Bioinformatics Algorithms

(Fundamental Algorithms, module 2)

Zsuzsanna Lipták

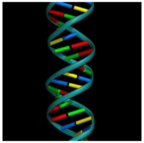
Masters in Medical Bioinformatics
academic year 2018/19, II. semester

Fragment Assembly with de Bruijn Graphs[1]

[1]These slides mainly based on Compeau, Pevzner, Tesler: *How to apply de Bruijn graphs to genome assembly*, Nature Biotechnology 29 (11).

Sequencing of a genome

From the DNA molecules (input of experiment) we want to get the sequence of the nucleotides (desired output).

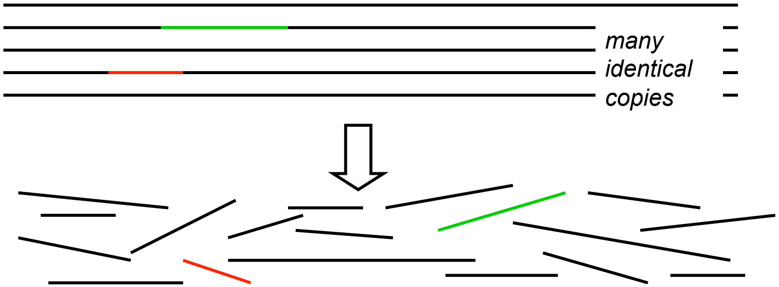


$\Rightarrow$

```
...AACAGTACCATGCTAGGTCAATCGA...
...TTGTCATGGTACGATCCAGTTAGCT...
```

Sequence assembly

Molecule (many identical copies) broken up into fragments.

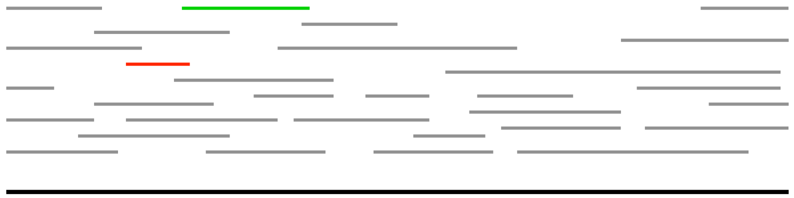


Sequence assembly

(also called Fragment Assembly Problem)

Input:

Many short sequences/strings (the fragments).



Goal:

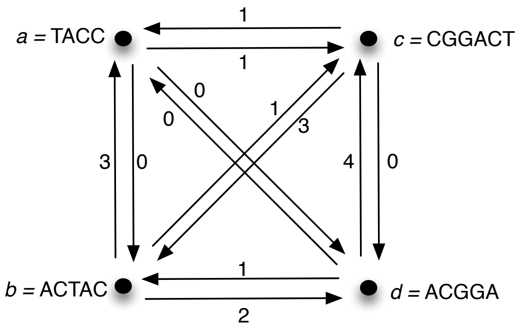
Reconstruct original string (the target sequence).

Overlap graph approach

(Recall from the first module of this course)

Previous approach (Sanger sequencing technology)

Shortest common superstring $\hat{=}$ a heaviest path in the overlap graph of $\mathcal{F} = \{\texttt{TACC}, \texttt{ACTAC}, \texttt{CGGACT}, \texttt{ACGGA}\}$ $\hat{=}$ a heaviest Hamiltonian path.



Sanger sequencing vs. short read sequencing (NGS)

NGS

Next generation sequencing technologies (Illumina, 454, SOLiD, ...) generate a much larger number of reads

- high-throughput: fast acquisition, low cost
- lower quality (more errors)
- short reads (Illumina: typically 60-100 bp)
- much higher number of reads

While overlap graph approach (with many additional details and modifications!) worked for Sanger type sequences, it no longer works for NGS data. Reason: Input too large, no efficient algorithms exist (efficient = polynomial time in input size), since SCS (and all other problem variants) are NP-hard.

Solution: Use Euler cycle/path approach

Solution:
Use Euler cycle/path in a de Bruijn graph (instead of heaviest Hamiltonian cycle/path in an overlap graph).

Solution: Use Euler cycle/path approach

Solution:

Use Euler cycle/path in a de Bruijn graph (instead of heaviest Hamiltonian cycle/path in an overlap graph).

Euler cycle/path vs. Hamiltonian cycle/path

- Hamiltonian cycle/path: uses every vertex exactly once
- Euler cycle/path: uses every edge exactly once

Solution: Use Euler cycle/path approach

Solution:

Use Euler cycle/path in a de Bruijn graph (instead of heaviest Hamiltonian cycle/path in an overlap graph).

Euler cycle/path vs. Hamiltonian cycle/path

- Hamiltonian cycle/path: uses every vertex exactly once
- Euler cycle/path: uses every edge exactly once

Fact

Finding an Euler cycle (or Euler path) can be solved in polynomial time.

Solution: Use Euler cycle/path approach

Solution:

Use Euler cycle/path in a de Bruijn graph (instead of heaviest Hamiltonian cycle/path in an overlap graph).

Euler cycle/path vs. Hamiltonian cycle/path

- Hamiltonian cycle/path: uses every vertex exactly once
- Euler cycle/path: uses every edge exactly once

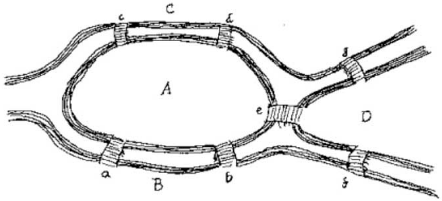
Fact

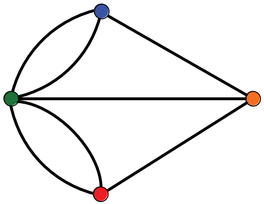
Finding an Euler cycle (or Euler path) can be solved in polynomial time.

But:

We have to find a way of modelling our problem in the right way.

Recall: Eulerian cycles and the bridges of Königsberg





Recall Euler cycle/path

Theorem

A directed graph has an Euler cycle (=Euler tour) if and only if it is connected and for all vertices v: $\mathit{indeg}(v) = \mathit{outdeg}(v)$ (i.e. all vertices are balanced). Such a graph is called Eulerian.

Theorem

A directed graph has an Euler path if and only if

- it is Eulerian, or
- it is connected, there are two vertices s, t, for which $\mathit{indeg}(s) = \mathit{outdeg}(s) - 1$ and $\mathit{indeg}(t) = \mathit{outdeg}(t) + 1$, and all other vertices are balanced.

Recall Euler cycle/path

Theorem

If G is Eulerian, then an Euler cycle can be found in time $O(|E|)$.

Proof

Use Hierholzer's algorithm:

- Start from any vertex v, go along so far untraversed edges. This is always possible, because every vertex is balanced.
- Eventually we get back to v (why?). Now if there are still untraversed edges, then there must be a vertex u in the cycle so far visited which has untraversed incident edges, since the graph is connected.
- Create a new cycle starting from u, unite the new cycle with the old one.
- Until no untraversed edges are left.

Note:

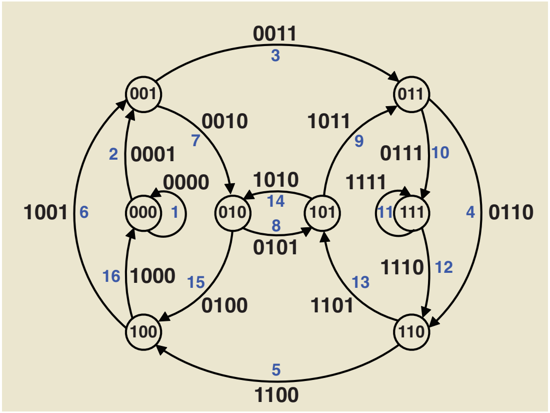
Similar for Eulerian path, start from s, will end up in t.

Application to the Fragment Assembly problem

We will use de Bruijn graph for modelling our problem:

- create a de Bruijn graph from the input fragments
- find an Eulerian path in this de Bruijn graph
- this Eulerian path will yield the desired string

De Bruijn graphs



The numbers give the order of the edges in an Eulerian cycle.— Named after Nicolaas de Bruijn, who introduced these graphs in 1946, for a different problem.

Definition of (full) de Bruijn graphs

Let Σ be our alphabet.
(E.g. $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ or $\Sigma = \{0, 1\}$ or $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$)

Definition[2]

The de Bruijn graph over Σ of order k is a directed graph $G = (V, E)$ s.t. $V = \Sigma^{k-1}$ and $(u, v) \in E$ if $u_2 \ldots u_{k-1} = v_1 \ldots v_{k-2}$.

(Equivalently: $(u, v) \in E$ if exists a word $w \in \Sigma^k$ s.t. u is the $(k-1)$-length prefix of w and v is the $(k-1)$-length suffix of w.)

N.B.

Note that $E = \Sigma^k$, and that the graph has loops (e.g. $(000, 000) \in E$).

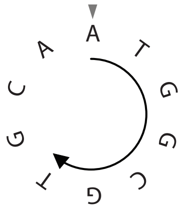
[2]Some people call these de Bruijn graphs of order $k-1$.

Modelling our problem with de Bruijn graphs

N.B.

For simplicity, for now our sequence to be reconstructed is assumed to be circular. E.g. bacterial genomes are circular.

a



String can be read as: `ATGGCGTGCA`, `TGGCGTGCAA`, `GGCGTGCAAT`, ...

Alternative definition of de Bruijn (sub)graphs

Let Σ be our alphabet.
(E.g. $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ or $\Sigma = \{0, 1\}$ or $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$)

Definition

A directed graph $G = (V, E)$ is called a de Bruijn (sub)graph of order k if $V \subseteq \Sigma^{k-1}$ and for all $u, v \in V$: if $(u, v) \in E$ then there exists a word $w \in \Sigma^k$ s.t. u is the $(k-1)$-length prefix of w and v is the $(k-1)$-length suffix of w.

Example

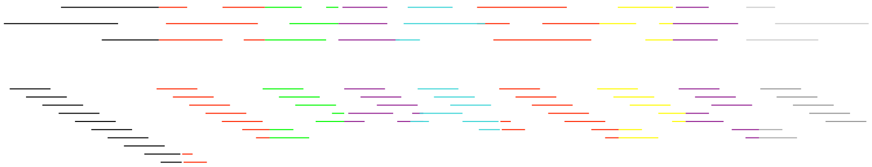
$u = \mathtt{GCA}, v = \mathtt{CAA}, w = \mathtt{GCAA}$.

N.B.

These are subgraphs of the original de Bruijn graph. Many researchers, esp. in bioinformatics call these graphs *de Bruijn graphs*. There exists also the version with multiple edges (multigraph, later).

Modelling our problem with de Bruijn graphs

Input: A collection $\mathcal{F}$ of strings.
First step: Generate all k-length substrings of fragments in $\mathcal{F}$.

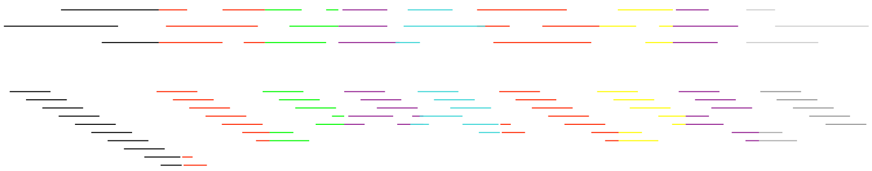


Example

$\mathcal{F} = \{\texttt{ATGGCGT}, \texttt{CAATGGC}, \texttt{CGTGCAA}, \texttt{GGCGTGC}, \texttt{TGCAATG}\}$.
For $k = 3$, we get:

Modelling our problem with de Bruijn graphs

Input: A collection $\mathcal{F}$ of strings.
First step: Generate all k-length substrings of fragments in $\mathcal{F}$.



Example

$\mathcal{F} = \{\texttt{ATGGCGT}, \texttt{CAATGGC}, \texttt{CGTGCAA}, \texttt{GGCGTGC}, \texttt{TGCAATG}\}$.
For $k = 3$, we get:
AAT, ATG, CAA, CGT, GCA, GCG, GGC, GTG, TGC, TGG.

Modelling our problem with de Bruijn graphs

Now from the k-mers, we generate the $(k-1)$-length prefixes and suffixes: `AA, AT, CA, CG, GC, GG, GT, TG`. These are the vertices. The edges are the k-mers.

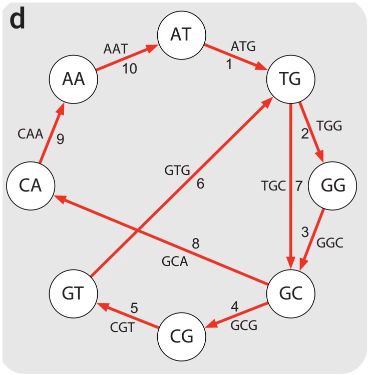
- $\mathcal{F} = \{\texttt{ATGGCGT}, \texttt{CAATGGC}, \texttt{CGTGCAA}, \texttt{GGCGTGC}, \texttt{TGCAATG}\}, k = 3$
- edges: `AAT`, `ATG`, `CAA`, `CGT`, `GCA`, `GCG`, `GGC`, `GTG`, `TGC`, `TGG`
- vertices: `AA`, `AT`, `CA`, `CG`, `GC`, `GG`, `GT`, `TG`

Modelling our problem with de Bruijn graphs

- edges: `AAT`, `ATG`, `CAA`, `CGT`, `GCA`, `GCG`, `GGC`, `GTG`, `TGC`, `TGG`
  (remember to only put an edge if the *k*-mer is present!)
- vertices: `AA`, `AT`, `CA`, `CG`, `GC`, `GG`, `GT`, `TG`

Modelling our problem with de Bruijn graphs

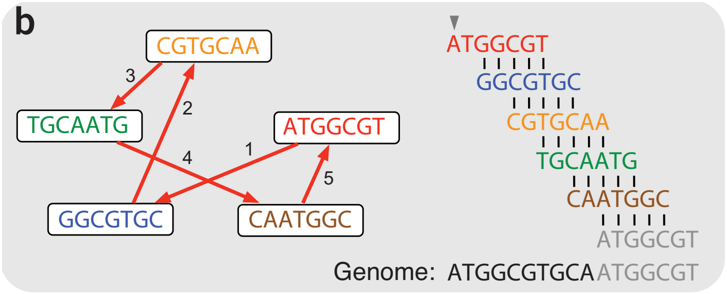
- edges: `AAT`, `ATG`, `CAA`, `CGT`, `GCA`, `GCG`, `GGC`, `GTG`, `TGC`, `TGG` (remember to only put an edge if the *k*-mer is present!)
- vertices: `AA`, `AT`, `CA`, `CG`, `GC`, `GG`, `GT`, `TG`



The numbers on the edges give an Eulerian cycle in this graph: `ATGGCGTGCA`

Comparison to other models

Compare to modelling the same problem with overlap graphs:
$\mathcal{F} = \{\texttt{ATGGCGT}, \texttt{CAATGGC}, \texttt{CGTGCAA}, \texttt{GGCGTGC}, \texttt{TGCAATG}\}$

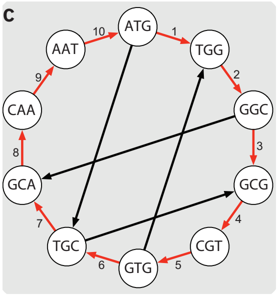


Note that not all non-zero weight edges are included in the figure. The numbers on the edges give a Hamiltonian cycle: `ATGGCGTGCA`.

Comparison to other models

Compare to modelling the same problem with overlap graphs using k-mers as nodes:

- $\mathcal{F} = \{\texttt{ATGGCGT}, \texttt{CAATGGC}, \texttt{CGTGCAA}, \texttt{GGCGTGC}, \texttt{TGCAATG}\}, k = 3$
- k-mers are nodes: `AAT`, `ATG`, `CAA`, `CGT`, `GCA`, `GCG`, `GGC`, `GTG`, `TGC`, `TGG`



Put an edge if the overlap equals $k - 1$. The numbers on the edges give a Hamiltonian cycle: `ATGGCGTGCA`.

Practical strategies for applying de Bruijn graphs: all *k*-mers

Generating nearly all *k*-mers

In reality, only a small fraction of all 100-mers (e.g.) are really sampled. Solution: Take shorter k than readlength. E.g. if reads have length approx. 100, then taking $k = 55$ will yield nearly all k-mers of the genome.

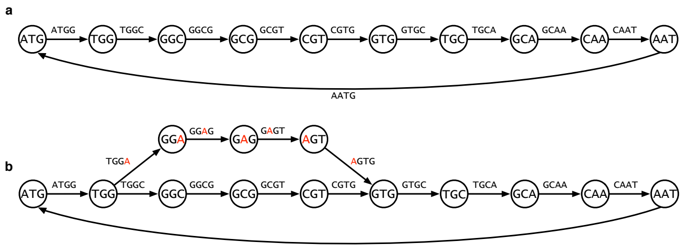
Ex.

In the example, not all 7-mers are present as reads, but all 3-mers are:

- genome: `ATGGCGTGCA`
- 7-mers: `ATGGCGT`, `CAATGGC`, `CGTGCAA`, `GGCGTGC`, `TGCAATG`
- 3-mers: `AAT`, `ATG`, `CAA`, `CGT`, `GCA`, `GCG`, `GGC`, `GTG`, `TGC`, `TGG`

Practical strategies for applying de Bruijn graphs: errors

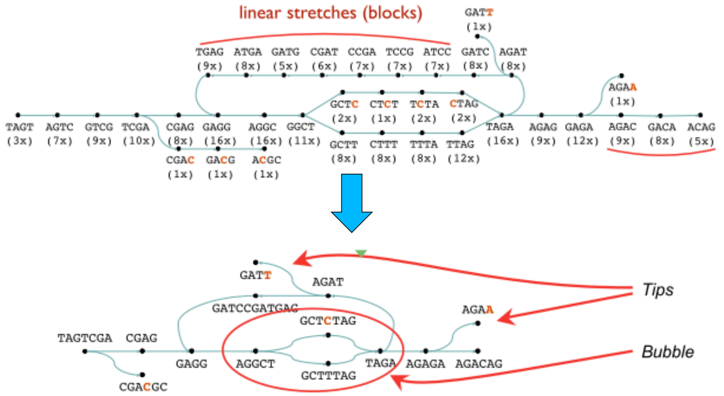
Errors is reads result in *bubbles* (= *bulges*) in the de Bruijn graph.



This can be detected and handled, using multiplicity of *k*-mers (multigraphs!), see next slide.

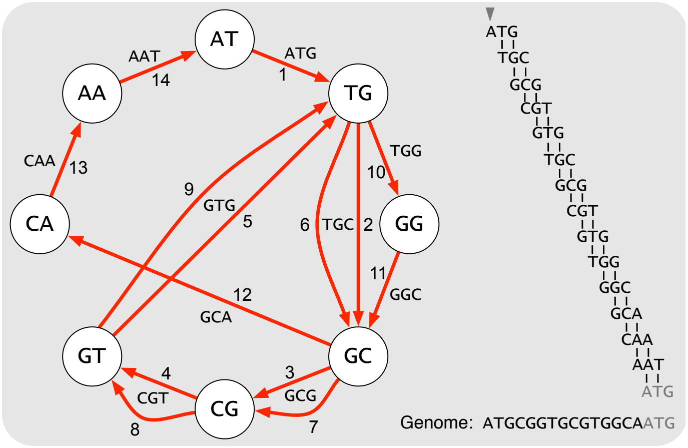
Practical strategies for applying de Bruijn graphs: errors

Errors is reads result in *bubbles* (= *bulges*) in the de Bruijn graph. This can be detected and handled via multiplicity of k-mers (multigraphs!) or of $(k-1)$-mers



E.g. the software Velvet (Zerbino and Birney, 2008) uses detection and elimination of bubbles and tips.

Practical strategies for applying de Bruijn graphs: repeats


AT
AA
TG
CA
GG
GT
GC
CG
AAT 14
ATG 1
CAA 13
TGG 10
9 GTG
5
6 TGC 2
11 GGC
12 GCA
4 CGT
3 GCG
8
7
ATG TGC GCG CGT GTG TGC GCG CGT GTG TGG GGC GCA CAA AAT ATG
Genome: ATGCGGTGCGTGGCAATG

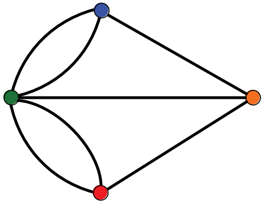
Repeats can be detected using multiplicity of *k*-mers (edges). Again, using multigraphs (edges have multiplicities).

Eulerian cycles in multigraphs

Theorem

A connected multigraph is Eulerian (has an Eulerian cycle) if and only if every vertex is balanced.

Now indegree = sum of multiplicities of incoming edges (= number of incoming edges counted with their multiplicities), outdegree defined similarly.



Recall the Bridges of Königsberg problem, that's a multigraph.

Sequencing By Hybridization

- Origin of de Bruijn graph approach to Fragment Assembly: Sequencing By Hybridization (SBH)
- suggested as alternative to SCS approach (Pevzner, 1988)
- DNA chip (DNA array) with all k-mers
- size 4^k
- entry (u, v) lights up if and only if uv is in the sample
- so we get a set (multiset?) of k-mers in the sample

Problems with Sequencing By Hybridization

SBH did not work because

- lack of fidelity of hybridization (mismatches!)
- array size: if longer k, better fidelity, but then array gets too big! (exponential in k)
  array size limited with current technology
- not practical (at present)
- But: it introduced the vastly successful approach of de Bruijn graphs to fragment assembly