Next-generation sequencing

Lecture 5

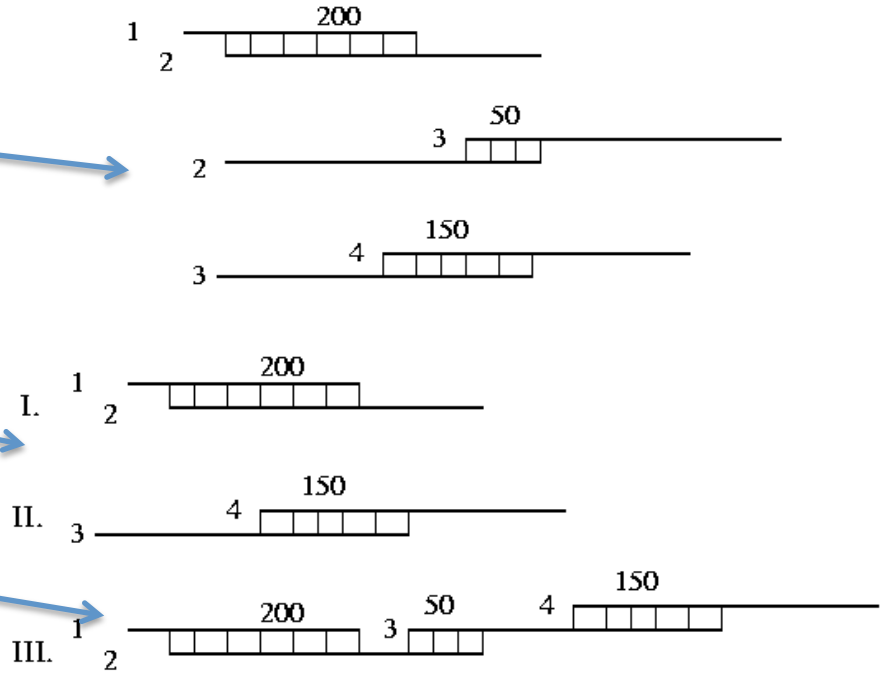
Assembly Algorithms

Main algorithm used:

- Greedy algorithms
- Overlap Layout Consensus
- De brujin graphs

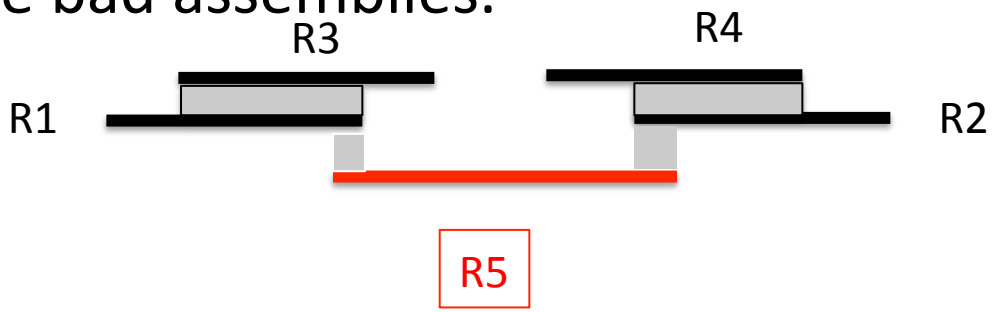
Greedy Assembly

- 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



Greedy Assembly

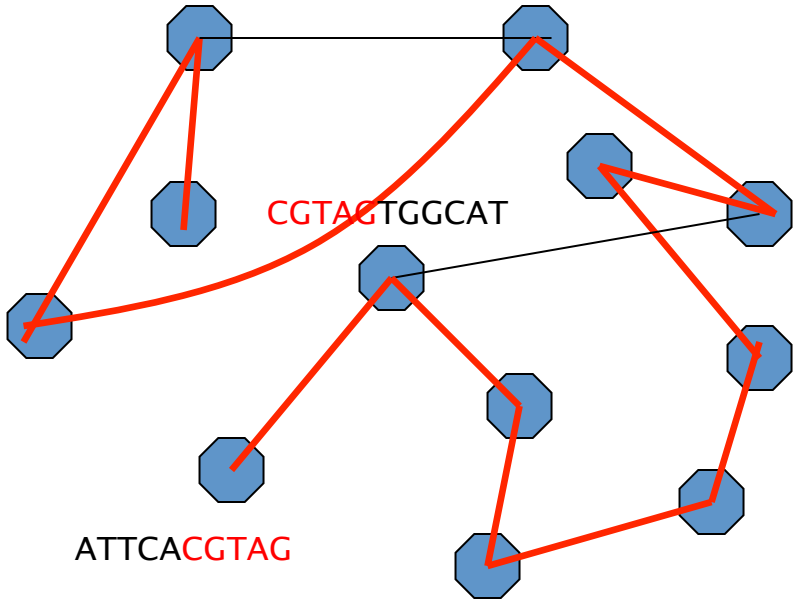
- Advantages:
  - Simple and easy to implement
  - effective
- Disadvantages
  - Since local information is considered at each step, the assembler can be easily confused by complex repeats, leading to mis-assemblies.
  - Local approach. Easy to be trapped into a local optimal solution (local minimum).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



Overlap-layout-consensus

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)



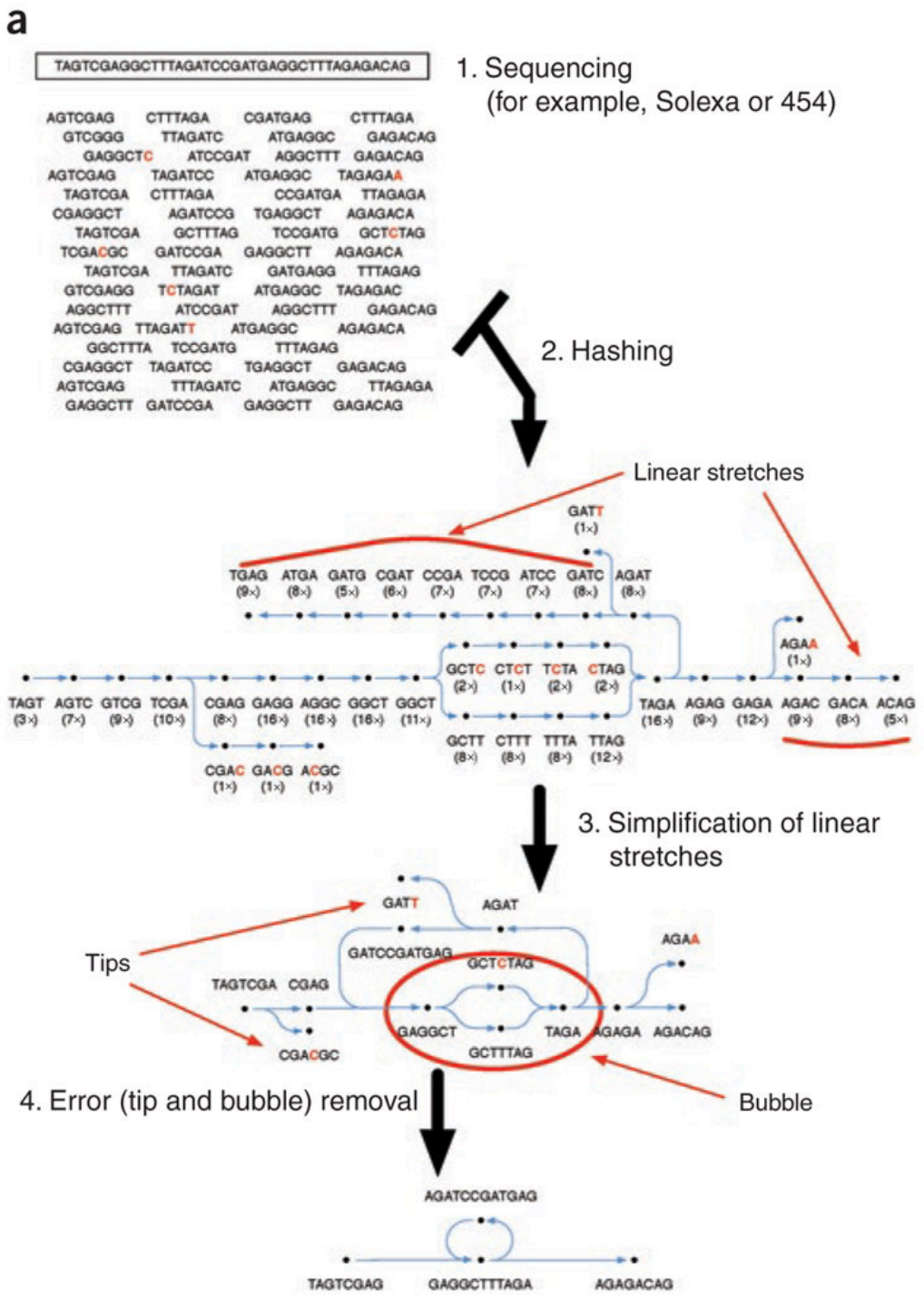
Overlap-layout-consensus

- 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.

Assembly Algorithms

Main algorithm used:

- Greedy algorithms
- Overlap Layout Consensus
- De bruijn graphs

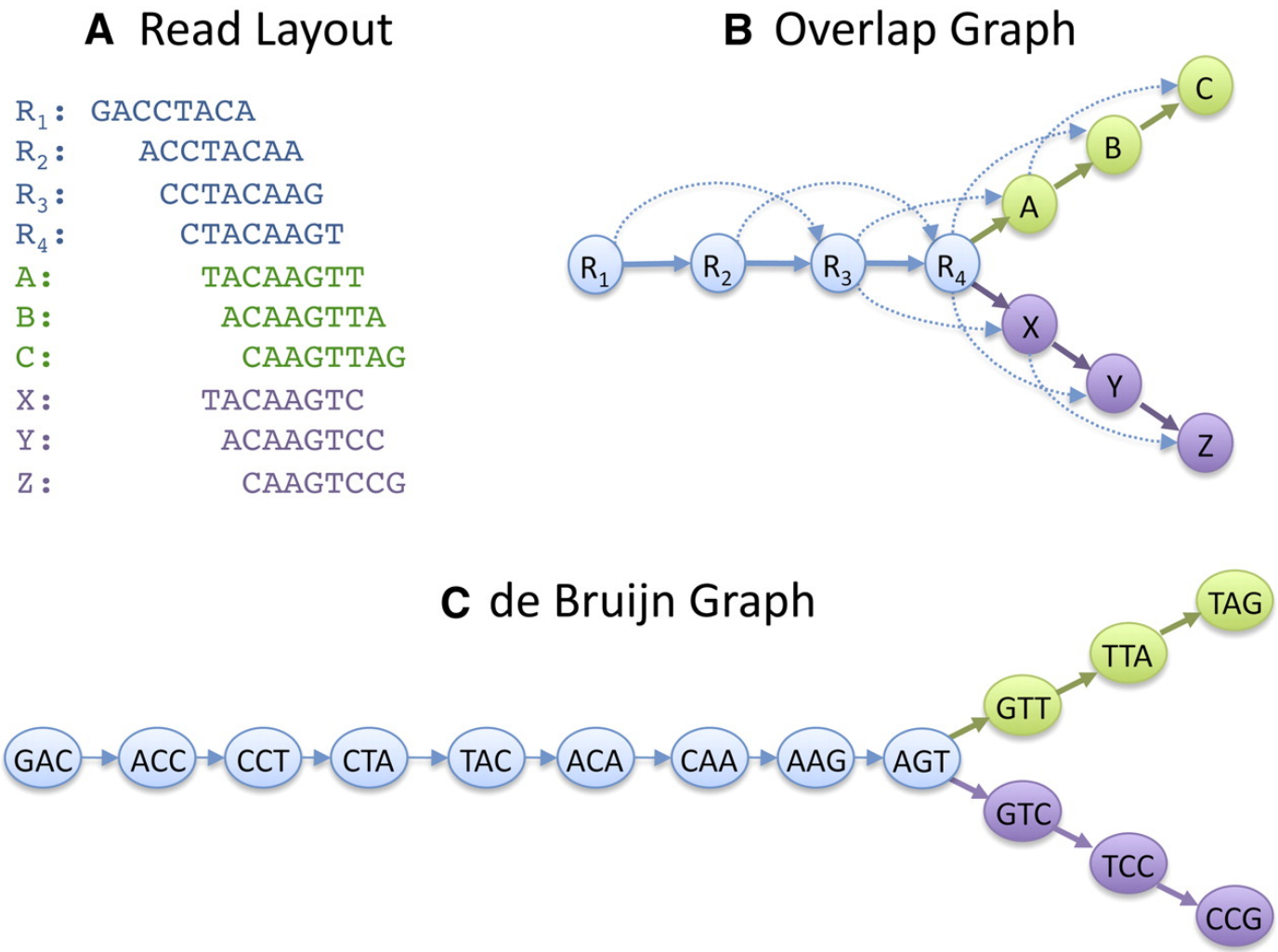


De Bruijn Graphs

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

Copyright © 2010 by Cold Spring Harbor Laboratory Press

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

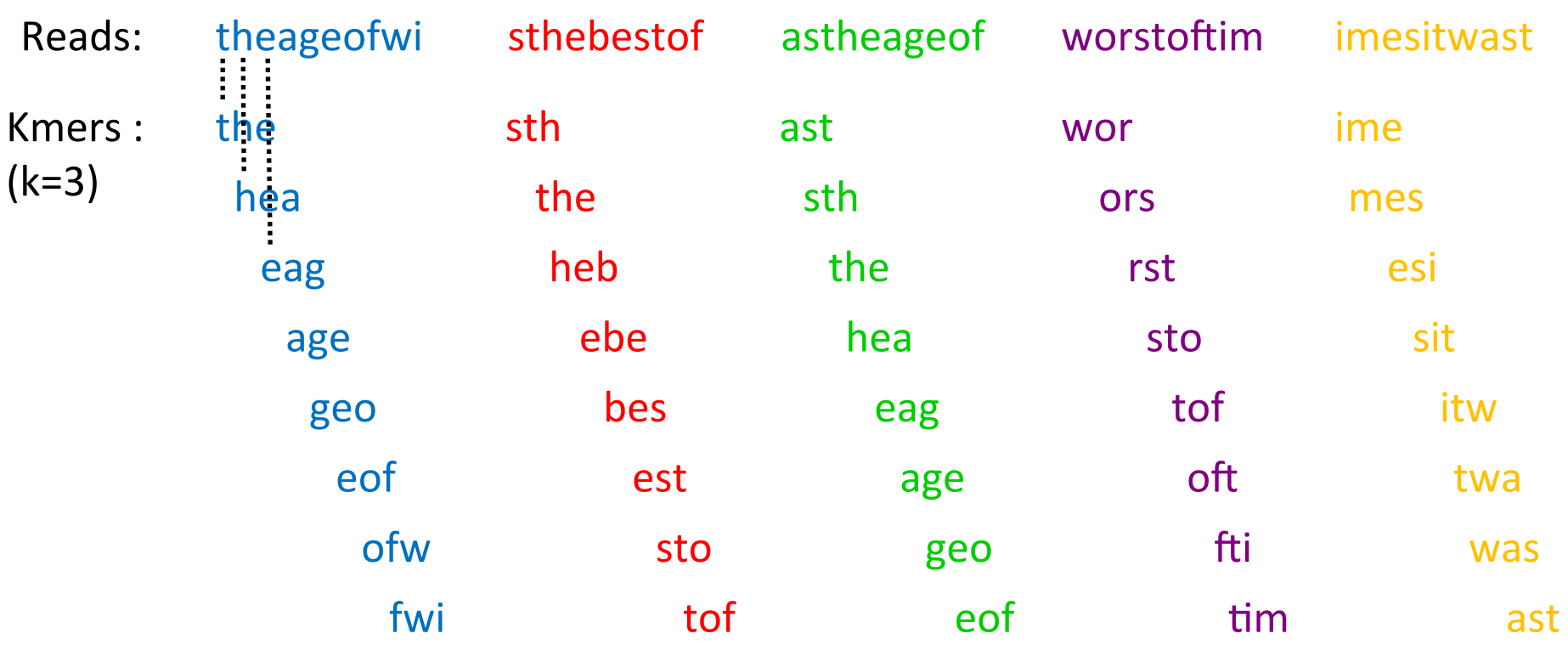
…etc. to 10’s of millions of reads

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

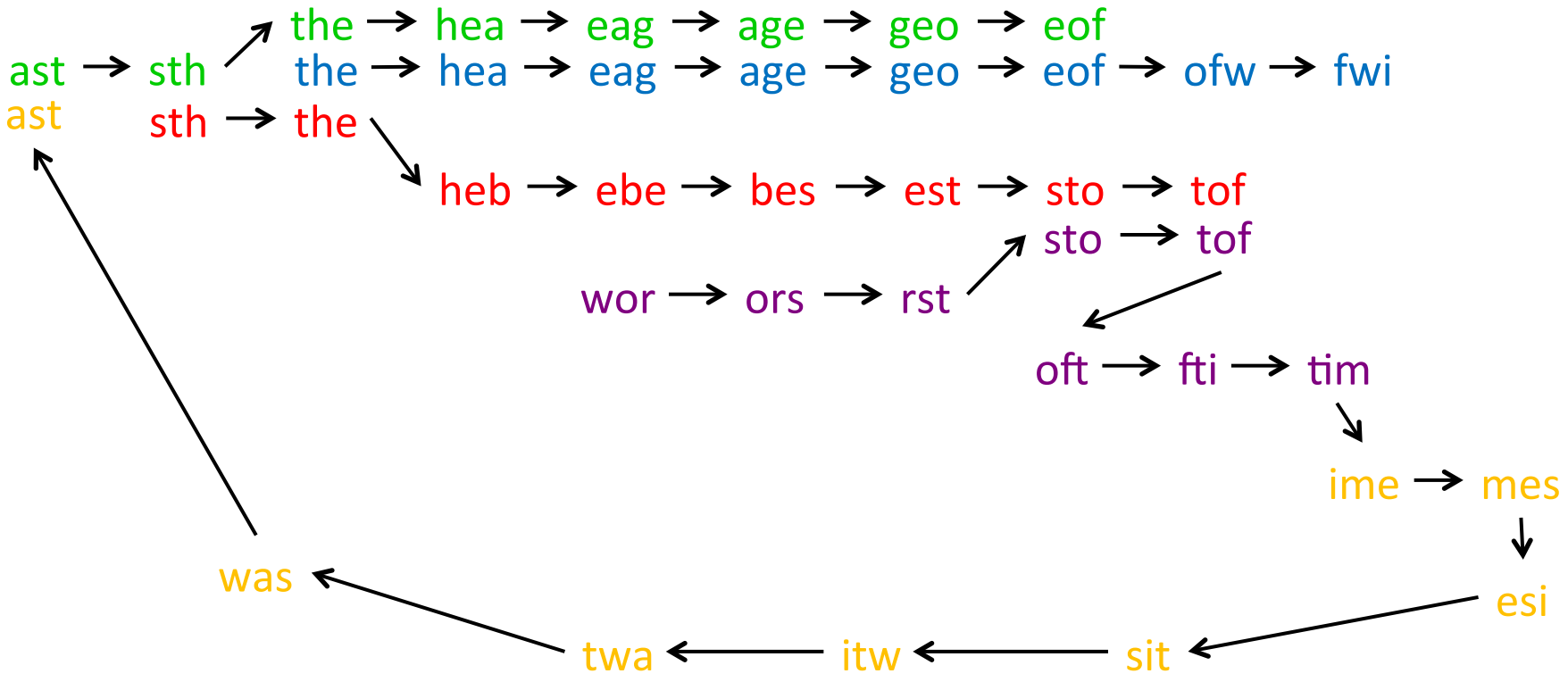


…..etc for all reads in the dataset

De Bruijn Graphs

Step2 Build the graph

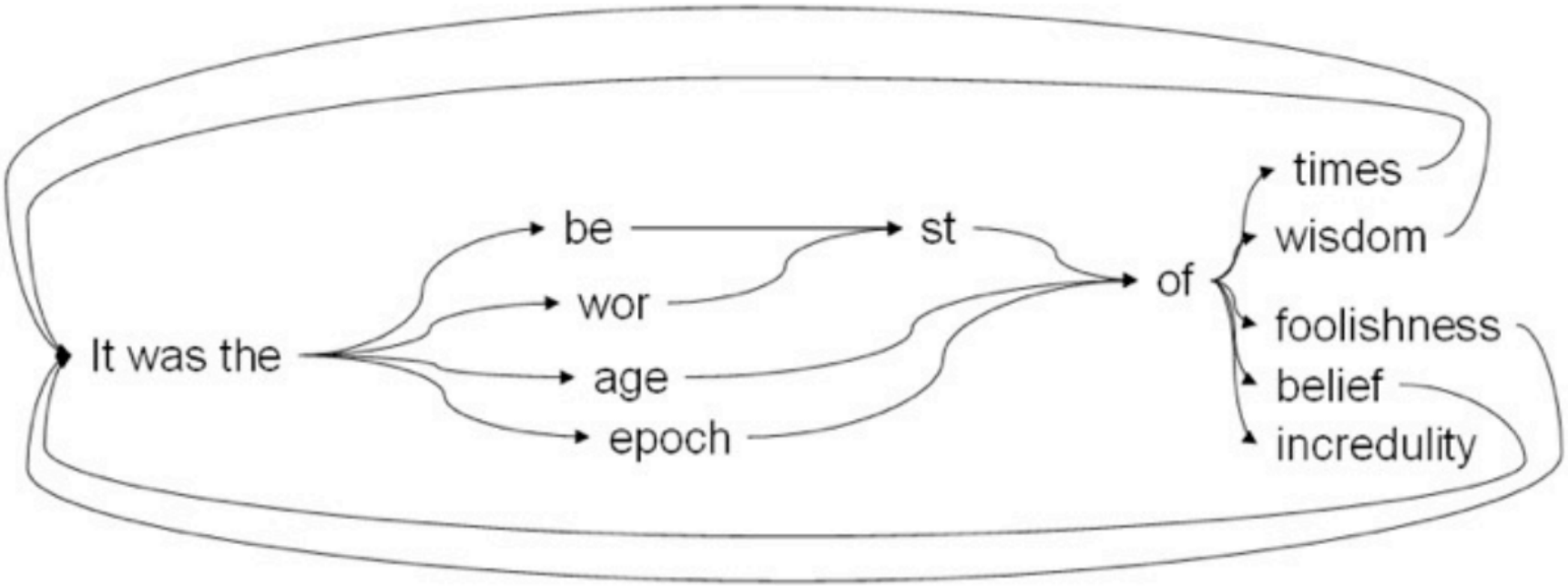
Look for k-1 overlaps: given by the reads



…..etc for all ‘kmers’ in the dataset

De Bruijn Graphs

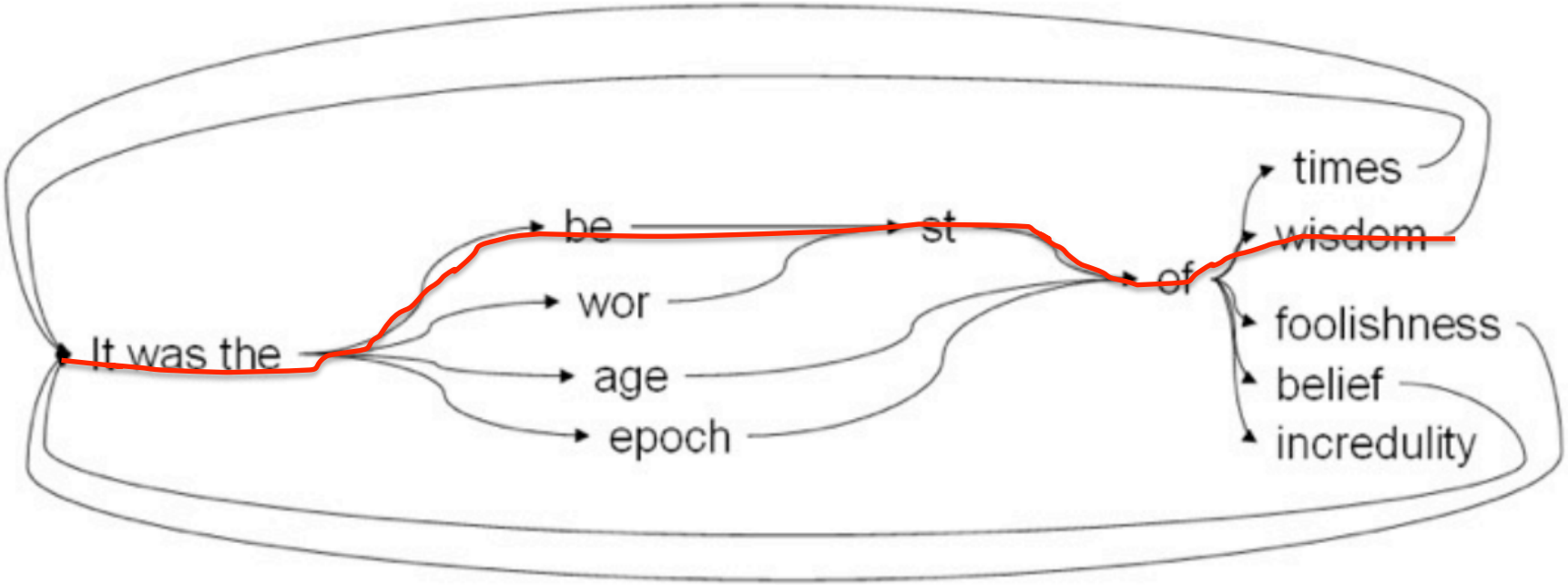
step3: simplify the graph



- 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

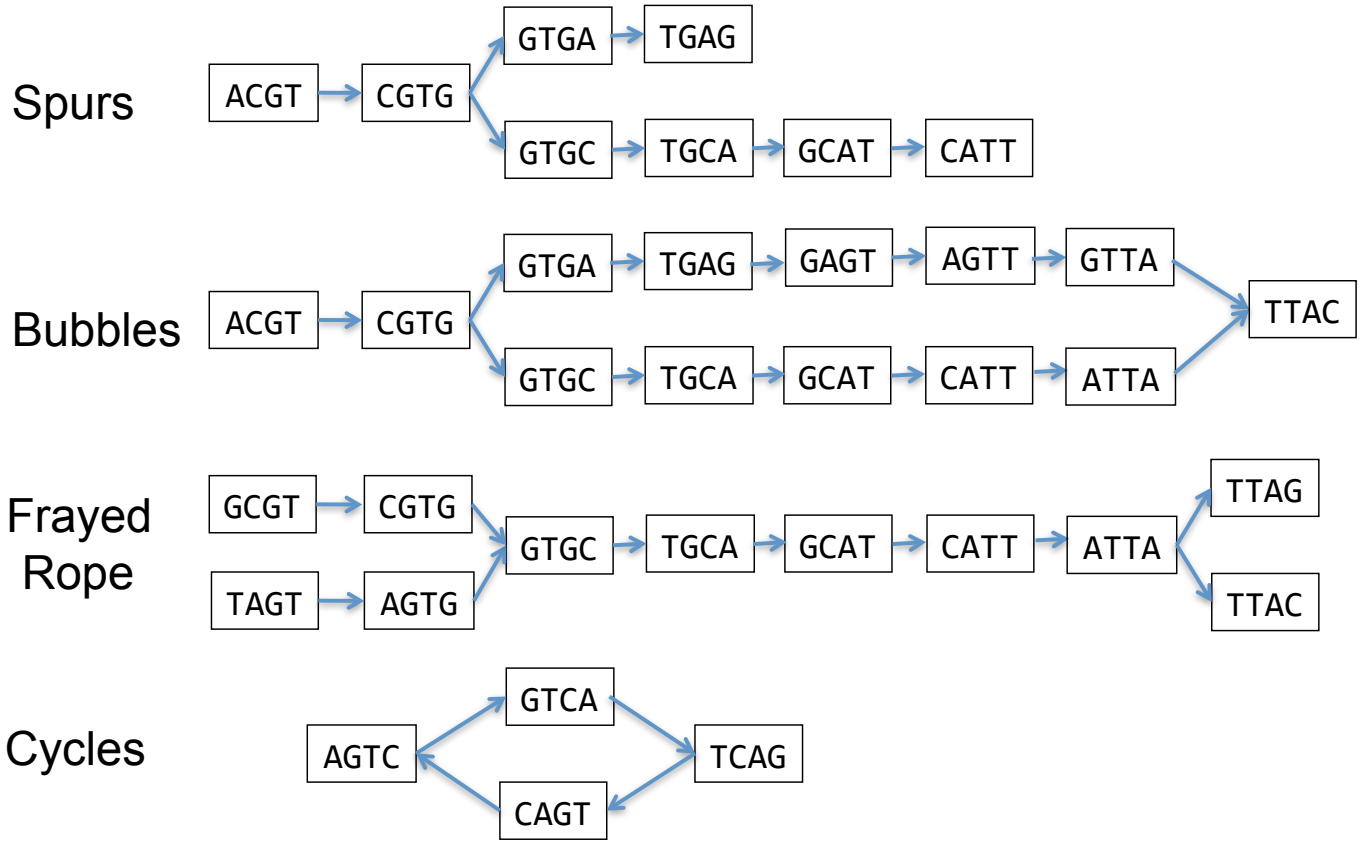
step4: Create contigs



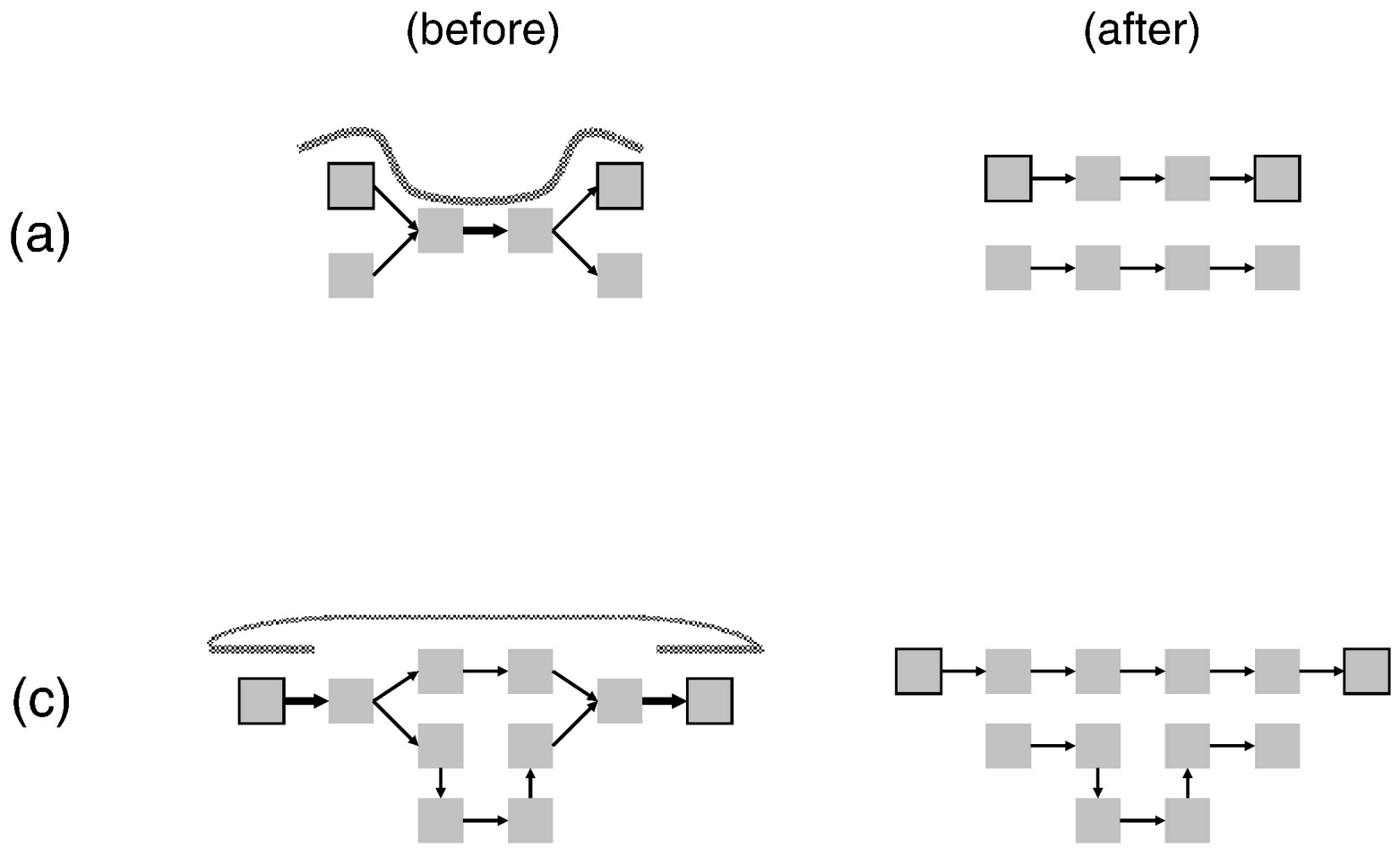
Find the Hamiltonian path or cycle in the De Bruijn graph. Each path in the simplified De Bruijn graph is a Hamiltonian path. A Hamiltonian path is a contig.

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

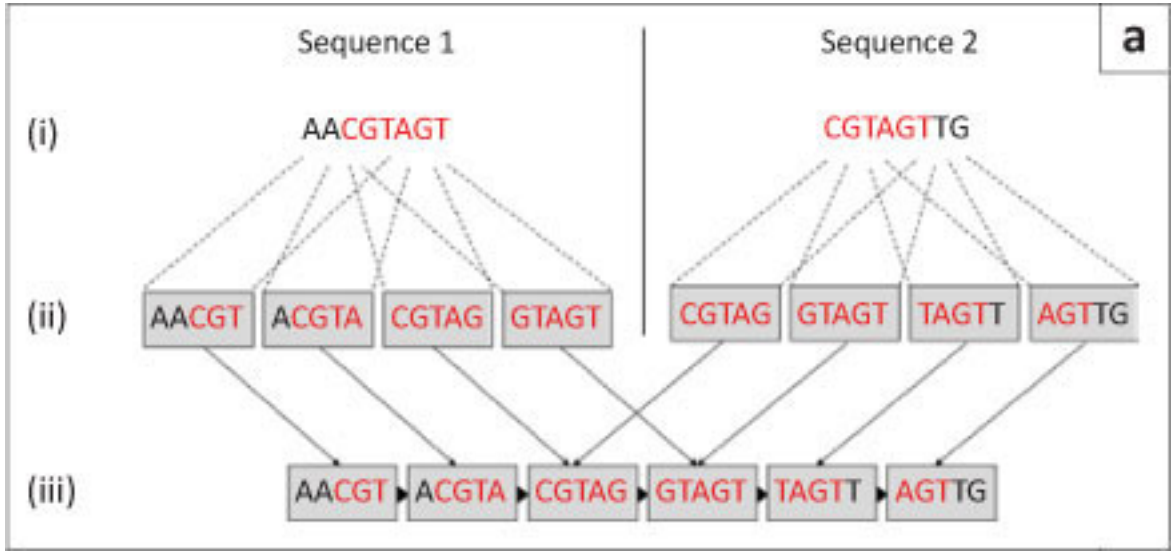
Strengths:

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

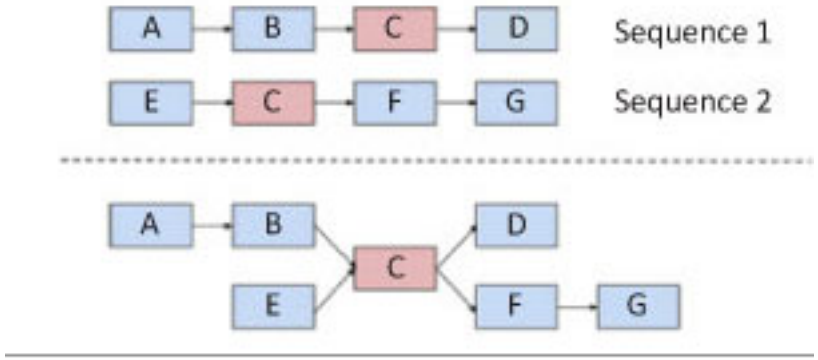
Problems:

- 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

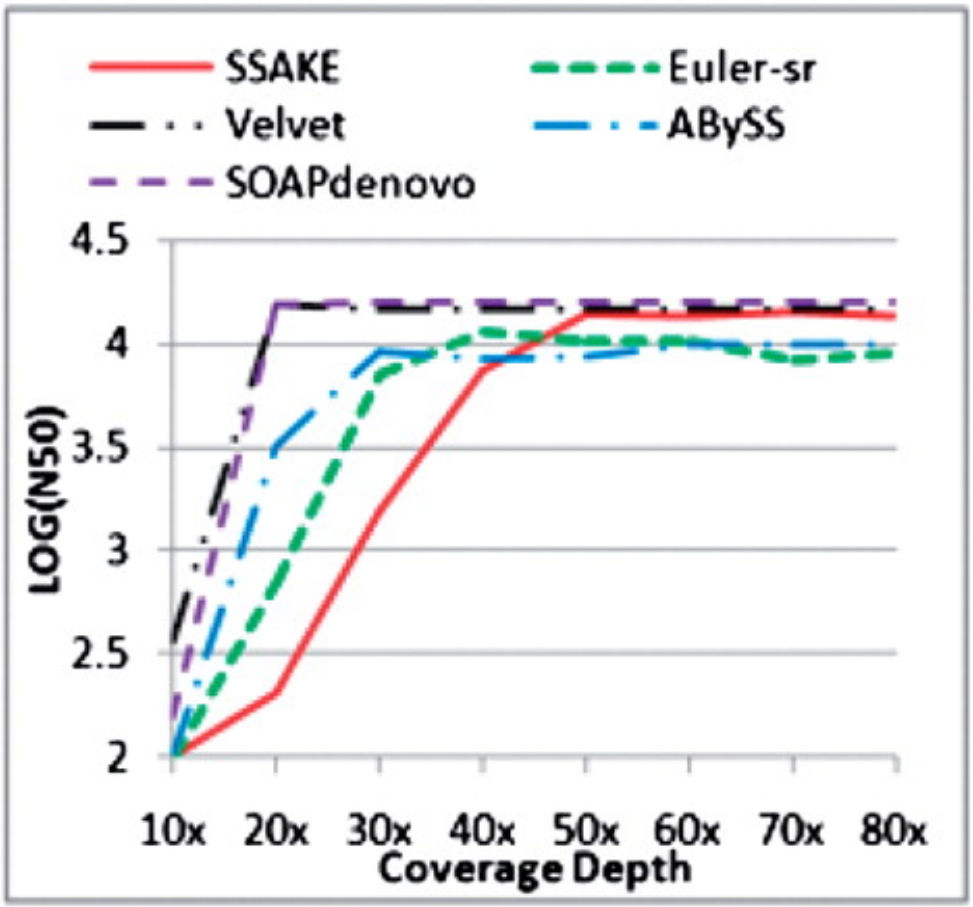
Miller, genomics, 2010, 95(6):315-27

Comparison of Assembly tools

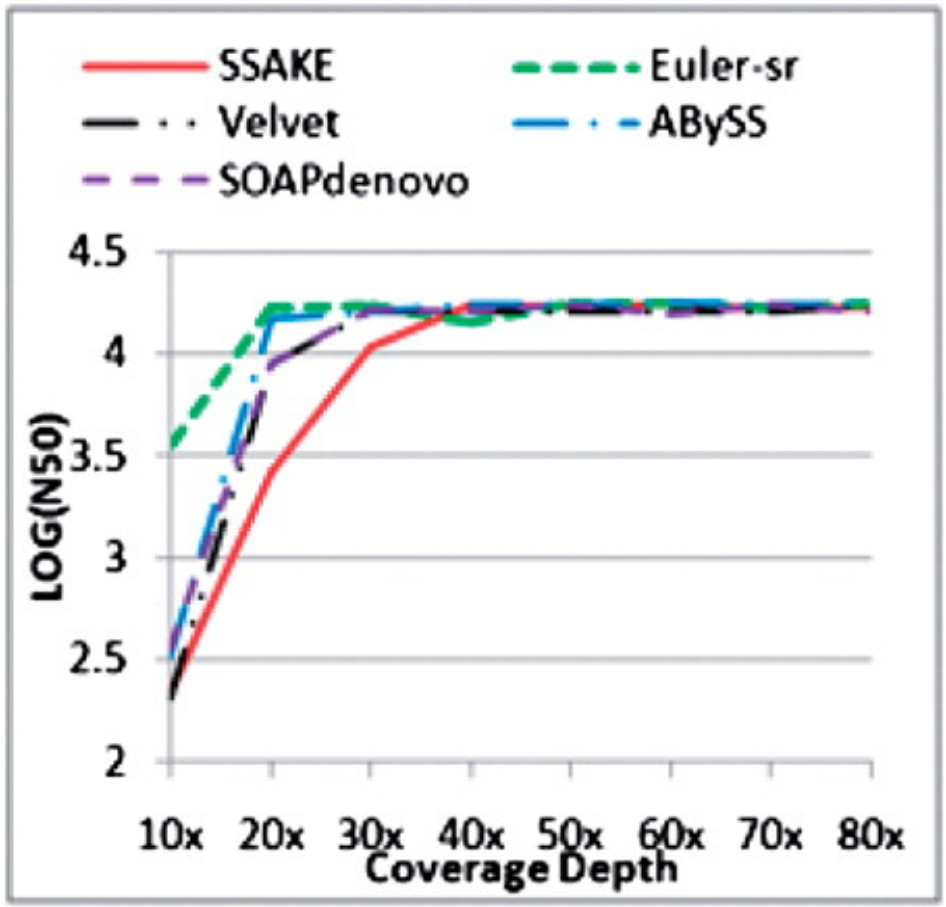
Algorithm Feature	Greedy Assemblers	OLC Assemblers	DBG Assemblers
Modeled features of reads			
Base substitutions			Euler, AllPaths, SOAP
Homopolymer miscount		CABOG	
Concentrated error in 3′ end			Euler
Flow space		Newbler	
Color space		Shorty	Velvet
Removal of erroneous reads			
Based on K-mer frequencies			Euler, Velvet, AllPaths
Based on K-mer freq and QV			AllPaths
For multiple values of K			AllPaths
By alignment to other reads		CABOG	
By alignment and QV	SHARCGS		
Correction of erroneous base calls			
Based on K-mer frequencies			Euler, SOAP
Based on Kmer freq and QV			AllPaths
Based on alignments		CABOG	

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

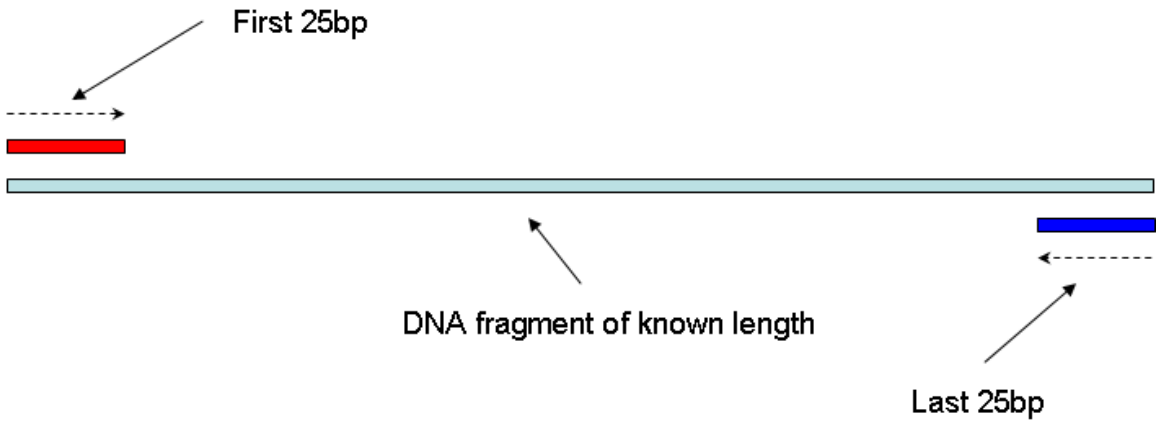
Lin Y et al. Bioinformatics 2011;27:2031-2037

Whole genome sequencing

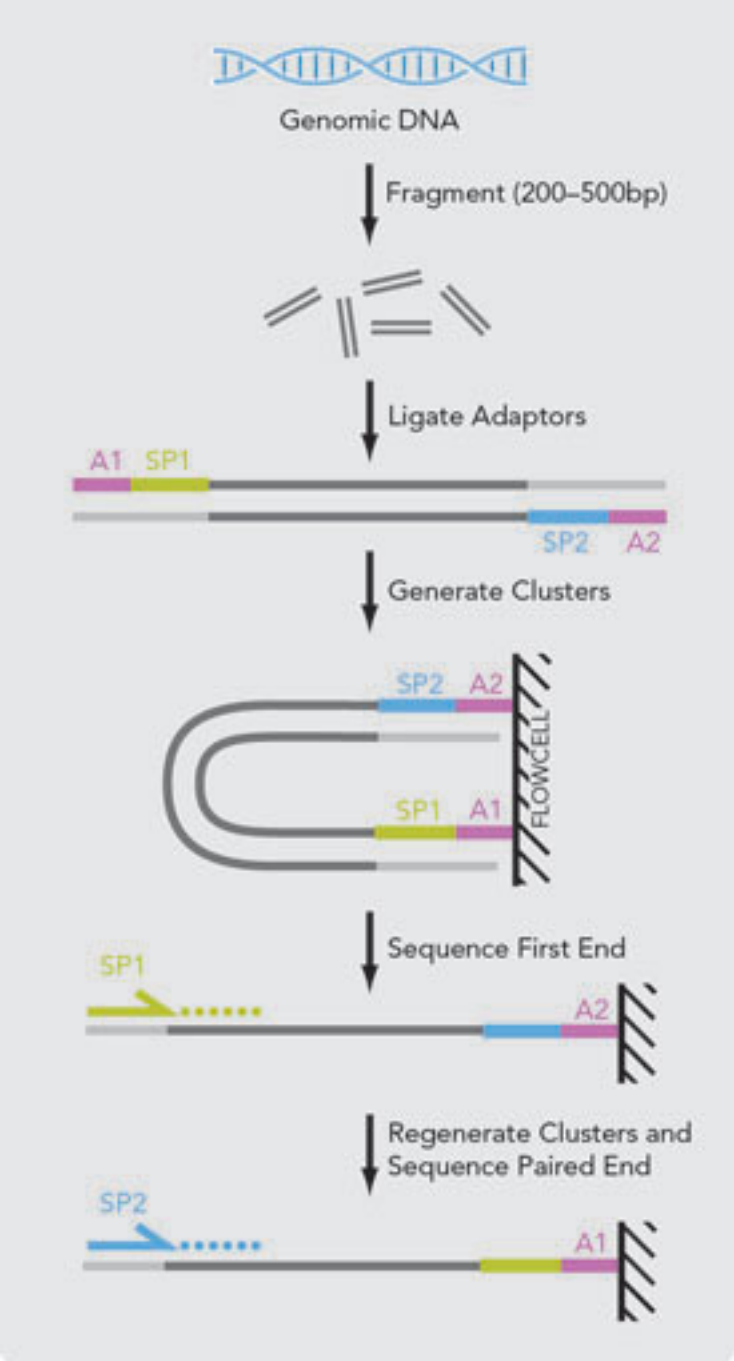
- *De Novo* whole genome sequencing
- Mapping assembly (Reference-guided assembly) (Resequencing)

Paired-end sequencing

- Paired-End sequencing (for Mate-pairs)
  - Sequence two ends of a fragment of known size.



  - Currently fragment length (insert size) can range from 200 bps – 10,000 bps
  - Paired-end sequencing is helpful for assembly and locating repeat. It also can detect rearrangements, including insertions and deletions (indels) and inversions.
  - As paired end reads are more likely to align to a reference, the quality of the entire data set improves



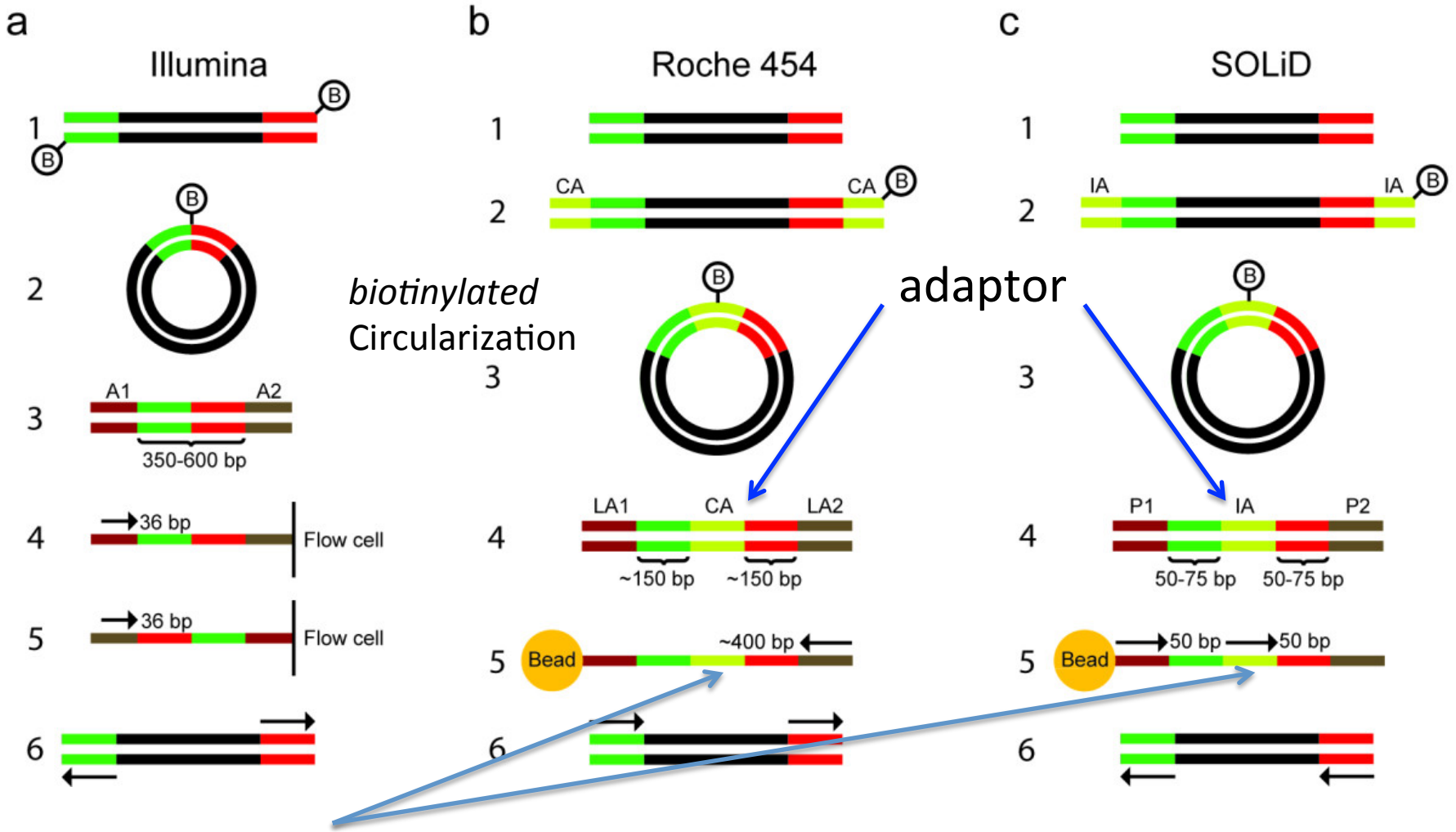
Paired-end sequencing by Illumina

Solid-phase amplification and Cyclic reversible termination

A simple modification to the standard single-read DNA library preparation.

Both the forward and reverse template strands of each cluster can be sequenced.

Mate-pair libraries



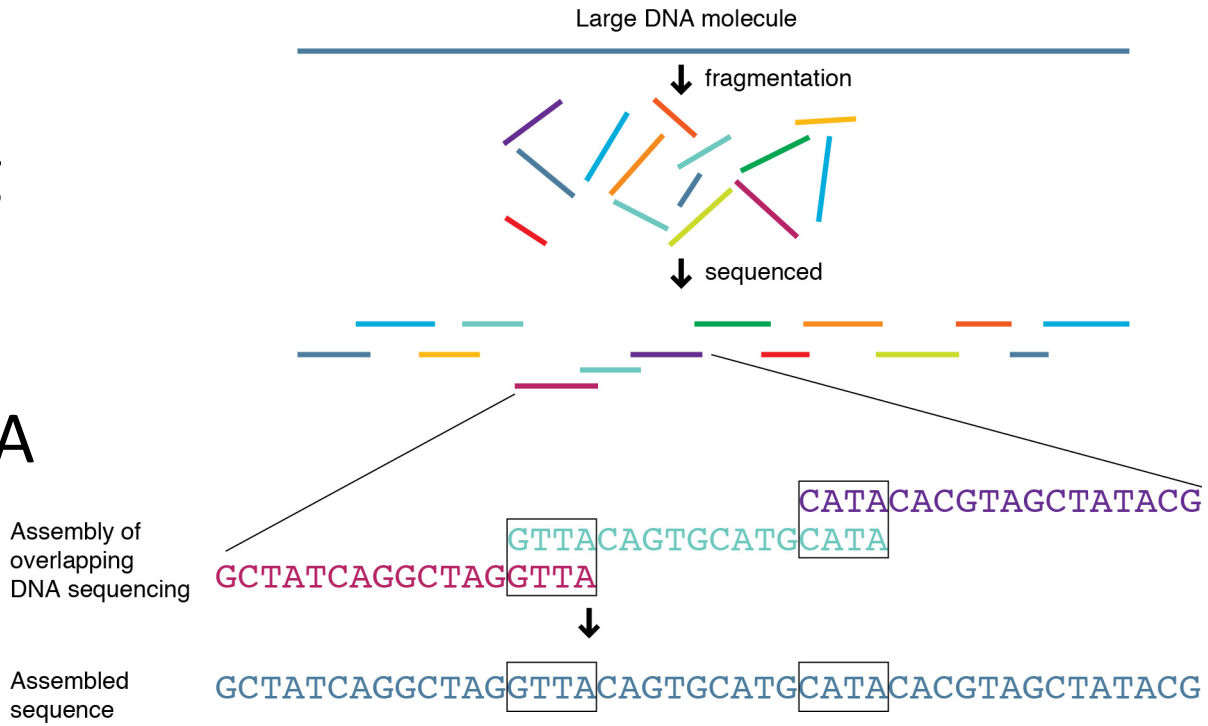
Berglund *et al. Investigative Genetics* 2011 **2**:23

De Novo sequencing

- New species/strains
- Challenge of assembly with short reads
  - 8x coverage of 3 GB genome = 750 million fragments
  - Exponential problem for all-vs-all algorithm (overlap)
- Big problem with repeats
- Assemble contigs, fill gaps
- Paired-end reads are essential

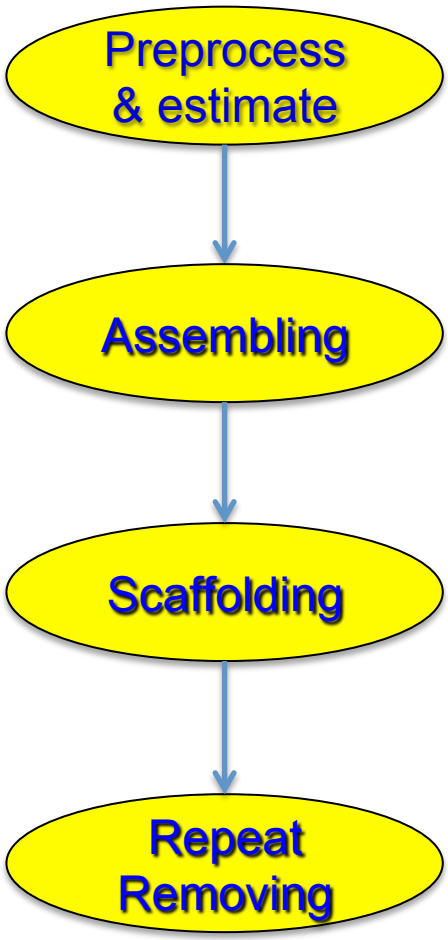
Shotgun Sequencing

Shotgun sequencing is a laboratory technique for determining the DNA sequence of an organism's genome.



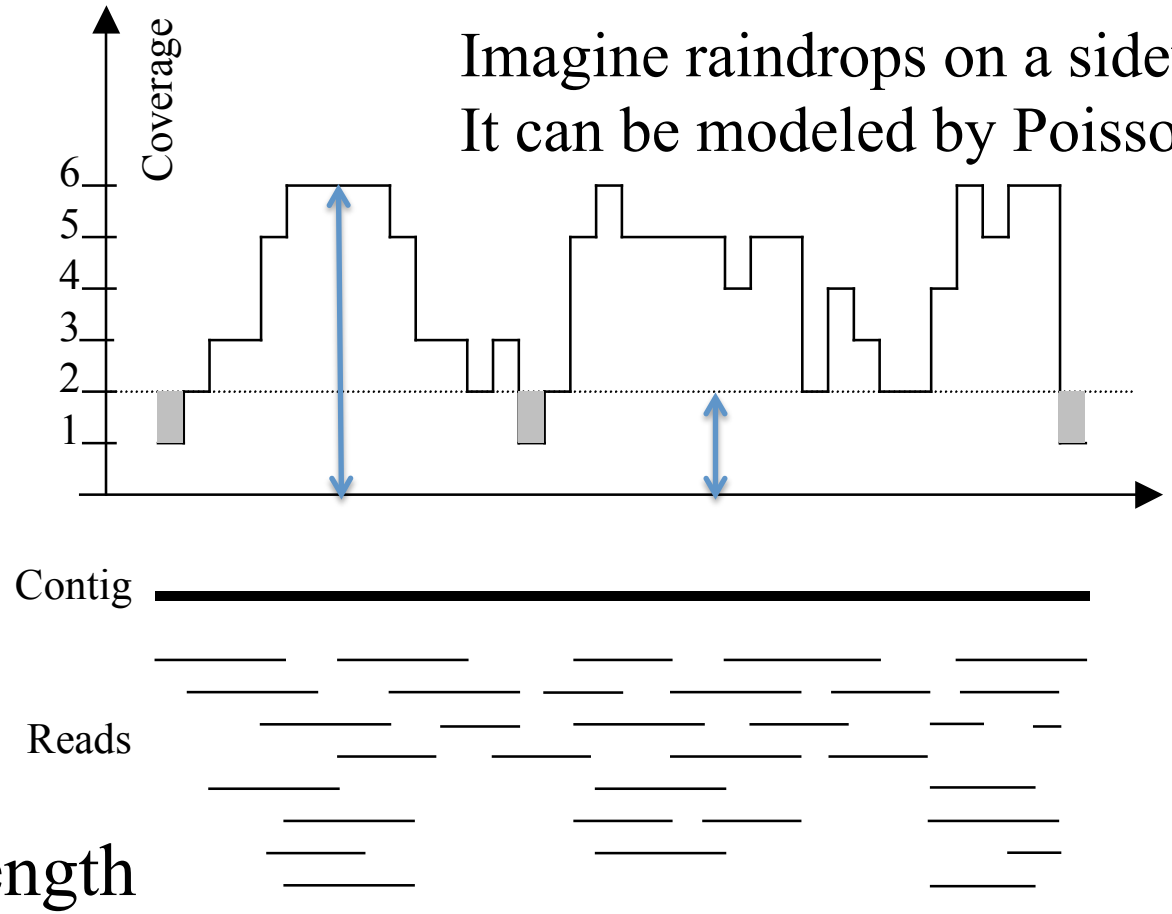
- Breaking the genome into a collection of small DNA fragments
- Sequencing.
- Reconstitute the genome.

Assembly Pipeline



Shotgun sequencing statistics

Typical contig coverage



Imagine raindrops on a sidewalk:
It can be modeled by Poisson distribution

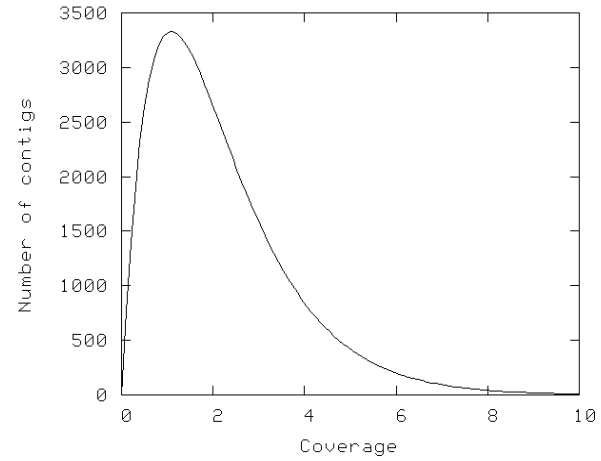
L = read length
G = genome size
N = number of reads
c = coverage= (NL / G)

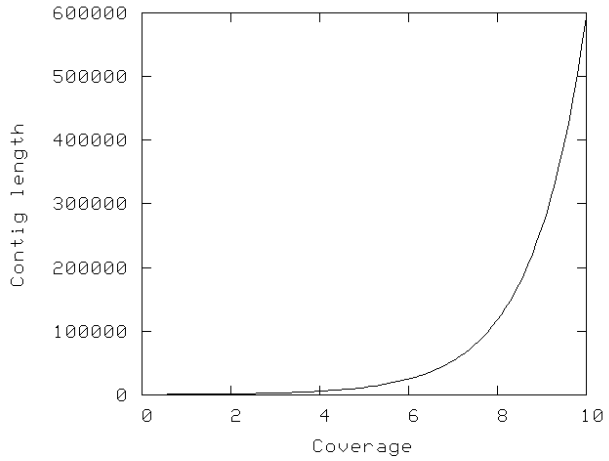
Average coverage

Lander-Waterman statistics

L = read length
G = genome size
N = number of reads
c = coverage =(NL / G)
T = minimum detectable overlap
$\sigma = 1 - T/L$

$E(\#\text{ of islands}) = Ne^{-c\sigma}$
$E(\text{island size}) = L((e^{c\sigma} - 1) / c + 1 - \sigma)$
contig = island with 2 or more reads





Smith-Waterman algorithm for sequence comparison

Example

Genome size: 1 Mbp Read Length: 600

c	N	#islands	#contigs	bases not in any read	bases not in contigs
1	1,667	655	614	698	367,806
3	5,000	304	250	121	49,787
5	8,334	78	57	20	6,735
8	13,334	7	5	1	335

Experimental data

X coverage	# ctgs	% > 2X	avg ctg size (L-W)	max ctg size	# ORFs
1	284	54	1,234 (1,138)	3,337	526
3	597	67	1,794 (4,429)	9,589	1,092
5	548	79	2,495 (21,791)	17,977	1,398
8	495	85	3,294 (302,545)	64,307	1,762
complete	1	100	1.26 M	1.26 M	1,329

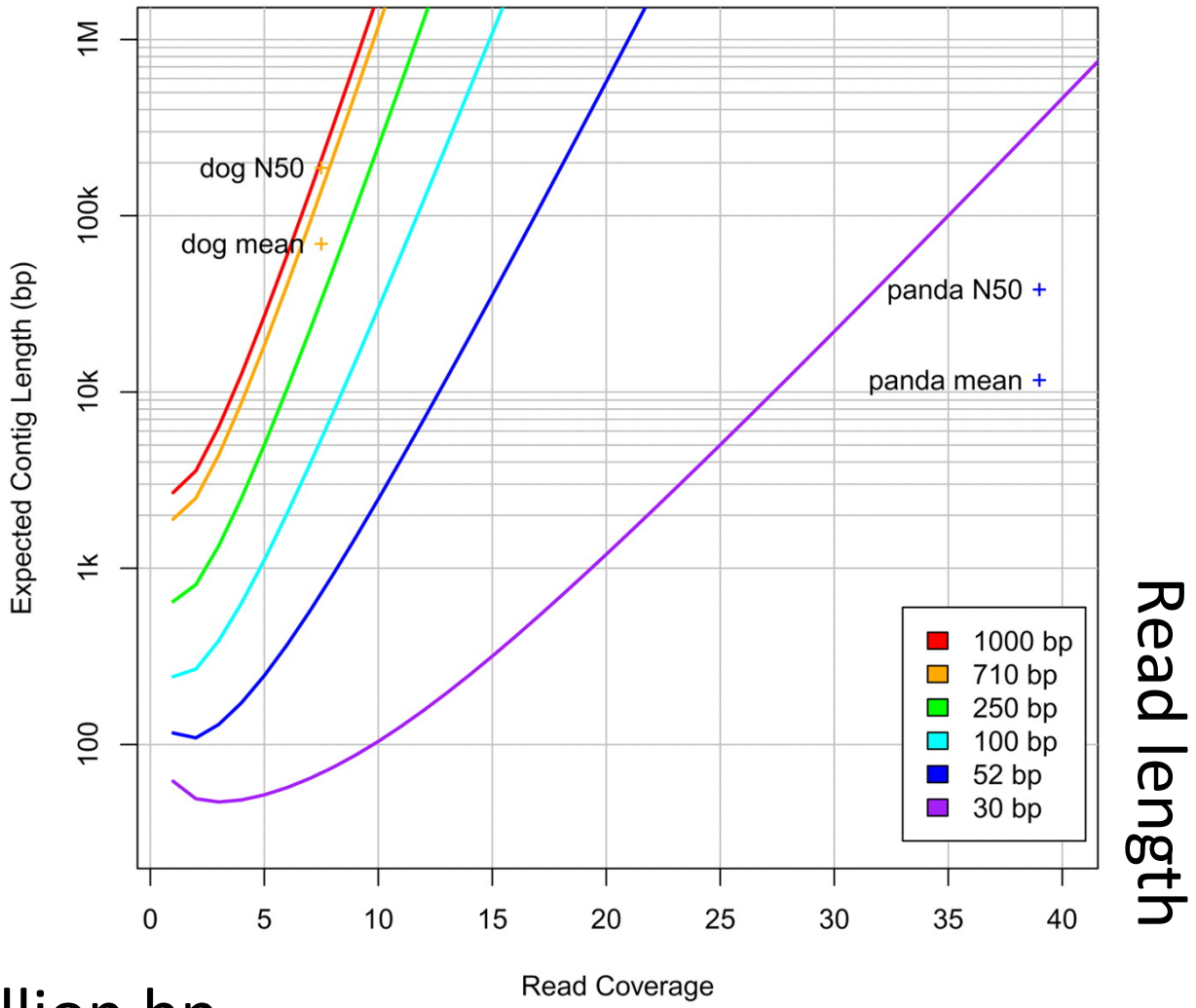
Numbers based on artificially chopping up the genome of *Wolbachia pipientis* dMel

Errors in Lander-Waterman Estimate

Lander-Waterman has errors:

- repeats
- GC/AT rich regions
- other low complexity regions
- cloning biases in shotgun libraries

Expected average contig length for a range of different read lengths and coverage values.



Dog: 2.5 billion bp

Panda: 3 billion bp

Copyright © 2010 by Cold Spring Harbor Laboratory Press

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

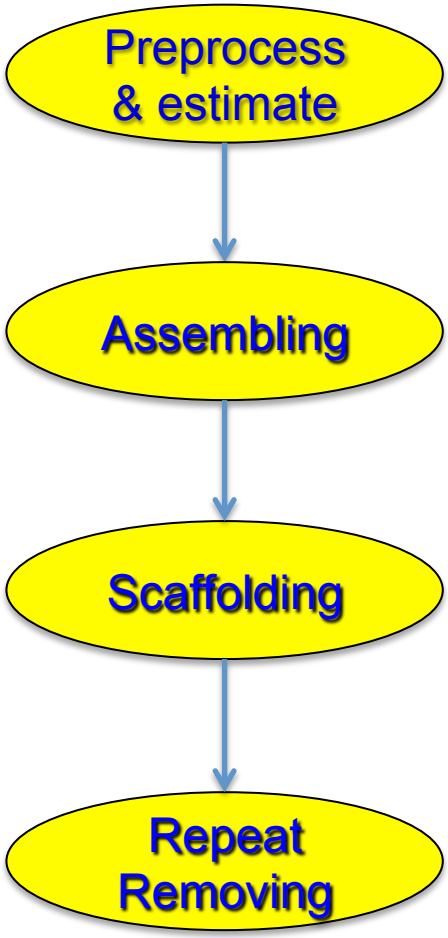
Organism/genome size	Assembler/status[a]	Input sequence						Contigs			
		Type	Pair size	Avgerage read (bp)	No. of reads	Read coverage[b]	Pair coverage[c]	No.	N50	Max	Total
Human (*H. sapiens*)/3.0 Gb	ABYSS published 2009	GA	210 bp	35–46	3.5 B	45×	120×	2.76 M	1.5 kb	18.8 kb	2.18 Gb
Grapevine (*V. vinifera*)/500 Mb	Myriad published 2007	Sanger	2–10 kb	579	5.95 M	6.9×	21×	58,611	18.2 kb	238 kb	531 Mb[d]
		Sanger	40 kb	460	144 k	0.13×	4.4×				
		Sanger	120 kb	369	68 k	0.02×	4.2×				
		454	None	169	12.5 M	4.2×	—				
Cucumber (*C. sativus*)/367 Mb	RePS2 published 2009	Sanger	2–6 kb	439	2.08 M	3.35×	9.9×	62,412	19,807	NR	226 Mb
		Sanger	40 kb	496	339 K	0.46×	16.7×				
		Sanger	140 kb	551	33.2 k	0.04×	5.6×				
		GA	200 bp	42	282 M	32.5×	76.8×	NR	2.6 kb	NR	204 Mb
		GA	400 bp	44	173 M	20.6×	94.4×				
		GA	2 kb	53	105 M	15.3×	286×	NR	12.5 kb	NR	190 Mb
Panda (*A. melanoleura*)/2.4 Gb	SOAPdenovo published 2010	GA	150	45	1.31 B	24.5×	43.3×	200,604	36,728	434,635	2.25 Gb
		GA	500	67	917 M	25.5×	90.2×				
		GA	2 kb	71	397 M	11.8×	192×				
		GA	5 kb	38	505 M	8.0×	533×				
		GA	10 kb	35	254 M	3.7×	571×				
Strawberry (*F. vesca*)/220 Mb	CABOG and Velvet announced	454	None	209	7.73 M	7.3×	—	16,487	28,072	215,349	202 Mb
		454	None	368	787 M	13.2×	—				
		454	2.5 kb	193	2.39 M	2.1×	6.9×				
		454	20 kb	236	1.58 M	1.7×	20×				
		GA	None	76	36 M	12.4×	—				
		SOLiD	2 kb	25	1.30 M	0.14×	6.4×				
Turkey (*M. gallopavo*)/1.1 Gb	CABOG announced	454	3 kb	180	6 M	1×	8×	128,271	12,594	90 kb	931 Mb
		454	20 kb	195	2 M	0.3×	18×				
		454	None	366	13 M	4×	—				
		GA	180 bp	74	200 M	13×	16×				
		GA	None	74	200 M	13×	—				

One more example

For yeast 12Mbp

- read length: 200-400 bp
- coverage: 50X (how many reads do we need?)
- paired-end read insert size: 8kb (better to make multiple libraries with different insert sizes.)

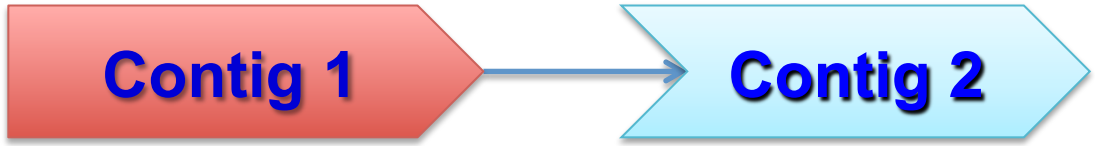
Assembly Pipeline



- **Velvet**: small genomes
- **ABySS**: large genome

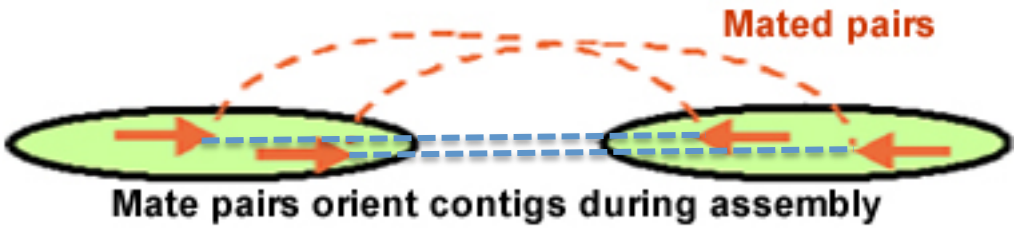
Scaffolding

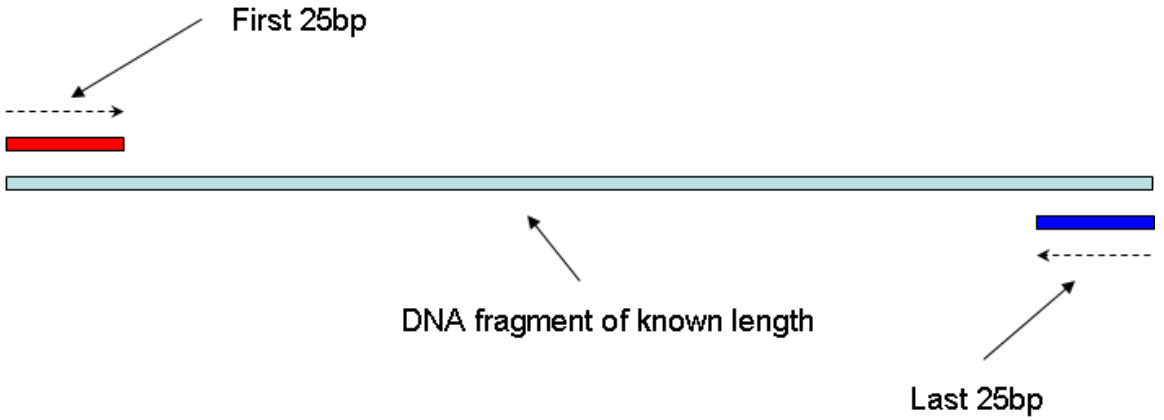
- Scaffolding groups contigs into subsets with known order and orientation.
- Nodes are contigs
- Directed edge is between two nodes if they are adjacent in the genome.



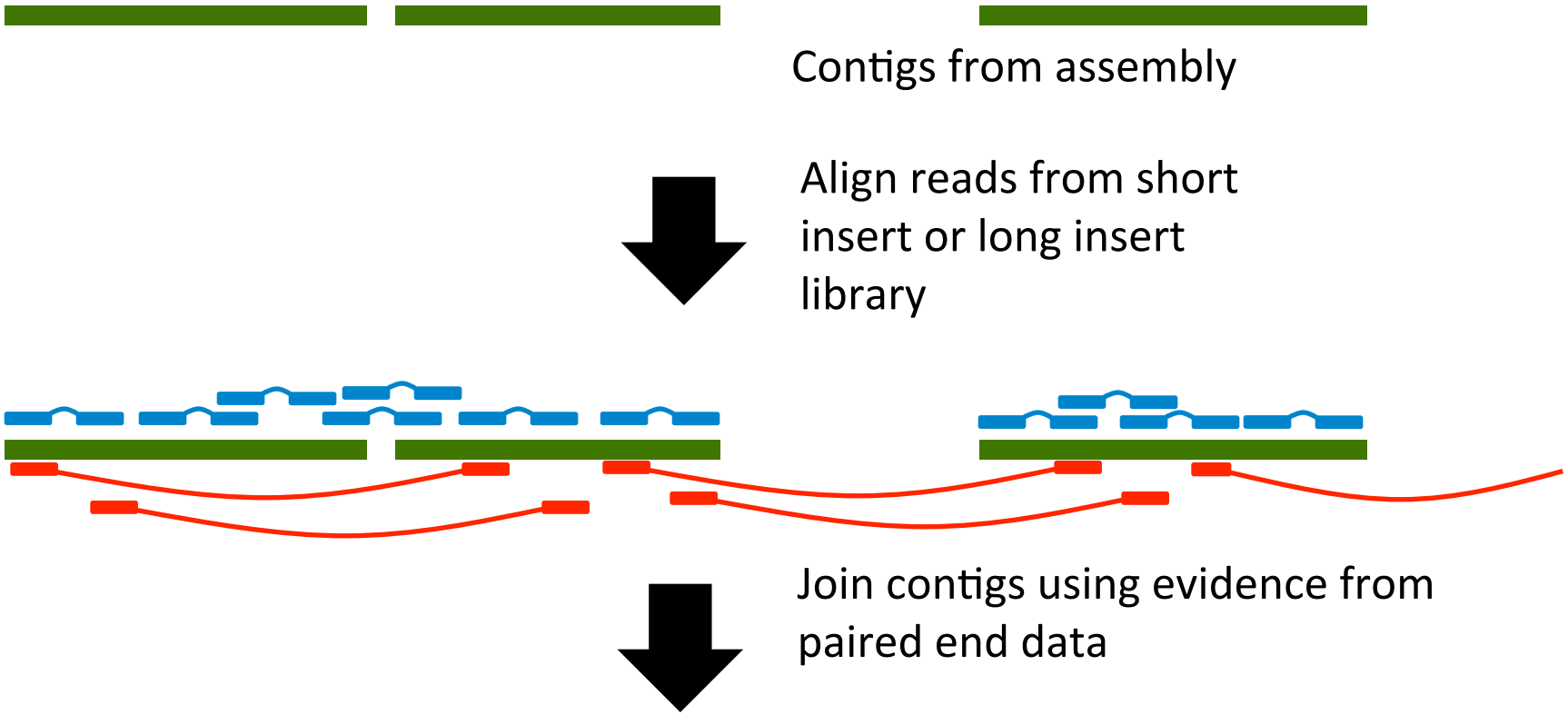
Scaffolding

- Mate pairs , if in different contigs, have a chance of being neighbors.





Scaffolding



Scaffolding Algorithm

- Find all connected components
- Find a consistent orientation for all nodes in the graph (all contigs).
  - Nodes (contigs) have two types of edges
    - Same orientation
    - Different orientation
  - Make sure linked contigs have consistent orientation.
  - Optimization problem – find the smallest number of edges to be removed so that all contigs have consistent orientation.
- Find the Hamiltonian path again.

Scaffolding software

- Some assembly software, such velvet, can do scaffolding as well.
- **Bambus** - http://www.cbcb.umd.edu/software/bambus
- **SSPACE** - http://www.baseclear.com/landingpages/basetools-a-wide-range-of-bioinformatics-solutions/sspacev12/
- **GRASS -** http://code.google.com/p/tud-scaffolding/

Additional techniques for orientation

- Physical mapping. Using information from Bacterial Artificial Chromosome (BAC)-based physical maps. Physical maps are built by clustering together of BACs sharing portions of a DNA "fingerprint," which is a pattern of DNA fragments of various sizes.
- Using markers along a DNA strand as independent information for scaffolding software. Markers are known sequences of nucleotides and tags. Markers are searched in the contigs.
- Using large scale maps of landmarks that lie along the the chromosomal DNA.

Scaffolding

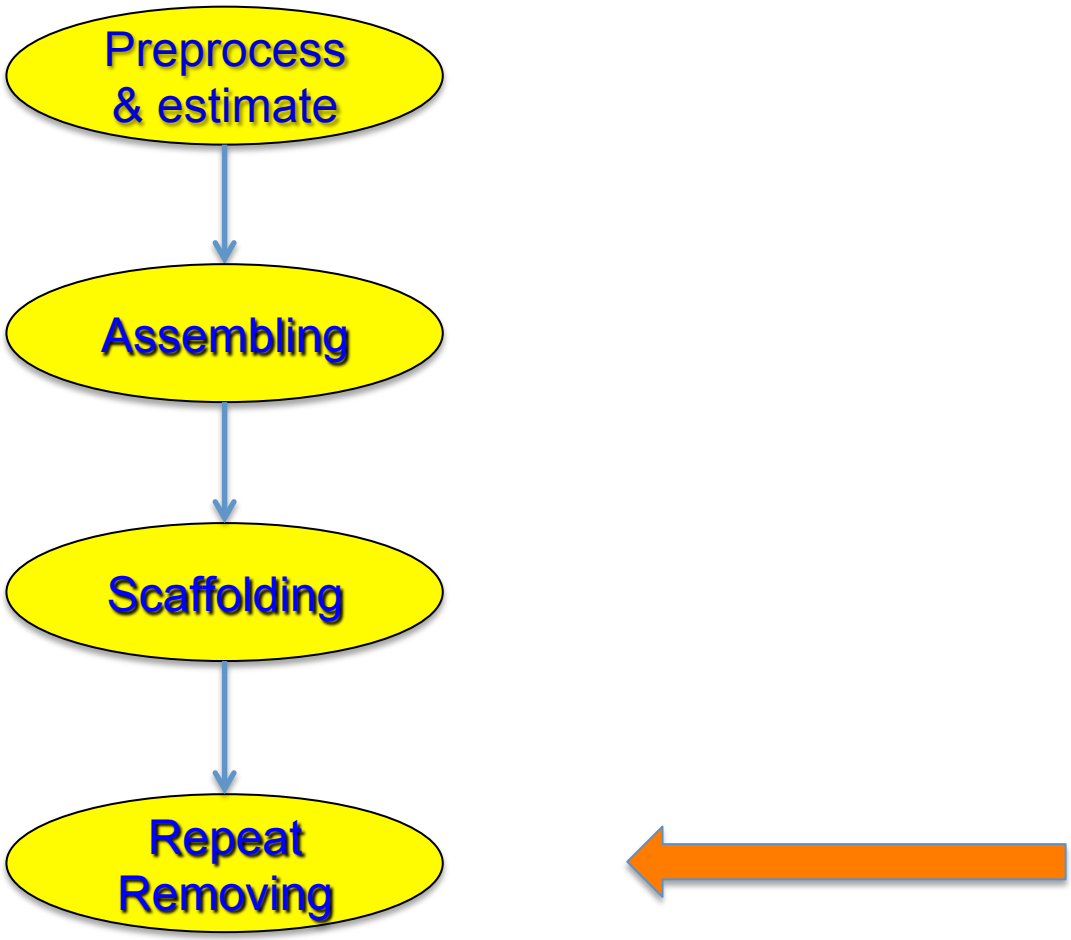
- Additional information is also useful:
  - Sequences of closely related organisms are also used as scaffolding information.

    Example: aligning scaffolds of a mouse genome to the human genome

Scaffolding: Issues

- Errors in length of inserts (affecting distances between clone mates)
- Physical mapping is error prone.
- first builds a sequence based on linking information with high confidence, then factors in linking information with lower confidence.

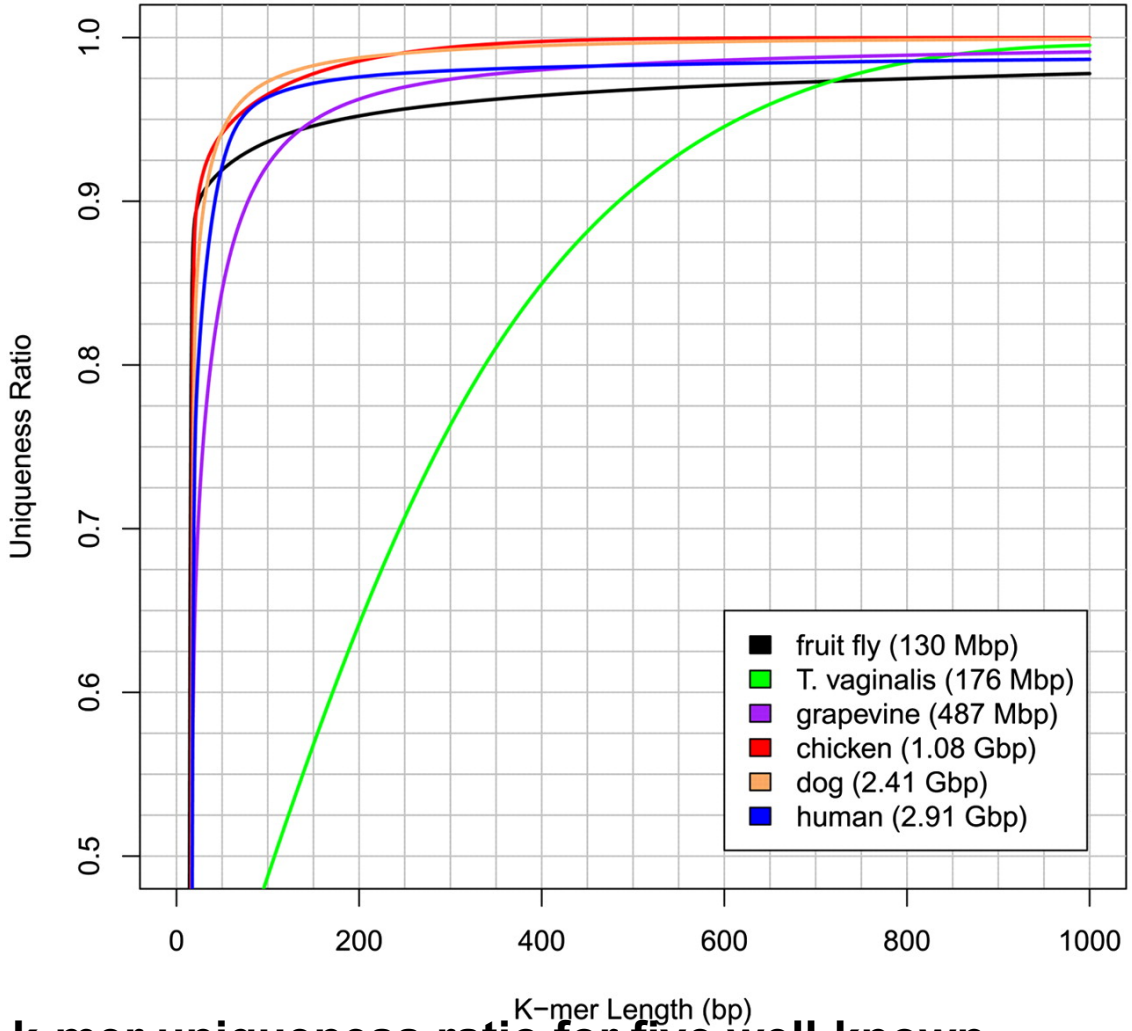
Assembly Pipeline



The variability in repetitiveness among species species.

The ratio == the percentage of the genome that is covered by unique sequences of length k or longer.

The figure shows how much of each genome would be covered by *k*-mers (reads) that occur exactly once.



The k-mer uniqueness ratio for five well-known organisms and one single-celled human parasite.

Copyright © 2010 by Cold Spring Harbor Laboratory Press

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

Repeat Control Issues

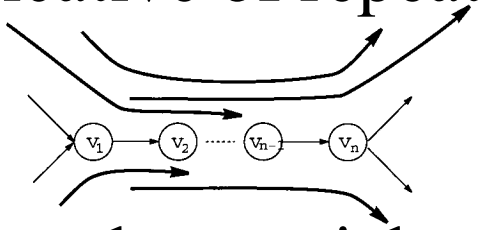
- Assembly programs should detect repeats in the assembly process and not after.
  - Incorrect genome reconstruction

- Assemblers should try to resolve correctly as many repeats as possible.
  - Avoid intensive human labor

Repeat Control – When? & How?

- **pre-assembly:** find fragments that belong to repeats
  - statistically (most existing assemblers)
  - repeat database (*RepeatMasker*)

- **during assembly:** detect "tangles" indicative of repeats (Pevzner, Tang, Waterman 2001)



- **post-assembly:** find repetitive regions and potential mis-assemblies.
  - *Reputer, RepeatMasker*
  - "unhappy" mate-pairs (too close, too far, mis-oriented)

Detecting repeats **pre-assembly:**

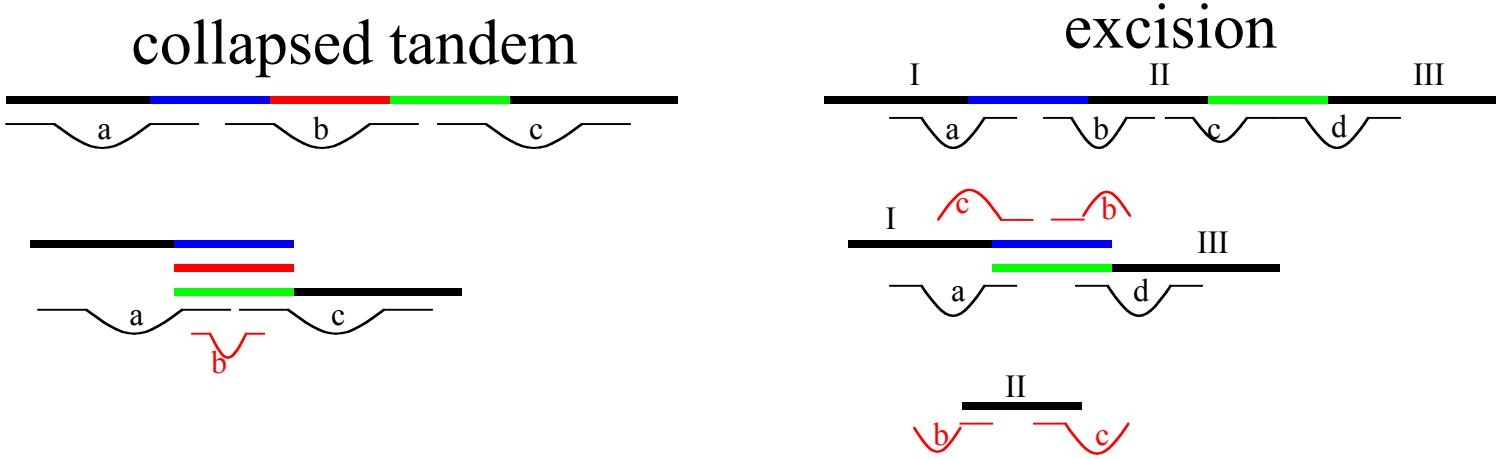
- Statistical methods
  - Assemblers assume that reads are sampled uniformly at random.
  - Significant deviations from average coverage flagged as repeats.
  - frequent k-mers are ignored
  - “arrival” rate of reads in contigs compared with theoretical value.

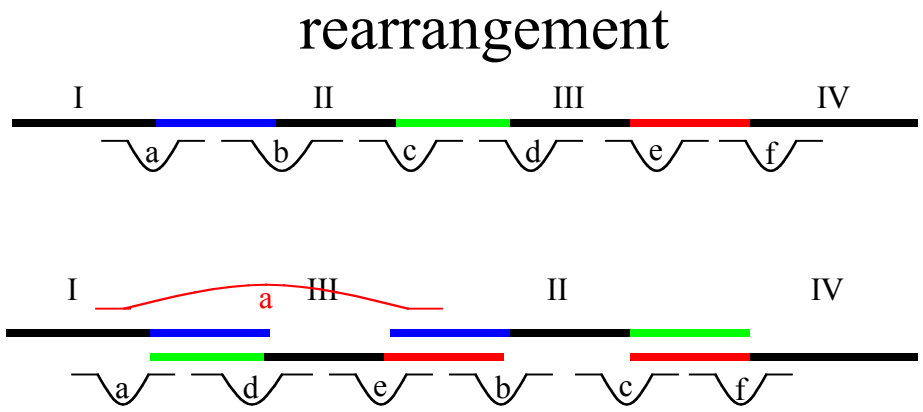
(e.g., 800 bp reads & 8x coverage - reads "arrive" every 100 bp)

Detecting repeats **during assembly**

- Example: In Euler assembly program
  - Finds repeats by complex parts of the graph constructed during the assembly process.
  - Researchers look into these complex areas to try and resolve repeats.
  - Assemblers can use clone mate information to find incorrect assemblies. This is based on finding clone-mate pairs too close or too far from one another. (“unhappy” mate-pairs)

Detecting repeats
post-assembly: Mis-assembled repeats





Repeat resolution

- Assemblers deduce that areas covered by a large number of reads may show an over-collapsed repeat.
- Problems with this - samples are not uniformly distributed (for example, non-random libraries and poor clonability regions). leads to false positives.
- Repeats with low copy number are missed - leads to false negatives.

Repeat resolution

- Techniques for repairing sequencing errors during repeat resolution
  - find clusters of reads where the clusters share differences.
    - For example, four reads contain an A , four contain a B. it is likely that the first four reads are from one copy and the last four from a different one.
  - Drawbacks are if certain areas of the sequence have low coverage.
  - Difficult to separate from true polymorphism

Discussion:
Virtual genome assembly

- Plant mitochondrion genome 500,000 bp DNA circular
- How can you get mitochondria DNA? What problems do we need to concern for this step?
- For DNA fragmenting, what sizes of DNA fragments will you use? A. 1Kbp, B. 5kbp, C. both
- Pair-ended or single ended?
- What depth do you sequence? how many lanes do you need if you use illumina hiseq 2000? or how many reads do you need to get?
- Which assembler will you use? Why?
- What computer do you used to do assemble? A. 4GB laptop B. 50GB workstation C. computer cluster in HCC
- According to your estimate, how long does it take for assemble? A. 30 minutes B.2 hours C. 12 hours D. 4 days
- What software do you used to do scaffold? how long does it take?
- What is longest gap in one scaffold? How do you fill gaps?
- How do you determine if your assembled genome is good enough?
- how do you annotate genes?

On Thursday, Jan 30.