Sequence Assembly

BMI/CS 576
www.biostat.wisc.edu/bmi576/
Mark Craven
craven@biostat.wisc.edu
Fall 2011

The sequencing problem

- We want to determine the identity of the base pairs that make up:
  - a single large molecule of DNA
  - the genome of a single cell/individual organism
  - the genome of a species
- But we can't (currently) "read" off the sequence of an entire molecule all at once

The strategy: substrings

- We *do* have the ability to read or detect *short* pieces (substrings) of DNA
  - Sanger sequencing: 500-800 bp/read
  - Latest technologies:
    - 454 Genome Sequencer FLX: 250-600 bp/read
    - Illumina Genome Analyzer: 35-150 bp/read

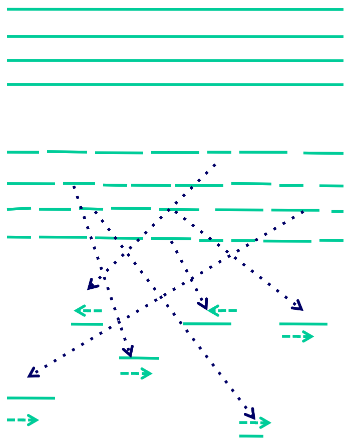
Shotgun sequencing fragment assembly

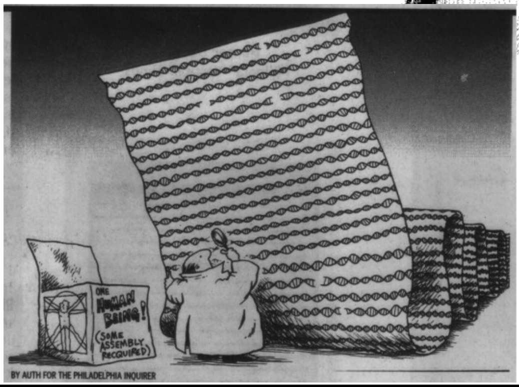
Multiple copies of sample DNA

Randomly fragment DNA

Sequence sample of fragments

Assemble reads




ONE HUMAN BEING! (SOME ASSEMBLY REQUIRED)
BY AUTH FOR THE PHILADELPHIA INQUIRER

Statistics for shotgun sequencing

- Given: G – genome length ($3 \cdot 10^9$ nts), L – read length (500 nts), N – number of reads (tbd)
- Calculate: coverage – $a=NL/G$
- Questions tbd by stats (Lander-Waterman):
  - How many contigs are there?
  - How big are the contigs?
  - How many reads are in each contig?
  - How big are the gaps?
- Requirement: 99% in contigs, 1% in gaps
  - $a=4.6$, $N=3 \cdot 10^7$, mean contig length 10^4, 100 reads/contig on average

The fragment assembly problem

- Given: A set of reads (strings) $\{s_1, s_2, \ldots , s_n\}$
- Do: Determine a large string s that "best explains" the reads

- What do we mean by "*best explains*"?
- What *assumptions* might we require?

Shortest superstring problem

- Objective: Find a string s such that
  - all reads $s_1, s_2, \ldots , s_n$ are substrings of s
  - s is as short as possible

- Assumptions:
  - Reads are 100% accurate
  - Identical reads must come from the same location on the genome
  - "best" = "simplest"

Shortest superstring example

- Given the reads:
  {ACG, CGA, CGC, CGT, GAC, GCG, GTA, TCG}
- What is the shortest superstring you can come up with?

  TCGACGCGTA (length 10)

Algorithms for shortest superstring problem

- This problem turns out to be *NP*-complete
- Simple *greedy* strategy:

```
while # strings > 1 do
    merge two strings with maximum overlap
loop
```

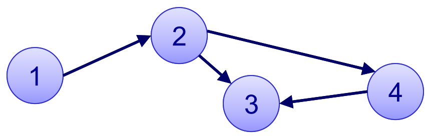
- Conjectured to give string with length $\leq 2 \times$ minimum length
- Other approaches are based on *graph theory…*

Graph basics

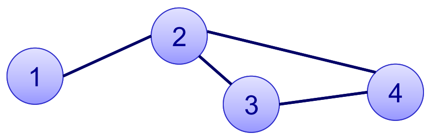
- a graph (*G*) consists of vertices (*V*) and edges (*E*)

  G = (V,E)

- edges can either be *directed* (*directed graphs)*

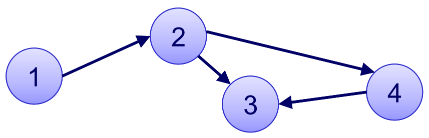


- or *undirected* (*undirected graphs)*



Vertex degrees

- the *degree* of a vertex: the # of edges incident to that vertex
- for directed graphs, we also have the notion of
  - *indegree:* The number incoming edges
  - *outdegree:* The number of outgoing edges



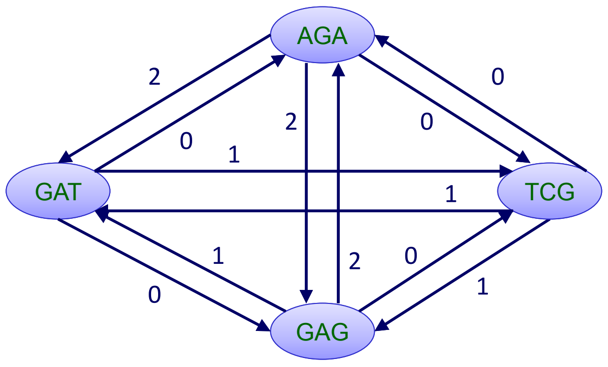
degree(v_2) *= 3*
indegree(v_2) *= 1*
outdegree(v_2) *= 2*

Overlap graph

- One representation that is commonly used for sequence assembly is an *overlap graph*
- For a set of sequence reads *S*, construct a directed weighted graph $G = (V,E,w)$
  - with one vertex per read (v_i corresponds *to* s_i)
  - edges between all vertices (a *complete* graph)
  - $w(v_i,v_j) = overlap(s_i, s_j)$ = length of longest suffix of s_i that is a prefix of s_j

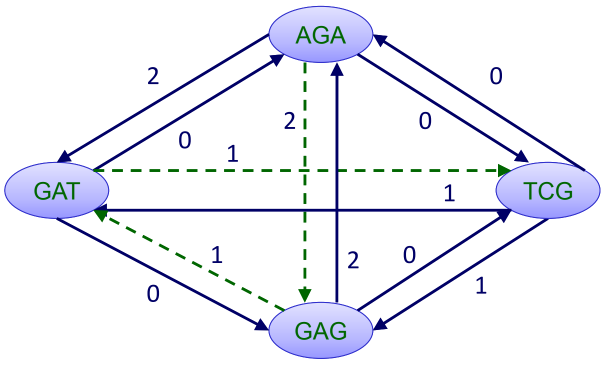
Overlap graph example

- Let *S* = {AGA, GAT, TCG, GAG}



Assembly as finding a Hamiltonian path

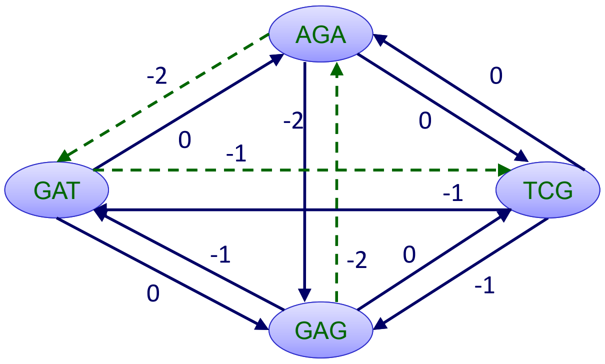
- *Hamiltonian path*: path through graph that visits each vertex exactly once



Path: AGAGATCG

Shortest superstring as TSP

- minimize superstring length ➔minimize weight of Hamiltonian path in overlap graph with edge weights negated



path: GAGATCG
path weight: -5
string length: 7

- this is essentially the Traveling Salesman Problem (also *NP*-complete)

Assembly as a Hamiltonian path

- finding Hamiltonian path is an NP-complete problem
- nevertheless overlap graphs are often used for sequence assembly
  - can detect repeats
  - heuristical hierarchical decomposition
    - unitigs (no forks, no conflicts) solved first
  - mate-pairs to scaffold

Sequencing by Hybridization (SBH)

- SBH array has probes for all possible *k*-mers
- For a given DNA sample, array tells us whether each *k*-mer is *PRESENT* or *ABSENT* in the sample
- the set of all *k*-mers present in a string *S* is called its *spectrum*

Example DNA array

S: ACTGATGCAT

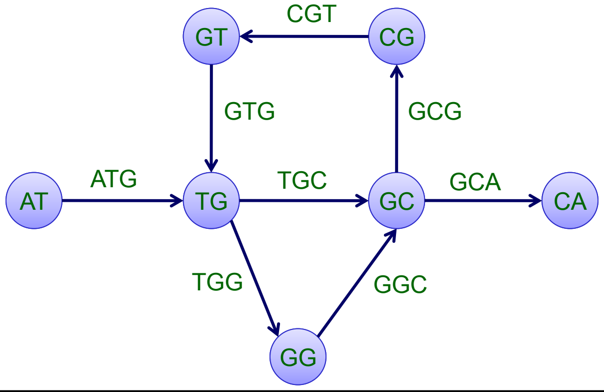
spectrum(*S*, 4) = {ACTG, ATGC, CTGA,GATG, GCAT,TGAT, TGCA}

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
AA																
AC															X	
AG																
AT										X						
CA																
CC																
CG																
CT									X							
GA															X	
GC				X												
GG																
GT																
TA																
TC																
TG				X	X											
TT																

de Bruijn graph

- in a de Bruijn graph
  - edges represent *k*-mers that occur in *spectrum*(*s*, *l*)
  - vertices correspond to (*k-1*)-mers

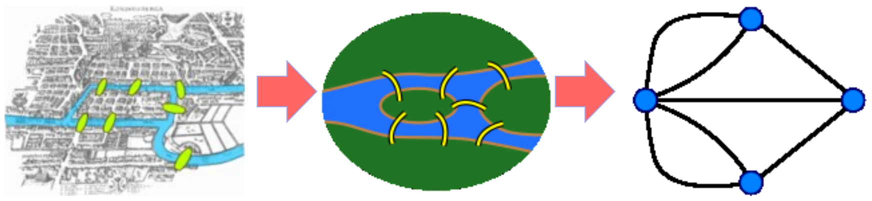
{ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT}



de Bruijn graph

- Can we find a DNA sequence containing all *k*-mers?
- In a de Bruijn graph, can we find a path that visits every edge of the graph exactly once?

Seven Bridges of Königsberg



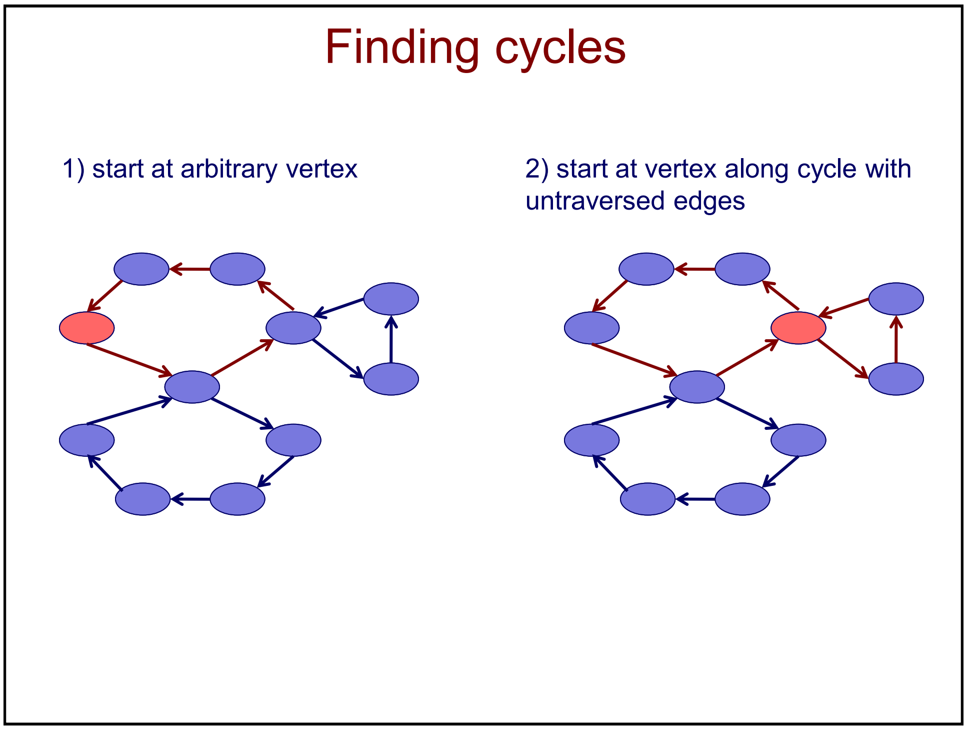
Euler answered the question: “Is there a walk through the city that traverses each bridge exactly once?”

Properties of Eulerian graphs

- *cycle:* a path in a graph that starts/ends on the same vertex
- *Eulerian cycle*: a path that visits every edge of the graph exactly once
- ***Theorem****:* A connected graph has an Eulerian cycle if and only if each of its vertices are *balanced*
- A vertex *v* is *balanced* if *indegree(v)* = *outdegree(v)*
- There is a linear-time algorithm for finding Eulerian cycles!

Eulerian cycle algorithm

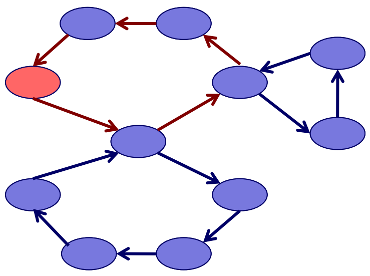
- start at any vertex *v*, traverse unused edges until returning to *v*
- while the cycle is not Eulerian
  - pick a vertex *w* along the cycle for which there are untraversed outgoing edges
  - traverse unused edges until ending up back at *w*
  - join two cycles into one cycle

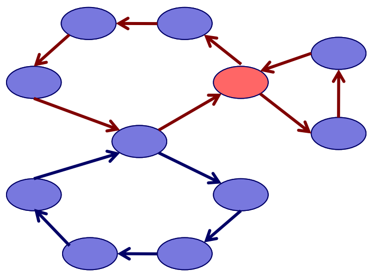


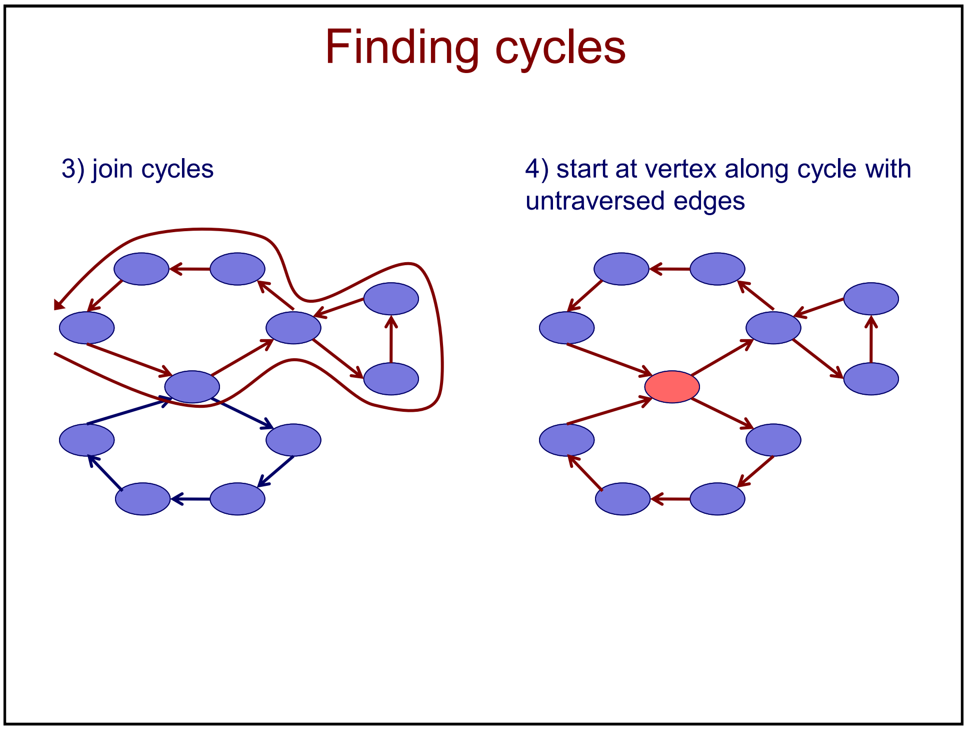
Finding cycles

1) start at arbitrary vertex

2) start at vertex along cycle with untraversed edges



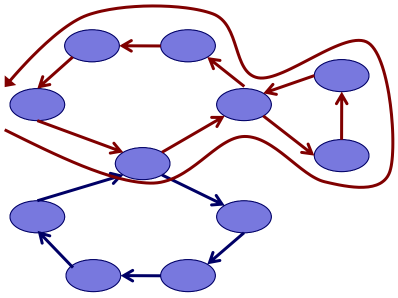


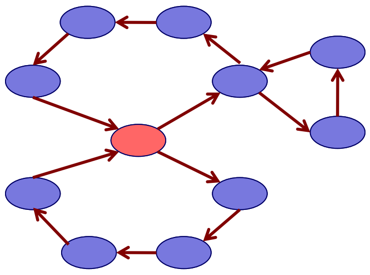


Finding cycles

3) join cycles

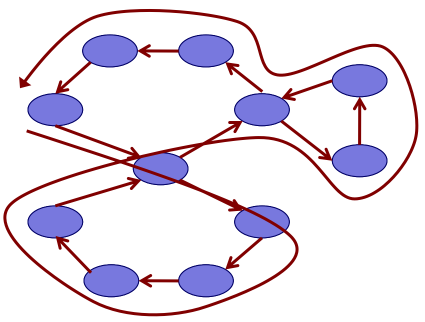
4) start at vertex along cycle with untraversed edges



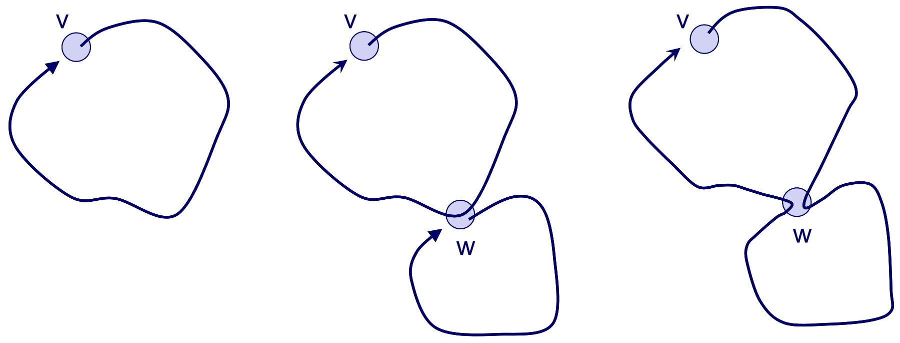


Finding cycles

5) join cycles

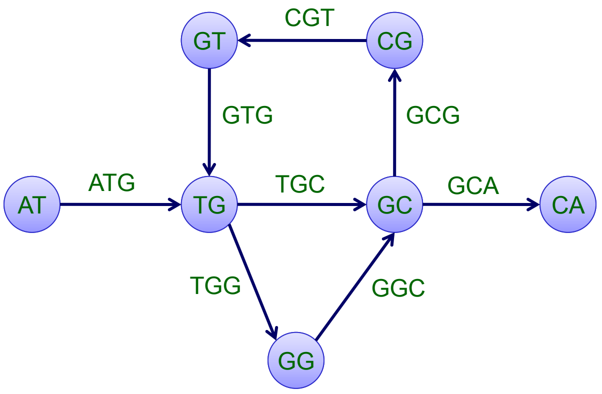


Joining cycles



Assembly as finding Eulerian *paths*

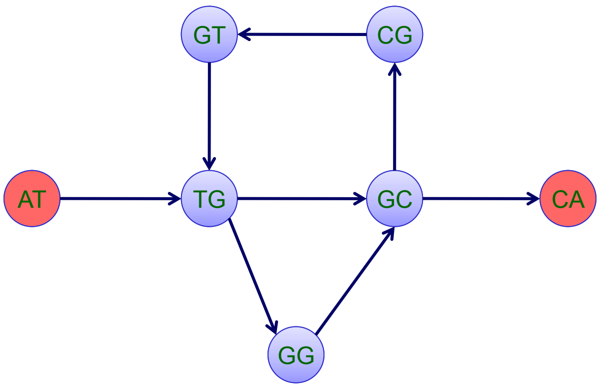
- *Eulerian path*: path that visits every edge exactly once
- we can frame the assembly problem as finding Eulerian paths in a de Bruijn graph
- resulting sequences contain all *k*-mers



- assembly: ATGGCGTGCA or ATGCGTGGCA

Eulerian *paths*

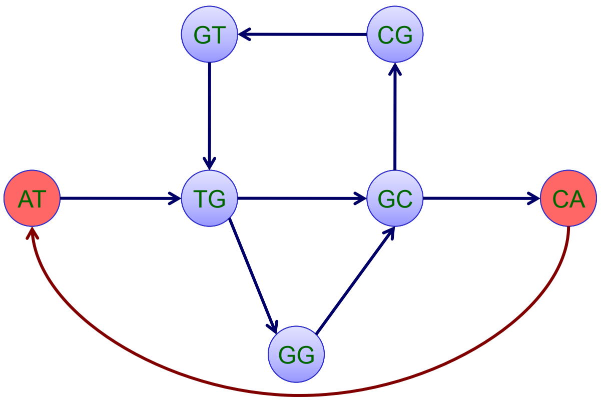
- a vertex v is semibalanced if $|indegree(v) - outdegree(v)| = 1$
- a connected graph has an *Eulerian path* if and only if it contains at most two semibalanced vertices



Eulerian path ➔ Eulerian cycle

- If a graph has an Eulerian Path starting at *w* and ending at *x* then
  - All vertices must be balanced, except for *w* and *x* which may have |*indegree(v)* – *outdegree(v)*| = 1
  - If and *w* and x are not balanced, add an edge between them to balance
    - Graph now has an Eulerian cycle which can be converted to an Eulerian path by removal of the added edge

Eulerian path ➔ Eulerian cycle

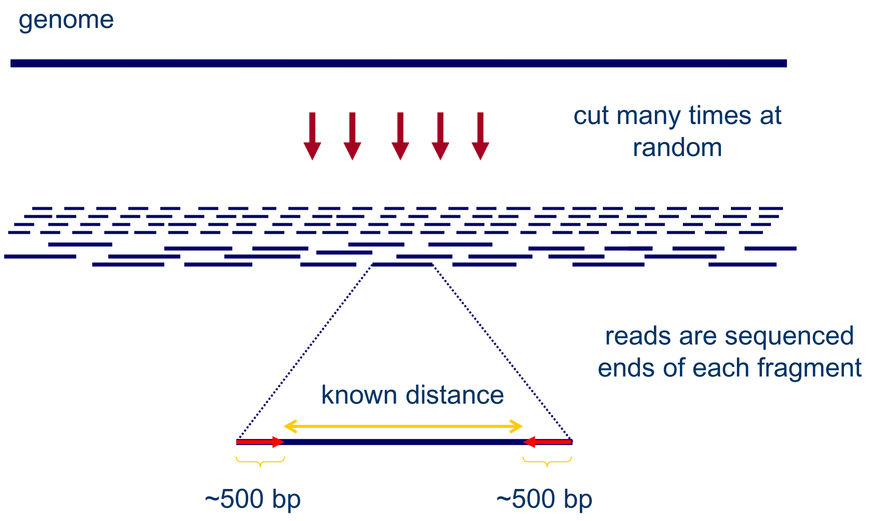


Sequence assembly in practice

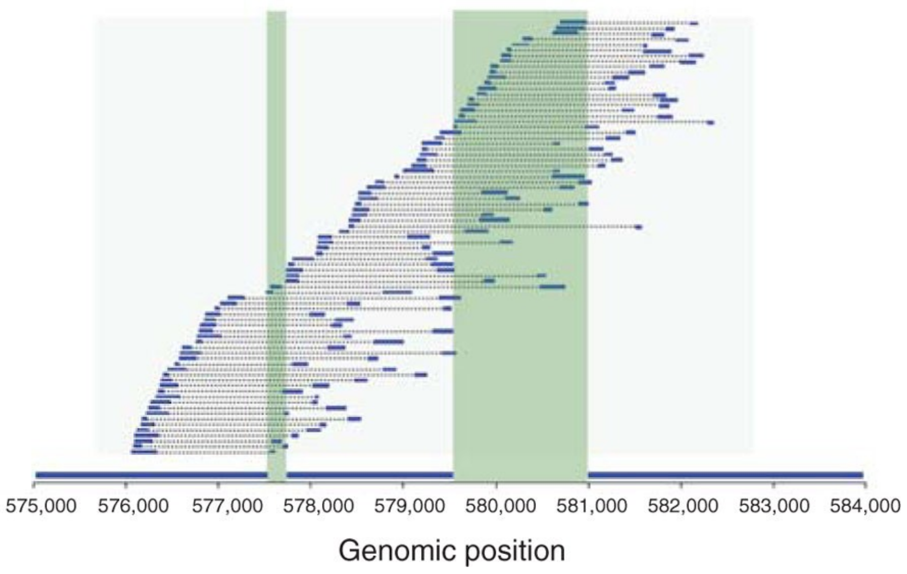
- approaches are based on these ideas, but include a lot of heuristics
- “best” approach varies depending on length of reads, amount of repeats in the genome, availability of paired-end reads

Paired end reads

- one approach to reducing ambiguity in assembly is to use *paired end* reads



Paired end reads



The Velvet assembler

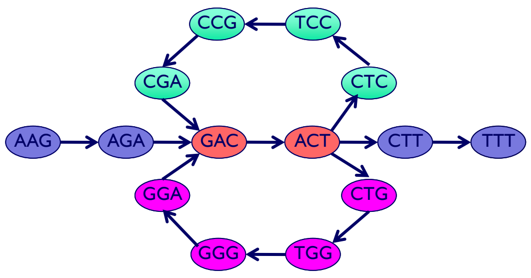
- based on de Bruijn graphs
- includes additional tricks for
  - reducing the size of the graph
  - trying to correct for errors in sequences
  - taking advantage of paired-end reads

Compressing the graph in Velvet

reads

AAGA
ACTC
ACTG
ACTT
AGAC
CCGA
CGAC
CTCC
CTGG
CTTT
GACT
GGAC
GGGA
TCCG
TGGG

de Bruijn Graph



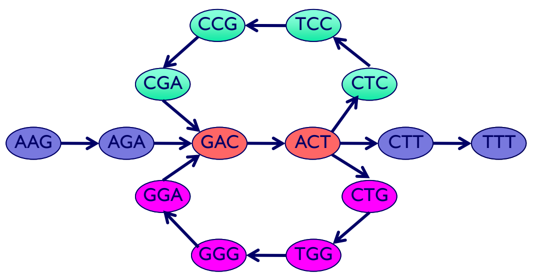
Potential Genomes

AAGACTCCGACTGGGACTTT

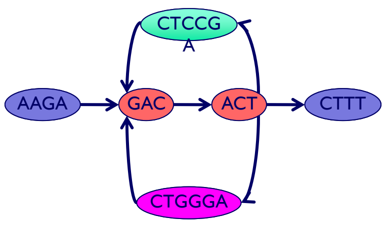
AAGACTGGGACTCCGACTTT

- human genome: ~ 3B nodes, ~10B edges

Compressing the graph in Velvet



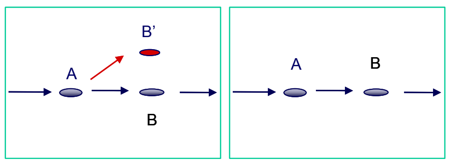
collapse linear subgraphs



Error correction in Velvet

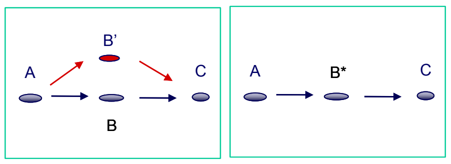
errors at end of read

- trim off ‘dead-end’ tips



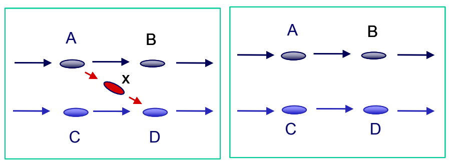
errors in middle of read

- pop bubbles



chimeric edges

- clip short, low coverage nodes



Summary

- The sequencing problem
  - Sequencing in vitro
  - Sequence assembly in silico
    - De novo versus resequencing
    - Approaches: greedy, overlap graph, Euler trail
      - Reads, contigs, scaffolding
  - Assembly validation
    - Statistical, viewers, comparative methods
- Still open problem
  - Costs, efficiency, reliability