NodeBasedAssembly

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

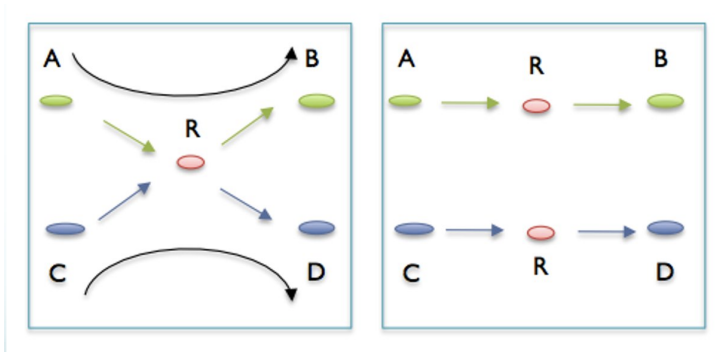
1. $G \leftarrow$ DBG build from R with parameter k
2. $G = (V, E) \leftarrow \texttt{collapse}(G)$
3. $G = (V, E) \leftarrow \texttt{remove\_tips}(G)$
4. $G = (V, E) \leftarrow \texttt{remove\_bubbles}(G)$
5. $G = (V, E) \leftarrow \texttt{remove\_low\_coverage\_nodes}(G, c)$
6. Return the set of sequences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

Next Steps: Connect contigs

- Expand short simple repeats
- Build scaffolds with paired-end information

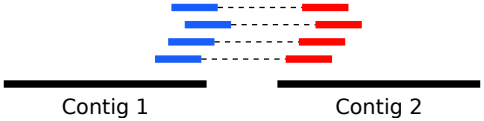
Removing Simple Repeats from de Bruijn Graphs

Reminder: Repeats are a fundamental problem for assembly.
Partial solution: Follow **original reads** along edges, split repetitive nodes (X-cut):



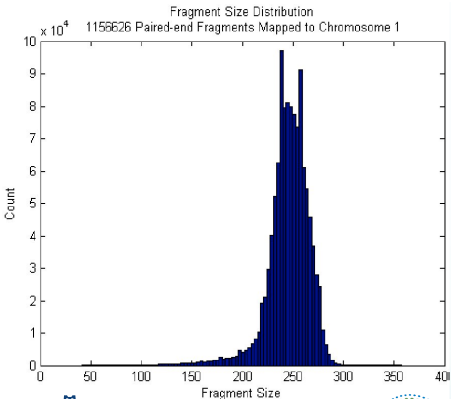
Pevzner et al. PNAS 2001

Scaffolding with Paired-End Reads



Example fragment length distribution

Fragments are not exactly the same length, but there's a clear peak around 250 nt, very few < 150 nt or > 300 nt



Technical Complication: Read Orientation

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

  antisense strand $\rightarrow$TGGACTGAG$\rightarrow$
  sense strand $\leftarrow$ACCTGACTC$\leftarrow$

- 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

- A DBG is in fact just a k-mer set (edge set).
  Nodes ($(k-1)$-mers) are implicitly defined by k-mer prefixes and suffixes.
- Any data structure that answers **set membership queries** can be used.
- Example for $k = 3$, possible nieghbors of ACG:

AAC		CGA
CAC	ACG	CGC
GAC		CGG
TAC		CGT

Representation of de Bruijn Graphs

Implicit Data Structures

- A DBG is in fact just a k-mer set (edge set).
  Nodes ($(k-1)$-mers) are implicitly defined by k-mer prefixes and suffixes.
- Any data structure that answers **set membership queries** can be used.
- Example for $k = 3$, possible nieghbors of ACG:

AAC		CGA
CAC	ACG	CGC
GAC		CGG
TAC		CGT

Edge traversal with implicit de Bruijn graphs

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

Exact and Probabilistic Set Membership Data Structures

Bit Arrays

Complete bit array

- Store a bit array of size $|\Sigma|^k$
- 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 for $k \geq 19$

Hash Tables

Simple hash table

- Use a hash function f to project k-mers to an array much smaller than $|\Sigma|^k$ 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

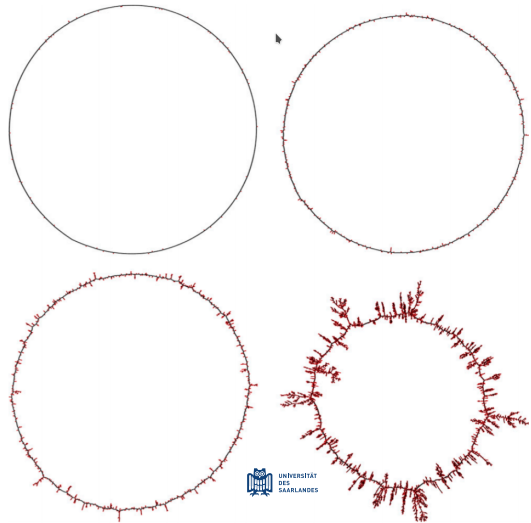
index	B
0	1
1	1
2	1
3	1
4	1
5	0
6	1

Querying a Bloom Filter

- Query a k-mer by testing if bits at all h addresses are 1
- If all bits are set, the k-mer **may 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%

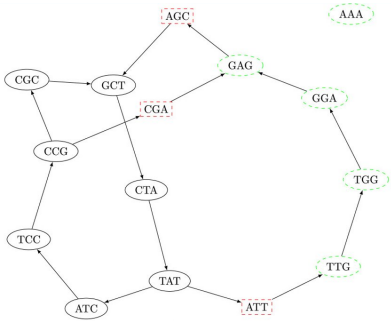


Pell et al., PNAS, 2012

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 an extra data structure (e.g. simple set).



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

Chikhi and Rizk, WABI, 2012

Assembly evaluation

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

Most Common Metric: N50

Definition

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

Example:

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

Important

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

Human Genome 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		
HG00733 (phased)	PacBio HiFi (33×, 13.4) and Strand-seq (5×)	Peregrine	2.90 (H1)	2,618 (H1)	28.0 (H1)	9,000[f]	91
			2.91 (H2)	2,557 (H2)	29.2 (H2)		

Logsdon, Vollger, and Eichler, Nature Reviews Genetics, June 2020

Sequence error correction before assembly

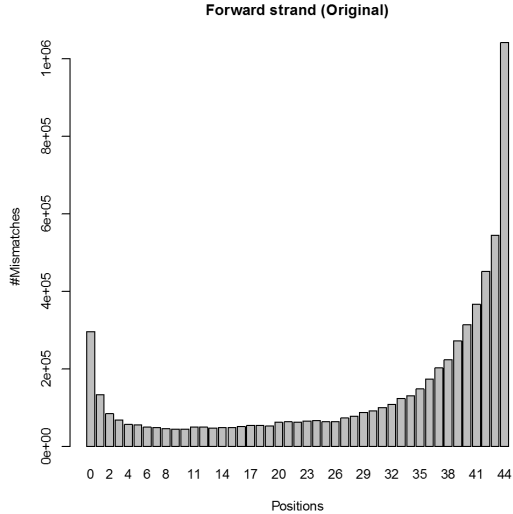
Correction of Sequencing Errors

Why?

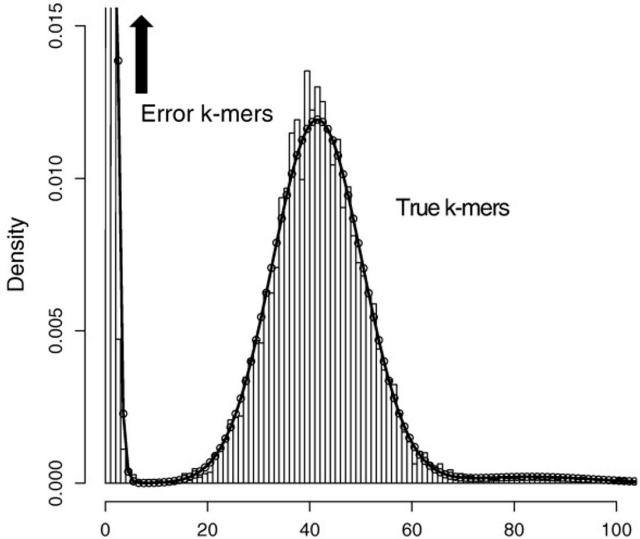
- Reads accumulate errors (esp. at the 3' end; see figure).
- Errors create tips and bubbles in the graph.
- Removing errors before DBG construction can save time and avoid "tangle".

How?

- Rare k-mers are proabbly not in from the genome, but errors.



Erroneous *k*-mers Occur at Low Frequency



Density:
normalized frequency of
k-mers with certain coverage

Source:
Kelley, Schatz, and Salzberg:
Genome Biology, 2010.

Error Correction from k-mer Spectrum (Spectral Alignment)

SpectralAlignment

Input: reads R, parameter k, 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 $H[v] < c$:
            $r[i \,..\, i+k] \leftarrow \texttt{BestHammingNeighbor}(v, c, H)$

Error Correction from k-mer Spectrum (Spectral Alignment)

SpectralAlignment

Input: reads R, parameter k, 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 $H[v] < c$:
            $r[i \,..\, i+k] \leftarrow \texttt{BestHammingNeighbor}(v, c, H)$

BestHammingNeighbor(v, c, H)

Return the Hamming-distance-1 neighbor of v with highest count above c.
(Found by evaluating all neighbors.)

Conclusion

Summary on Assembly Paradigms

Overlap-layout consensus paradigm (Hamilton path):

- Nodes are reads.
- Overlap between reads is represented as an overlap graph.
- Computing a maximum weight Hamilton path in the overlap graph produces a genome assembly (NP hard).
- Takes advantage of long reads.

Summary on Assembly Paradigms

Overlap-layout consensus paradigm (Hamilton path):

- Nodes are reads.
- Overlap between reads is represented as an overlap graph.
- Computing a maximum weight Hamilton path in the overlap graph produces a genome assembly (NP hard).
- Takes advantage of long reads.

de Bruijn graph paradigm (k-mers)

- Nodes are $(k-1)$-mers of reads.
- Edges are k-mers of reads.
- De Bruijn graph models exact overlap of k-mers in reads.
- Based on fast (probabilistic?) set membership data structures.

Summary

- Genome assembly paradigms: OLC, DBG
- Overlap graph
  - construction (pairwise overlaps)
  - Hamiltonian path problem (NP-hard)
- de Bruijn graph
  - definition
  - simplification
  - traversal
  - error correction
- Representations for de Bruijn graphs (k-mer sets)
  - bit array (huge)
  - hash table
  - Bloom filters (exact, inexact)

Possible exam questions

- What are the main approaches to genome assembly?
- Define the overlap graph.
- What is the problem when computing the overlap graph? How can it be reduced?
- What is a (maximum weight) Hamilton path? How can it be computed?
- Why is a Hamilton path in the overlap graph a layout for genome assembly?
- Define the 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 them.
- Mention different representations for de Bruijn graphs.
- Which representation of a DBG is the most space-efficient?
- What is a Bloom filter? Why can it give false positive answers to queries?
- What is a critical false positive?

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