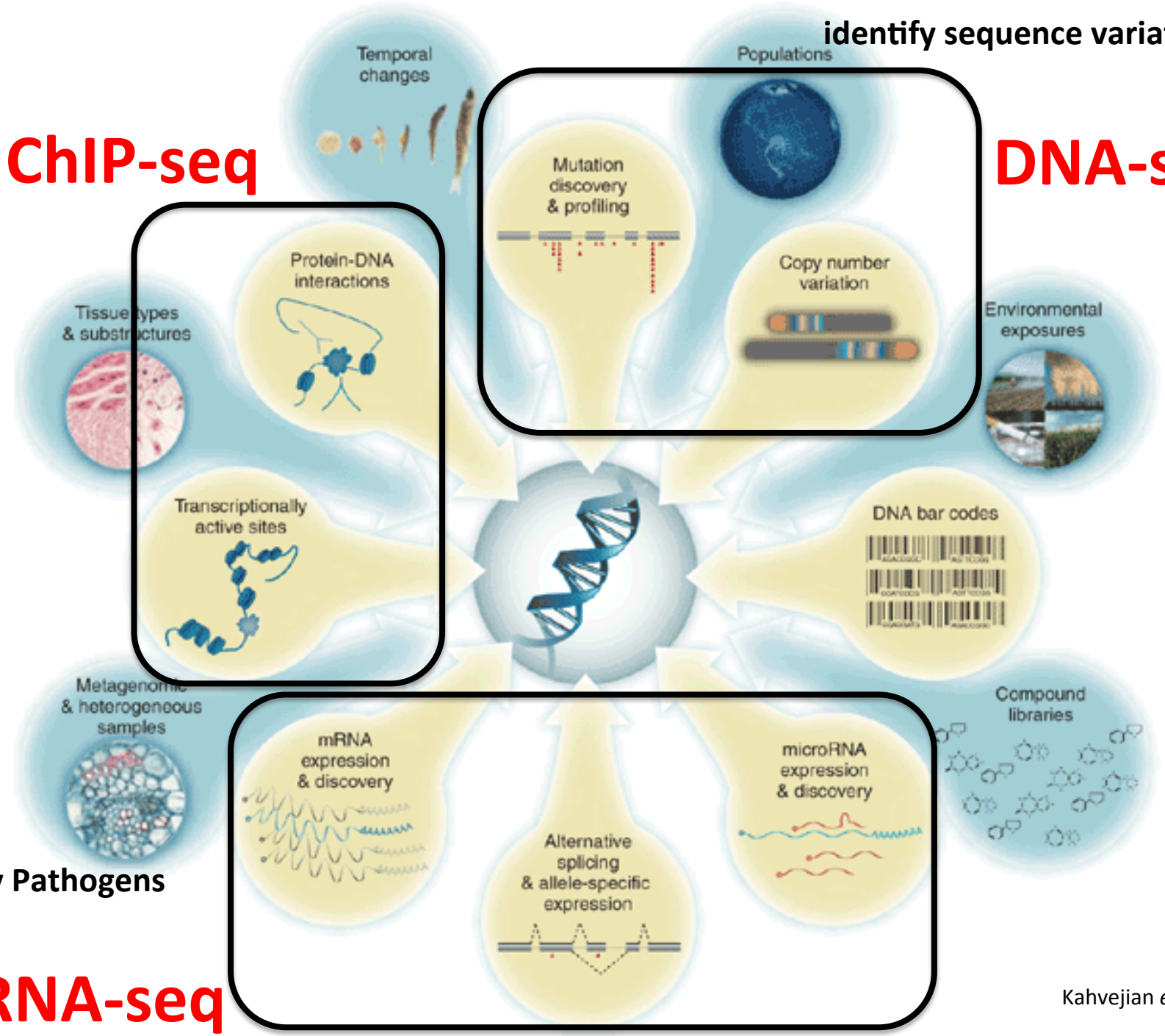
Next-generation sequencing

Lecture 4

identify sequence variations



Identify Pathogens

Kahvejian *et al*, 2008

NGS

- Introduction to the background
- NGS workflow and accuracy
- Data format, quality control, data management
- Assembly: **sequence assembly** refers to aligning and merging fragments of a much longer DNA sequence in order to reconstruct the original sequence.
- RNA-seq
  - Aligner
  - Analysis tools
  - Applications, such as MiRNA
- Chip-seq
  - Applications

Goals of Assembly

- Reconstruction of unknown genomes
  - viruses, bacteria
  - Eucaryotes (individual genomes)
  - metagenomics (environmental samples, microbial communities)
- RNA-Seq => transcriptomes
  - for organisms without reference genomes
  - novel transcripts
  - fusion genes
  - viral integration
- Local assembly for detection of insertions and genomic rearrangements
  - unmapped reads from WGS
  - transposable elements
  - viral integration

Next-generation sequencing

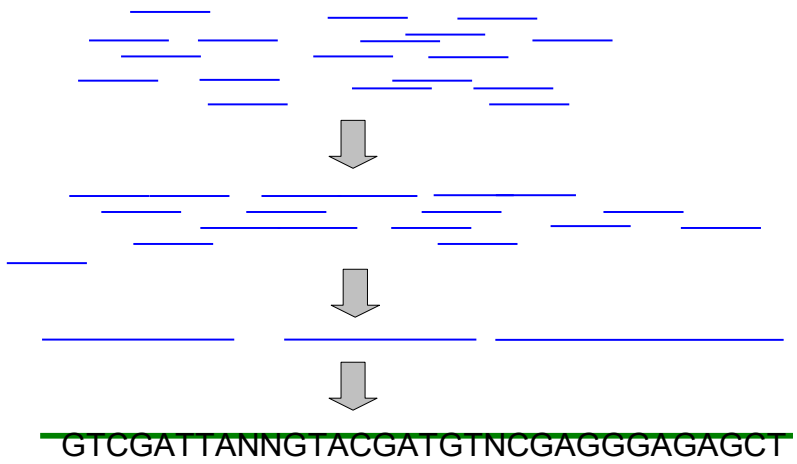
- Much higher throughput (1-4gbps / day)
- Lower cost / base pair
- Inherent ability to do paired-end (mate-pair) sequencing

Assembly

- **sequence assembly** refers to aligning and merging fragments of a much longer DNA sequence in order to reconstruct the original sequence.
- Taking many copies of a book, passing each of them through a shredder with a different cutter, and piecing the text of the book back together just by looking at the shredded pieces.
- Combines short sequencing reads into contigs based on sequence similarity and overlap between reads.
- Find the shortest common sequence of a set of reads

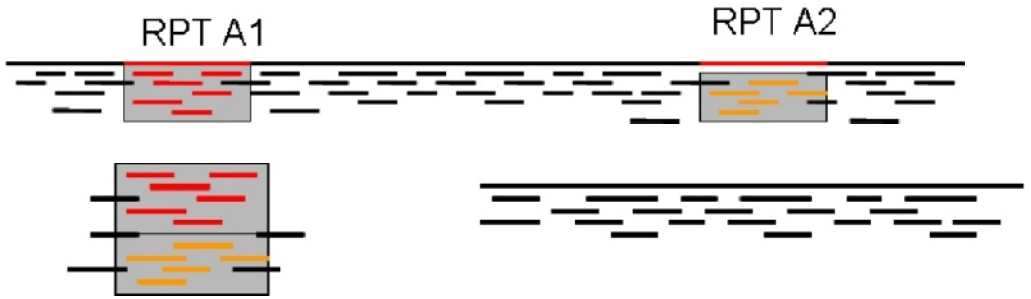
Assembly

- **Spanner**: single read that spans a repeat instance with sufficient unique sequence on either side of the repeat.
- **Contig**: contiguous sequence formed by several overlapping reads with **no gaps**.
- **Supercontig (scaffold)**: ordered and oriented set of contigs, usually by mate pairs. Relative distance known => fill gaps between contigs with "NNNNNNNNN…"
- **Consensus sequence**: sequence derived from the multiple alignment of reads in a contig.



Challenges for Assembly

- Repeats or similar parts in the genome. The reads originating from different copies of a repeat appear identical to the assembler and cause assembly errors.



- Non-random shearing
- Lose DNA fragments during library preparation
- Bias during amplification
- Very short fragment lengths (25-200bps)
- High error rate

Assembly

- Assembly algorithms
- De novo whole genome assembling strategies
- Mapping assembling strategies

Assembly algorithms

- Assembly, finding an optimal path that connect all short reads (or part of short reads) once time, is NP-hard problem.
- No efficient solution.
  - We need to use some approximation algorithm.

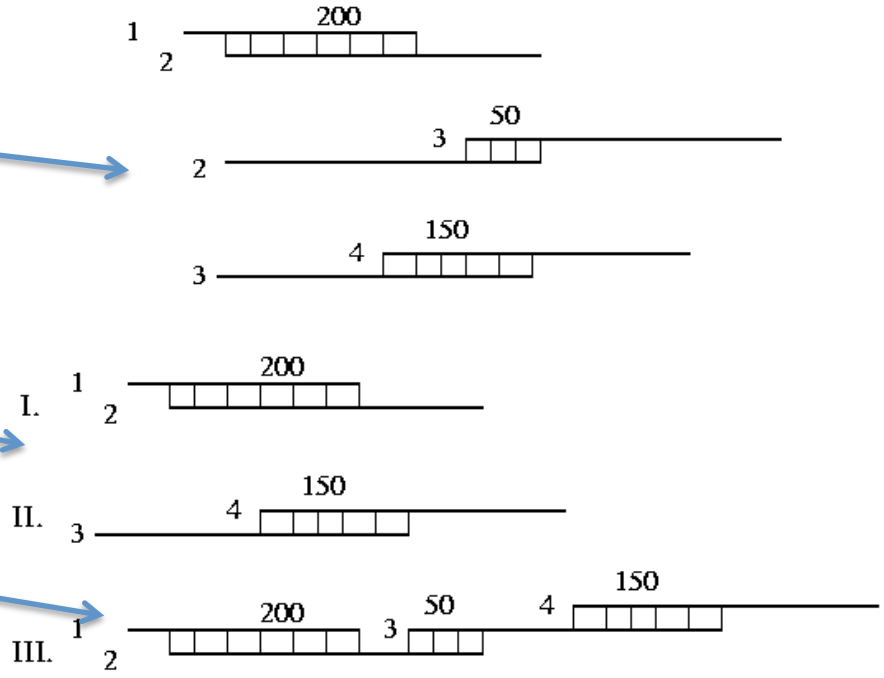
Assembly Algorithms

Main algorithm used:

- Greedy algorithms
- Overlap Layout Consensus
- De brujin graphs

Greedy Assembly

- Build a rough map of fragment overlaps (pairwise alignment)
- Pick the largest scoring overlap
- Merge the two fragments
- Repeat until no more merges can be done

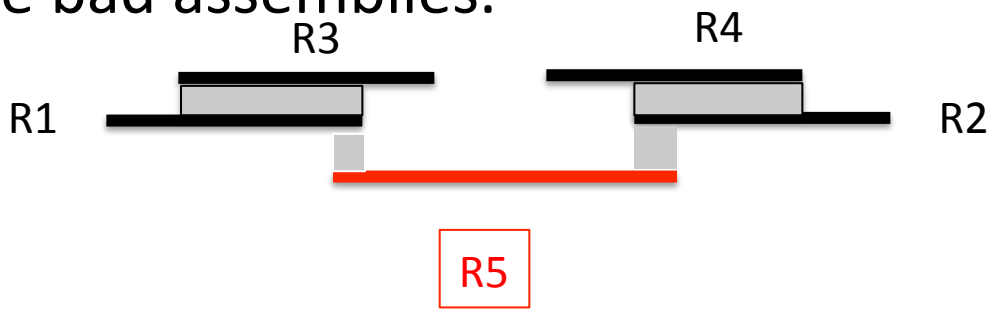


Alternative Greedy Algorithm

- Instead of calculating overlaps of all reads:
  1. Start with a random read as an initial contig (seed)
  2. Go over all unassembled reads, pick the one that best fits the 3' end of the contig and elongate the contig by this read.
  3. Repeat step 2 until no elongation is possible anymore.
  4. Repeat step 2 for the 5' end of the reverse complement of the contig.
  5. Stop in case of a conflict (fork)
     - if two reads that do not overlap with each other could elongate the contig equally well.

Greedy Assembly

- Advantages:
  - Simple and easy to implement
  - effective
- Disadvantages
  - Since local information is considered at each step, the assembler can be easily confused by complex repeats, leading to mis-assemblies.
  - Local approach. Easy to be trapped into a local optimal solution (local minimum).
  - Early mistakes create bad assemblies.



Greedy Assemblers

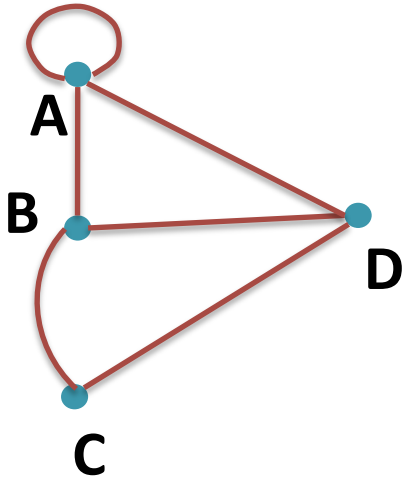
- TIGR Assembler: ftp://ftp.jcvi.org/pub/software/assembler/, Sanger, 2003
- SSAKE: http://www.bcgsc.ca/platform/bioinfo/software/ssake , small genomes, Solexa/Illumina, 2007
- SHARCGS: http://sharcgs.molgen.mpg.de/ , small genomes, Solexa/Illumina, SOLiD, Sanger, 454, 2007
- VCAKE: http://sourceforge.net/projects/vcake , small genomes, Solexa/Illumina, 2007
- Phrap: http://www.phrap.org/ , Sanger, 454, Solexa, 1995-2008

Assembly Algorithms

Main algorithm used:

- Greedy algorithms
- Overlap Layout Consensus
- De brujin graphs

Graph model for Assembly



Graphs are made up by vertices (nodes) and edges (links).
***G*=(*V*,*E*)**
***V*: a finite set of vertices.**
***E*: edges of the graph**

Graph for Assembly:

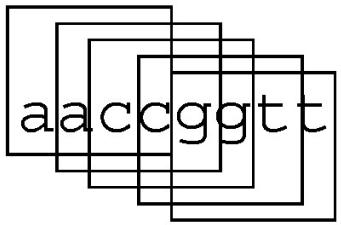
- **Graph model. A node is a read or a k-mer subsequence**
- **Edge: if there are over lap between two k-mer subsequence**

(a)

```
aaccgg
  ccggtt
```

(b)

(c)



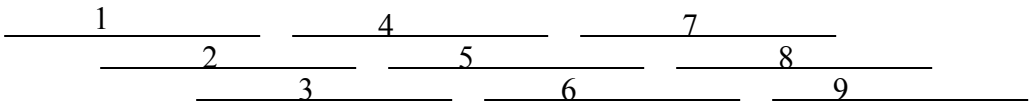
Miller, 2010

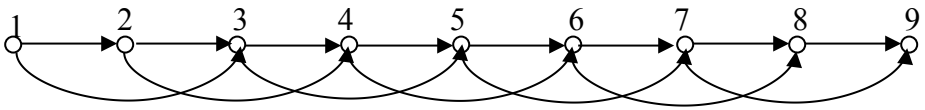
Overlap-layout-consensus

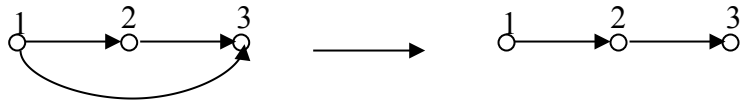
Main entity: read
Relationship between reads: overlap

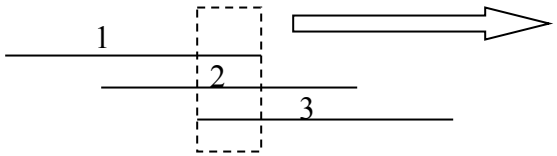
Graph model:

- **A node is a**
- **Edge: if there are overlap between two reads**







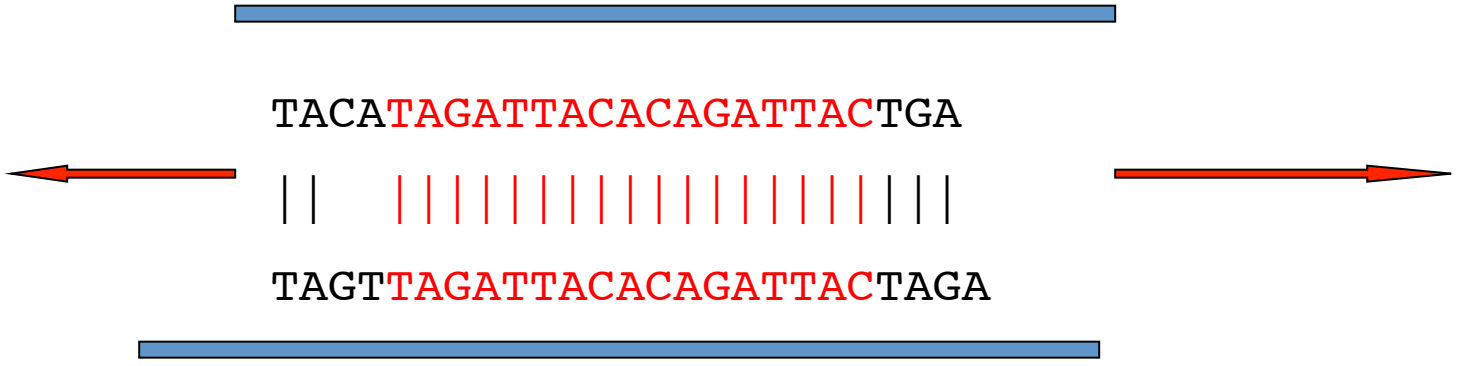


ACCTGA
ACCTGA
AGCTGA
ACCAGA

Overlap-layout-consensus

Step 1: Find Overlapping Reads

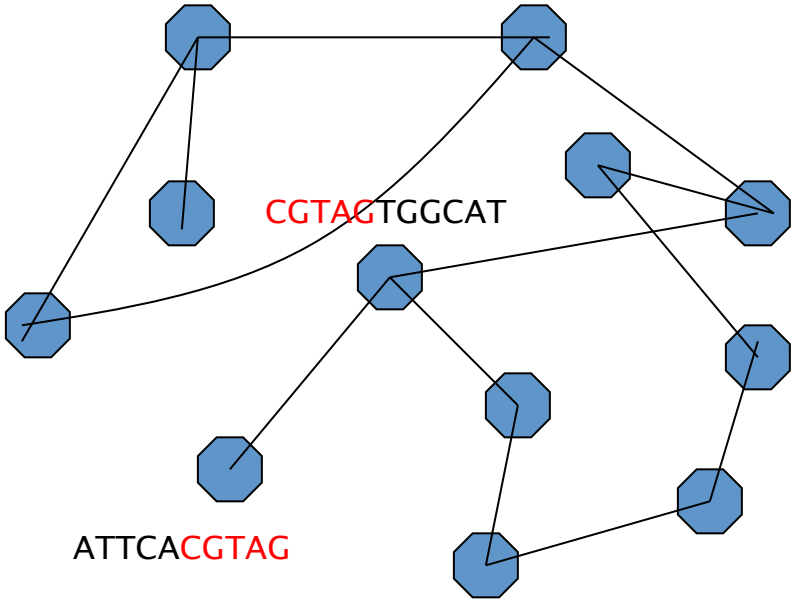
- What reads are intersecting ?
- Issues:
  1. Need efficient alignment algorithm (parallelization and index based strategies)
  2. Doesn't scale well when number of read is high
  3. Use seed based alignment with extension



Overlap-layout-consensus

Step 2: Construct overlap graph

- A graph is constructed:
  - Nodes are reads
  - Edges represent overlapping reads
- Need to simplify graph, such as remove redundant nodes and edges.



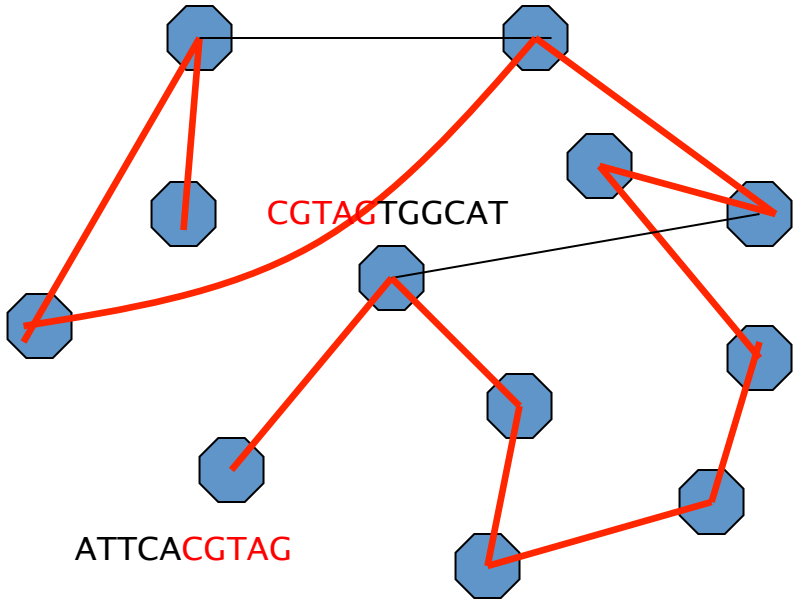
Overlap graph

Overlap-layout-consensus

Step 3: Consensus stage, Find Contigs

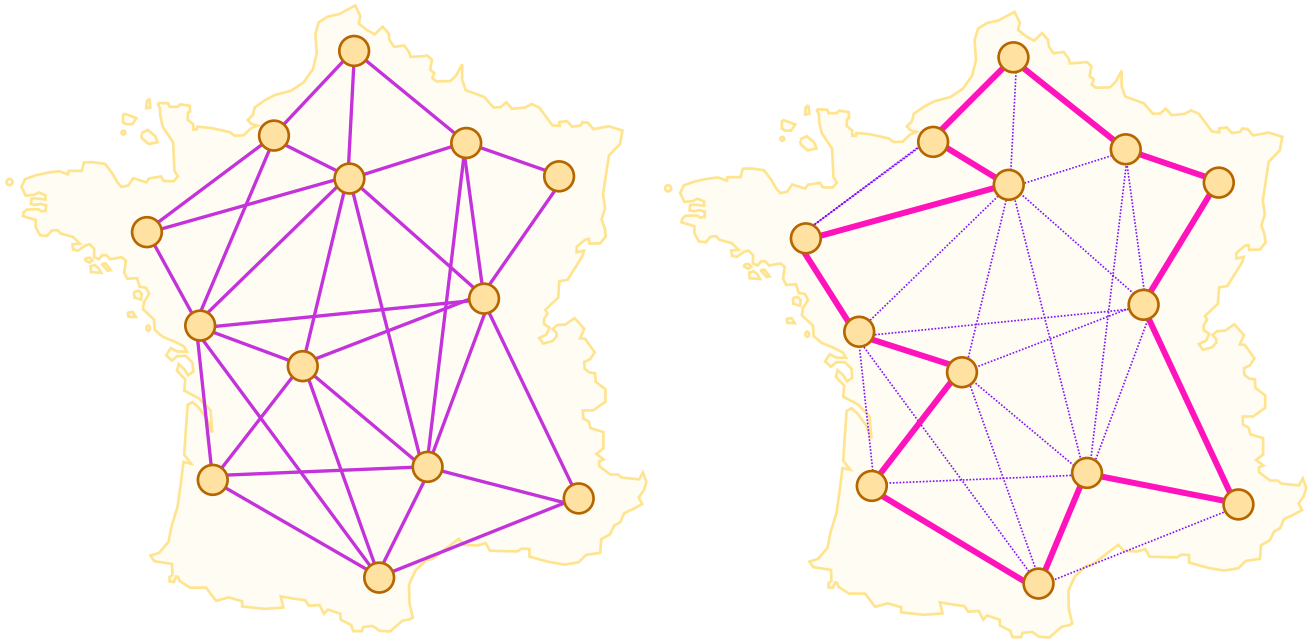
Try to find the Hamiltonian path:

- A path in the graph contains each node exactly once.
- Following the Hamiltonian path, combine the overlapping sequences in the nodes into the sequence of the genome
- Computationally expensive (NP-hard problem)



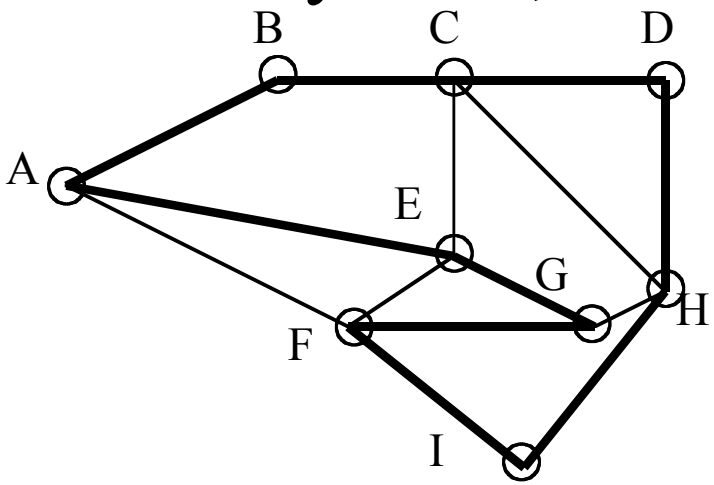
Travelling salesman problem

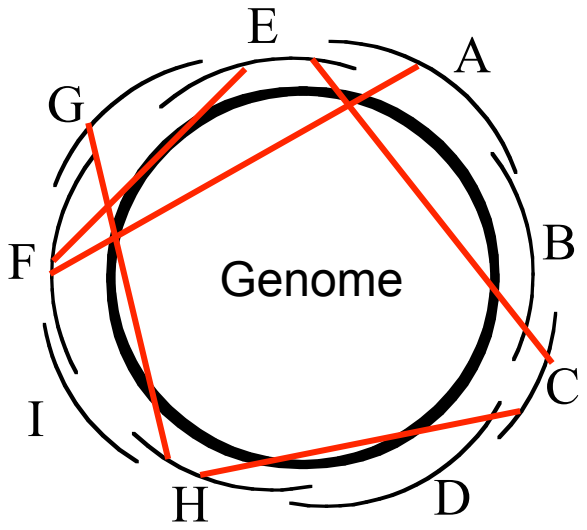
- The travelling salesman problem (TSP) asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?



Circular genome

- Hamiltonian circuit: visit each node (city) exactly once, returning to the start





Overlap-layout-consensus

- Better than Greedy algorithm. It can generate correct order of contigs that the Greedy algorithms may have errors.
- No efficient algorithm to find the Hamiltonian path
- Short fragment length = very small overlap therefore many false overlaps.
- Overlap discovery is sensitive to minimum overlap length and minimum percent identity required for an overlap.
- Overlap discovery is also time consuming.
- Large number of reads + short overlap + higher error are challenging for the overlap - layout - consensus approach
- Can't assemble repeat longer than read length
- It is mostly used with Sanger or 454 data.

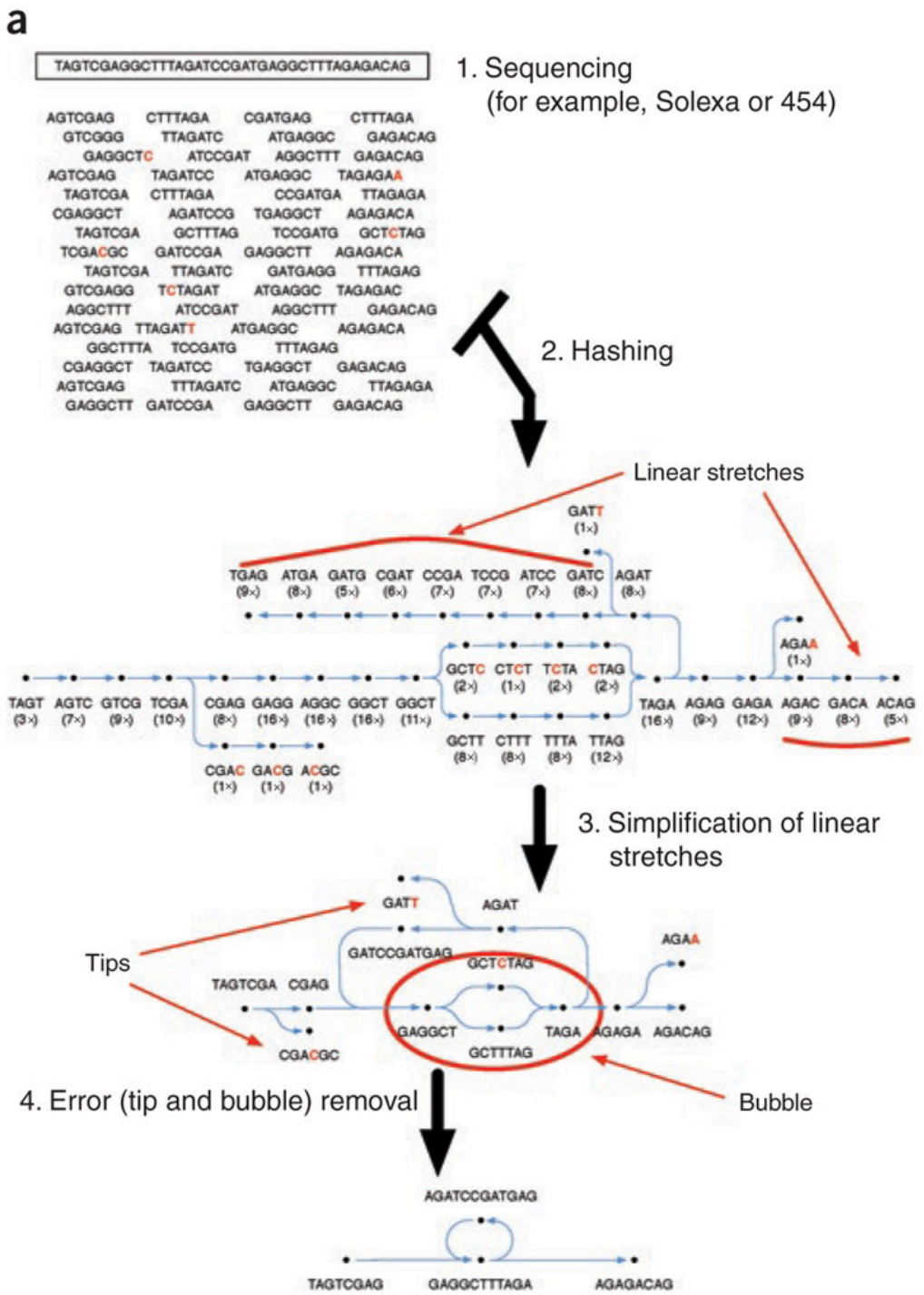
Overlap-layout-consensus

- Celera (CABOG): http://www.jcvi.org/cms/research/projects/celera-assembler/overview/ , large genome, Sanger, 454,Solexa, 2004/2010
- Newbler: http://www.454.com/ , 454, Sanger, 2009
- Mira: http://sourceforge.net/apps/mediawiki/mira-assembler/ Sanger, 454, Solexa, 1998/2011
- Edena: http://www.genomic.ch/edena.php Snger, 454, 2008/2013

Assembly Algorithms

Main algorithm used:

- Greedy algorithms
- Overlap Layout Consensus
- De bruijn graphs

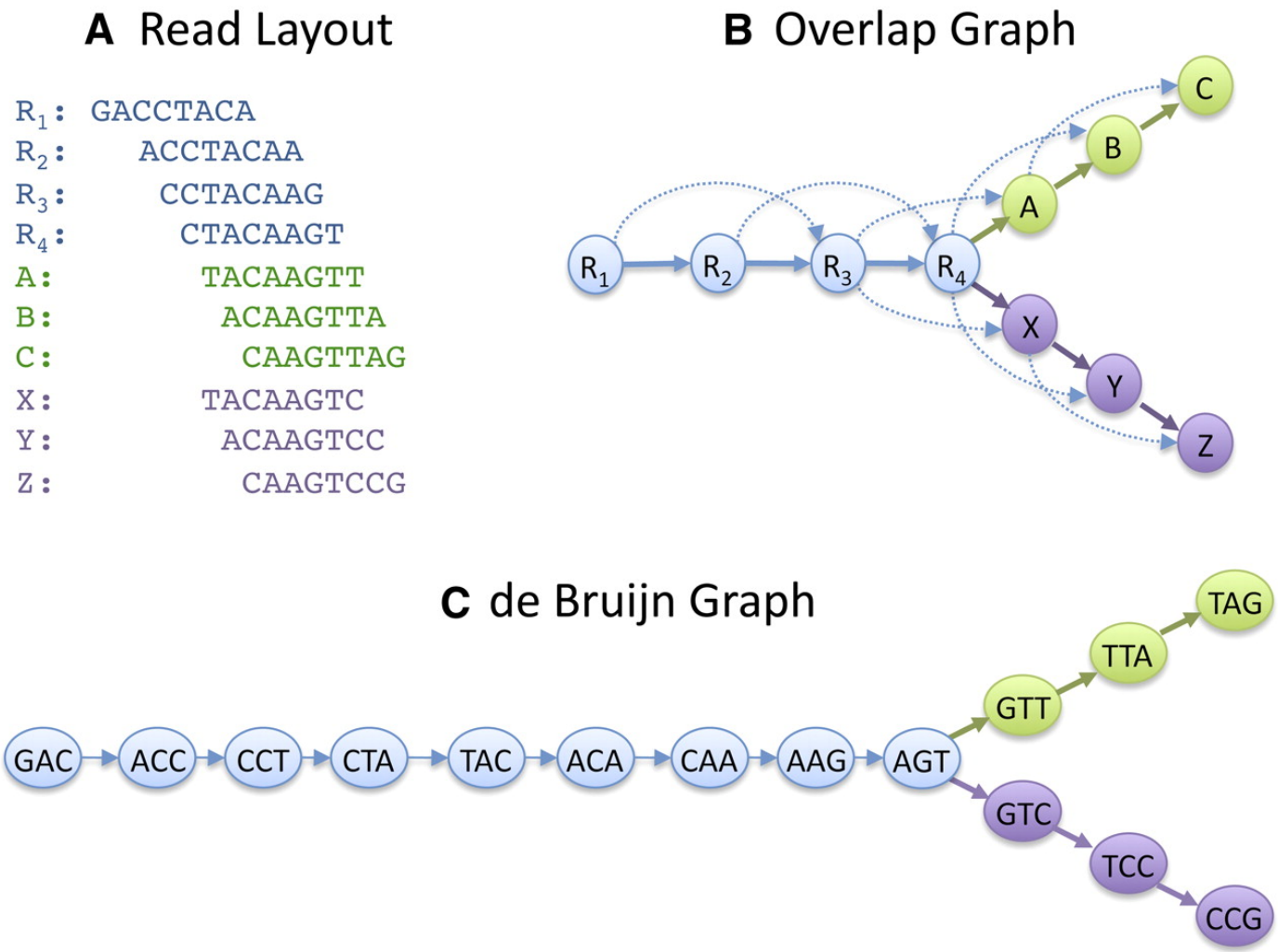


De Bruijn Graphs

1. Get k-bp (k-mer) subsequences for reads.
2. k-mers in the reads are collected into nodes and the coverage at each node is recorded. Link two k-mer nodes if they have overlap.
3. the graph is simplified to combine nodes that are associated with the continuous linear stretches into single, larger nodes of various k-mer sizes.
4. error correction removes the tips and bubbles that result from sequencing errors and creates a final graph structure that accurately and completely describes in the original genome sequence.

Flicek, Nature Methods, 2009

Differences between an overlap graph and a de Bruijn graph for assembly.



Schatz M C et al. Genome Res. 2010;20:1165-1173

De Bruijn Graphs example

- For the purposes of illustration, we can use human readable text to explore how assemblies work.
- This is an example taken from Leipzig et al 2001.
- In it he uses the opening paragraph from Dickens' "A tale of two cities".
- It is an appropriate example because like genomes, it contains strings that are repeat over and over.

"It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity,.... "

Dickens, Charles. A Tale of Two Cities. 1859. London: Chapman Hall

Velvet example courtesy of J. Leipzig 2010

De Bruijn Graphs example

itwasthebestoftimesitwastheworstoftimesitwastheageofwisdomitwastheageoffoolishness…

Generate random ‘reads’

How do we assemble?

fincreduli geoffoolis Itwasthebe Itwasthebe geofwisdom itwastheep epochofinc timesitwas stheepocho nessitwast wastheageo theepochof stheepocho hofincredu estoftimes eoffoolish lishnessit hofbeliefi pochofincr itwasthewo twastheage toftimesit domitwasth ochofbelie eepochofbe eepochofbe astheworst chofincred theageofwi iefitwasth ssitwasthe astheepoch efitwasthe wisdomitwa ageoffooli twasthewor ochofbelie sdomitwast sitwasthea eepochofbe ffoolishne eofwisdomi hebestofti stheageoff twastheepo eworstofti stoftimesi theepochof esitwasthe heepochofi theepochof sdomitwast astheworst rstoftimes worstoftim stheepocho geoffoolis ffoolishne timesitwas lishnessit stheageoff eworstofti orstoftime fwisdomitw wastheageo heageofwis incredulit ishnessitw twastheepo wasthewors astheepoch heworstoft ofbeliefit wastheageo heepochofi pochofincr heageofwis stheageofw fincreduli astheageof wisdomitwa wastheageo astheepoch olishnessi astheepoch itwastheep twastheage wisdomitwa fbeliefitw bestoftime epochofbel theepochof sthebestof lishnessit hofbeliefi Itwasthebe ishnessitw sitwasthew ageofwisdo twastheage esitwasthe twastheage shnessitwa fincreduli fbeliefitw theepochof mesitwasth domitwasth ochofbelie heageofwis oftimesitw stheepocho bestoftime twastheage foolishnes ftimesitwa thebestoft itwastheag theepochof itwasthewo ofbeliefit bestoftime mitwasthea imesitwast timesitwas orstoftime estoftimes twasthebes stoftimesi sdomitwast wisdomitwa theworstof astheworst sitwasthew theageoffo eepochofbe theageofwi foolishnes incredulit ofbeliefit chofincred beliefitwa beliefitwa wisdomitwa eageoffool eoffoolish itwastheag mesitwasth epochofinc ssitwasthe itwastheep astheageof stheageoff sitwasthee thebestoft oolishness heepochofb ochofbelie wastheepoc bestoftime mesitwasth ebestoftim pochofincr

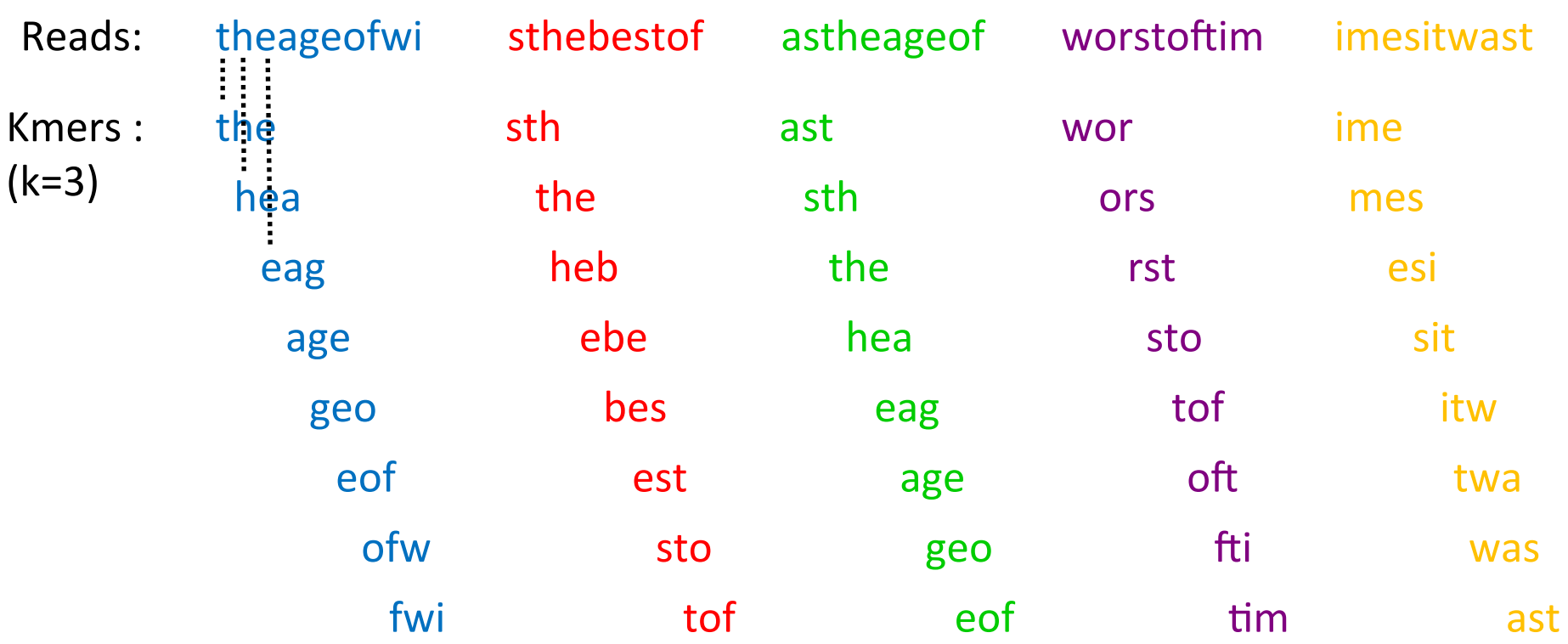
…etc. to 10’s of millions of reads

Traditional all-vs-all comparisons of datasets this size require immense computational resources.

De Bruijn solution: Construct a graph efficiently

De Bruijn Graphs

Step 1: “Kmerize” the data

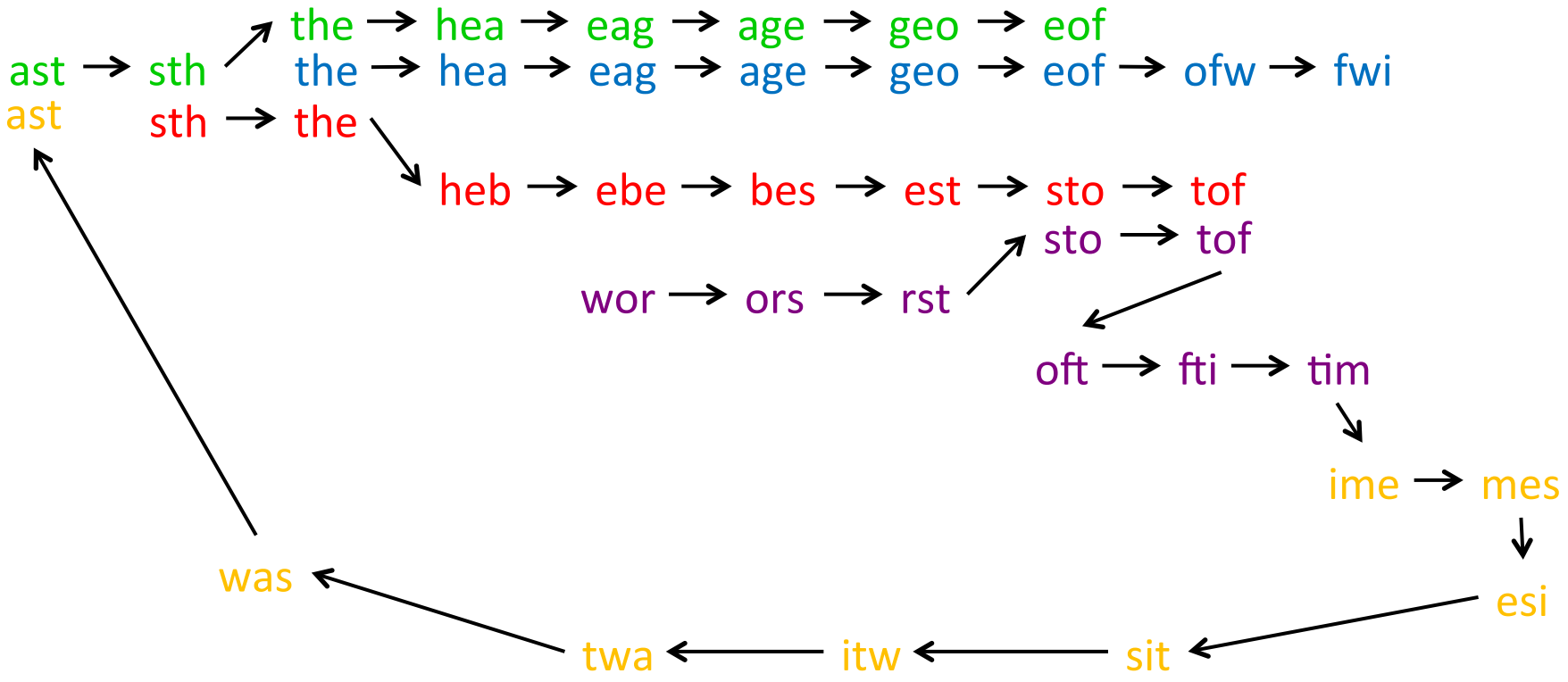


…..etc for all reads in the dataset

De Bruijn Graphs

Step2 Build the graph

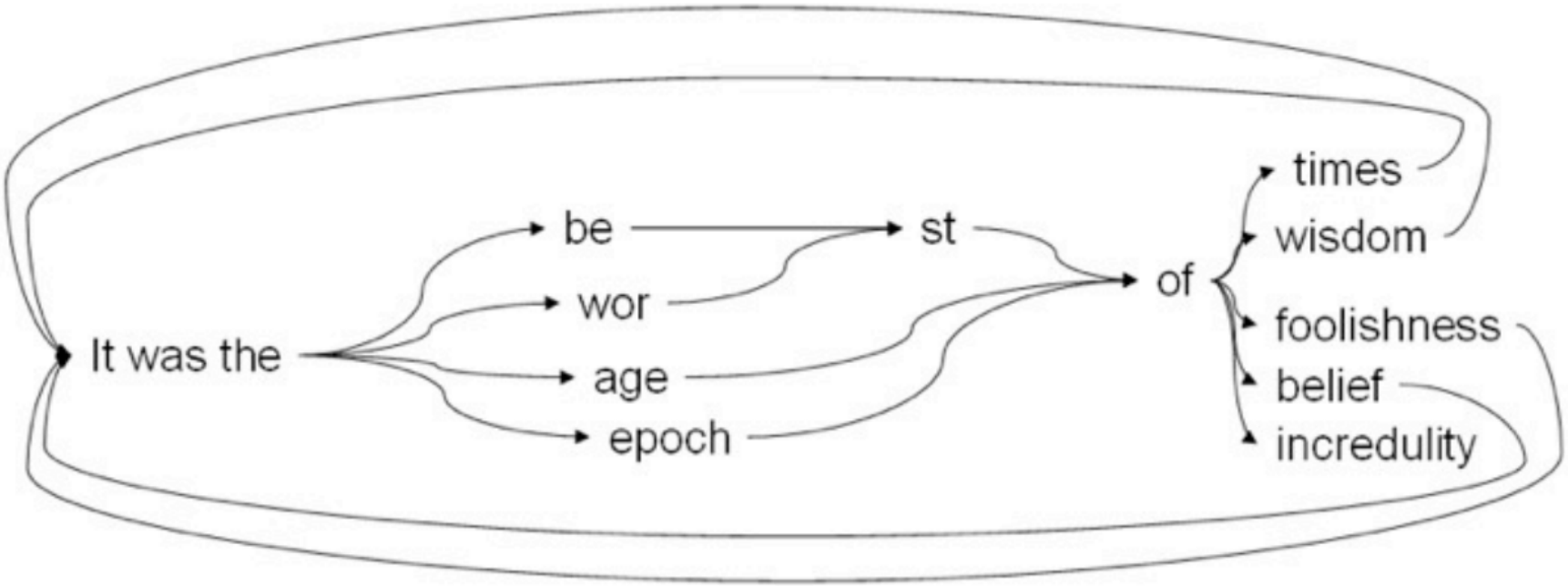
Look for k-1 overlaps: given by the reads



…..etc for all ‘kmers’ in the dataset

De Bruijn Graphs

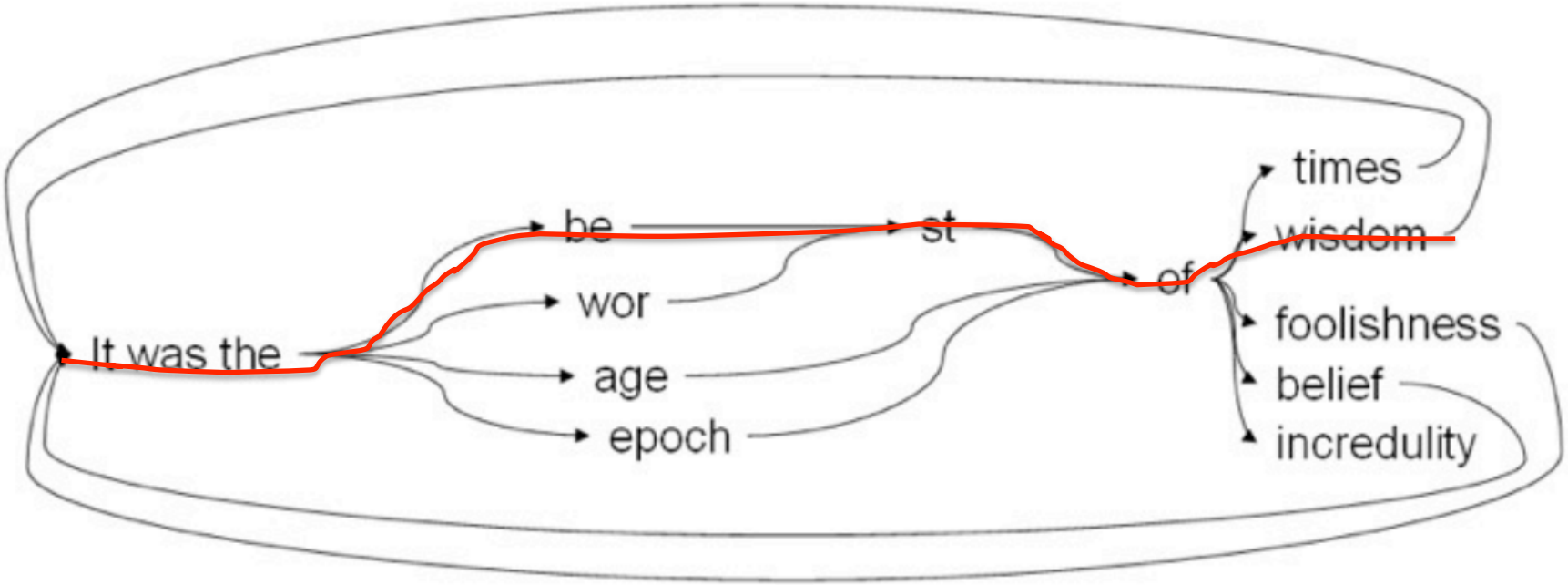
step3: simplify the graph



- The final step is to remove redundancy, result in the final De Bruijn Graph representation of our genome.
- the overlaps between reads are implicit in the graph, so all the millions v.s millions of comparisons are not required.
- On the downside, information is lost as repetitive sequences are “collapsed” into a single representation.

De Bruijn Graphs

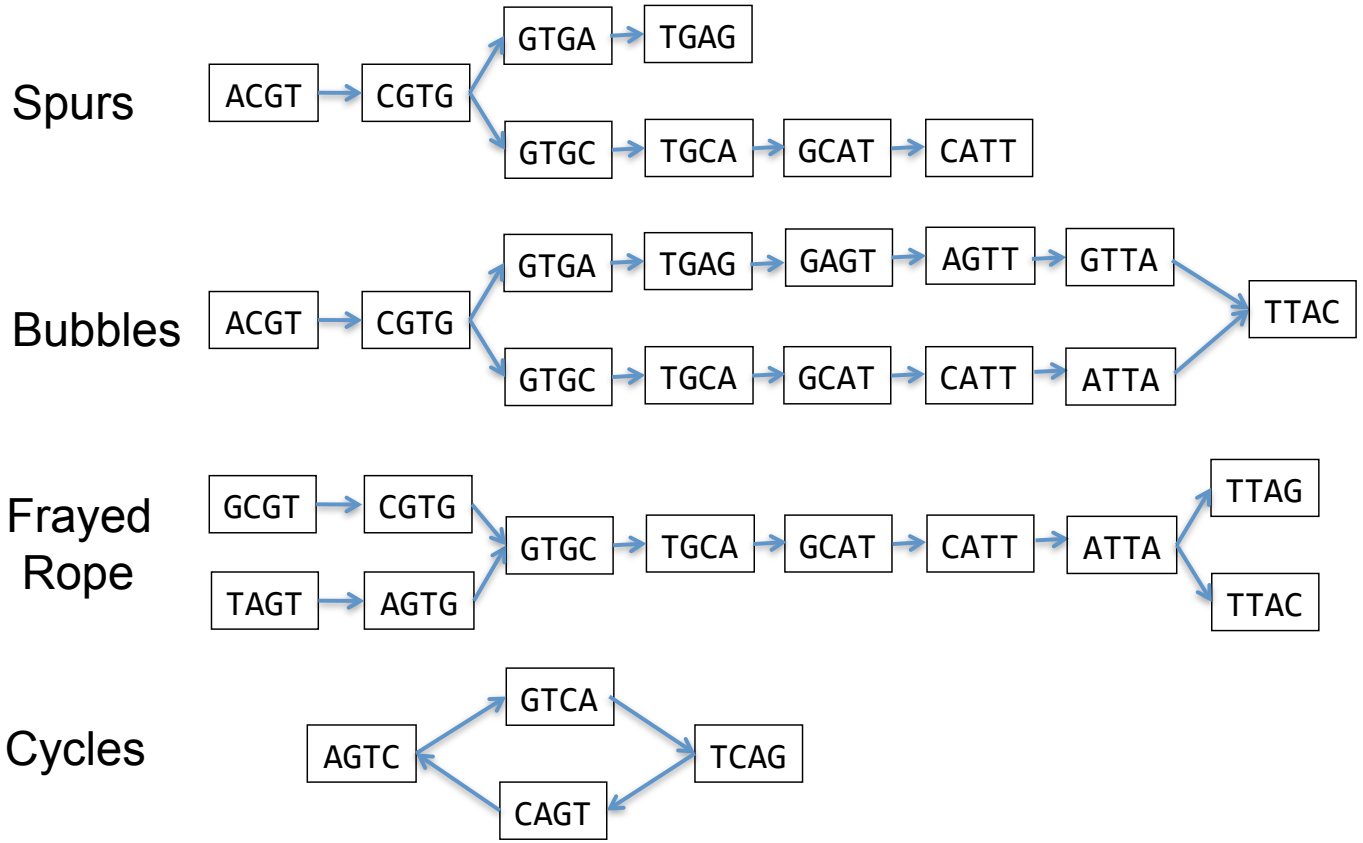
step4: Create contigs



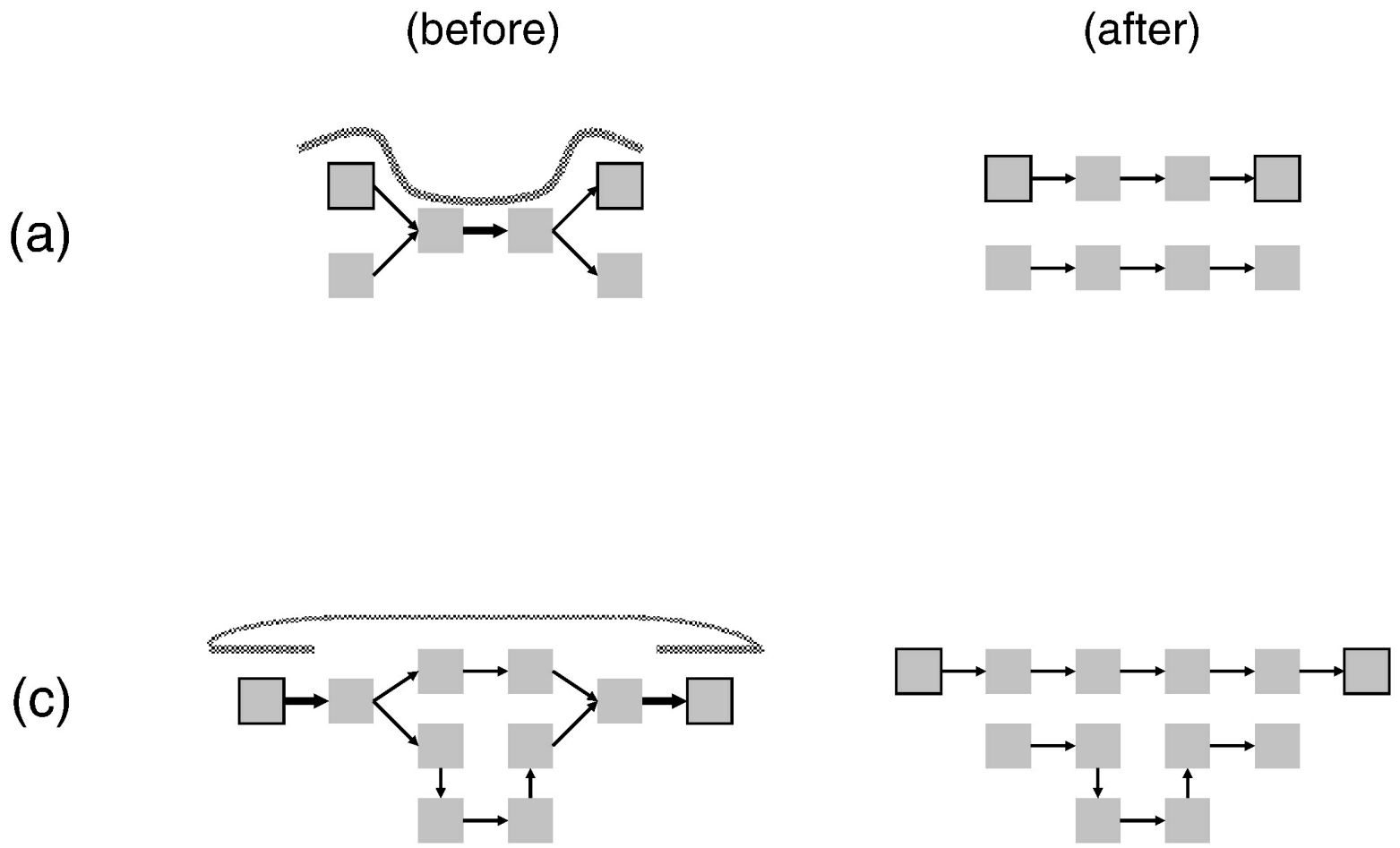
Find the Hamiltonian path or cycle in the De Bruijn graph. Each path in the simplified De Bruijn graph is a Hamiltonian path. A Hamiltonian path is a contig.

Common Problems

- Spurs: dead-end sequences
- Bubbles: divergent paths that then converge
- Frayed rope: convergent then divergent paths
- Cycles: paths convergent upon themselves



Resolve graph complexity



Strengths and problems of De Bruijn approach

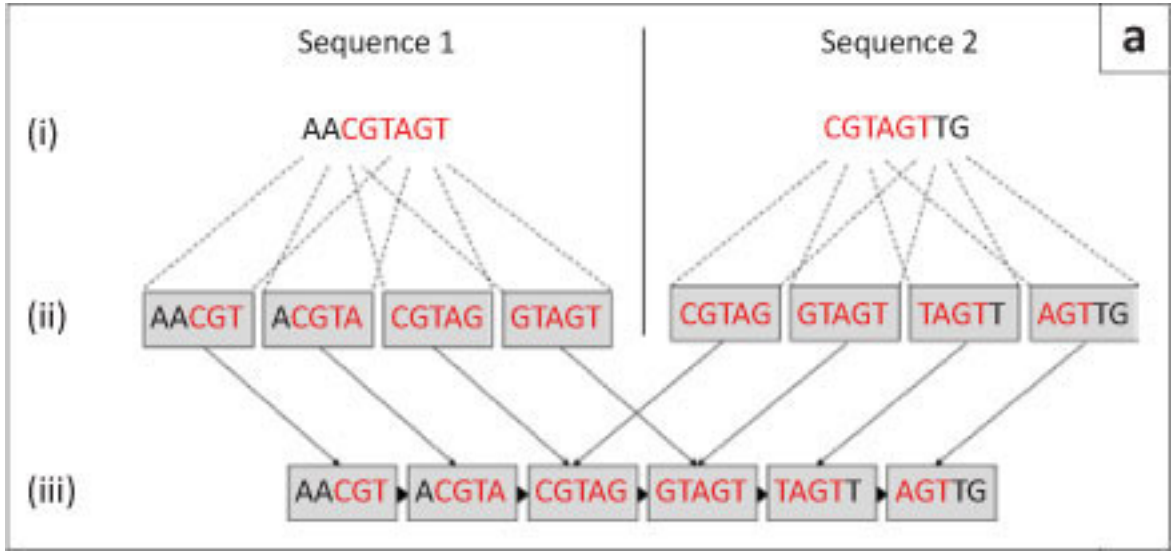
Strengths:

- No need to calculate the overlaps
- Size of the final graph is proportional to the genome size
- successfully for very short reads (<50bp)

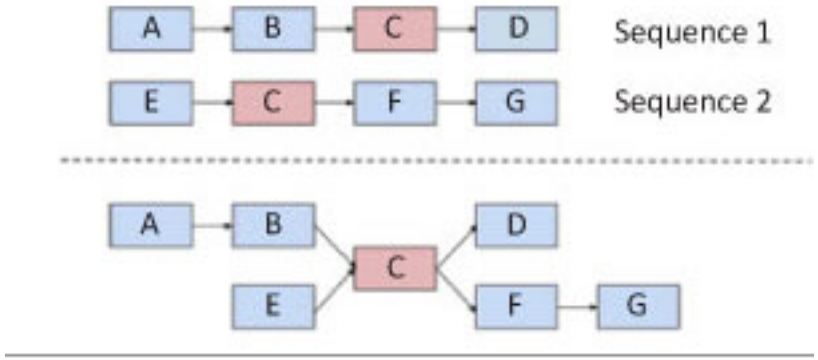
Problems:

- The main drawback to the de Bruijn approach is the loss of information caused by decomposing a read into a path of *k*-mers.
- require an enormous amount of computer space
- Can only resolve k long repeat
- Loose connectivity when create the contigs

Strengths and problems of De Bruijn approach



Schlebusch, 2012



Pros: correctly links two sequences without having to compute overlap score. (above case)

Cons: two sequences are linked without any real overlap. (left case)

De Bruijn Assemblers

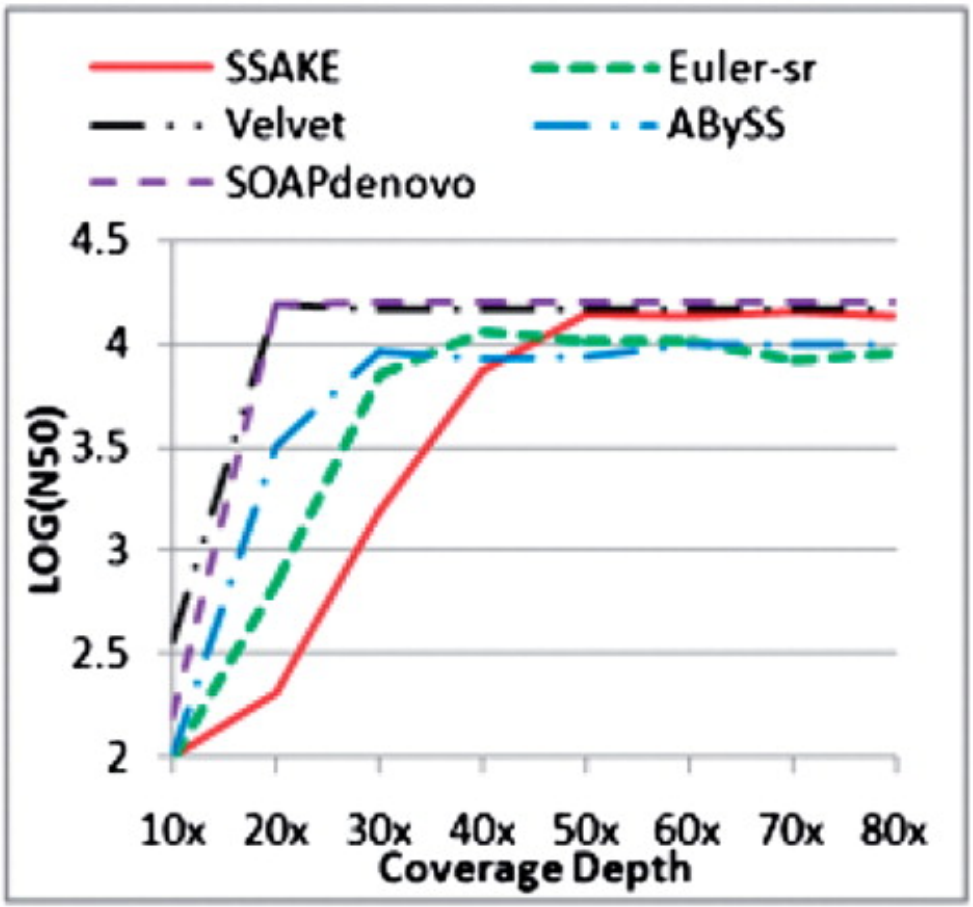
- Euler: http://nbcr.sdsc.edu/euler/ , Sanger, 454, 2001-2006
- **Velvet**: http://www.ebi.ac.uk/~zerbino/velvet/, small genomes, Sanger, 454, Solexa, SOLiD, 2007-2009 (very good for small genome)
- **ABySS**: http://www.bcgsc.ca/platform/bioinfo/software/abyss, large genome, Solexa, SOLiD, 2008-2011 (for very large genome)
- **SOAP-denovo**: http://soap.genomics.org.cn/soapdenovo.html, Solexa, 2009
- ALLPATH-LG: http://www.broadinstitute.org/software/allpaths-lg/blog/, large genome, Solexa, SOLiD, 2011 (very good performance bu require 2 lib of different insert sizes)
- IDBA-UD: http://i.cs.hku.hk/~alse/hkubrg/projects/idba_ud/, Sanger, 454,Solexa, 2010 (metagenomic, doesn't rely on coverage to remove error)

Comparison of Assembly tools

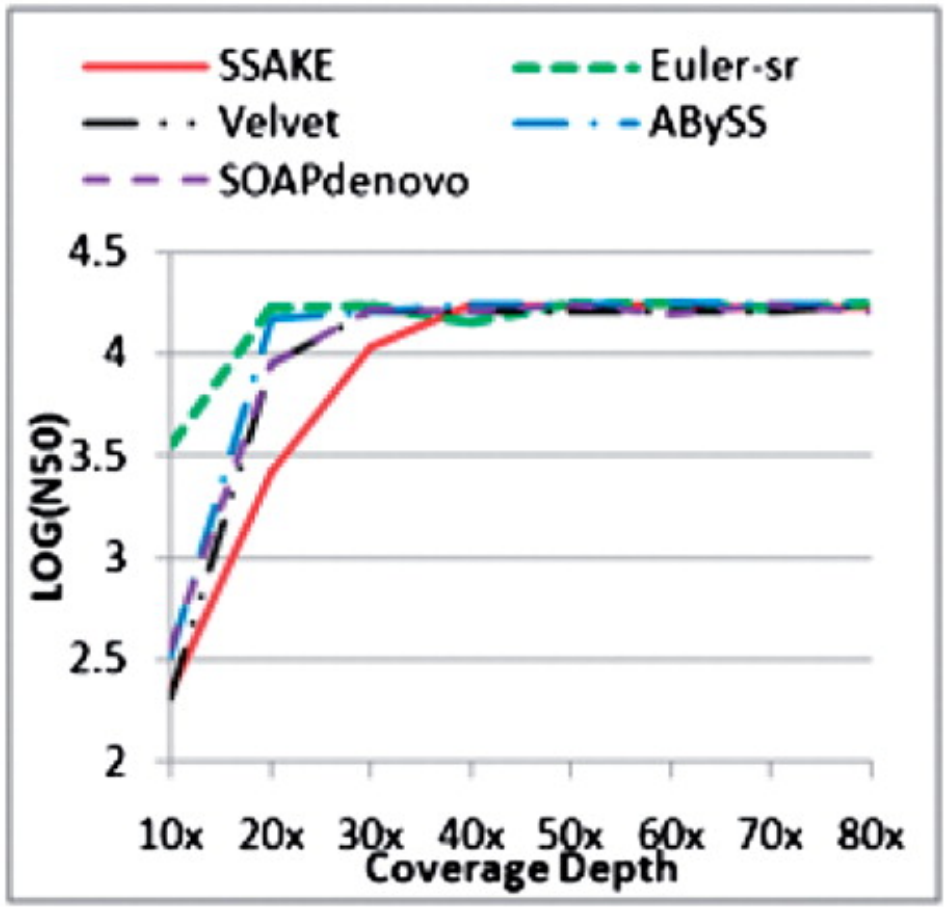
Algorithm Feature	Greedy Assemblers	OLC Assemblers	DBG Assemblers
Approaches to graph construction			
Implicit	SSAKE, SHARCGS, VCAKE		
Reads as graph nodes		CABOG, Newbler, Edena	
K-mers as graph nodes			Euler, Velvet, ABySS, SOAP
Simple paths as graph nodes			AllPaths
Multiple values of K			Euler
Multiple overlap stringencies	SHARCGS		
Approaches to graph reduction			
Filter overlaps		CABOG	
Greedy contig extension	SSAKE, SHARCGS, VCAKE		
Collapse simple paths		CABOG, Newbler	Euler, Velvet, SOAP
Erosion of spurs		CABOG, Edena	Euler, Velvet, AllPaths, SOAP
Transitive overlap reduction		Edena	
Bubble smoothing		Edena	Euler, Velvet, SOAP
Bubble detection			AllPaths
Reads separate tangled paths			Euler, SOAP
Break at low coverage			Velvet, SOAP
Break at high coverage		CABOG	Euler
High coverage indicates repeat		CABOG	Velvet
Special use of long reads		Shorty	Velvet
Graph partitions			
Partition by K-mers			ABySS
Partition by scaffolds			AllPaths

Miller, genomics, 2010, 95(6):315-27

Comparison of the effect of various coverage depths on average contig length



Read length =35



Read length =75

Lin Y et al. Bioinformatics 2011;27:2031-2037

Couple issues

- Try different assemblers and compare their results.
- Need a big fat memory computer (from 16GB to 1TB).
- Running time is long: from several hours to several days.

Running time

	Runtime (s)				
	Bench.Seq (Length: bp)	*E.coli* (4.6M)	*C.ele* (20.9M)	*H.sap*-2 (50.3M)	*H.sap*-3 (100.5M)
SE	SSAKE	2,776	---	---	---
	VCAKE	1,672	16,742	---	---
	Euler-sr	1,689	11,961	29,622	---
	Edena	895	8,450	17,043	---
	Velvet	205	1,003	2,786	6,098
	ABySS	265	1,300	3,307	6,608
	SOAPdenovo	62	253	560	1,029
PE	SSAKE	9,163	---	---	---
	Euler-sr	1,455	15,068	---	---
	Velvet	229	1,351	55,581	---
	ABySS	458	3,081	9,199	21,683
	SOAPdenovo	78	374	889	2,257

Lin Y et al. Bioinformatics 2011;27:2031-2037

'–' denotes runtime is too long (>10 days) to get assembly results

RAM used

	RAM (MB)				
	Bench.Seq (Length: bp)	*E.coli* (4.6M)	*C.ele* (20.9M)	*H.sap*-2 (50.3M)	*H.sap*-3 (100.5M)
SE	SSAKE	9,933	---	---	---
	VCAKE	4,099	17,408	---	---
	Euler-sr	1,536	7,065	13,312	---
	Edena	1,741	7,557	30,720	---
	Velvet	1,229	4,045	9,830	22528
	ABySS	1,126	3,993	8,909	18432
	SOAPdenovo	935	2,867	8,089	18227
PE	SSAKE	16,384	---	---	---
	Euler-sr	1,638	7,578	---	---
	Velvet	1,331	5,324	30,720	---
	ABySS	950	4,505	9,830	18,432
	SOAPdenovo	1,638	5,939	10,342	19,456

'–' denotes the RAM of computer is not enough

Lin Y et al. Bioinformatics 2011;27:2031-2037

Whole genome sequencing

- *De Novo* whole genome sequencing
- Mapping assembly (Reference-guided assembly) (Resequencing)