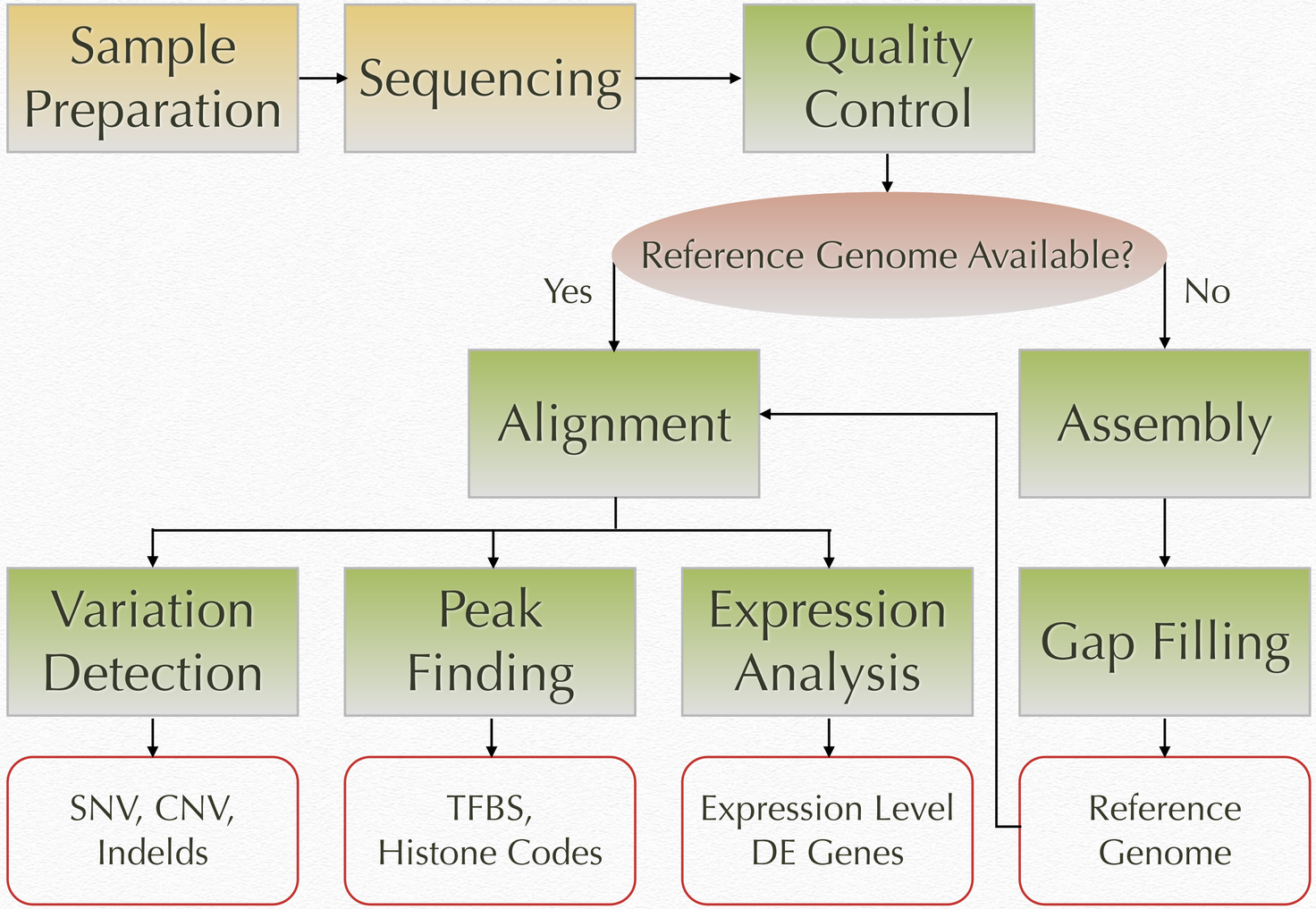
In the name of the most high

Introduction to Bioinformatics

Genome Assembly

Ali Sharifi-Zarchi
Department of Computer Engineering, Sharif University of Technology

Analysis Pipeline



Massively Parallel Sequencing

- Generating billions of 100-200 bp nucleotide sequences (reads) in a few days

  - Resequencing: Alignment to the already known reference genome

  - *De novo* Assembly: Building the genome from scratch

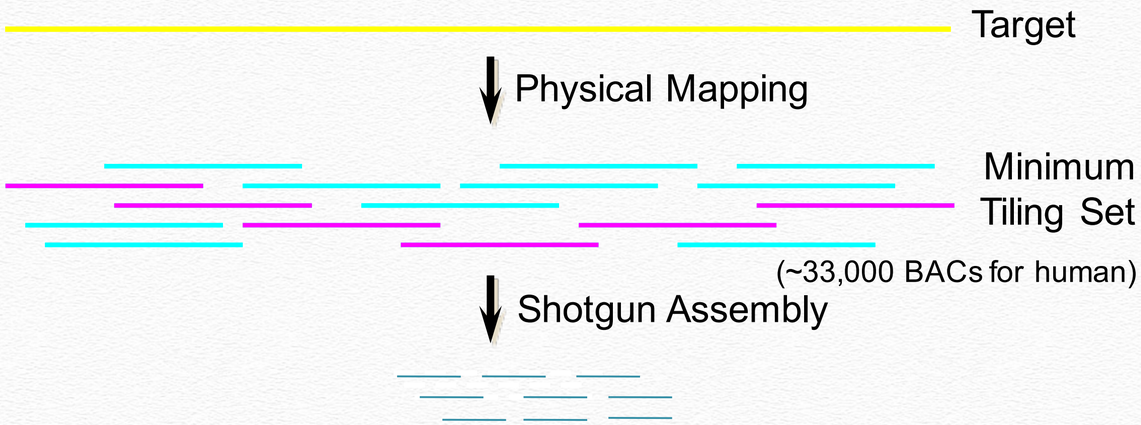
Resequencing vs. Assembly

- Resequencing
  - Faster
  - Less memory required
  - Needs fewer number of reads
  - Does not require long reads

Assembly vs. Resequencing

- *De Novo* Assembly
  - More sensitive to small indels
  - More sensitive to structural variations
  - No coverage bias to AT-rich or CG-rich regions
  - The only choice when reference genome is not available!

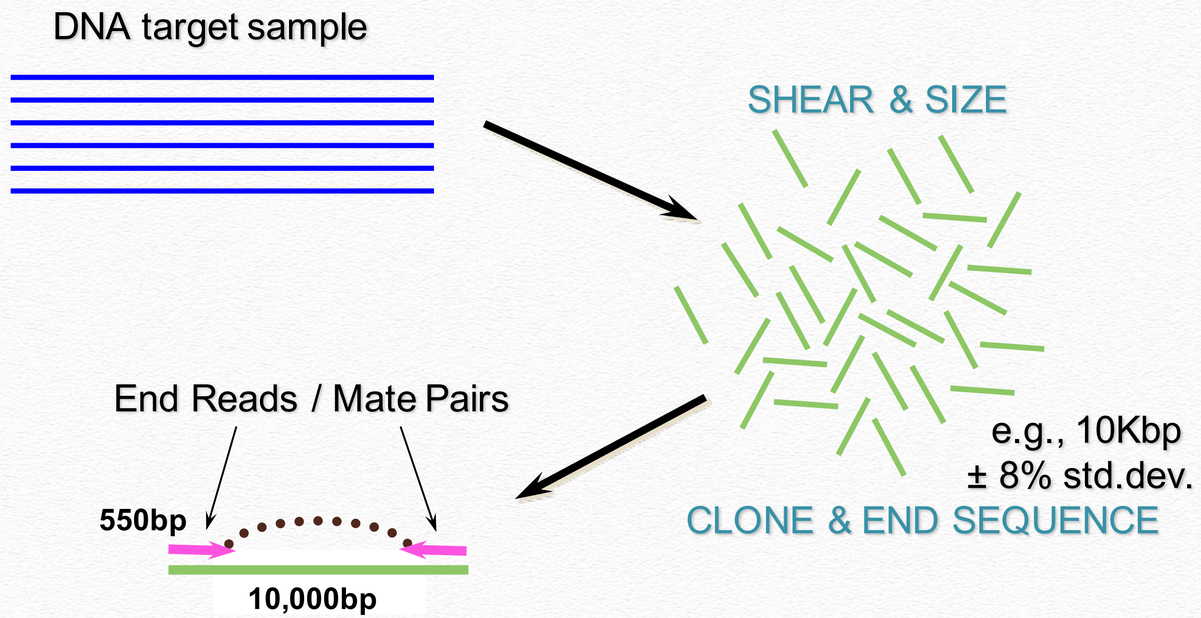
Clone-by-Clone Sequencing



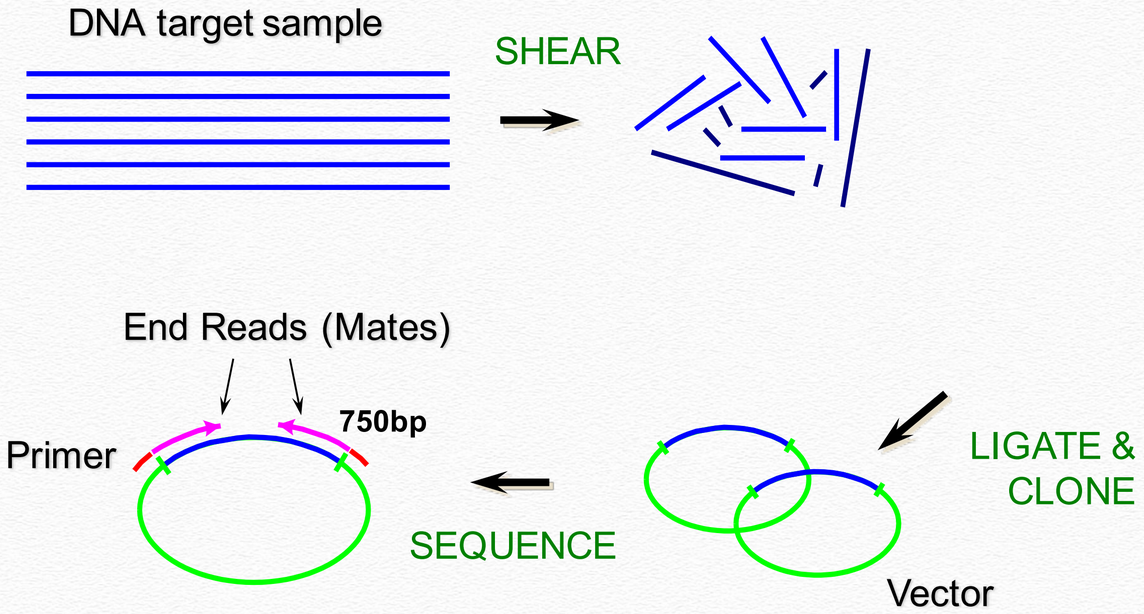
Clone-by-Clone Sequencing

- – Took ~11 years (1990-2001) to sequence 1 human and publish draft genome (complete genome: 2003)
- $3B Budget, labs in 20 countries
- – 2 Separate processes
- – Clone libraries unstable, maps hard to complete
- – Sequencing libraries must be made for every clone
- + Assembly problem ‘easy’ and well understood

Whole-genome Shotgun Sequencing



Whole-genome Shotgun Sequencing (2nd Method)



Whole-genome Shotgun Sequencing

- + Took ~3 years to sequence 1 human
- $300M budget, labs by one company!
- – Assembly problem ‘challenging’

Human Genome Project

Sanger Sequencing Assemblers

- GigAssembler (International Human Genome Project, Kent & Haussler)
- TIGR Assembler (TIGR)
- Phrap (Univ. of Washington)
- WGS (Celera)
- Arachne (Batzouglou, Broad Institute)
- Phusion (Mullikin et. al.)
- Atlas (Havlak, Baylor College of Medicine)
- CAP3, PCAP (Iowa State Univ.)
- AMOS (Univ. of Maryland)
- RePS (Wang et. al.)
- Euler (Pevzner, Univ. of California)
- gsAssembler (Roche, 454 Life Science)
- DNA Baser (Cubic Design)
- Elvira
- SHARCGS (Genome Res., 2007)
- SAKE (Bioinformatics, 2007)
- SHRAP (PlusONE 2007)
- New Euler (Genome Res. 2007)
- Sequencher
- JAZZ (Aparicio et. al)

Algorithmic View

- Given some strings $s_1, s_2, \ldots, s_n$
- We want to find the shortest string S such that all $s_1, s_2, \ldots, s_n$ are substrings of S
- The Shortest Common Superstring (SCS) of {“CAT”, “CGCC”, “CCCA”} is “CGCCCAT”
- It is equal to Traveling Salesman Problem
- So it is NP-Hard

Greedy Algorithm

Let T = set of all fragments

do {

For the pair (s,t) in T with maximum overlap do

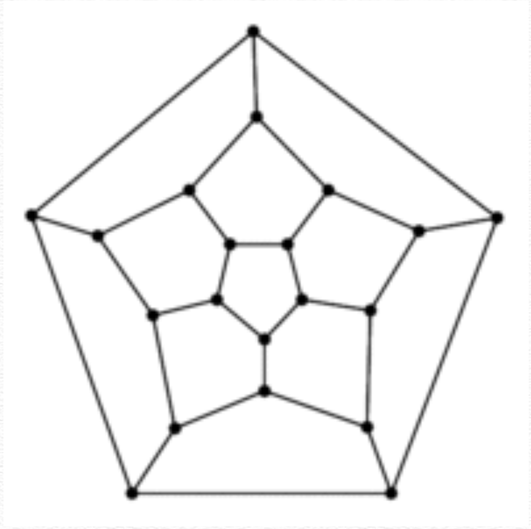
merge(s,t)

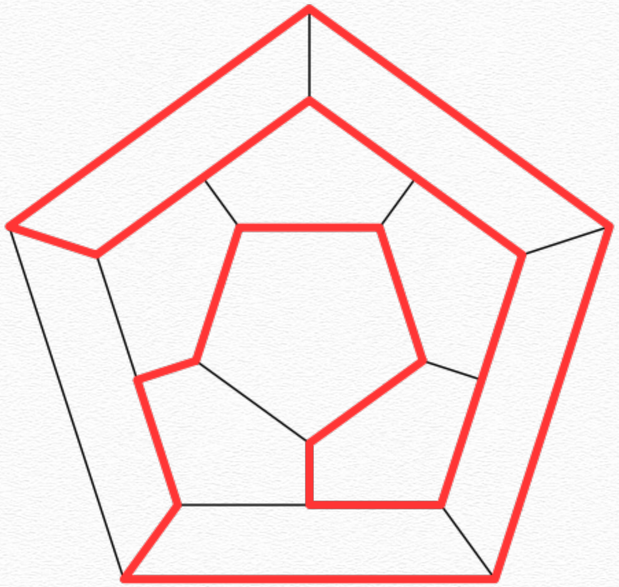
} while there are at least two strings in T

Output the only string in T

- This algorithm is 2.75 times optimal
- It ignores the biological problems of: orientation, read errors, insertions and deletions
- It is used in TIGR Assembler, Phrap and CAP3 and is the base of some other assemblers

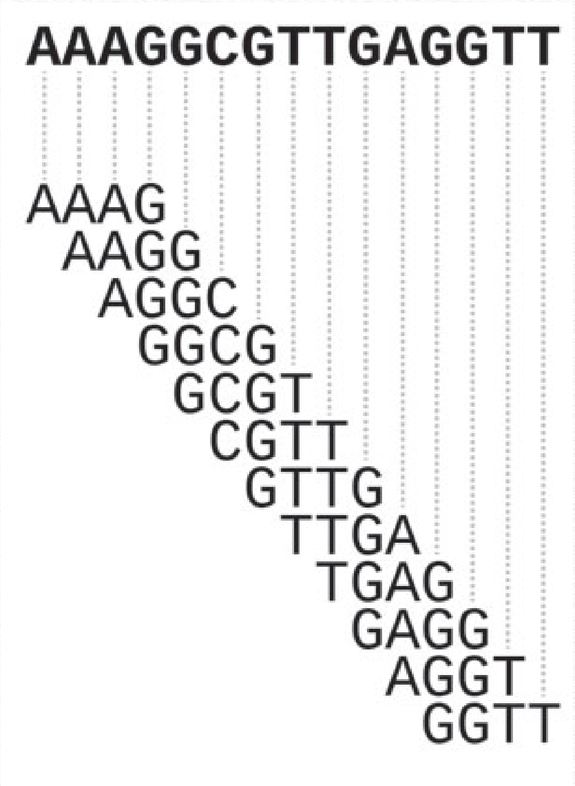
Hamiltonian Path/Cycle

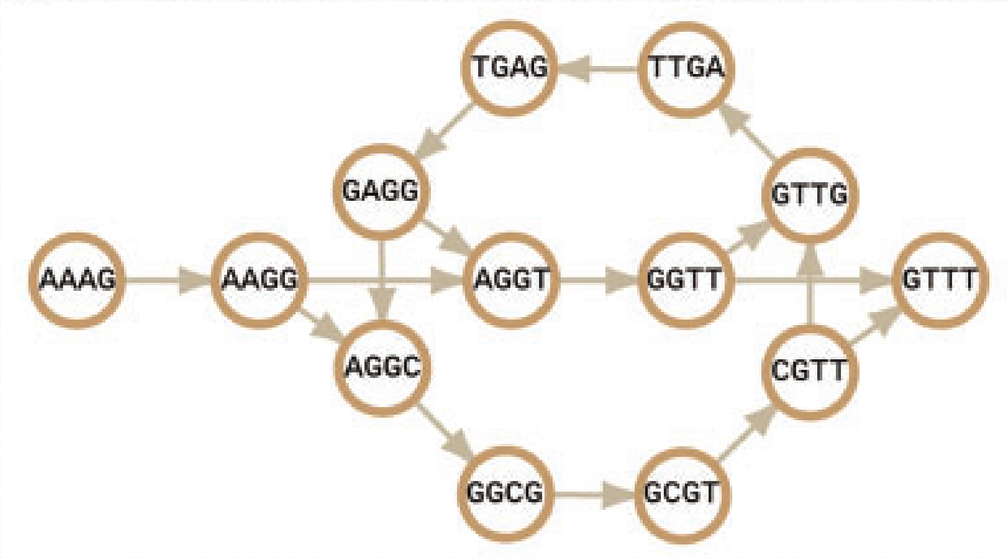




Wikipedia

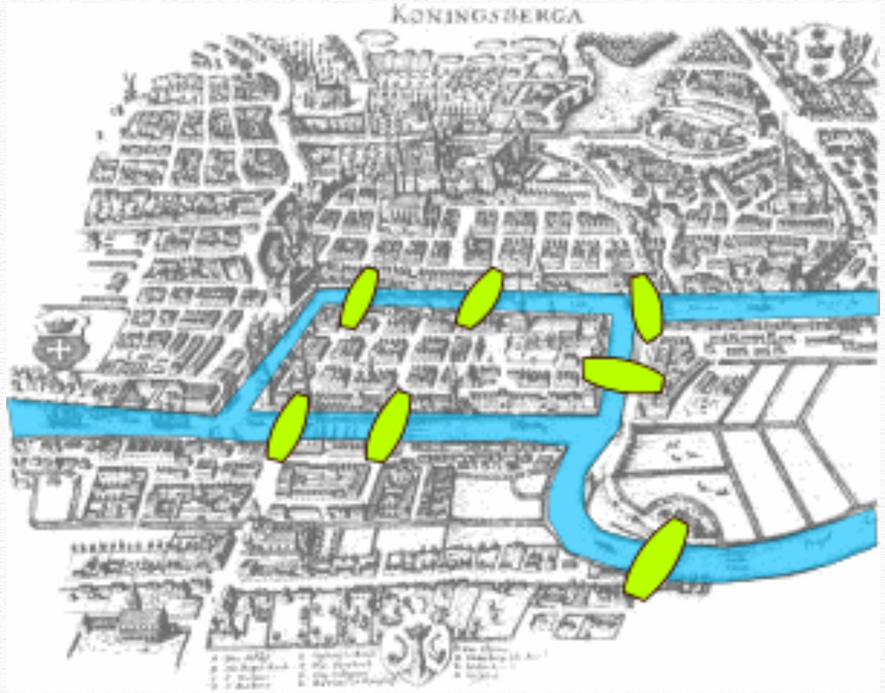
Hamiltonian Path

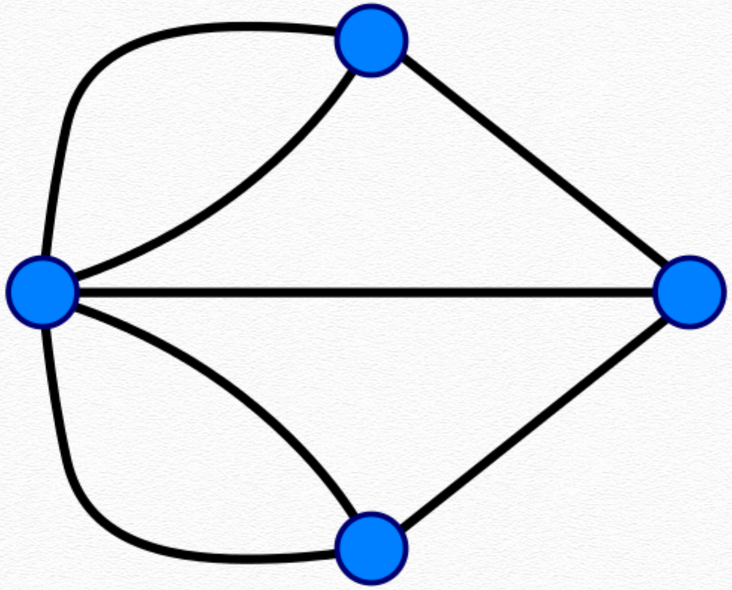




Jang-il Sohn and Jin-Wu Nam, Briefings in Bioinformatics, 2016

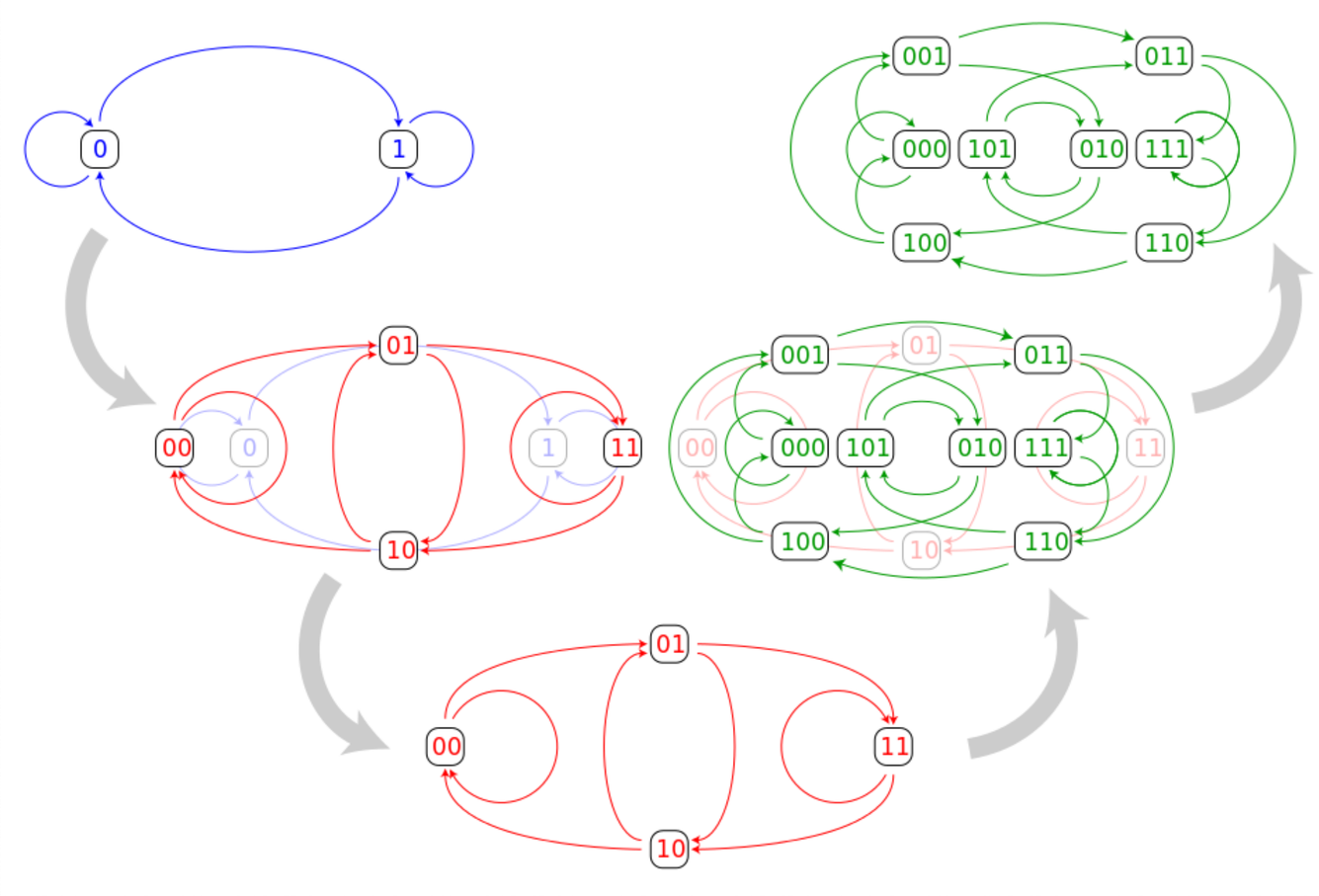
Eulerian Tour





Wikipedia

De Bruijn Graph



Wikipedia

Question: How To Pronounce "De Bruijn"?

17

De Bruijn graphs have become a popular of late for solving genome assembly problems. However, I'm not sure if I've ever heard "de Bruijn" pronounced correctly--or even pronounced at all (I'm usually reading about it). As opposed to made-up programs or format names, de Bruijn is someone's actual name from an actual culture (Dutch perhaps?), so there should be a "correct" pronunciation, am I correct?

Here are the possibilities I've considered.

- de **BRAIN**
- de **BRINE**
- de **BROIN**
- de **BROON**

Is any of these correct, or is my Dutch *that bad*? :)

• 9.5k views

ADD COMMENT • link • Not following

modified 3.7 years ago by Andreas • **2.3k** • written 6.0 years ago by Daniel Standage ♦ **3.6k**

6.0 years ago by
Daniel Standage ♦
3.6k
Davis, California, USA

8

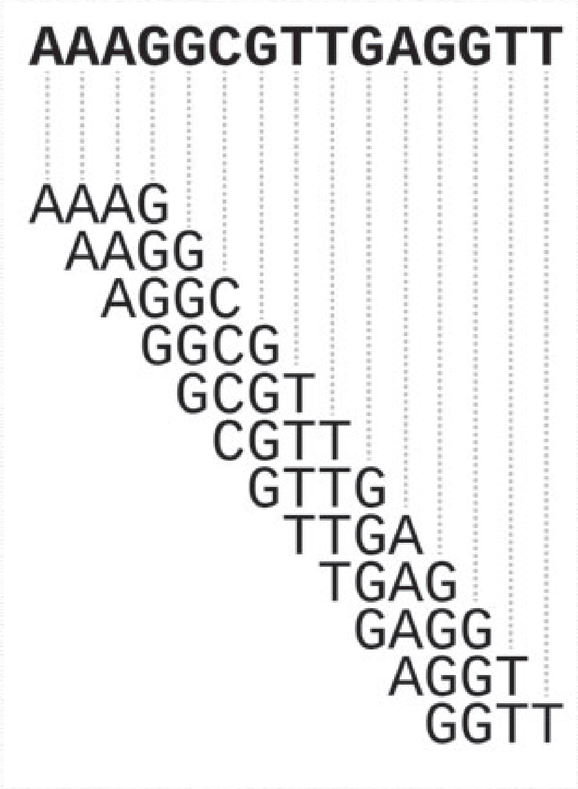
No offense to the non-Dutch people here, but if you're not Dutch (or Flemish or Afrikaans), you probably won't be able to pronounce this, unless you practice a lot. This is based on years and years of experience with people trying to pronounce my last name correctly .... :) The UI sound just doesn't seem to exist in many other languages ....

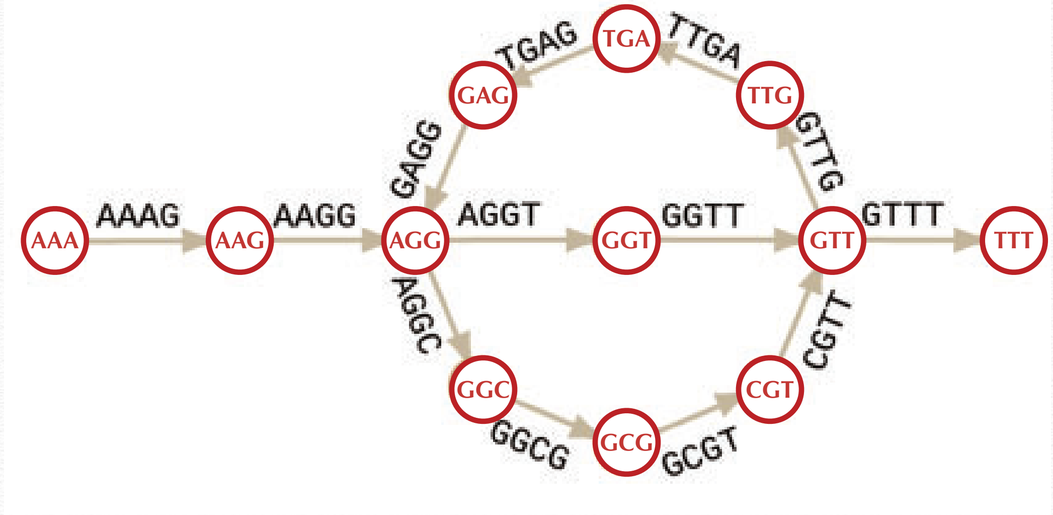
ADD COMMENT • link

written 6.0 years ago by Bert Overduin • **3.5k**

6.0 years ago by
Bert Overduin • **3.5k**
Edinburgh Genomics,
The University of
Edinburgh

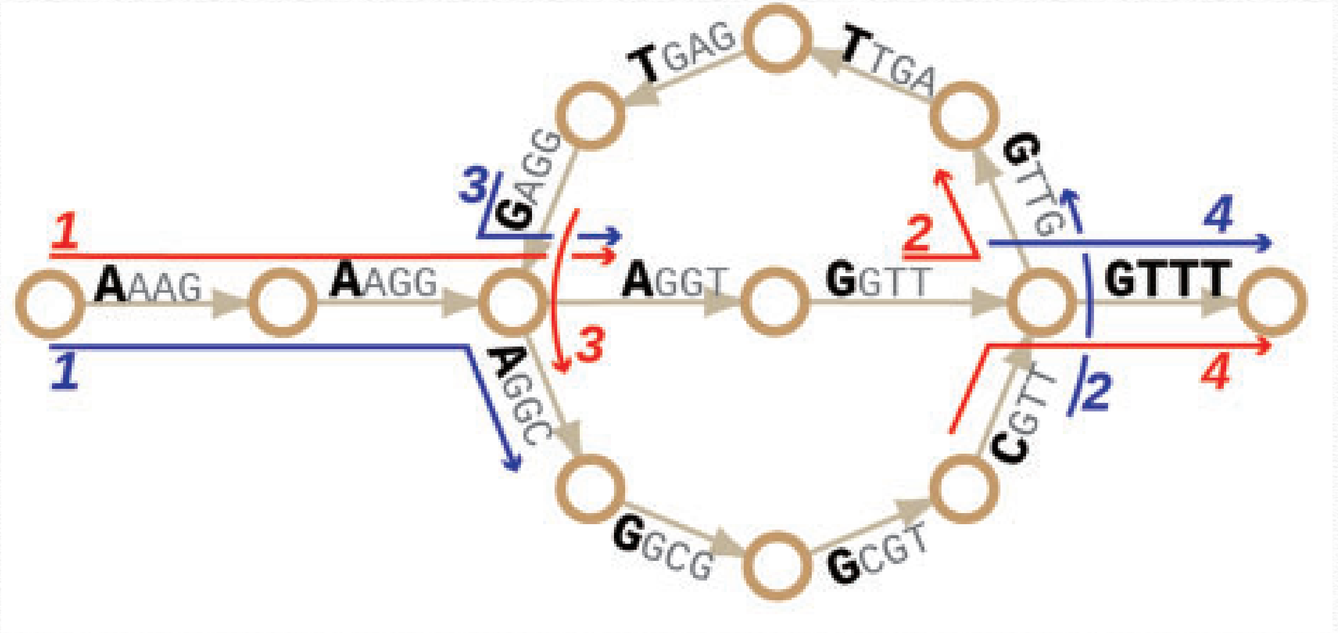
Euclidean Tour on De Bruijn Graph





Jang-il Sohn and Jin-Wu Nam, Briefings in Bioinformatics, 2016

The Repeats Challenge

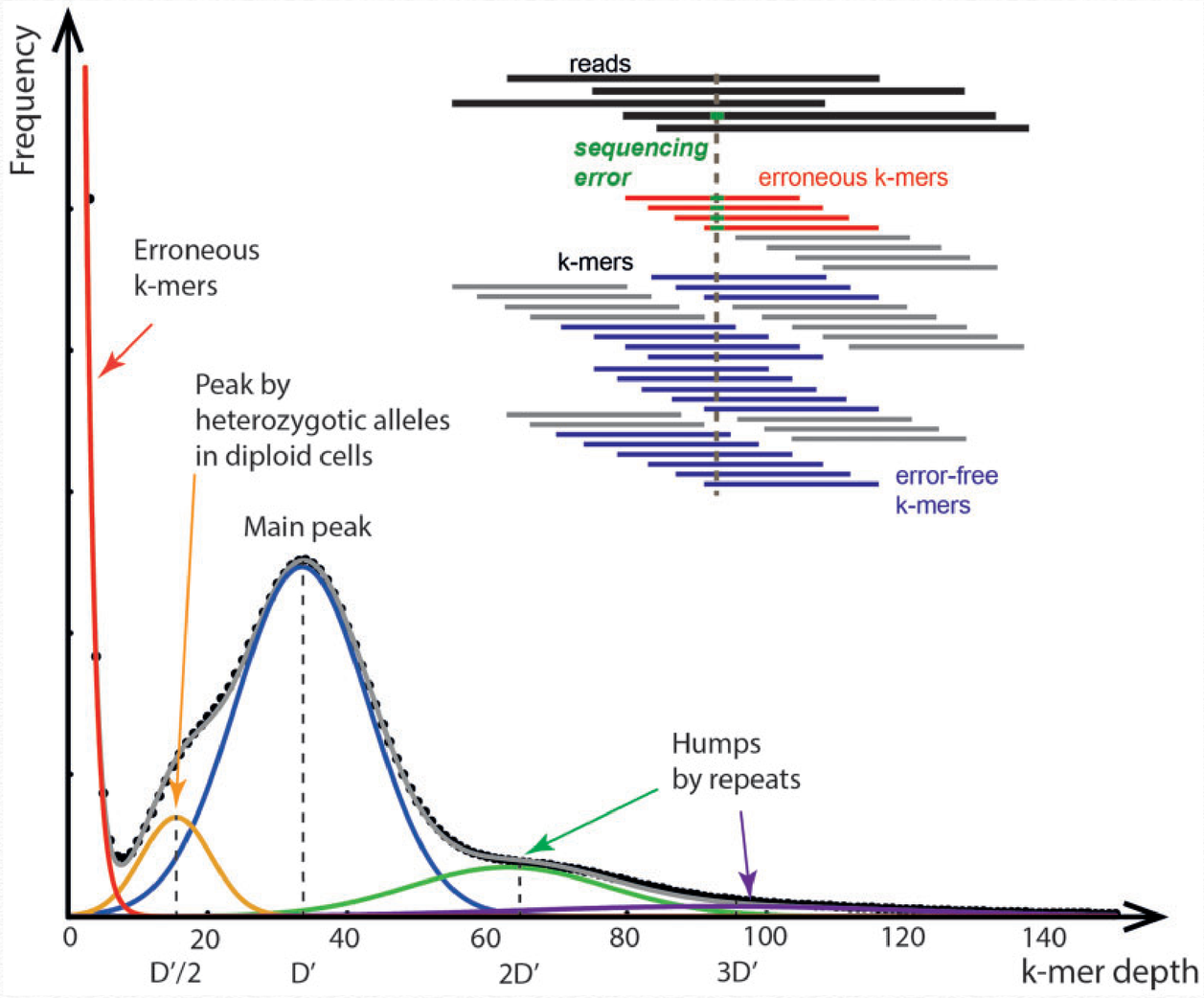


AAAGGTTGAGGCGTTT

AAAGGCGTTGAGGTTT

Jang-il Sohn and Jin-Wu Nam, Briefings in Bioinformatics, 2016

k-mer Histogram



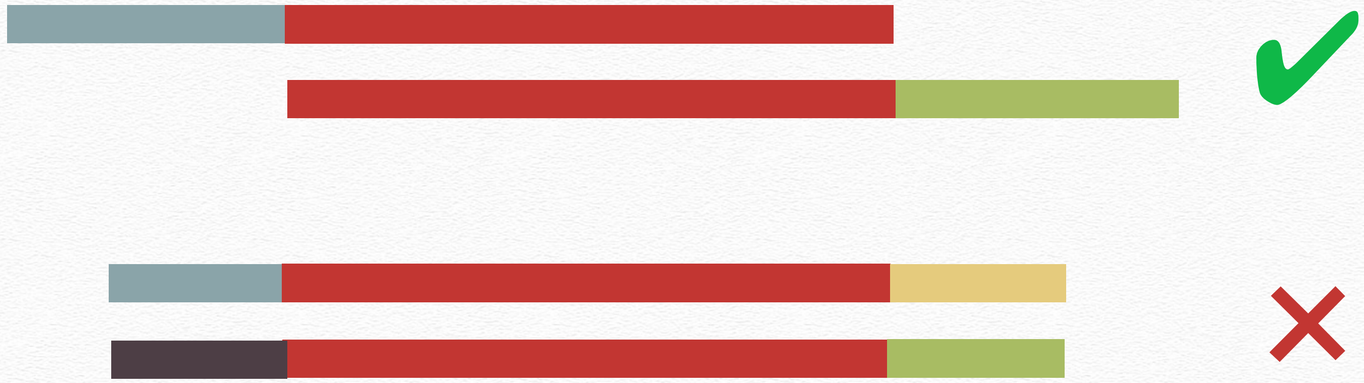
Jang-il Sohn and Jin-Wu Nam, Briefings in Bioinformatics, 2016

General Assembly Procedure

- The basic principle: two overlapping reads presumably originate from the same region of the genome
- For the repetitive parts of the genome, this assumption is invalid!
- Based on sequence similarity between reads, assembler coombines all sequencing reads into contiguous sequences, or “Contigs”
- Rarely the entire chromosomes are reconstructed into a single contig
- So a finishing step is performed

Basic Principle

Two overlapping reads presumably originate from the same genomic region



For the repeat elements, this principle is false!

Contigs

- Based on sequence overlaps between reads, assembler combines them into contiguous sequences, or Contigs
- Rarely the entire chromosomes are reconstructed into a single contig
- In practice there are hundreds or thousands of contigs generated

Finishing

- The task of closing the gaps between contigs and obtaining a complete molecule is called Finishing
- First, we try to make larger structures called Scaffolds

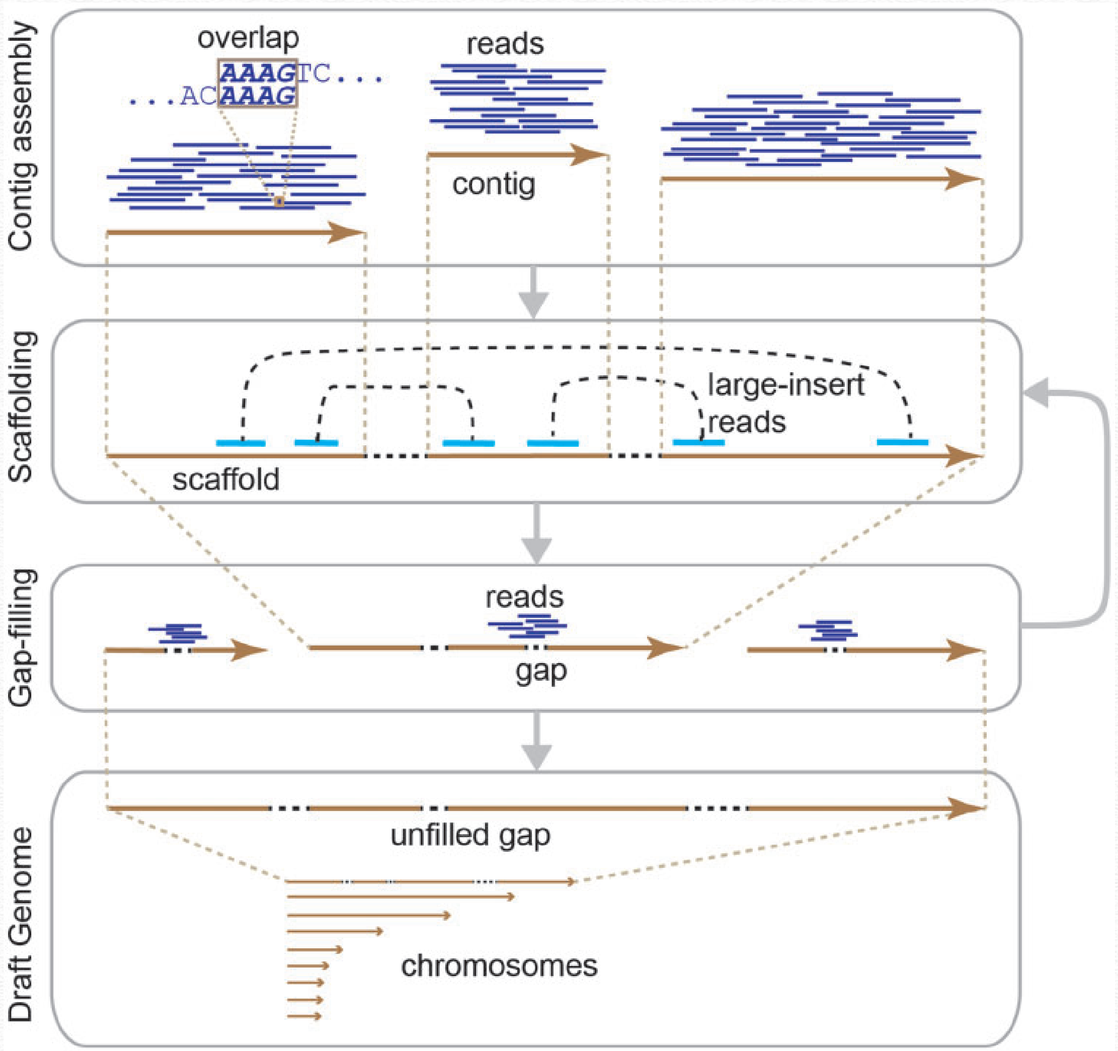
Gap Filling

- Two types of gaps will remain:

  - Sequence gaps: The gaps between contigs of same scaffold. They are covered by retrieving the original clone inserts (walking)

  - Physical gaps: The gaps betweens scaffolds. Filling them involves large amount of manual labor and complex laboratory techniques

Hierarchy

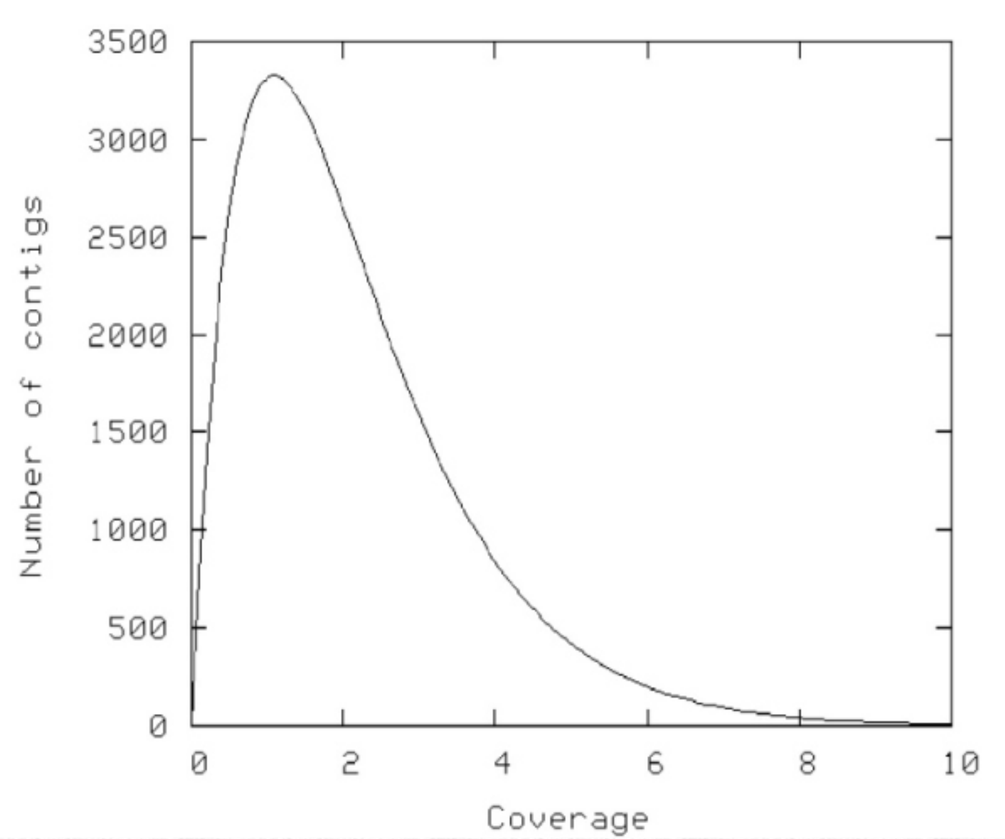


Jang-il Sohn and Jin-Wu Nam, Briefings in Bioinformatics, 2016

Coverage

- Total size of reads / size of genome

- Studied by Eric Lander and Michael Waterman in 1988:



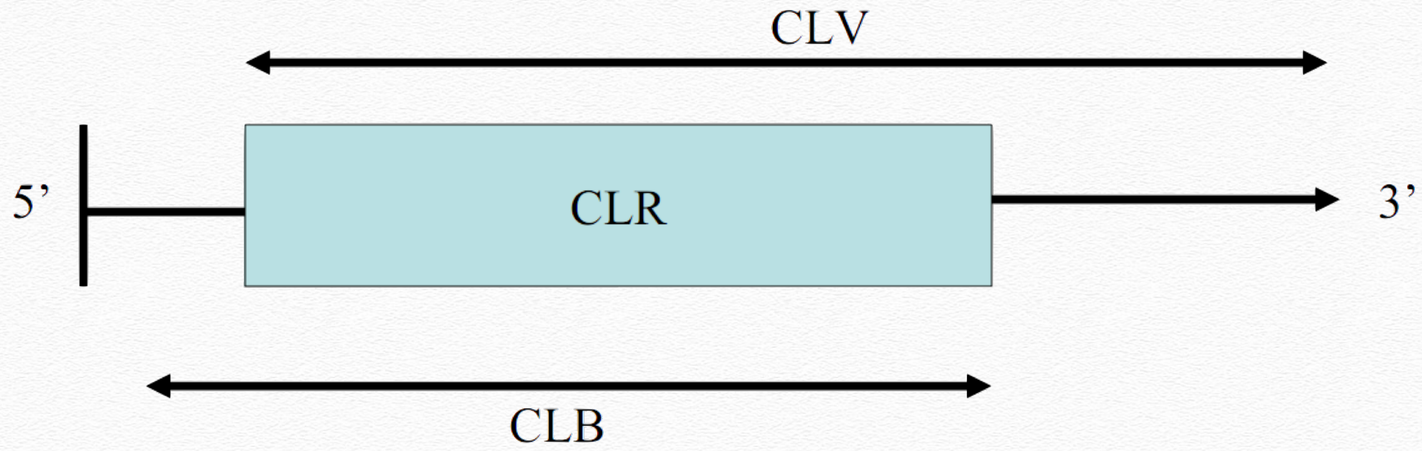
Assembly Complexities

- Repeats
- Sequencing errors
- Bias in reads
- Imperfect coverage
- Unknown orientation
- Repeat polymorphism

Celera Assembler

Overall procedure

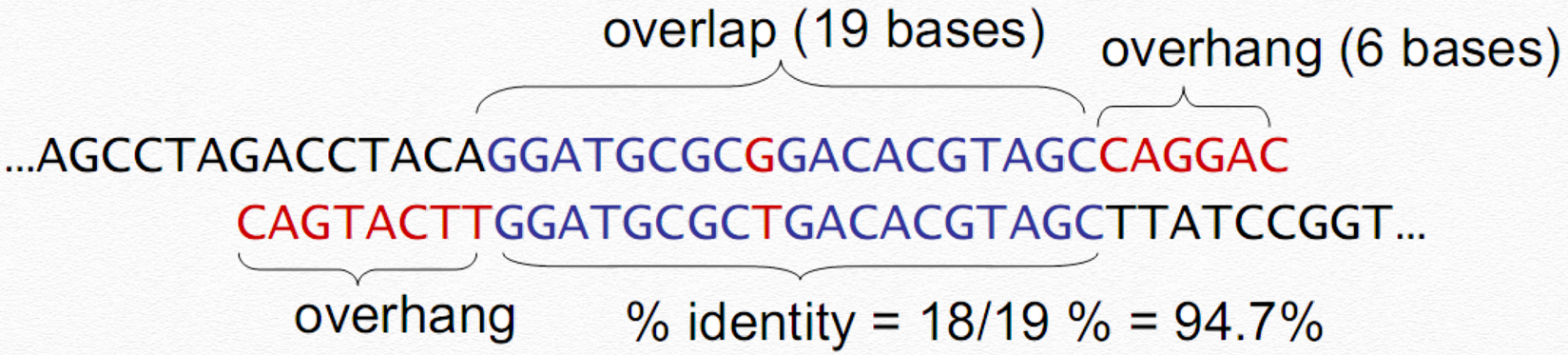
Trim & Screen



- CLV: Region free of vector
- CLB: Region free of bad quality reads
- Trimming identifies the regions of good quality for the assembler to use (CLR), as the intersection of CLV and CLB

Overlapper

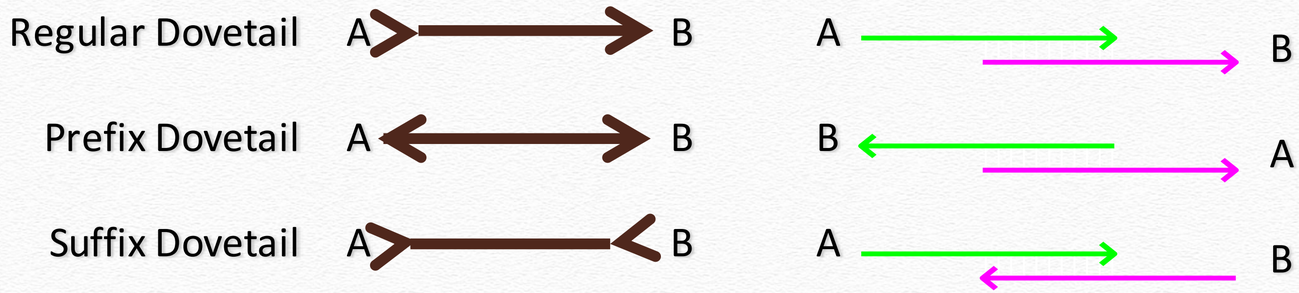
- The assembler screens merges based on:
  - Length of overlap
  - % Identity in overlap region
  - Maximum overhang size

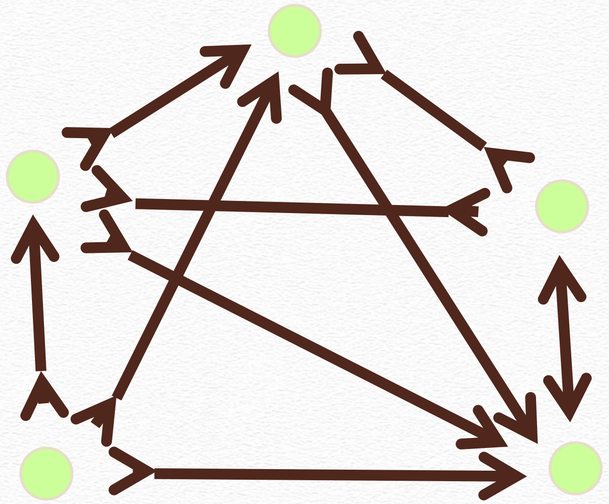


Overlapper

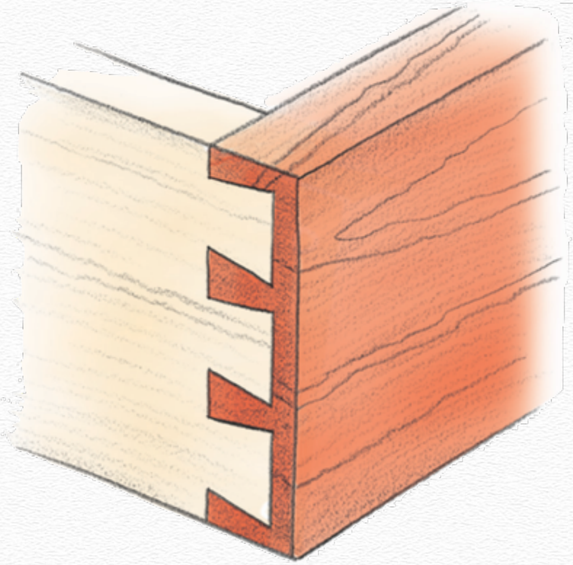
- A simple solution to find all overlaps: all pairs alignment, O(n^2) pairs (n=number of reads)
- Better idea: O(n x coverage) algorithm
  - Build a table of k-mers in sequences
  - Generate the pairs from one pass read of the table

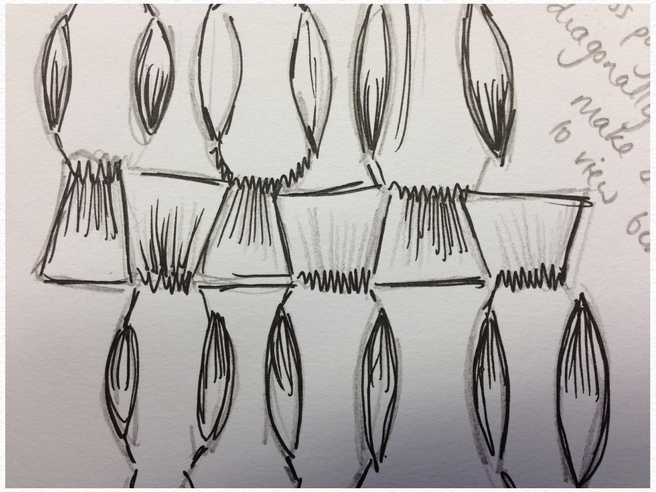
Overlap Types



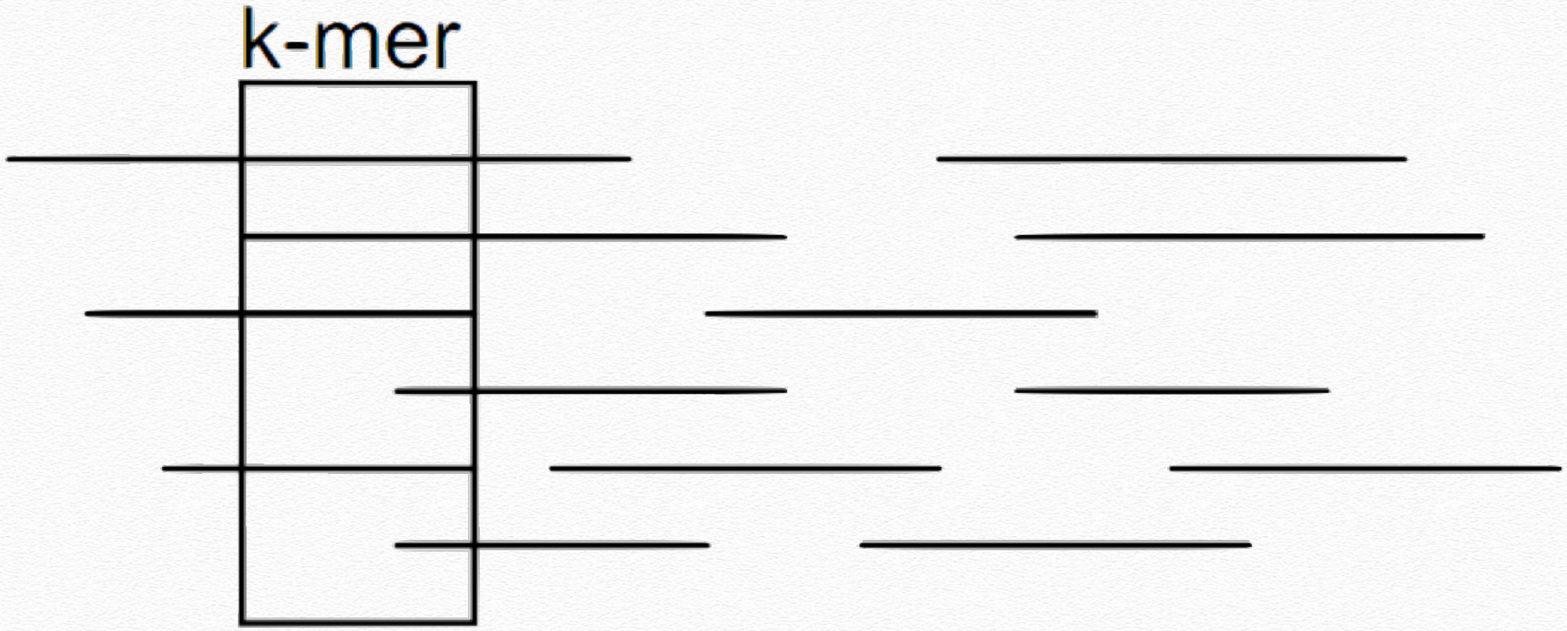


Dovetail?



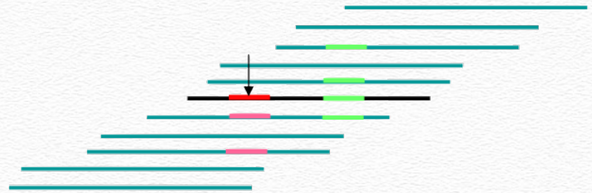


Overlapper



Error Correction

- If the k-mers from overlapping reads match, their bases are confirmed



- If a base is not confirmed, there is a vote for error correction

ACGTACCGATATGACAC

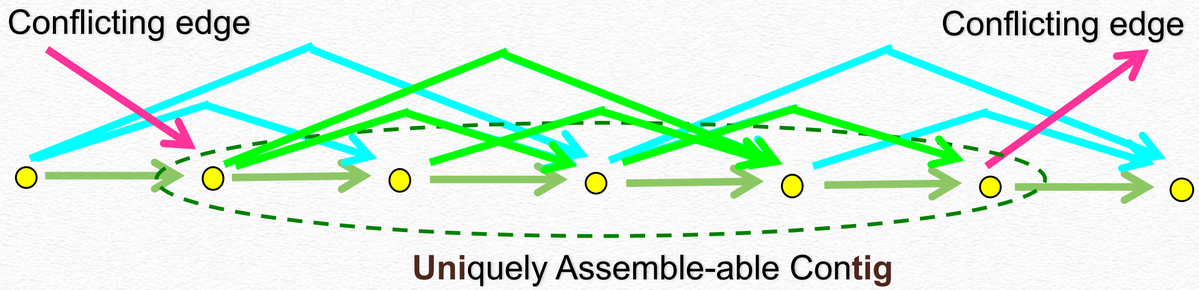
ACGTACCGTTATGACAC

ACGTACCGATATGACAC

ACGTACCGATATGACAC

Unitiger

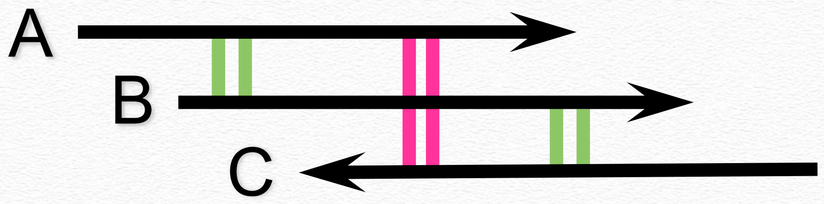
- Compute all overlap consistent sub-assemblies
- Finds Unitigs (Uniquely Assembled Contig)

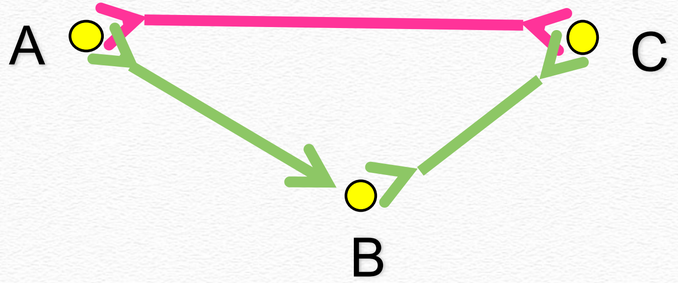


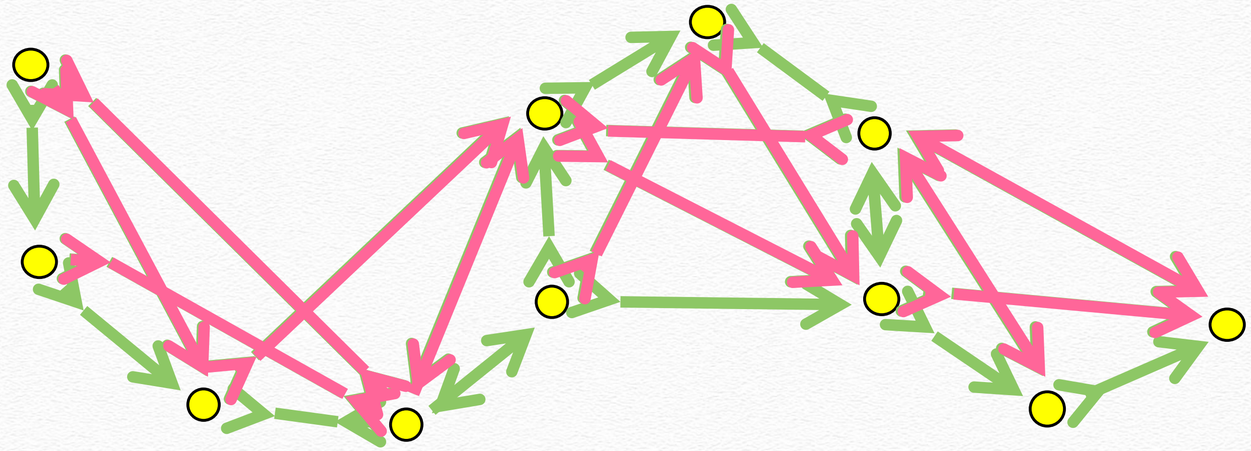
Unitiger Algorithm

- Remove contained fragments
- Remove transitively inferred edges
- Collapse into unitigs

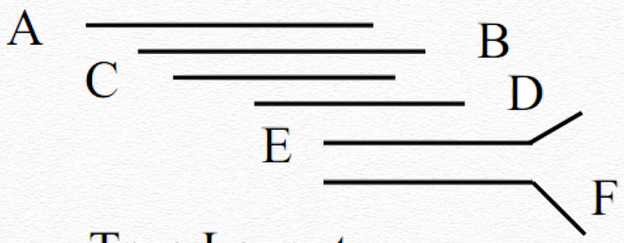
Transitive Edges Removal



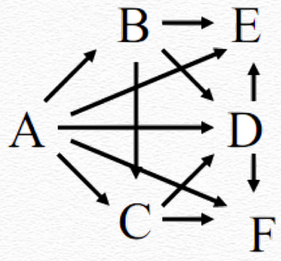




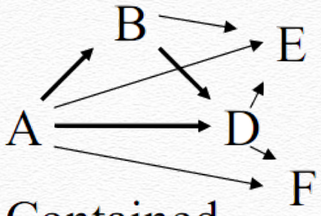
Unitiger



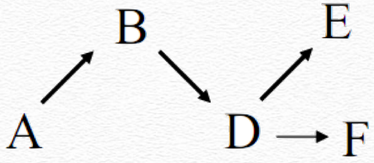
True Layout



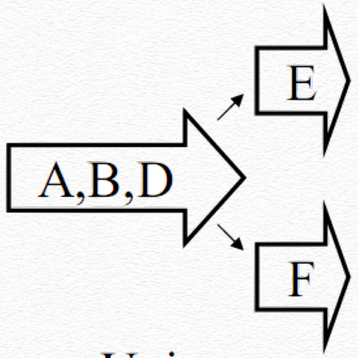
Original Overlap Graph



Contained Read Removal



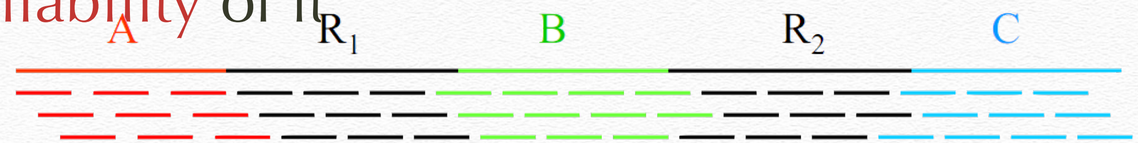
Transitive Edge Removal

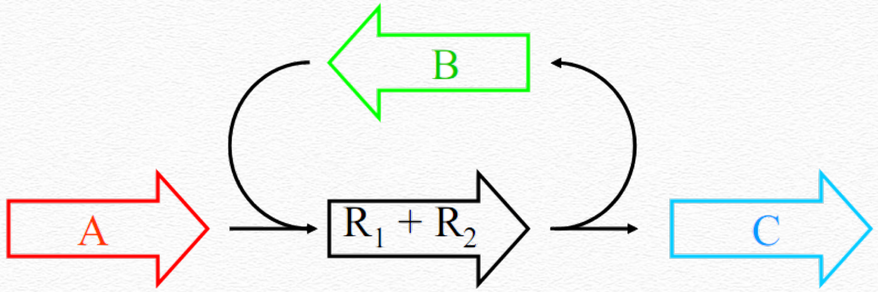


Unique Join Collapsing

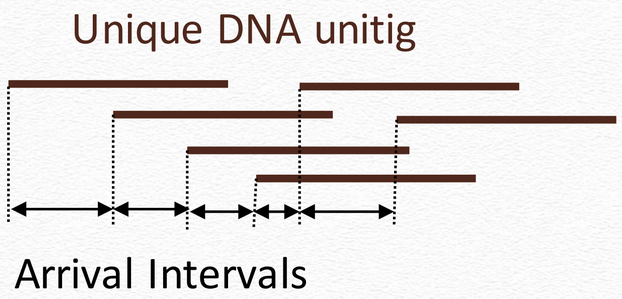
Unitig Scoring

- Compute the statistical arrival rate for each unitig, considering that all reads have uniform distribution
- The higher arrival rate for a unitig, the lower reliability of it





A-stat Scoring



A-stat Scoring

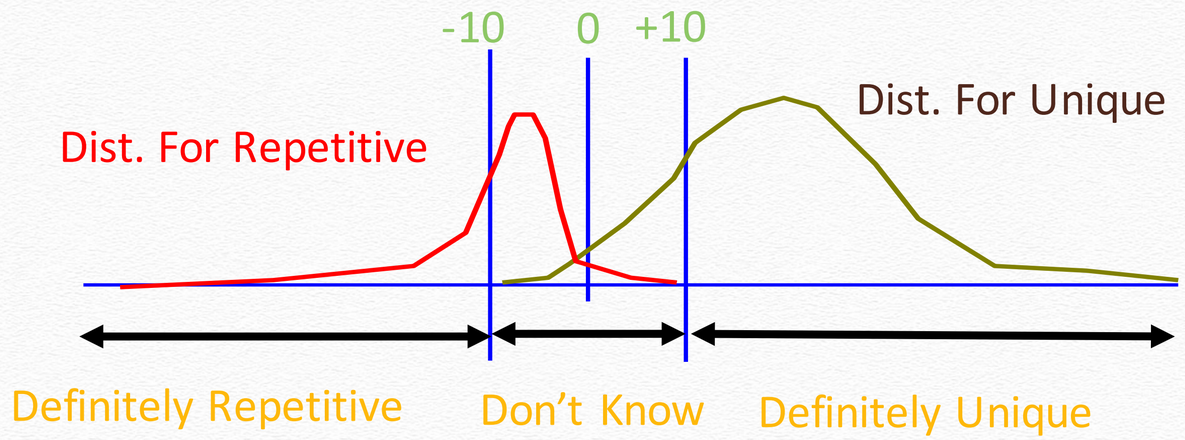
- Suppose there are F fragments in a database and the genome size is estimated as G
- For a unitig with k fragments and distance ρ between the start of its first fragment and the start of its last fragment, the probability of seeing the k – 1 start points in the interval of length ρ, given the unitig is not oversampled, is $[(\rho F/G)^k/k!] \exp(-\rho F/G)$.

A-stat Scoring

- If the unitig was the result of collapsing two repeats, then the probability is $[(2\rho F/G)^k/k!]\exp(-2\rho F/G)$

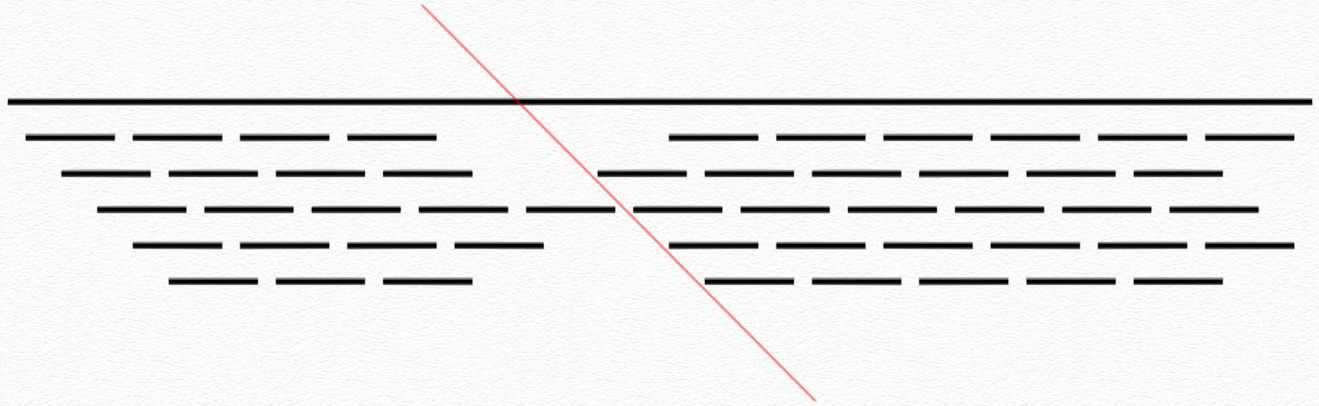
Discriminator A-statistics

- Log-odds ratio of probability unitig is unique DNA versus 2-copy DNA
- A-stat (unitig X) = log [Prob (X is single-copy | coverage (X)) / Prob (X is two-copy | coverage (X))]



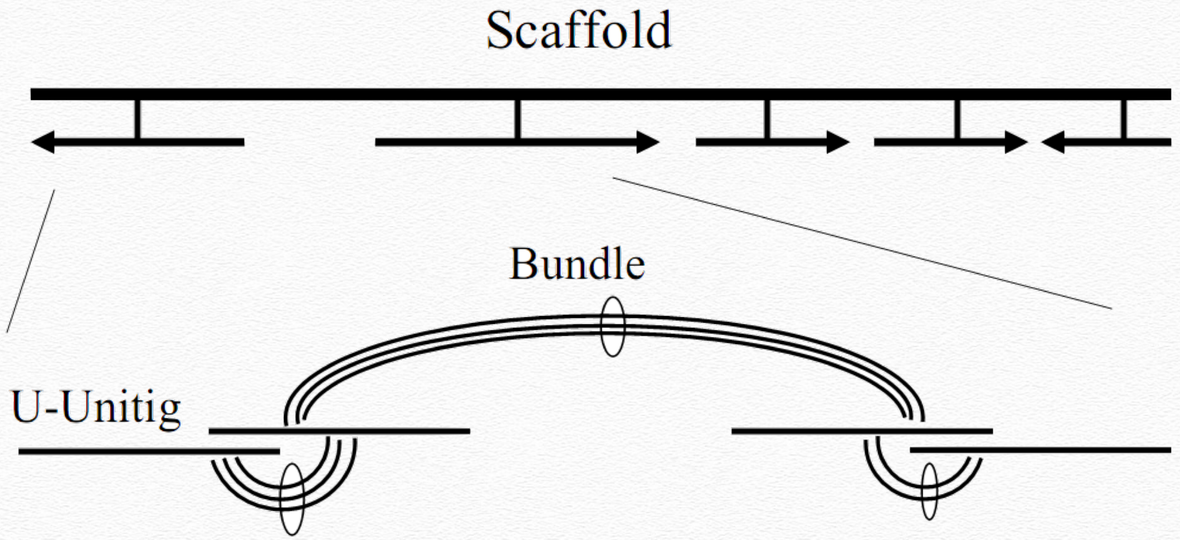
Uniting Splitting

- Unitigs are split when the coverage level drops below a threshold, and there are no mates connecting the Unitig



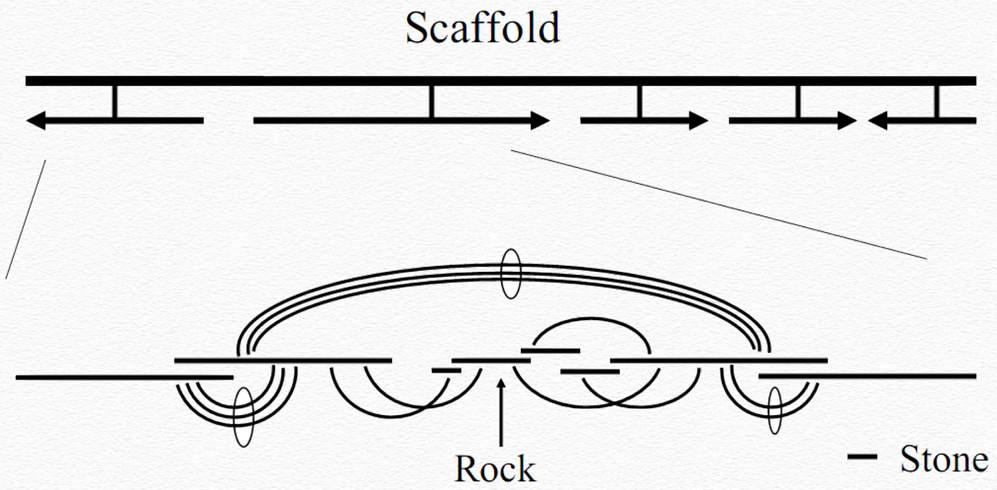
Scaffolding

- Create a initial scaffold of unique unitigs (U-Unitigs) whose A-stat > 5
- Crete borderline unitigs whose A-stat is > 2 having consistent mates with the U-Unitigs



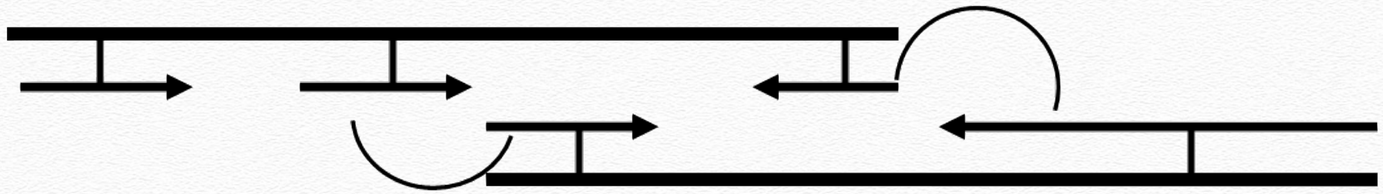
Finishing

- Place rocks (A-stat > 0 with multiple consistent mates), and stones (single mate and overlap path with placed objects) into the gaps
- Pebbles, unitigs lacking mates, are no longer incorporated regardless of overlap qualities



Scaffold Merging

- After placing borderline unitigs and rocks, there may be sufficient mates to merge scaffolds (mates from stones are not considered)
- If multiple orientations are possible, choose the scaffold merge with the happiest mates
- This in turn may allow for new rocks and stones to be placed, so iterate these steps until the scaffold stabilizes



Celera Summary

1. Compute Overlaps between reads
2. Simplify Overlap Graph into Unitigs
3. Score Unitigs based on Coverage
4. Create Contigs & Scaffold of Unique Unitigs
5. Fill in gaps with repetitive unitigs

Resource

Velvet: Algorithms for de novo short read assembly using de Bruijn graphs

Daniel R. Zerbino and Ewan Birney[1]

EMBL-European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, United Kingdom

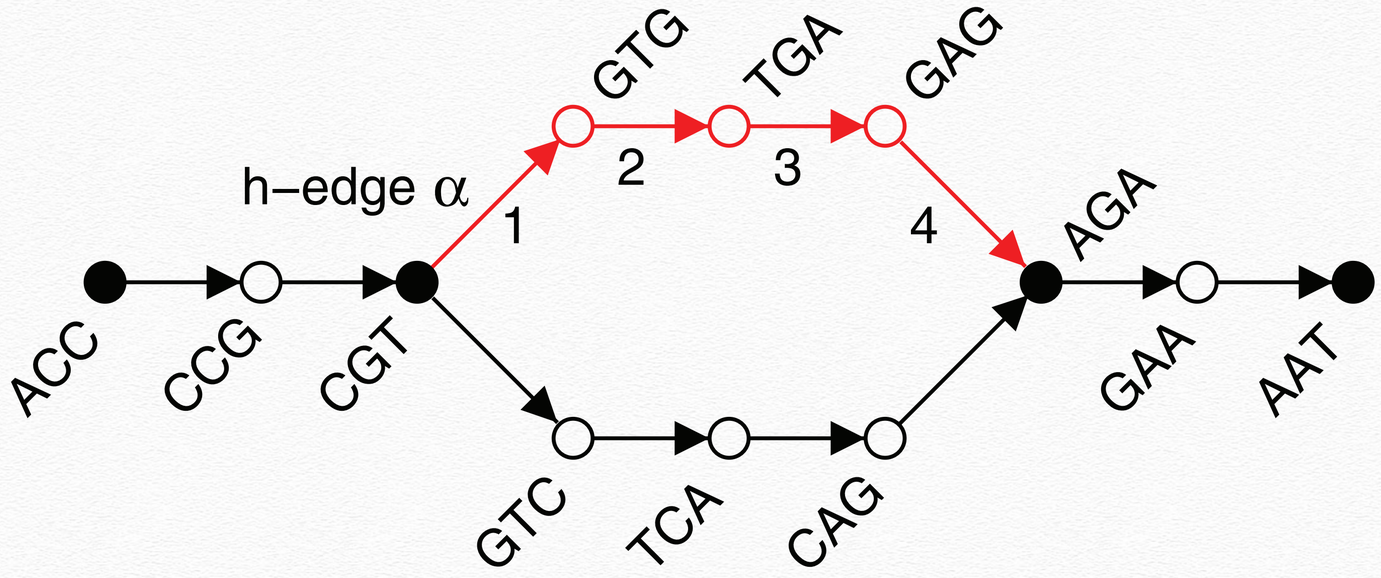
We have developed a new set of algorithms, collectively called "Velvet," to manipulate de Bruijn graphs for genomic sequence assembly. A de Bruijn graph is a compact representation based on short words (*k*-mers) that is ideal for high coverage, very short read (25–50 bp) data sets. Applying Velvet to very short reads and paired-ends information only, one can produce contigs of significant length, up to 50-kb N50 length in simulations of prokaryotic data and 3-kb N50 on simulated mammalian BACs. When applied to real Solexa data sets without read pairs, Velvet generated contigs of ~8 kb in a prokaryote and 2 kb in a mammalian BAC, in close agreement with our simulated results without read-pair information. Velvet represents a new approach to assembly that can leverage very short reads in combination with read pairs to produce useful assemblies.

[Supplemental material is available online at www.genome.org. The code for Velvet is freely available, under the GNU Public License, at http://www.ebi.ac.uk/~zerbino/velvet.]

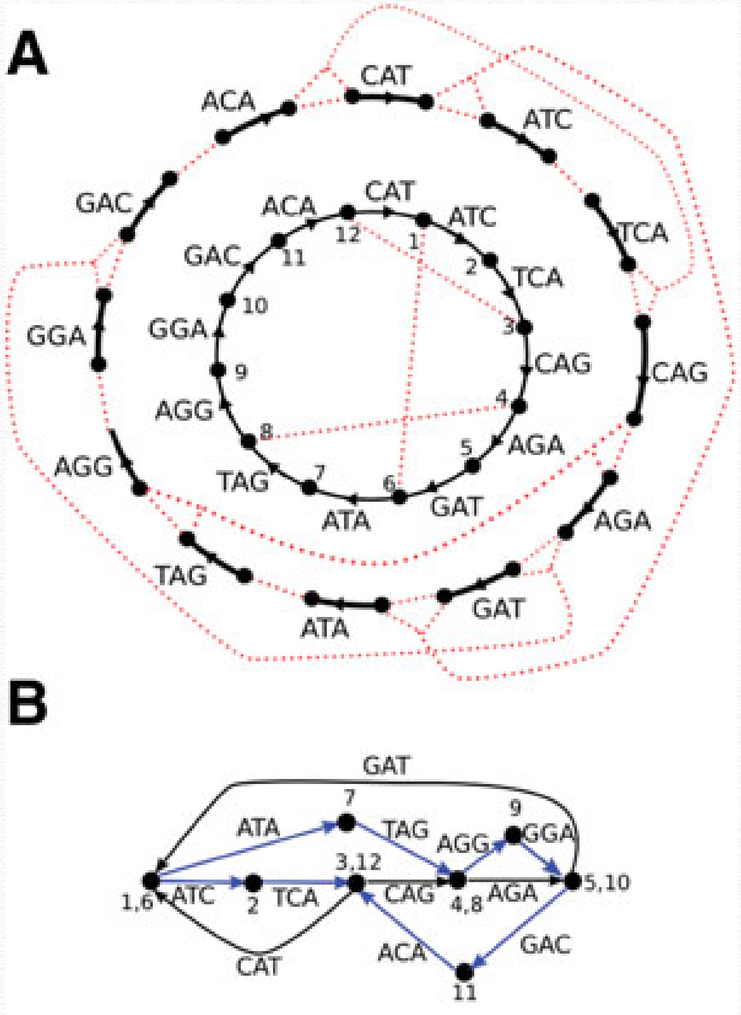
Sequencing remains at the core of genomics. Applications in- proach. Because of their length, they must be produced in large

SPAdes

h-paths, h-reads



3-mers Graph


A
CAT
ATC
TCA
CAG
AGA
GAT
ATA
TAG
AGG
GGA
GAC
ACA
1
2
3
4
5
6
7
8
9
10
11
12
B
GAT
ATA
TAG
AGG
GGA
ATC
TCA
CAG
AGA
GAC
ACA
CAT
1,6
2
3,12
4,8
5,10
11

3,4-mers Graph