Graphs and Genome Assembly

Michael Schatz

Bioinformatics Lecture 3
Quantitative Biology 2010

Sequence Alignment Review

DP Alignment

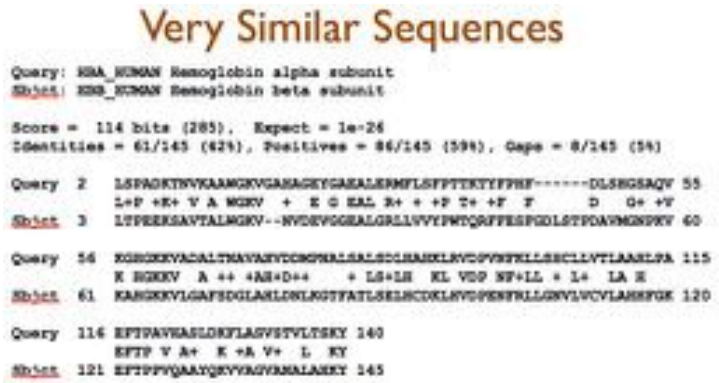
		A	C	A	C	A	C	T	A
	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
G	2	1	1	2	3	4	5	6	7
C	3	2	1	2	2	3	4	5	6
A	4	3	2	1	2	2	3	4	5
C	5	4	3	2	1	2	2	3	4
A	6	5	4	3	2	1	2	3	3
C	7	6	5	4	3	2	1	2	3
A	8	7	6	5	4	3	2	2	2

D[AGCACACA,ACACACTA] = 2

```
AGCACAC-A
|*|||||*|
A-CACACTA
```

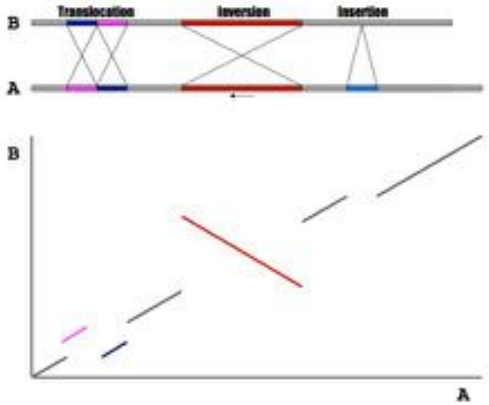
Guaranteed optimal, but slow

BLAST



Seed-and-extend for "good" matches to a DB

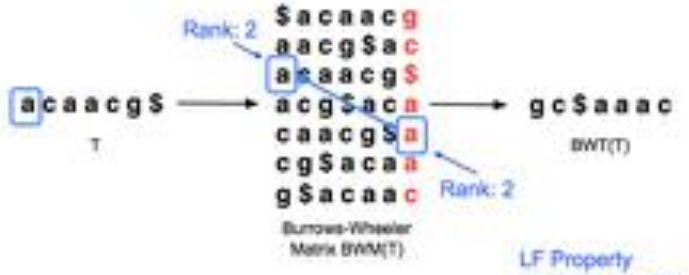
MUMmer



Whole Genome Alignment w/ Suffix Tree

Bowtie

- Reversible permutation of the characters in a text



- BWT(T) is the index for T

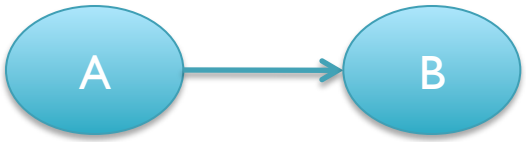
Fast searching for short read mapping

Outline

1. Graphs and Graph Theory

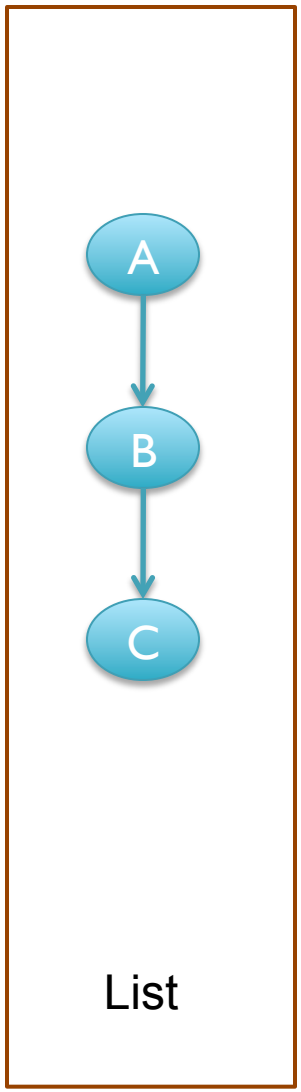
2. Genome Assembly
    1. Assembly Validation

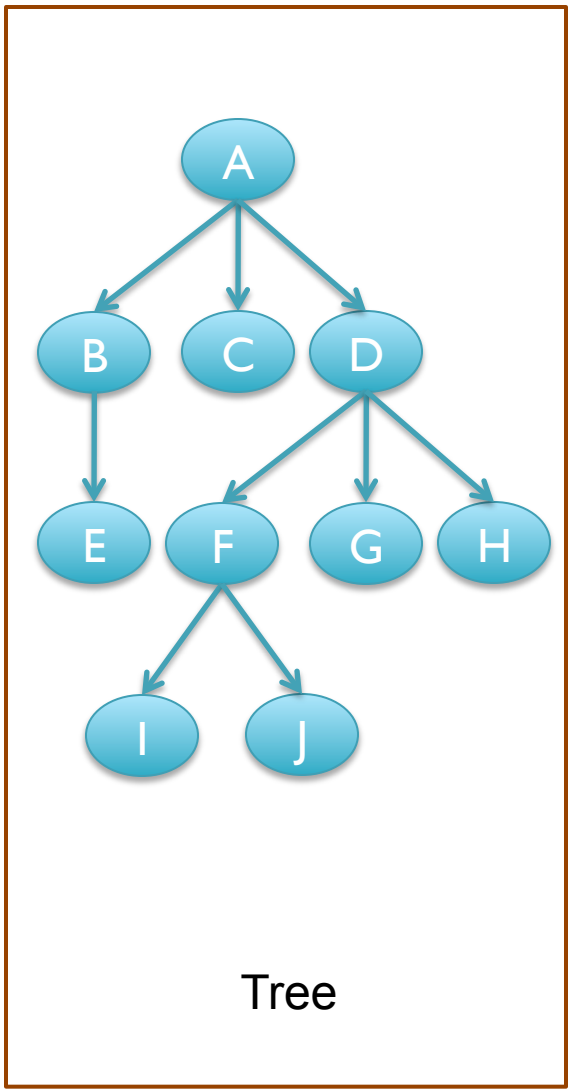
Graphs

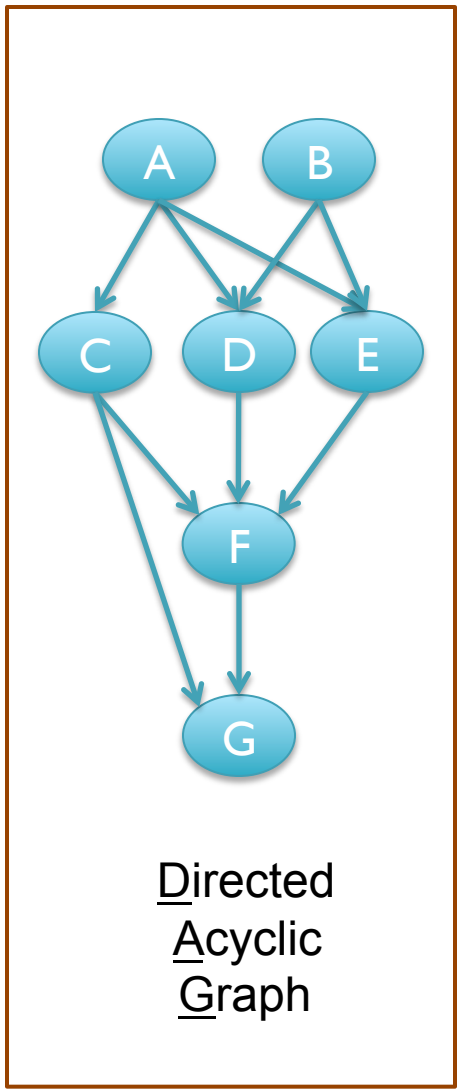


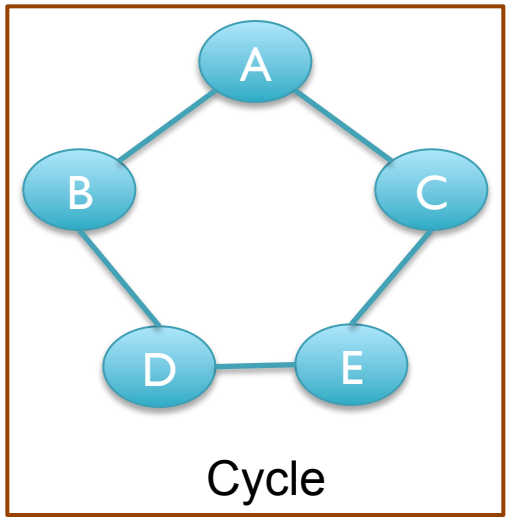
- Nodes
  - People, Proteins, Genes, Neurons, Sequences, Numbers, …
- Edges
  - A is connected to B
  - A is related to B
  - A regulates B
  - A precedes B
  - A interacts with B
  - A is related to B
  - …

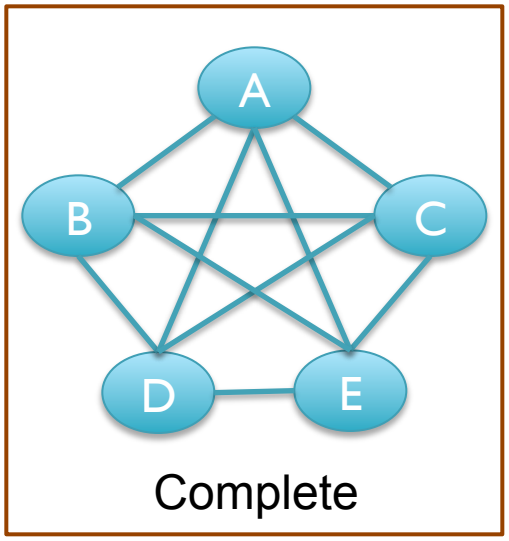
Graph Types


A
B
C
List

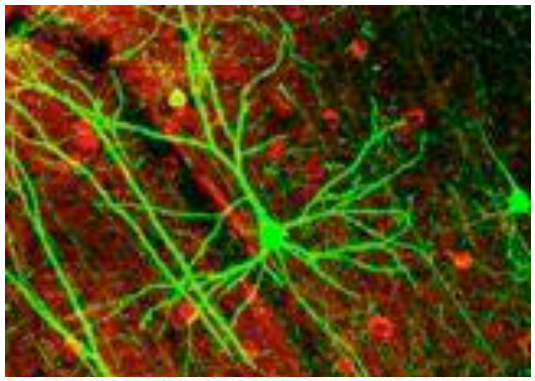

A
B
C
D
E
F
G
H
I
J
Tree


A
B
C
D
E
F
G
Directed
Acyclic
Graph


A
B
C
D
E
Cycle


A
B
C
D
E
Complete

Biological Networks



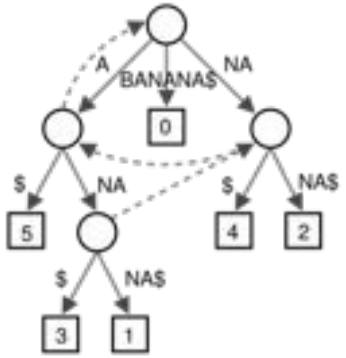
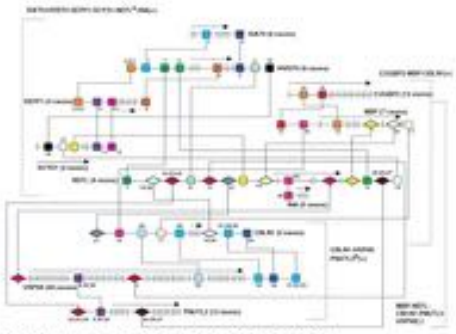
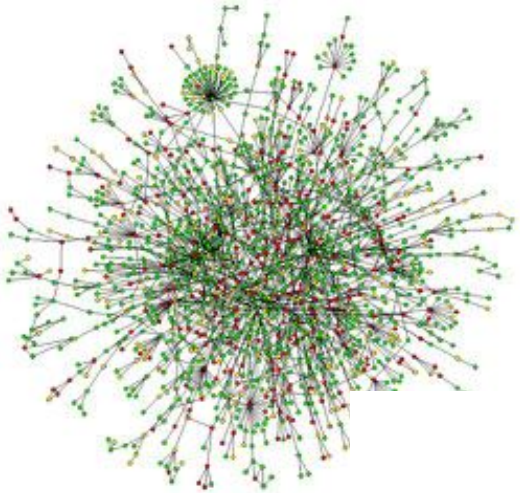
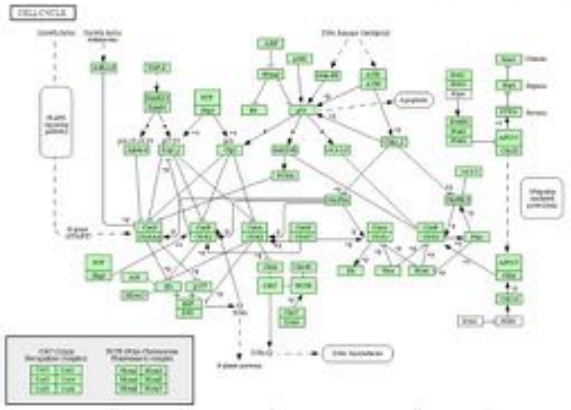


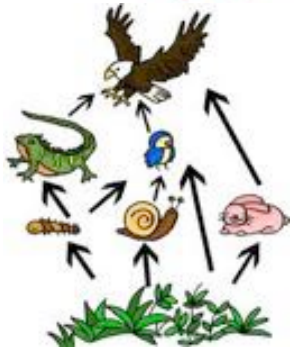
Figure 5 Putative regulatory elements shared between groups of correlated and anticorrelated genes

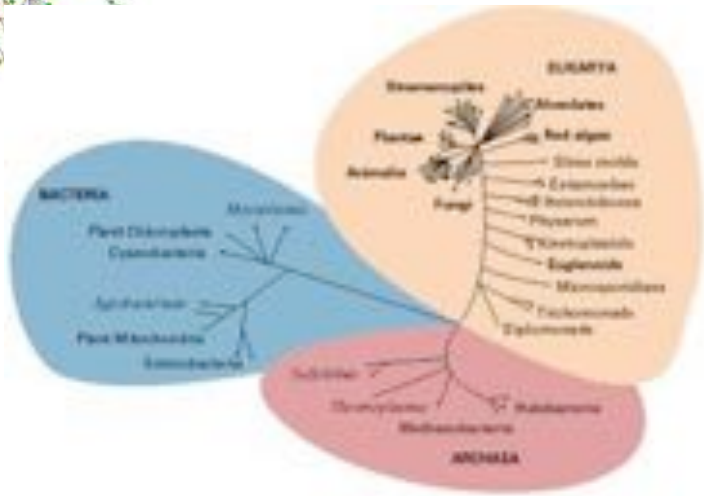


Vanessa M. Brown et al. Genome Res. 2002; 12: 868-884



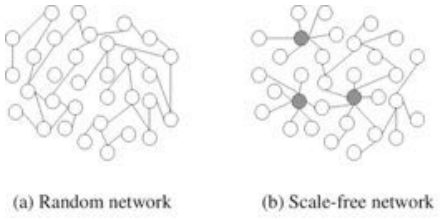


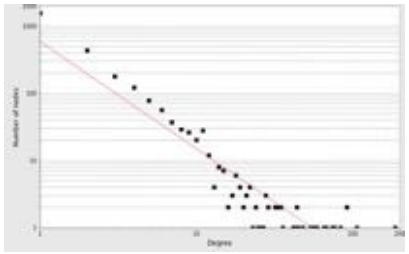


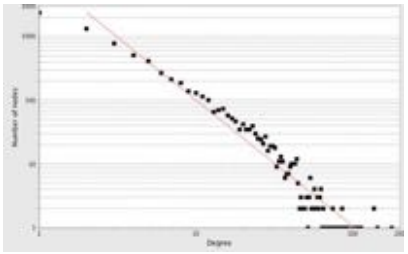


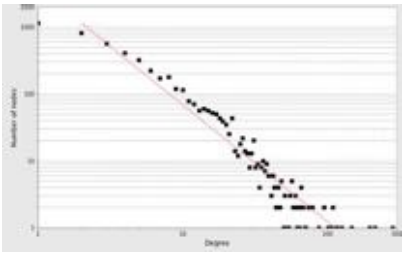
Network Characteristics

	C. elegans	*D. melanogaster*	*S. cerevisiae*
# Nodes	2646	7464	4965
# Edges	4037	22831	17536
Avg. / Max Degree	3.0 / 187	6.1 / 178	7.0 / 283
# Components	109	66	32
Largest Component	2386	7335	4906
Diameter	14	12	11
Avg. Shortest Path	4.8	4.4	4.1
Data Sources	2H	2x2H, TAP-MS	8x2H, 2xTAP, SUS









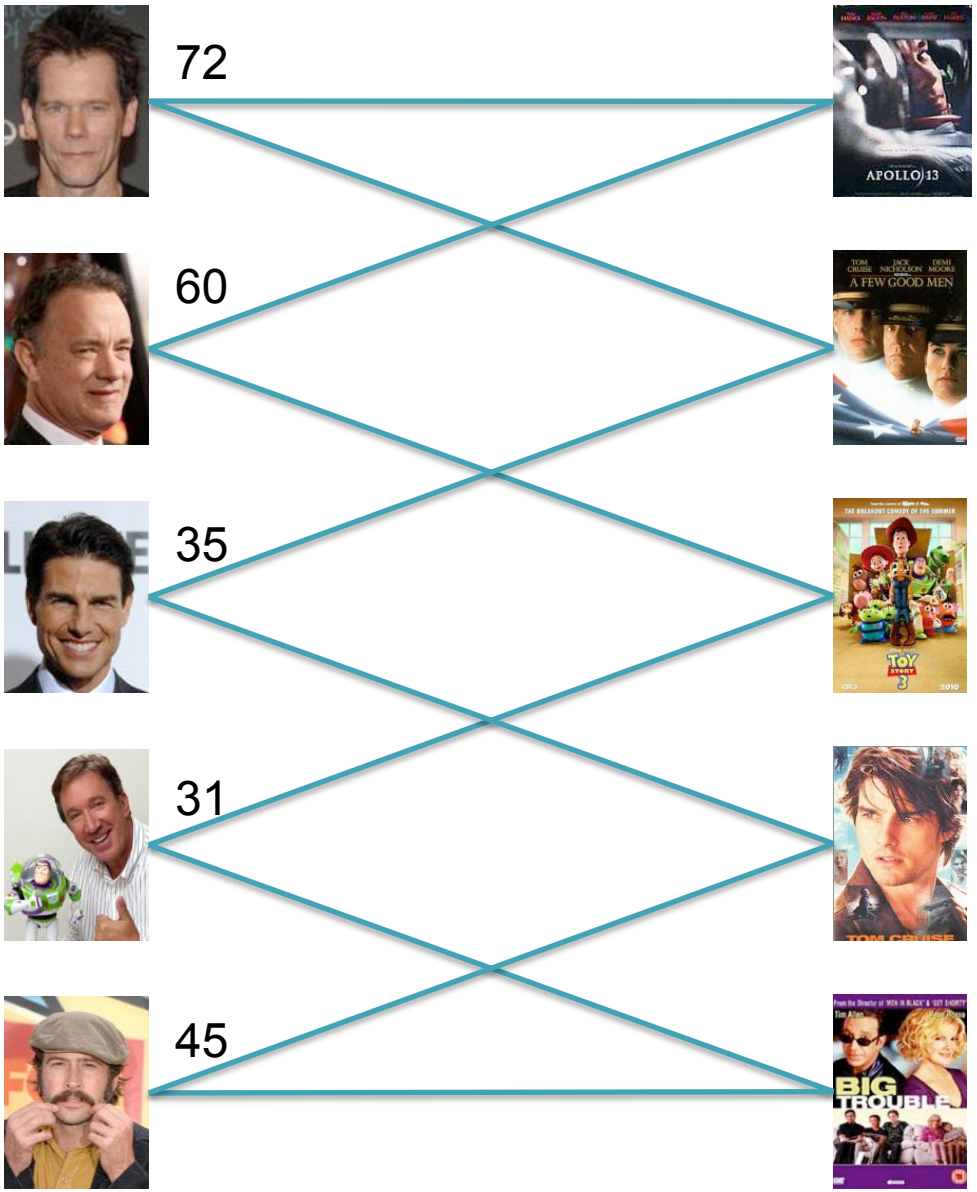
Small World: Avg. Shortest Path between nodes (proteins) is small
Scale Free: Power law distribution of degree – preferential attachment

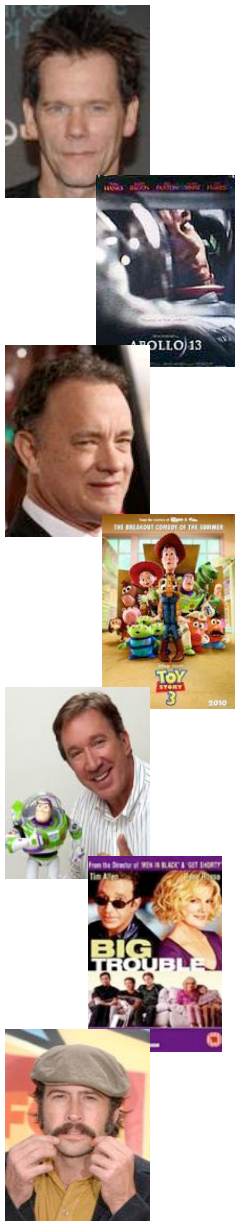
Kevin Bacon and Bipartite Graphs

Q1:
Find ***any*** path
from
Kevin Bacon
to
Jason Lee

Depth First Search:
6 hops

Bacon Distance:
3



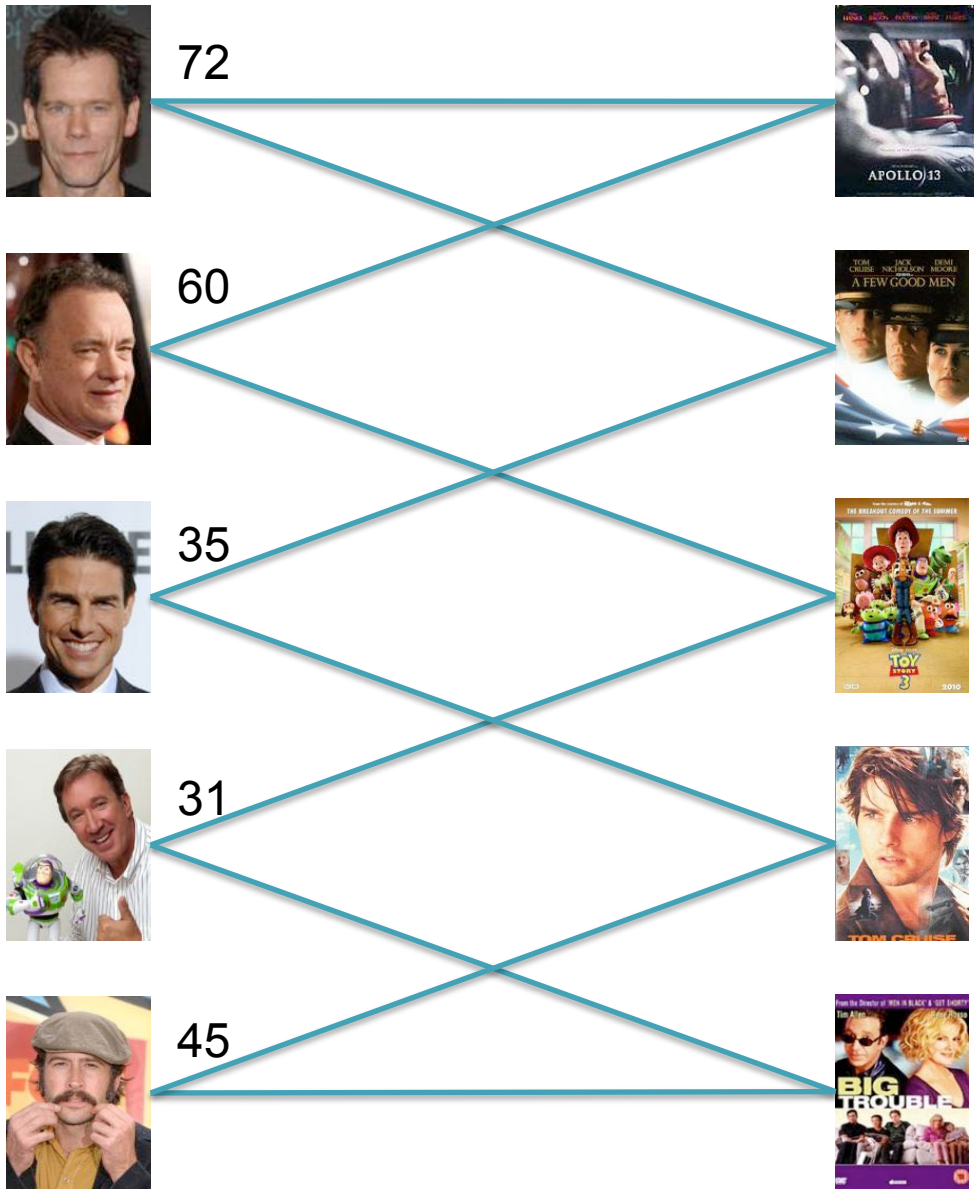


Kevin Bacon and Bipartite Graphs

Q2:
Find the ***shortest*** path from Kevin Bacon to Jason Lee

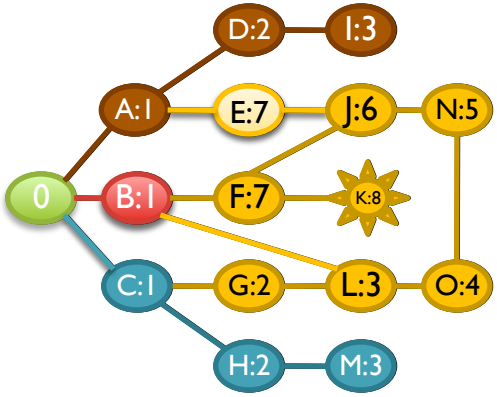
Breadth First Search:
4 hops

Bacon Distance:
2



DFS

```
DFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
   cur = list.end()
   if (cur == stop)
      print cur.dist;
   else
      foreach child in cur.children
         if (child.dist == -1)
            child.dist = cur.dist+1
            list.addEnd(child)
```



0
A,B,C
A,B,G,H
A,B,G,M
A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E
A,B
A
D
I

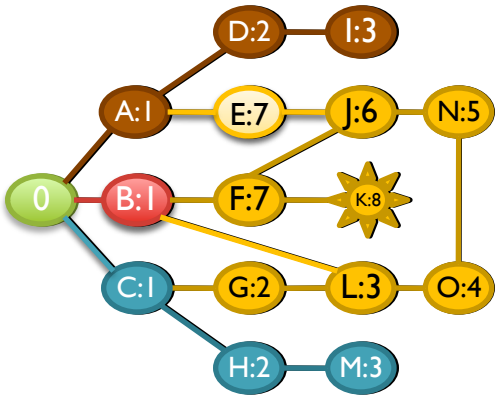
[How many nodes will it visit?]

[What's the running time?]

[What happens for disconnected components?]

DFS

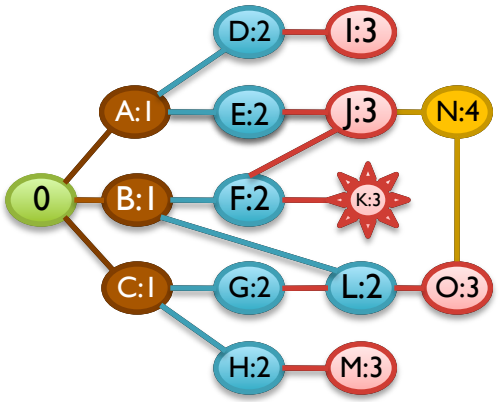
```
DFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
   cur = list.end()
   if (cur == stop)
      print cur.dist;
   else
      foreach child in cur.children
         if (child.dist == -1)
            child.dist = cur.dist+1
            list.addEnd(child)
```



0
A,B,C
A,B,G,H
A,B,G,M
A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E
A,B
A
D
I

BFS

```
BFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
   cur = list.begin()
   if (cur == stop)
      print cur.dist;
   else
      foreach child in cur.children
         if (child.dist == -1)
            child.dist = cur.dist+1
            list.addEnd(child)
```



0
A,B,C
B,C,D,E
C,D,E,F,L
D,E,F,L,G,H
E,F,L,G,H,I
F,L,G,H,I,J
L,G,H,I,J,K
G,H,I,J,K,O
H,I,J,K,O
I,J,K,O,M
J,K,O,M
K,O,M,N
O,M,N
M,N
N

BFS and TSP

- BFS computes the shortest path between a pair of nodes in $O(|E|) = O(|N|^2)$

- What if we wanted to compute the shortest route visiting every node once?
  - Traveling Salesman Problem

ABDCA: 4+2+5+3 = 14

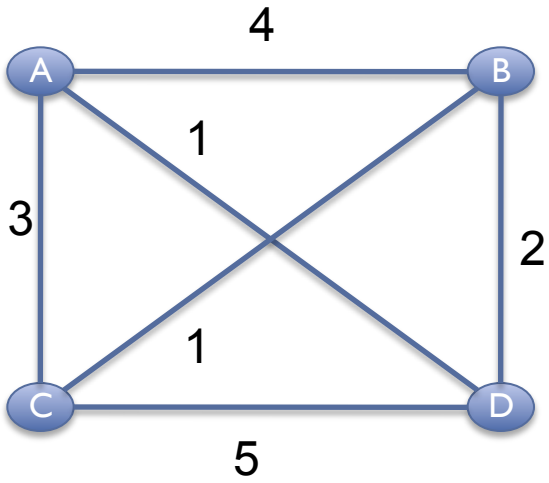
ACDBA: 3+5+2+4 = 14*

ABCDA: 4+1+5+1 = 11

ADCBA: 1+5+1+4 = 11*

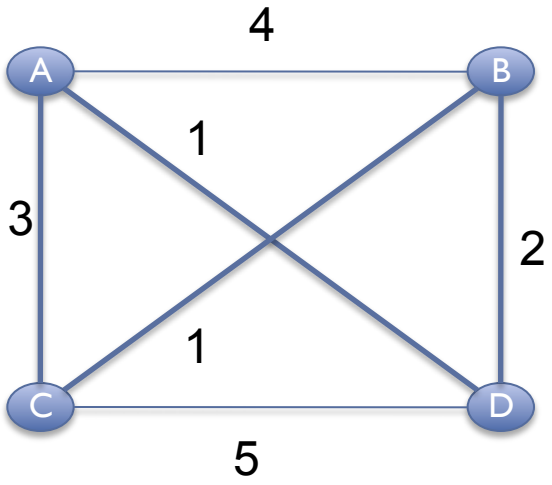
ACBDA: 3+1+2+1 = 7

ADBCA: 1+2+1+3= 7 *



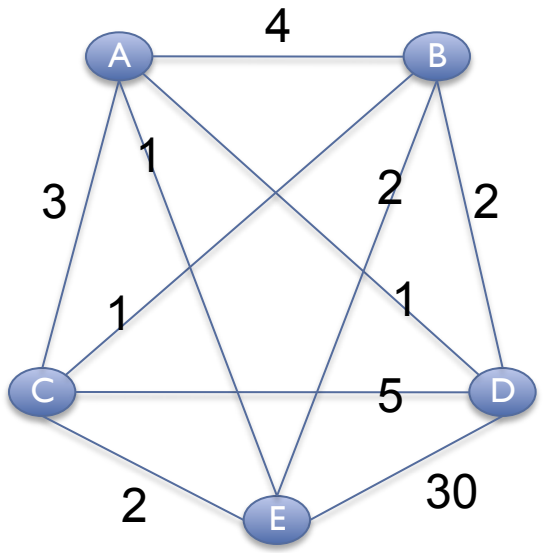
TSP Hardness

- No known way to partition the problem
  - Knowing optimal tour through n cities doesn't seem to help much for n+1 cities



[How many possible tours for n cities?]

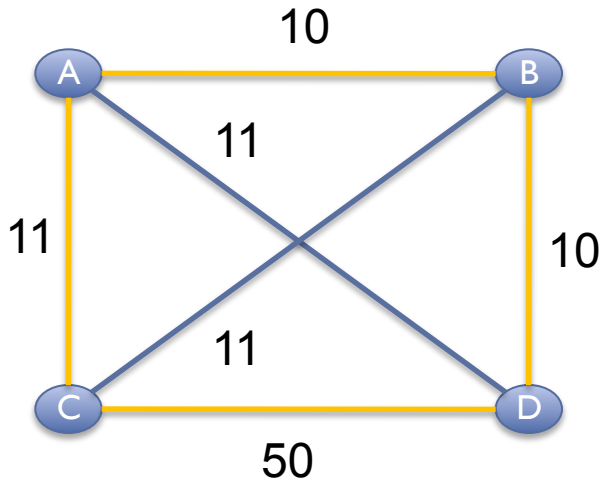
- Extensive searching is the only known provably correct algorithm
  - Brute Force: O(n!)
    - ~20 cities max
    - 20! = 2.4 x 10^{18}



Greedy Search

Greedy Search

```
cur=graph.randNode()
while (!done)
    next=cur.getNextClosest()
```



Greedy: ABDCA = 10+10+50+11= 81

Optimal: ACBDA = 11+11+10+11 = 43

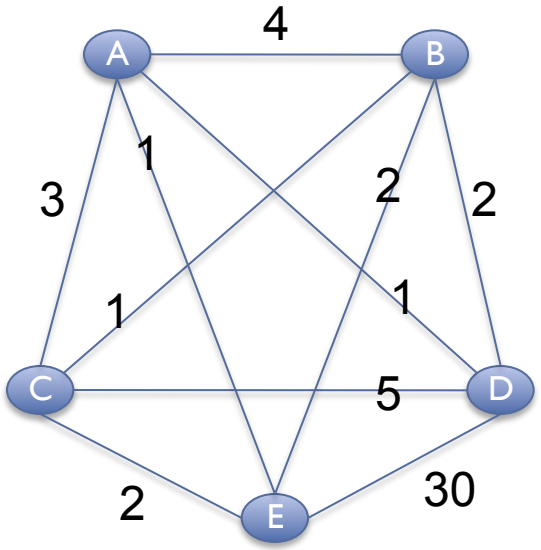
Greedy finds the global optimum only when

1. Greedy Choice: Local is correct without reconsideration
2. Optimal Substructure: Problem can be split into subproblems

Optimal Greedy: Making change with the fewest number of coins

Branch-and-Bound

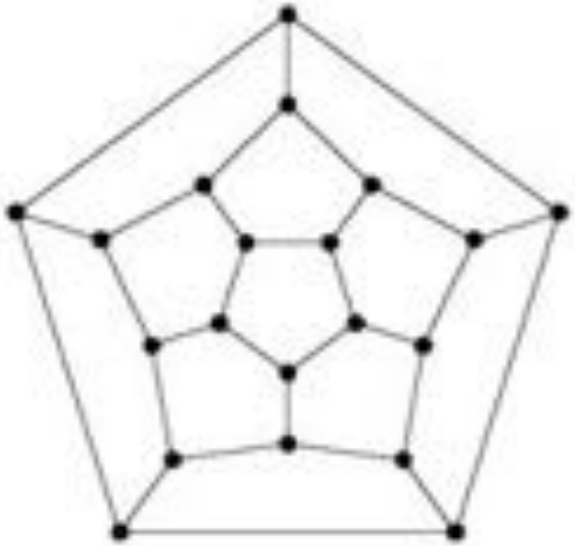
- Abort on suboptimal solutions as soon as possible
  - ADBECA = 1+2+2+2+3 = 10
  - ABDE = 4+2+30 > 10
  - ADE = 1+30 > 10
  - AED = 1+30 > 10
  - …



- Performance Heuristic
  - Always gives the optimal answer
  - Doesn't always help performance, but often does
  - Current TSP record holder:
    - 85,900 cities
    - 85900! = 10^{386526}

[When not?]

TSP and NP-complete



- TSP is one of many extremely hard problems of the class NP-complete
  - Extensive searching is the only way to find an exact solution
  - Often have to settle for approx. solution
- WARNING: Many optimization problems are in this class
  - Find a tour the visits every node once
  - Find the smallest set of vertices covering all the edges
  - Find the largest clique in the graph
  - Find a set of items with maximal value but limited weight
  - Maximizing the number of tetris pieces played
  - …
  - http://en.wikipedia.org/wiki/List_of_NP-complete_problems

Shortest Common Superstring

Given: $S = \{s_1, \ldots, s_n\}$

Problem: Find minimal length superstring of S

s_1 CACCC

s_2 CCGGGTGC

s_3 CCACC

$s_{1,}s_2,s_3$ = CACCCGGGTGCCACC 15

s_1,s_3,s_2 = CACCCACCGGGTGC14

s_2,s_1,s_3 = CCGGGTGCACCCACC 15

s_2,s_3,s_1 = CCGGGTGCCACCC 13

s_3,s_1,s_2 = CCACCCGGGTGC 12

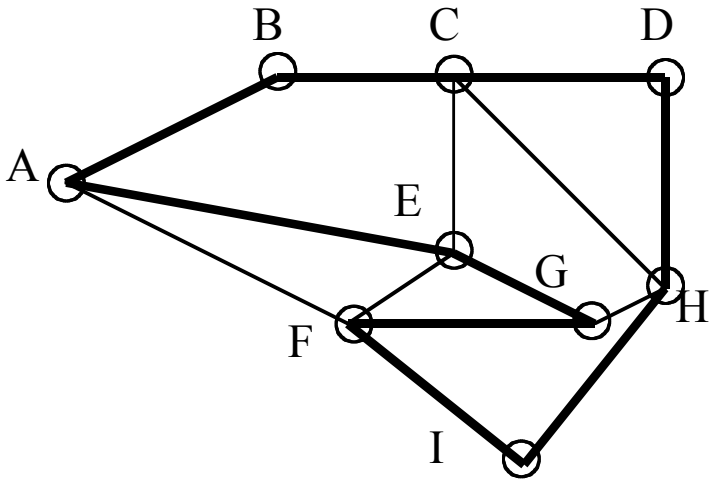
s_3,s_2,s_1 = CCACCGGGTGCACCC 15

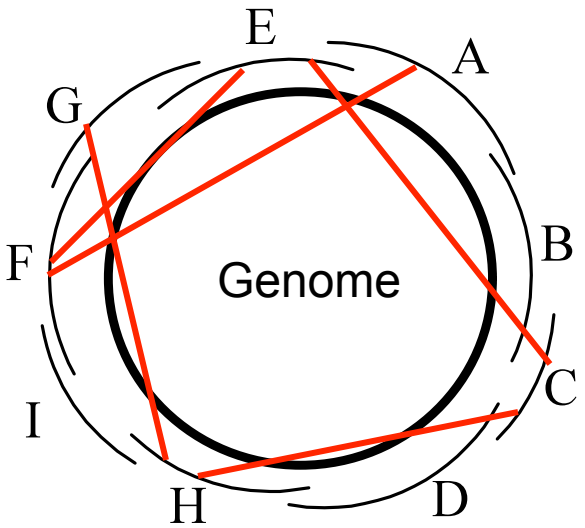
NP-Complete by reduction from VERTEX-COVER and later DIRECTED-HAMILTONIAN-PATH

Paths through graphs and assembly

- Hamiltonian circuit: visit each node (city) exactly once, returning to the start
  - If we could do this fast, we could exactly assemble genomes as the shortest common superstring

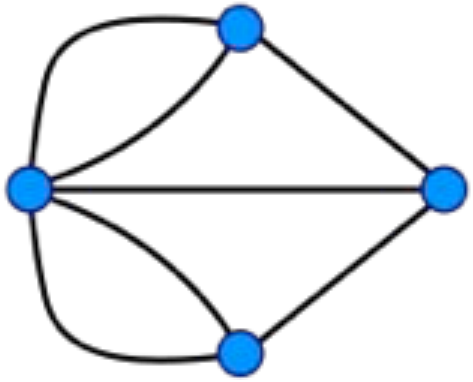
    [Is this the right model for assembly?]





Eulerian Cycle Problem

- **Seven Bridges of Königsberg**
  - Find a cycle that visits every ***edge*** exactly once



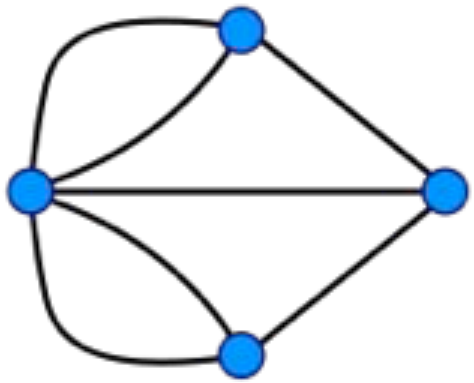
[Can you find the cycle?]

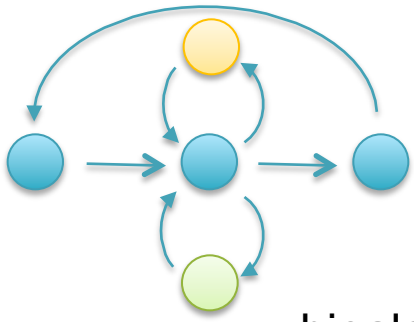
Euler Theorem

- A graph is ***balanced*** if for every vertex the number of incoming edges equals to the number of outgoing edges:

$$in(v)=out(v)$$

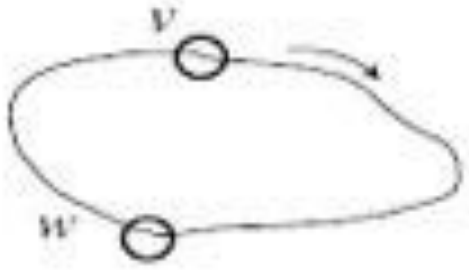
- **Theorem**: *A connected graph is Eulerian if and only if each of its vertices is balanced.*





Algorithm for Constructing an Eulerian Cycle

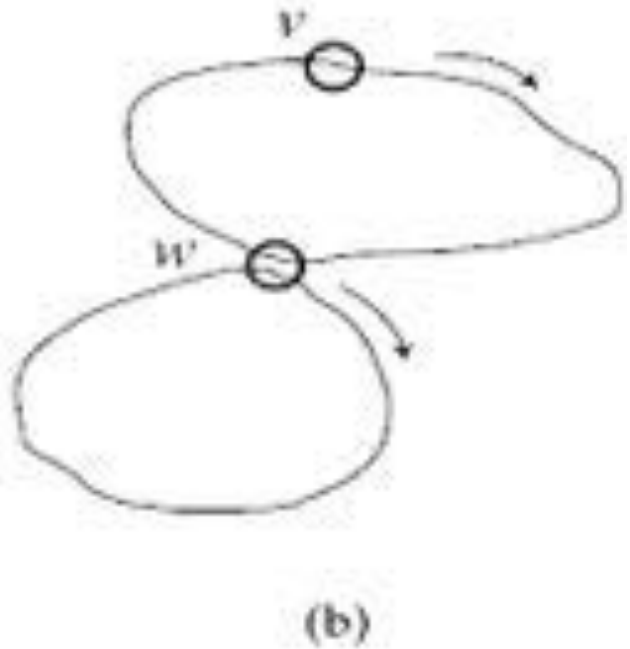
a. Start with an arbitrary vertex *v* and form an arbitrary cycle with unused edges until a dead end is reached. Since the graph is Eulerian this dead end is necessarily the starting point, i.e., vertex *v*.



(a)

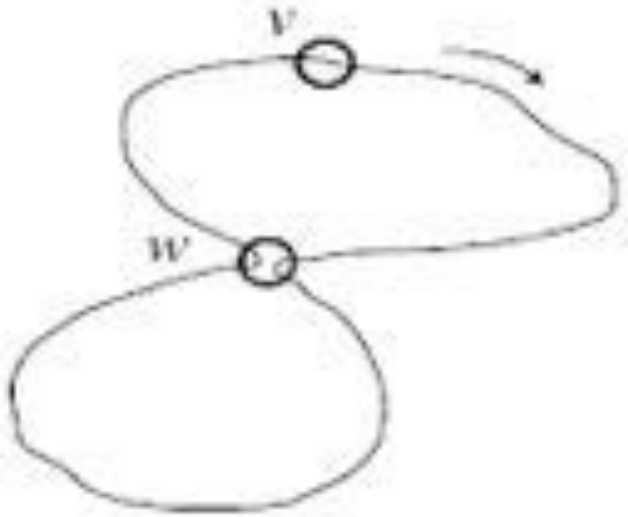
Algorithm for Constructing an Eulerian Cycle (cont'd)

b. If cycle from (a) above is not an Eulerian cycle, it must contain a vertex *w*, which has untraversed edges. Perform step (a) again, using vertex *w* as the starting point. Once again, we will end up in the starting vertex *w*.



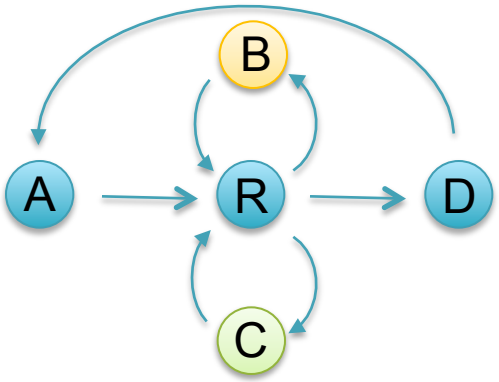
Algorithm for Constructing an Eulerian Cycle (cont'd)

c. Combine the cycles from (a) and (b) into a single cycle and iterate step (b).



(c)

Counting Eulerian Tours



ARBRCRD
or
ARCRBRD

Generally an exponential number of compatible sequences

– Value computed by application of the BEST theorem (Hutchinson, 1975)

$$\mathcal{W}(G,t) = (\det L)\left\{\prod_{u\in V}(r_u-1)!\right\}\left\{\prod_{(u,v)\in E} a_{uv}!\right\}^{-1}$$

L = *n x n* matrix with r_u-a_{uu} along the diagonal and -a_{uv} in entry uv

r_u = $d^+(u)+1$ if $u=t$, or $d^+(u)$ otherwise

a_{uv} = multiplicity of edge from *u* to *v*

Assembly Complexity of Prokaryotic Genomes using Short Reads.
Kingsford C, Schatz MC, Pop M (2010) *BMC Bioinformatics*.

Break


PHASE TWO: INTERPRETATION
I THINK I FOUND A CORNER PIECE.
3 BILLION PIECES
GENOME

Milestones in Genome Assembly


articles

Nucleotide sequence of bacteriophage ΦX174 DNA

F. Sanger, G. M. Air*, B. G. Barrell, N. L. Brown*, A. R. Coulson, J. C. Fiddes, C. A. Hutchison III*, P. M. Slocombe* & M. Smith*

1977. Sanger *et al.*
1st Complete Organism
5375 bp

1995. Fleischmann *et al.*
1st Free Living Organism
TIGR Assembler. 1.8Mbp

1998. C.elegans SC
1st Multicellular Organism
BAC-by-BAC Phrap. 97Mbp

2000. Myers *et al.*
1st Large WGS Assembly.
Celera Assembler. 116 Mbp

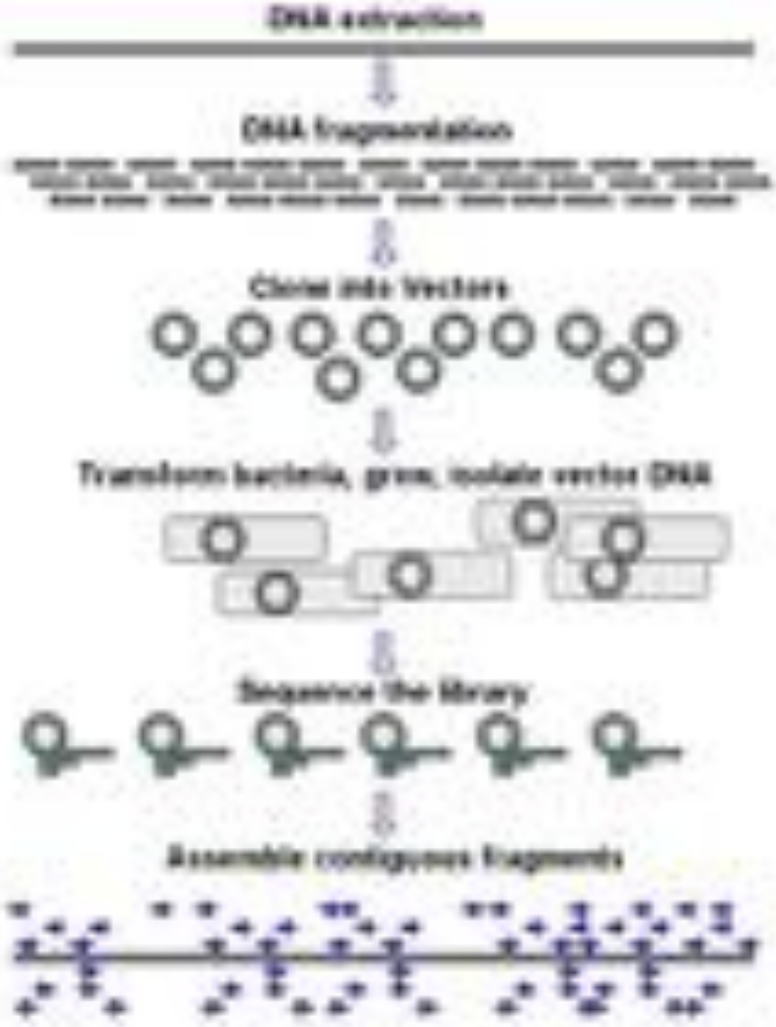
2001. Venter *et al.*, IHGSC
Human Genome
Celera Assembler/GigaAssembler. 2.9 Gbp

2010. Li *et al.*
1st Large SGS Assembly.
SOAPdenovo 2.2 Gbp

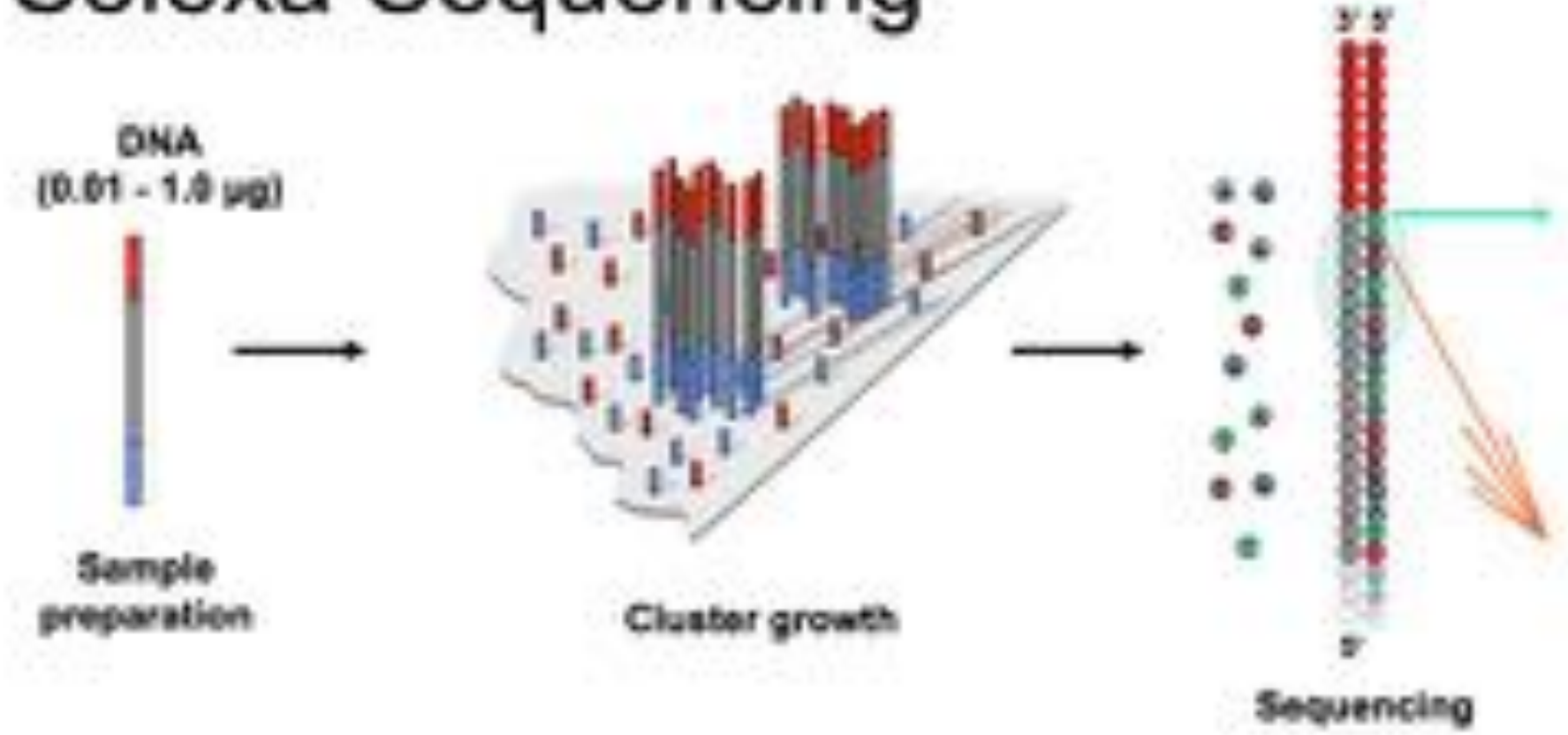
"old" way of genome sequencing

Cloning and clone handling are very labor intensive

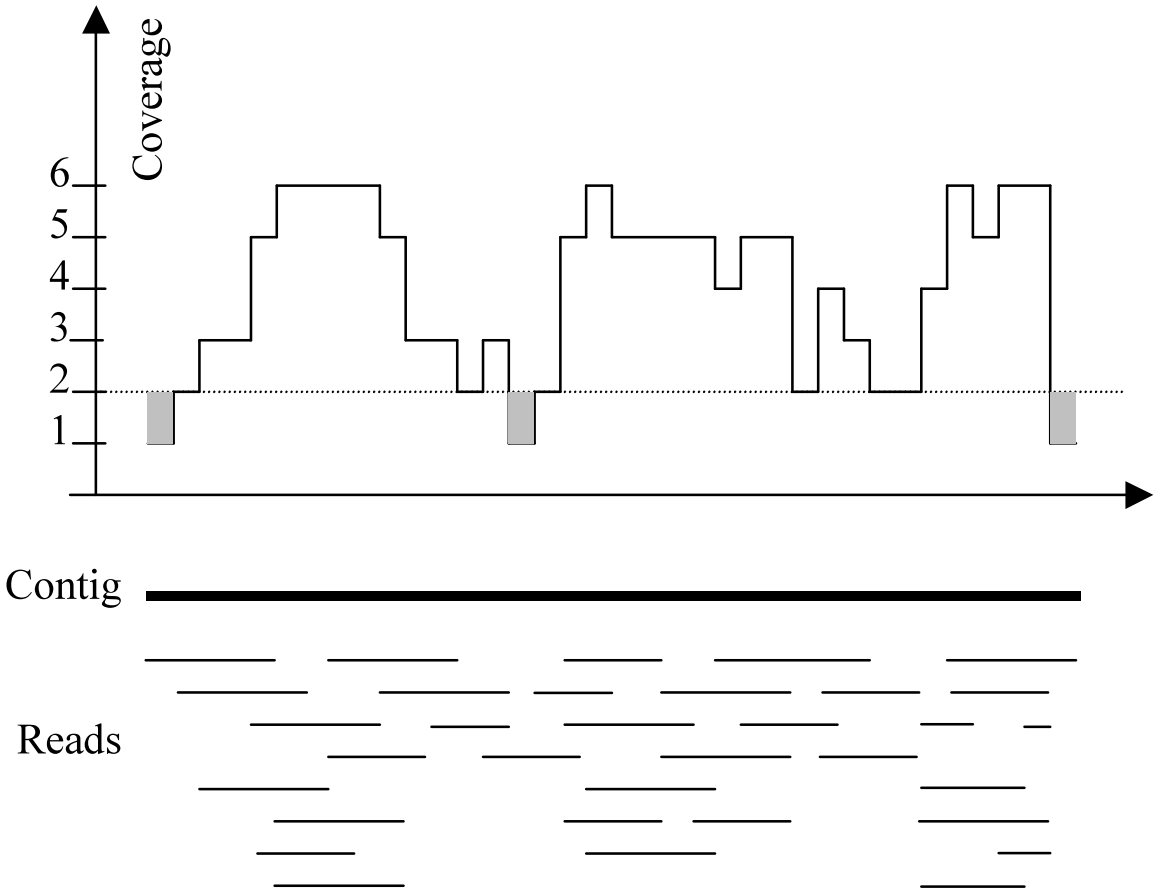
Throughput of capillary sequencing machines is limited



Solexa Sequencing

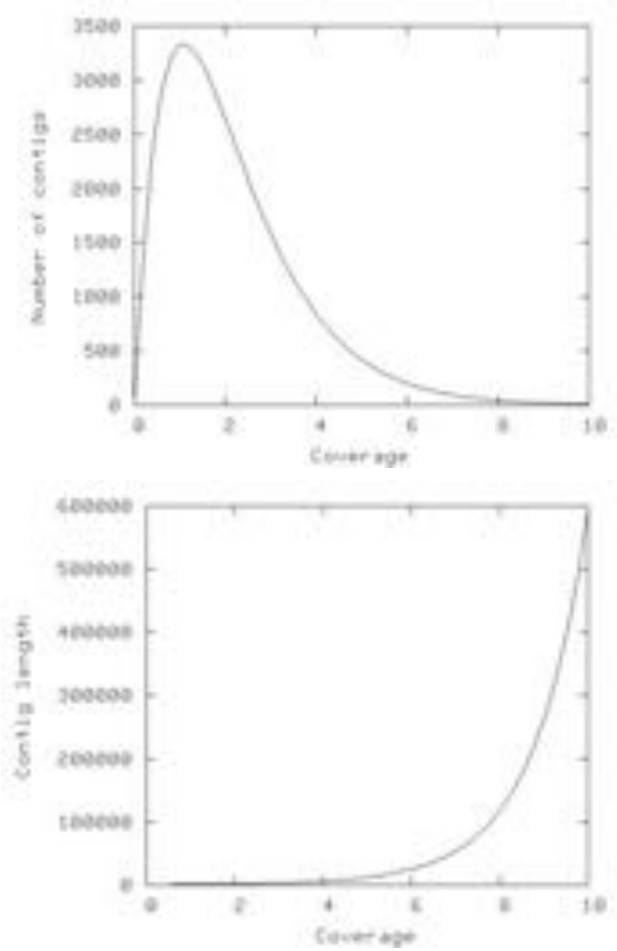


Typical contig coverage



Imagine raindrops on a sidewalk

Lander-Waterman statistics



L = read length

T = minimum overlap

G = genome size

N = number of reads

c = coverage (NL / G)

$\sigma = 1 - T/L$

$E(\#islands) = Ne^{-c\sigma}$

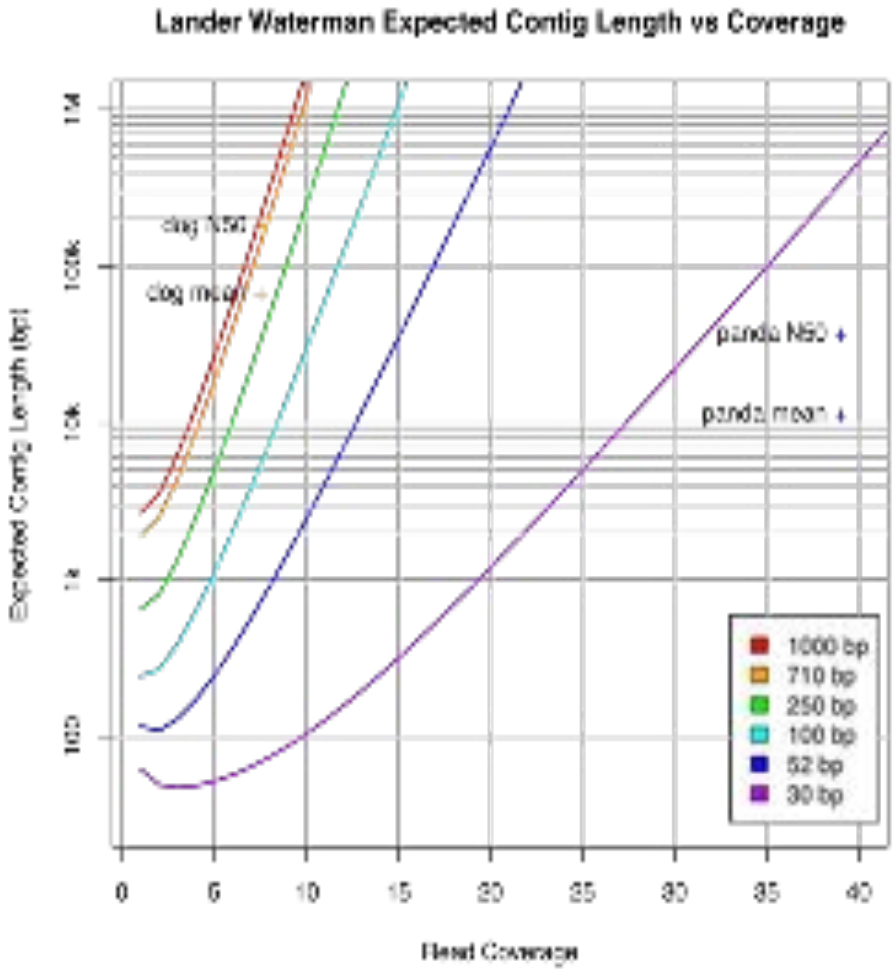
$E(island\ size) = L(e^{c\sigma} - 1) / c + 1 - \sigma$

contig = island with 2 or more reads

Genome Coverage

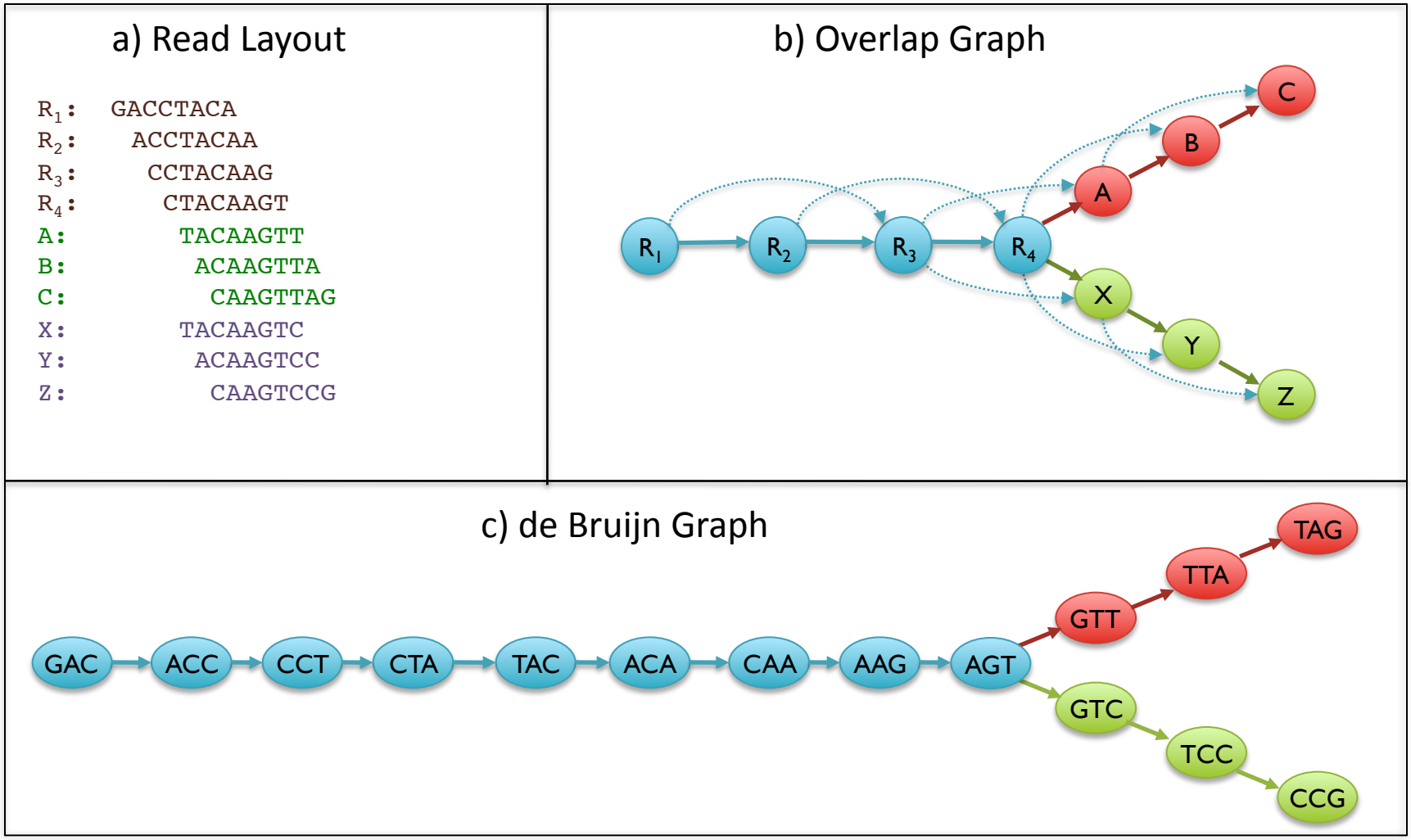
Idealized assembly

- Uniform probability of a read starting at a given position
  - p = G/N
- Poisson distribution in coverage along genome
  - Contigs end when there is no overlapping read
- Contig length is a function of coverage and read length
  - Short reads require much higher coverage



Assembly of Large Genomes using Second Generation Sequencing
Schatz MC, Delcher AL, Salzberg SL (2010) *Genome Research* 20, 1165-73.

Two Paradigms for Assembly



Assembly of Large Genomes using Second Generation Sequencing
Schatz MC, Delcher AL, Salzberg SL (2010) *Genome Research* 20, 1165-73.

Overlap-Layout-Consensus

Assemblers: ARACHNE, PHRAP, CAP, TIGR, CELERA

Overlap: find potentially overlapping reads

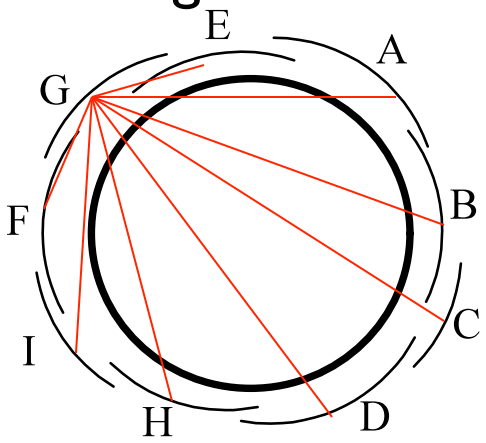
Layout: merge reads into contigs and contigs into supercontigs

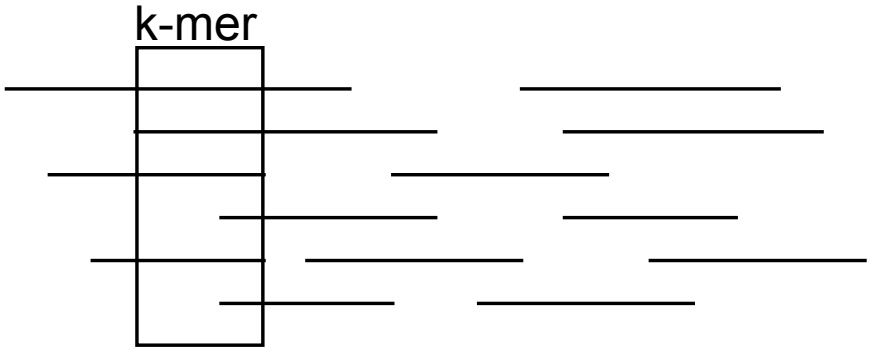
Consensus: derive the DNA sequence and correct read errors

..ACGATTACAATAGGTT..

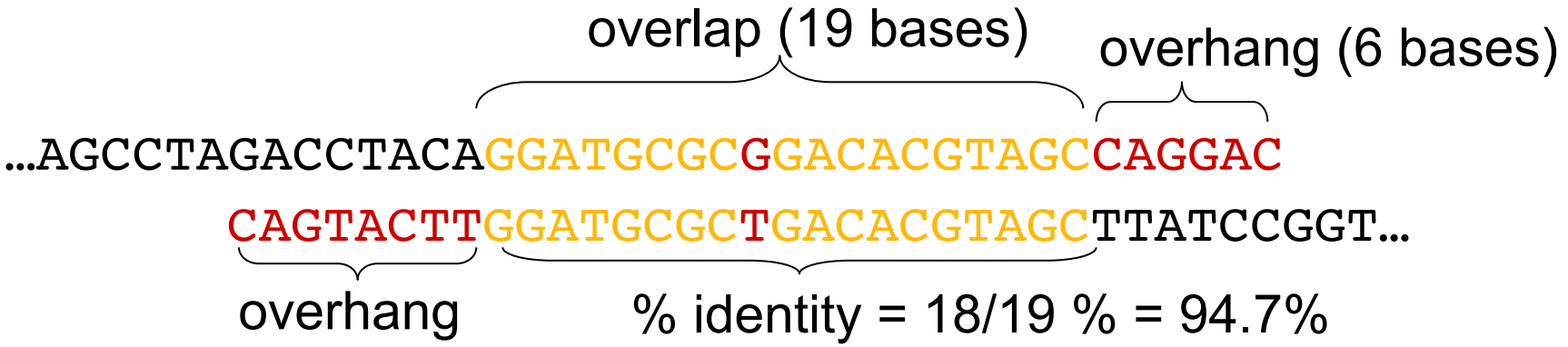
All pairs alignment

- Needed by the assembler
- Try all pairs – must consider ~ n^2 pairs
- Smarter solution: only n x coverage (e.g. 8) pairs are possible
  - Build a table of k-mers contained in sequences (single pass through the genome)
  - Generate the pairs from k-mer table (single pass through k-mer table)





Overlap between two sequences



overlap - region of similarity between regions
overhang - un-aligned ends of the sequences

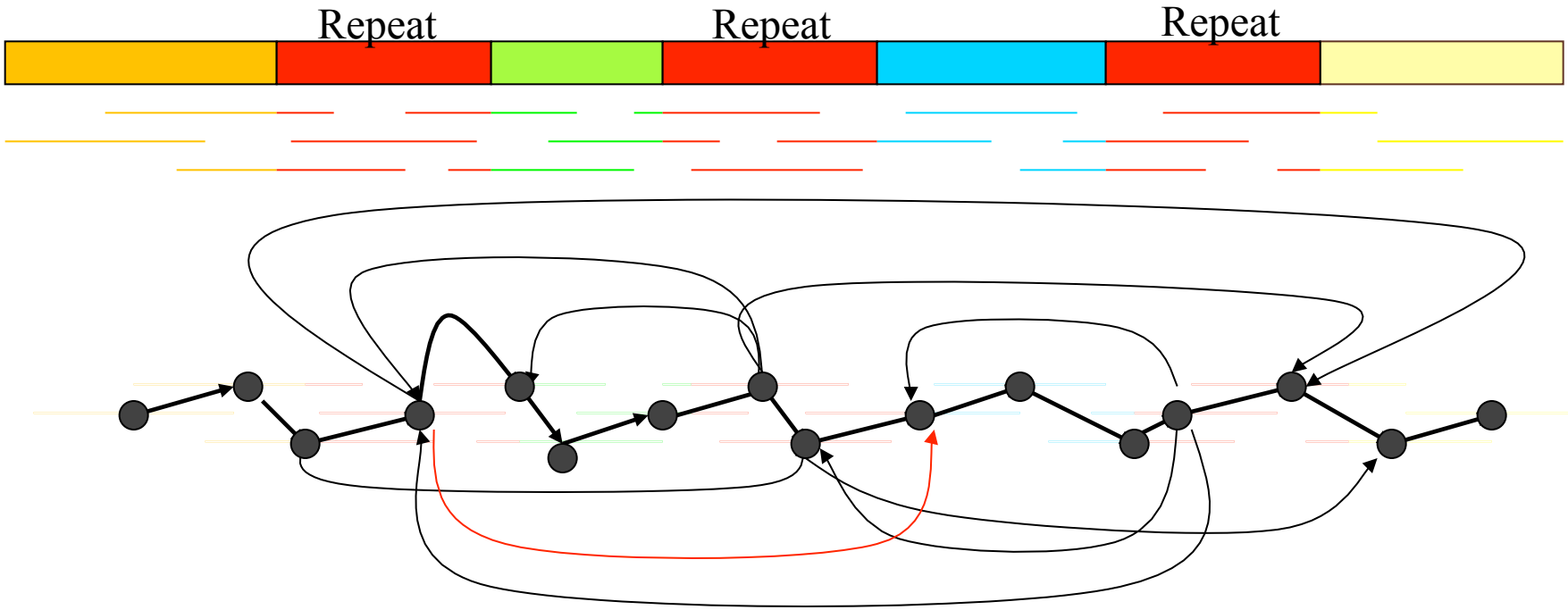
The assembler screens merges based on:

- length of overlap
- % identity in overlap region
- maximum overhang size.

[How do we compute the overlap?]

Overlap Graph: Hamiltonian Approach

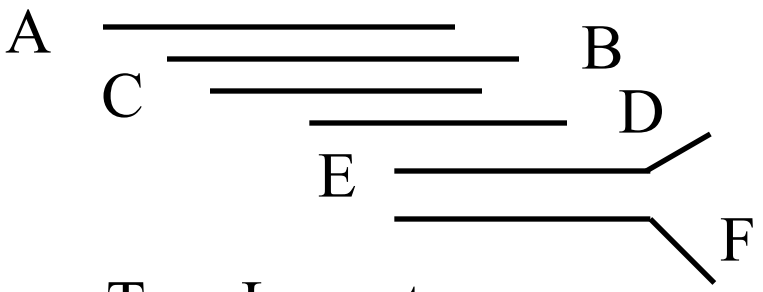
Each vertex represents a read from the original sequence.
Vertices from repeats are connected to many others.



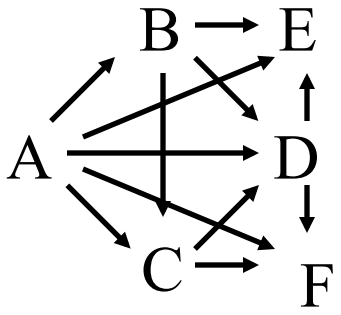
Repeat Types

- **Low-Complexity DNA** (e.g. ATATATATACATA…)
- **Microsatellite repeats** $(a_1 \ldots a_k)^N$ where k ~ 3-6
  (e.g. CAGCAGTAGCAGCACCAG)
- **Transposons/retrotransposons**
  - **SINE** Short Interspersed Nuclear Elements
    (e.g., *Alu*: ~300 bp long, 10^6 copies)
  - **LINE** Long Interspersed Nuclear Elements
    ~500 - 5,000 bp long, 200,000 copies
  - **LTR retroposons** Long Terminal Repeats (~700 bp) at each end
- **Gene Families** genes duplicate & then diverge
- **Segmental duplications** ~very long, very similar copies
- A large fraction of the genome is repetitive
  => any repeat longer than the read length may be problematic

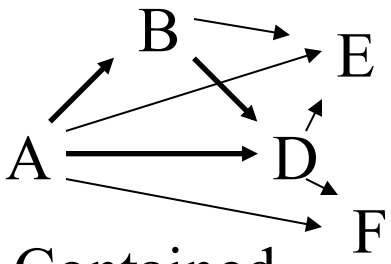
Unitigging: Pruning the Overlap



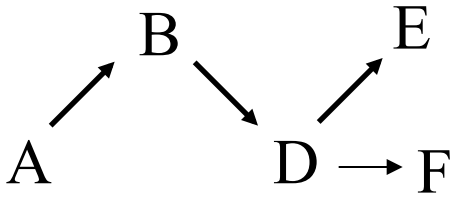
True Layout



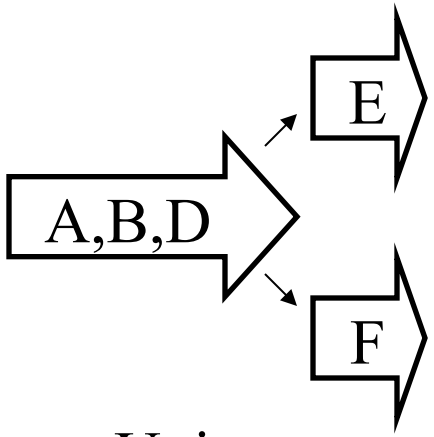
Original Overlap Graph



Contained Read Removal



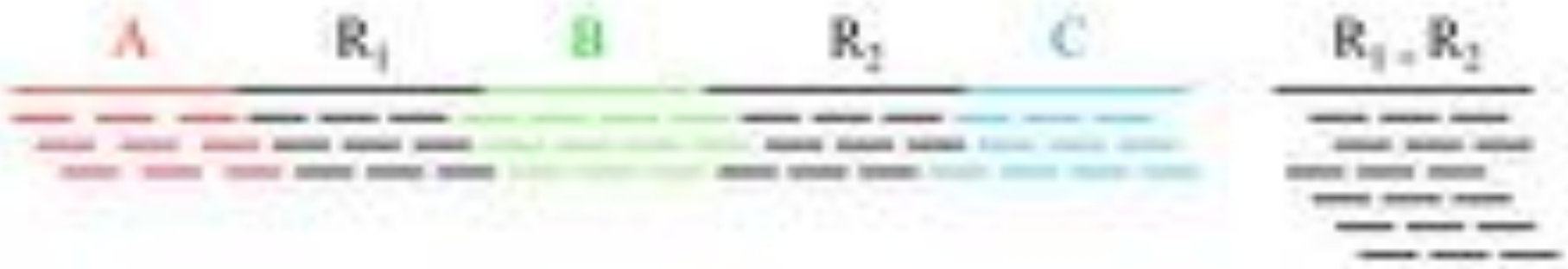
Transitive Edge Removal



Unique Join Collapsing

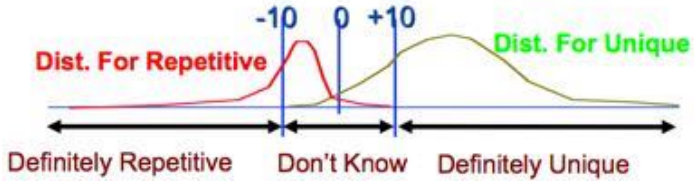
Theorem: SCS of unitigs = SCS of reads

A-stat



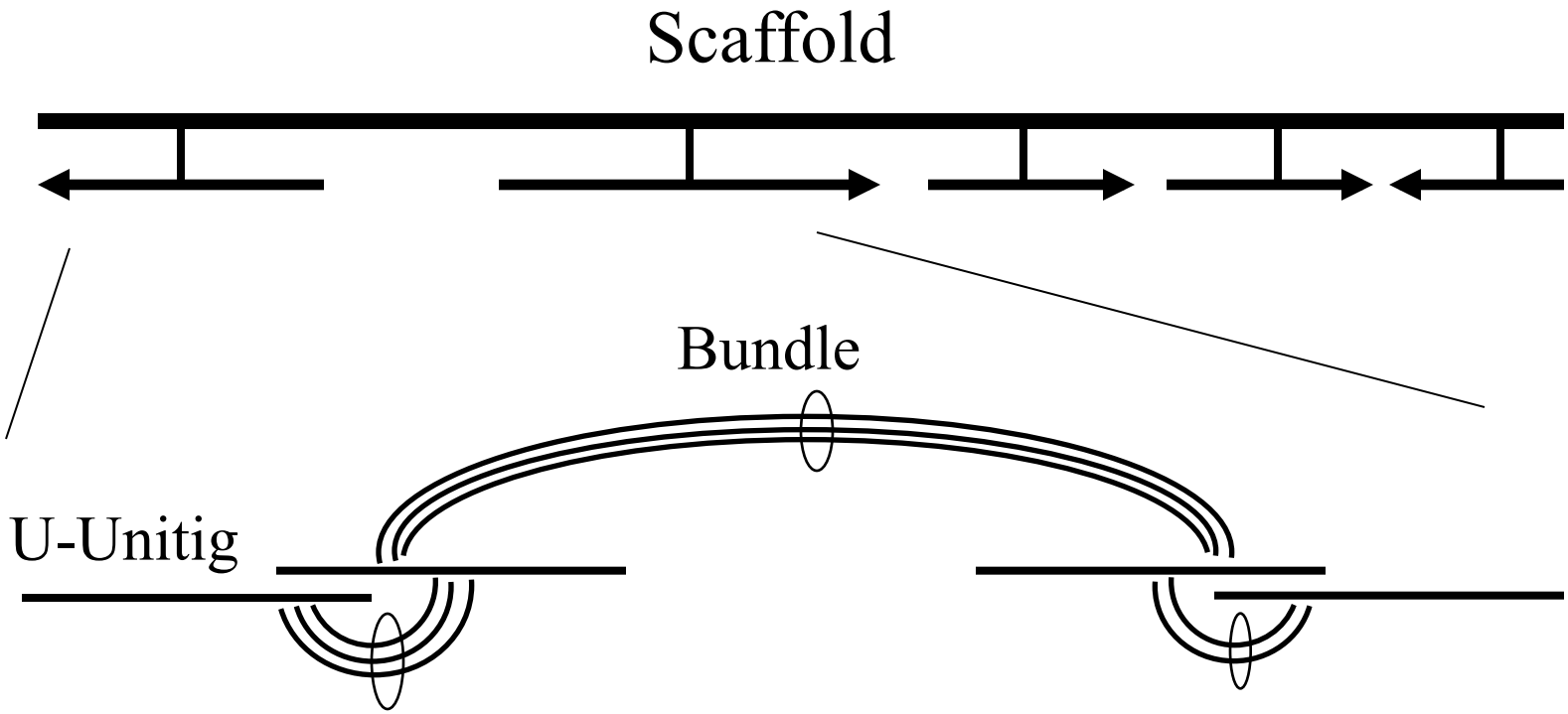
- If n reads are a uniform random sample of the genome of length G, we expect $k=n\Delta/G$ reads to start in a region of length Δ.
  - If we see many more reads than k (if the arrival rate is > λ), it is likely to be a collapsed repeat
  - Requires an accurate genome size estimate

Identify those that cover unique DNA = U-unitigs



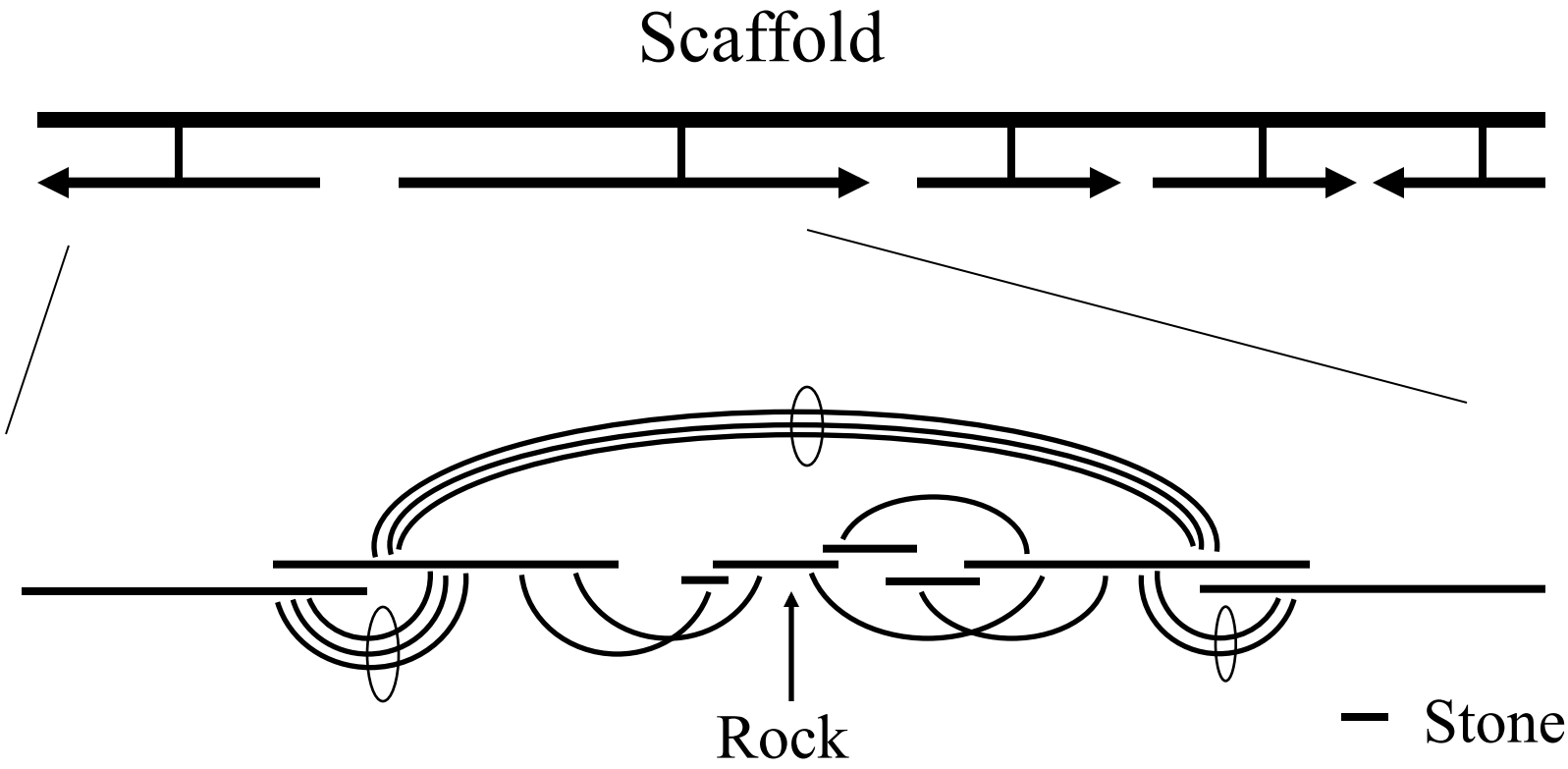
$$A(\Delta,\lambda) = \ln\left(\frac{P(k \le \cdots)}{P(k \ge \cdots)}\right) = \ln\left(\frac{\frac{(\lambda n/G)^{\cdots}\, e^{-\frac{\lambda n}{G}}}{k!}}{\frac{(\lambda n/G)^{\cdots}\, e^{-\frac{\lambda n}{G}}}{k!}}\right) = \frac{n\Delta}{G} - k\ln 2$$

Initial Scaffolding



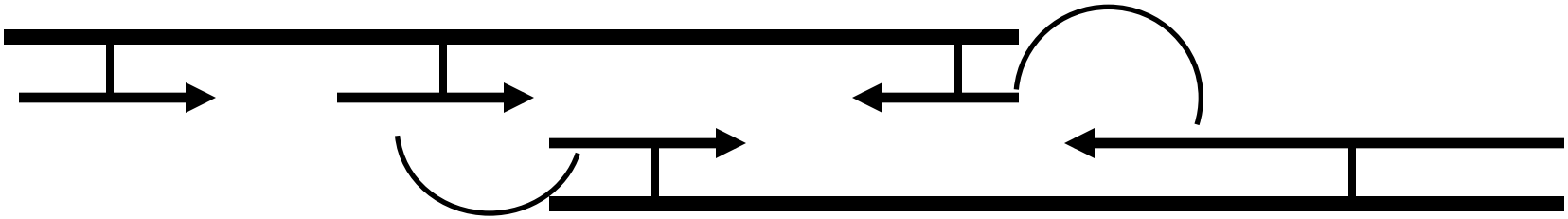
Create a initial scaffold of unique unitigs (U-Unitigs) whose A-stat > 5. Also recruit borderline unitigs whose A-stat is > 2 and have consistent mates with the U-Unitigs.

Repeat Resolution



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

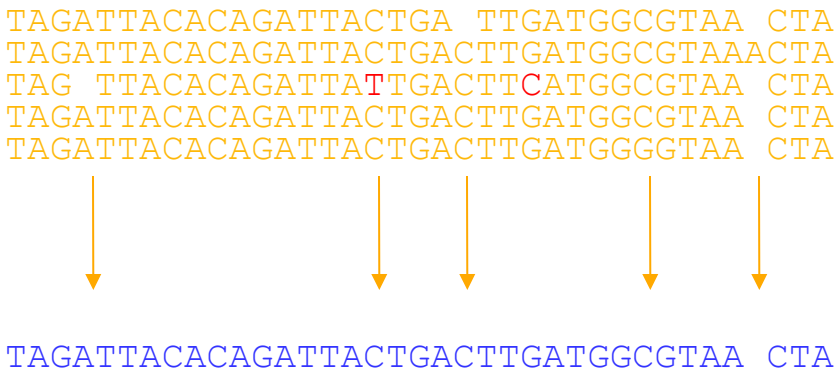
Scaffold merging



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.

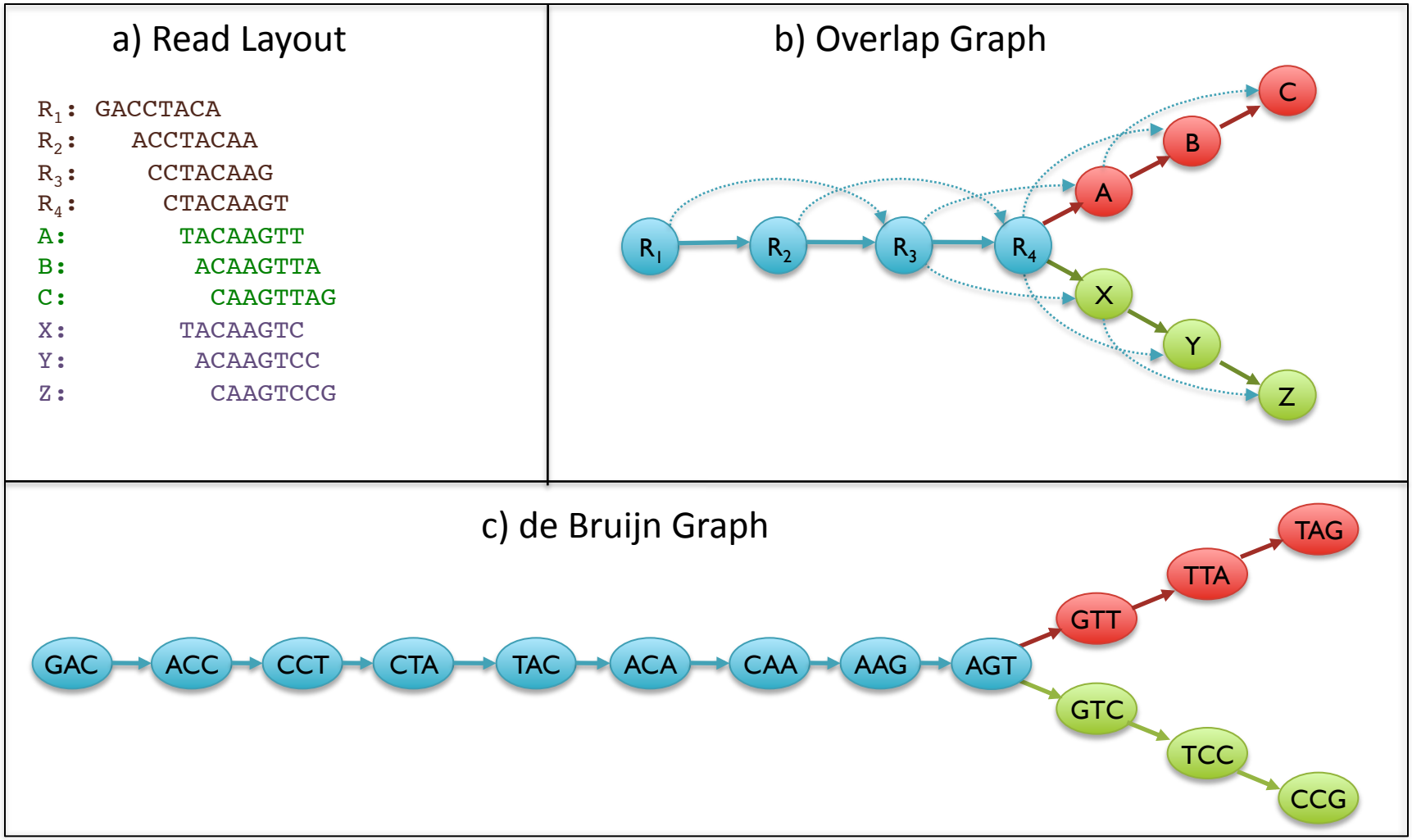
Derive Consensus Sequence



Derive multiple alignment from pairwise read alignments

Derive each consensus base by weighted voting

Two Paradigms for Assembly

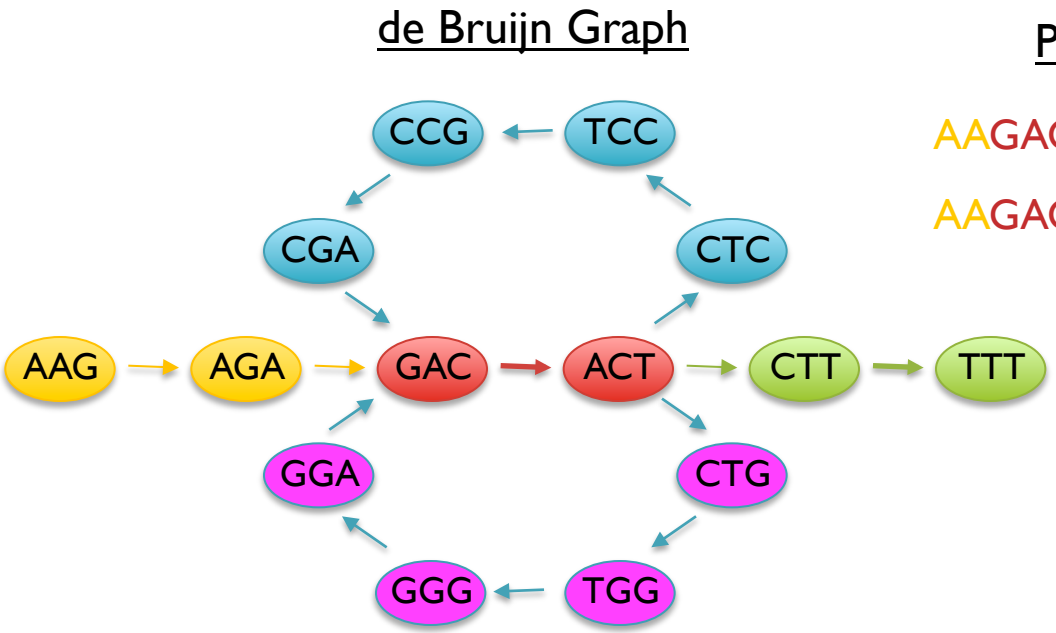


Assembly of Large Genomes using Second Generation Sequencing
Schatz MC, Delcher AL, Salzberg SL (2010) *Genome Research* 20, 1165-73.

Short Read Assembly

Reads

AAGA
ACTT
ACTC
ACTG
AGAG
CCGA
CGAC
CTCC
CTGG
CTTT
…



Potential Genomes

AAGACTCCGACTGGGACTTT

AAGACTGGGACTCCGACTTT

- Genome assembly as finding an Eulerian tour of the de Bruijn graph
  - Human genome: >3B nodes, >10B edges
- The new short read assemblers require tremendous computation
  - Velvet (Zerbino & Birney, 2008) serial: > 2TB of RAM
  - ABySS (Simpson *et al.*, 2009) MPI: 168 cores x ~96 hours
  - SOAPdenovo (Li *et al.*, 2010) pthreads: 40 cores x 40 hours, >140 GB RAM

Short Read Genome Assemblers

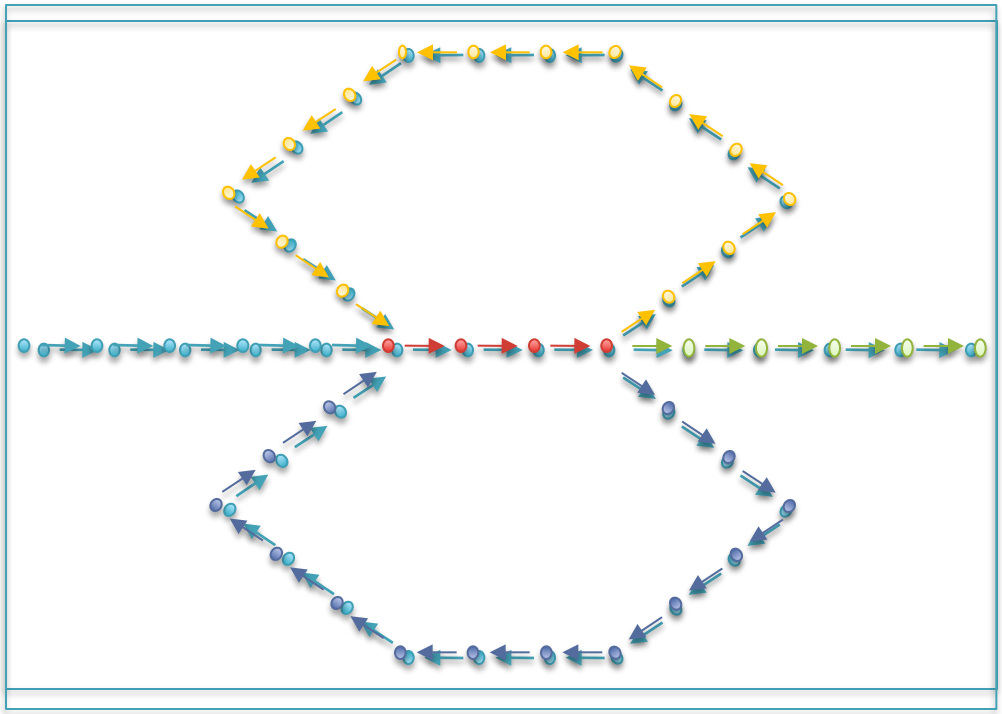
- Several new assemblers developed specifically for short read data
  - Old assemblers incompatible for technical and algorithmic reasons
  - Variations on compressed de Bruijn graphs
    - Velvet (Zerbino & Birney, 2008)
    - ALLPATHS (Butler et al, 2008)
    - EULER-USR (Chaisson et al, 2009)
    - ABySS (Simpson et al, 2009)

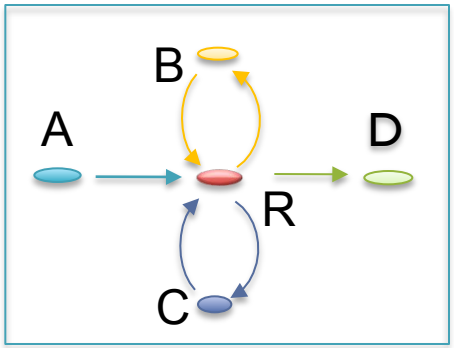
- Short Read Assembler Overview
  1. Construct compressed de Bruijn Graph
  2. Remove sequencing error from graph
  3. Use mate-pairs to resolve ambiguities in the graph

- Very successful for small to medium genomes
  - 2Mbp bacteria – 100Mbp flies

de Bruijn Graph Construction

- Map: Scan reads and emit (k_i,k_{i+1}) for consecutive k-mers
  - Also consider reverse complement k-mers, build bi-directed graph
- Reduce: Save adjacency representation of graph (n, (nodeinfo, ni))





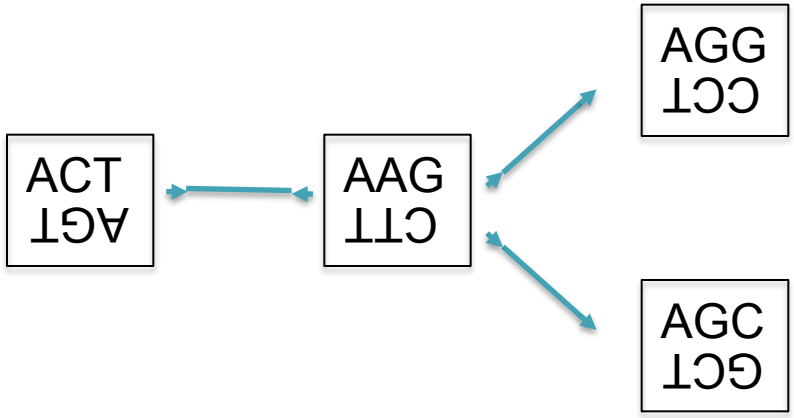
Bidirectional de Bruijn Graph

- Designate a representative mer for each mer/rc(mer) pair
  - Use the lexigraphically smaller mer

- Bidirected edges record if connection is between forward or reverse mer

- In practice, keep separate adjacency lists for the forward and reverse mers

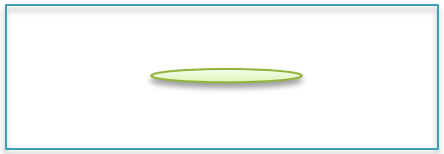
AAGG [CCTT]: AAG^{+} -> AGG^{+}
ACTT [AAGA]: ACT^{+} -> AAG^{-}
GCTT [AAGC]: AGC^{-} -> AAG^{-}
AAG^{+} -> AGC^{+}



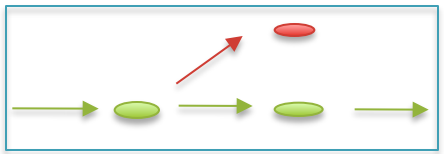
(Medvedev et al, 2007)

Node Types

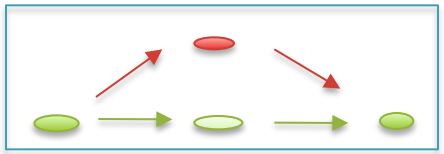
(Chaisson, 2009)



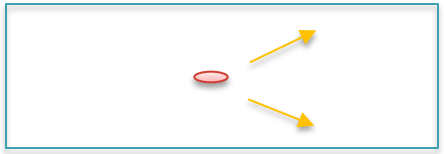
Isolated nodes (10%)



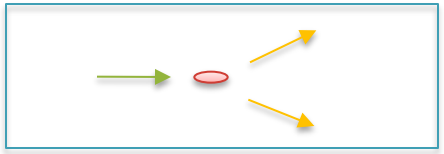
Tips (46%)



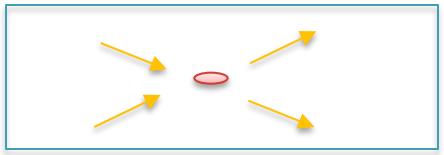
Bubbles/Non-branch (9%)



Dead Ends (.2%)



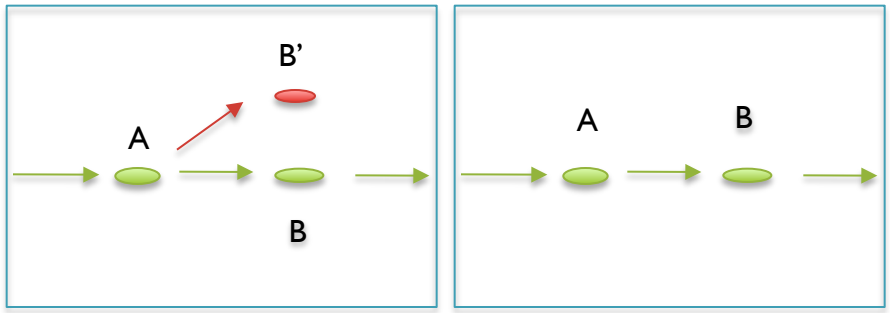
Half Branch (25%)



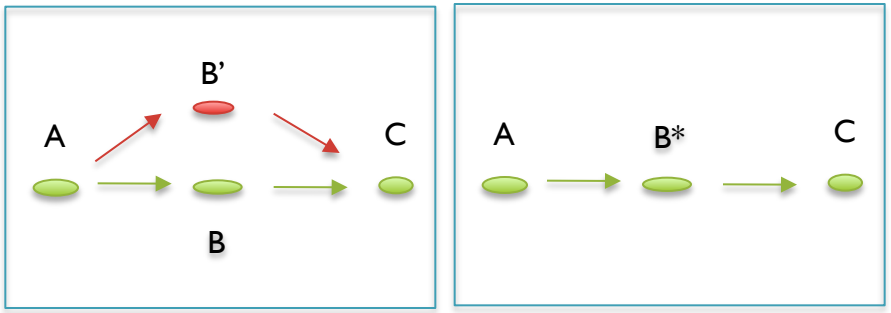
Full Branch (10%)

Error Correction

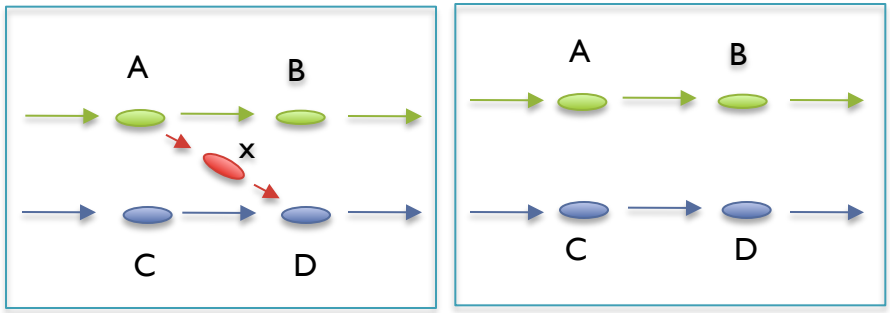
- Errors at end of read
  - Trim off 'dead-end' tips



- Errors in middle of read
  - Pop Bubbles

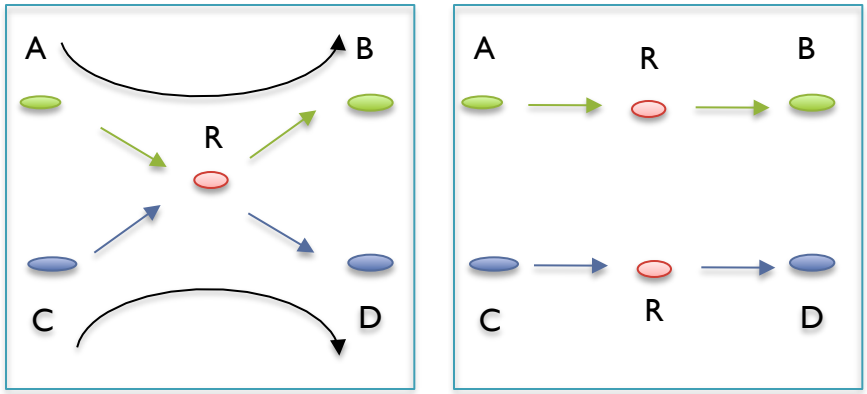


- Chimeric Edges
  - Clip short, low coverage nodes

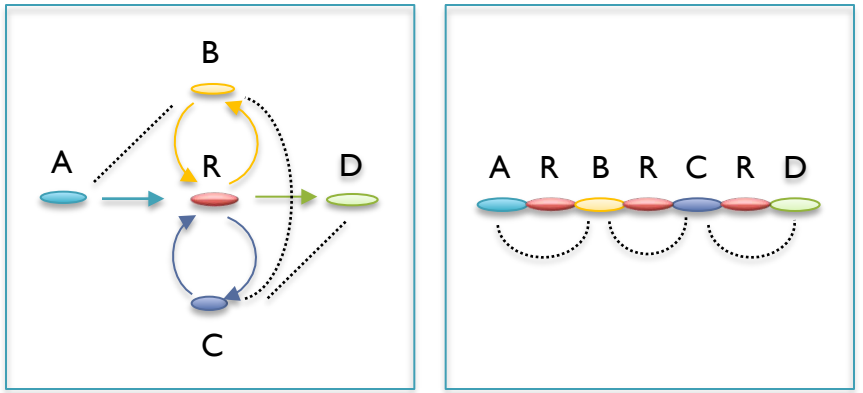


Repeat Analysis

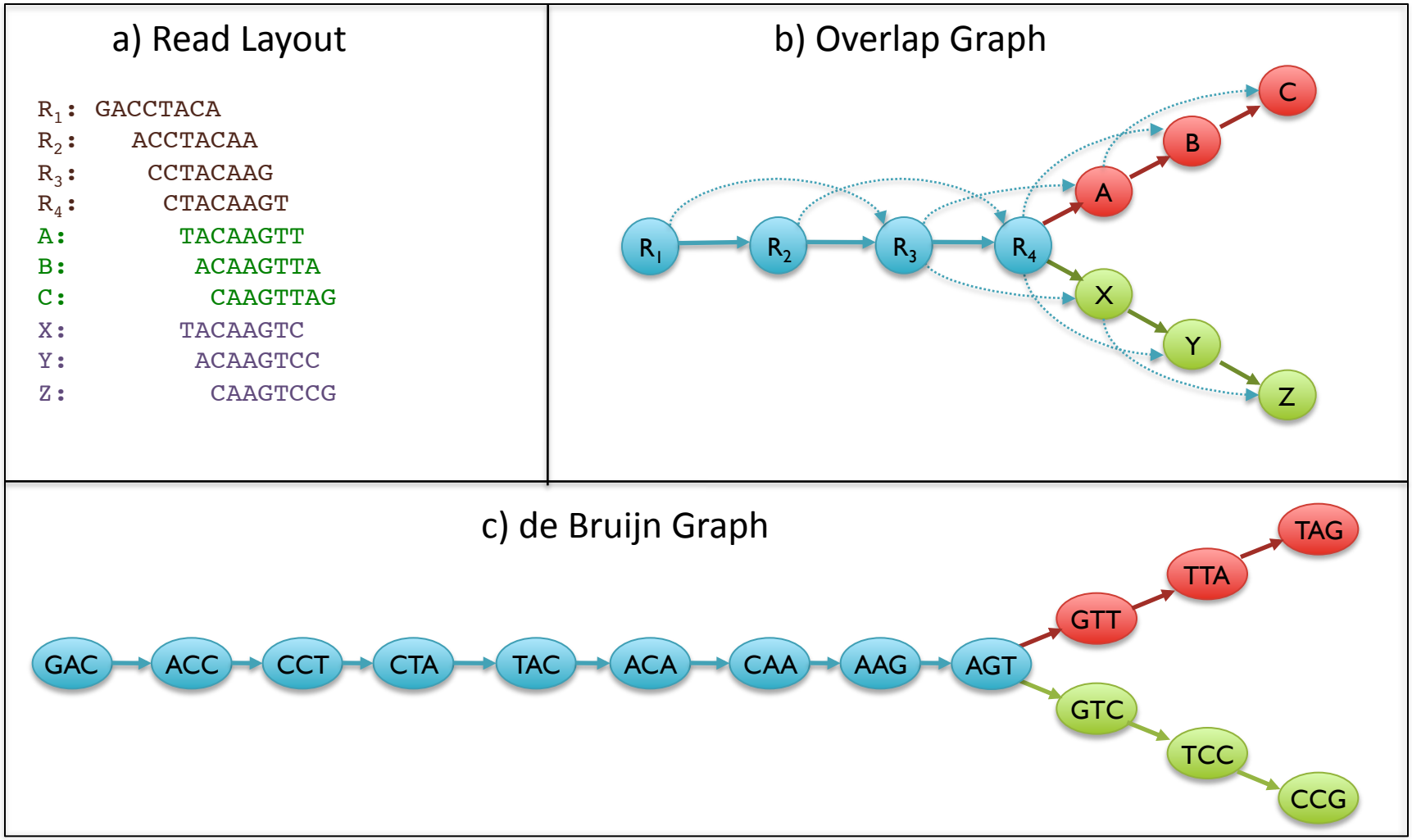
- X-cut
  - Annotate edges with spanning reads
  - Separate fully spanned nodes
    - (Pevzner *et al.*, 2001)



- Scaffolding
  - If mate pairs are available search for a path consistent with mate distance
  - Conceptually very similar to old techniques

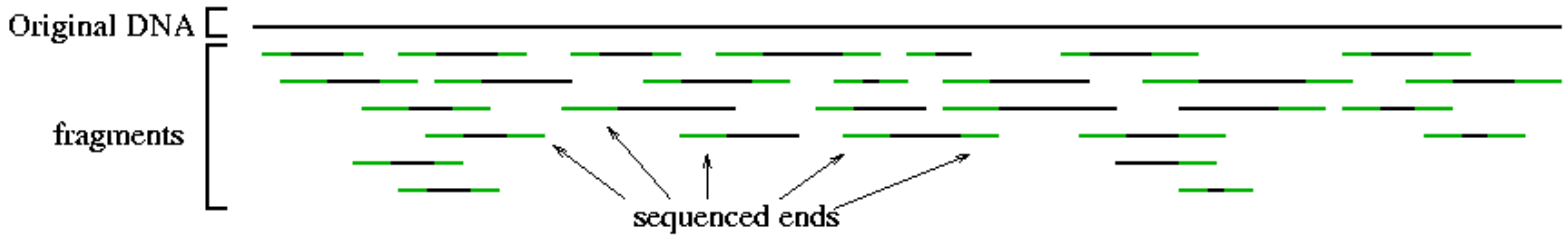


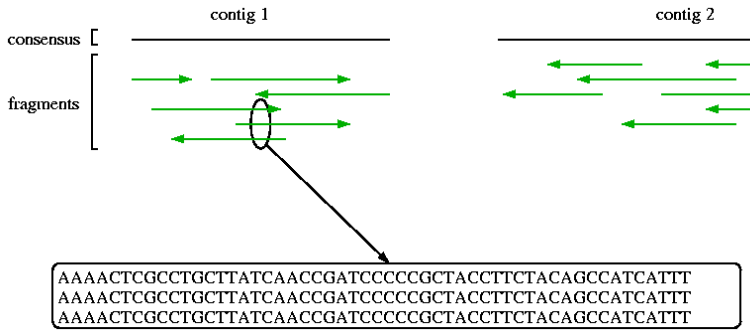
Two Paradigms for Assembly



Assembly of Large Genomes using Second Generation Sequencing
Schatz MC, Delcher AL, Salzberg SL (2010) *Genome Research* 20, 1165-73.

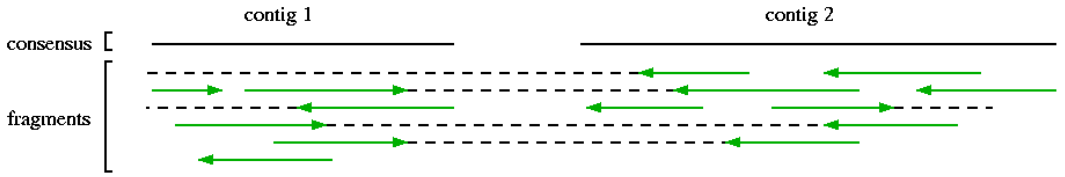
Unifying view of assembly



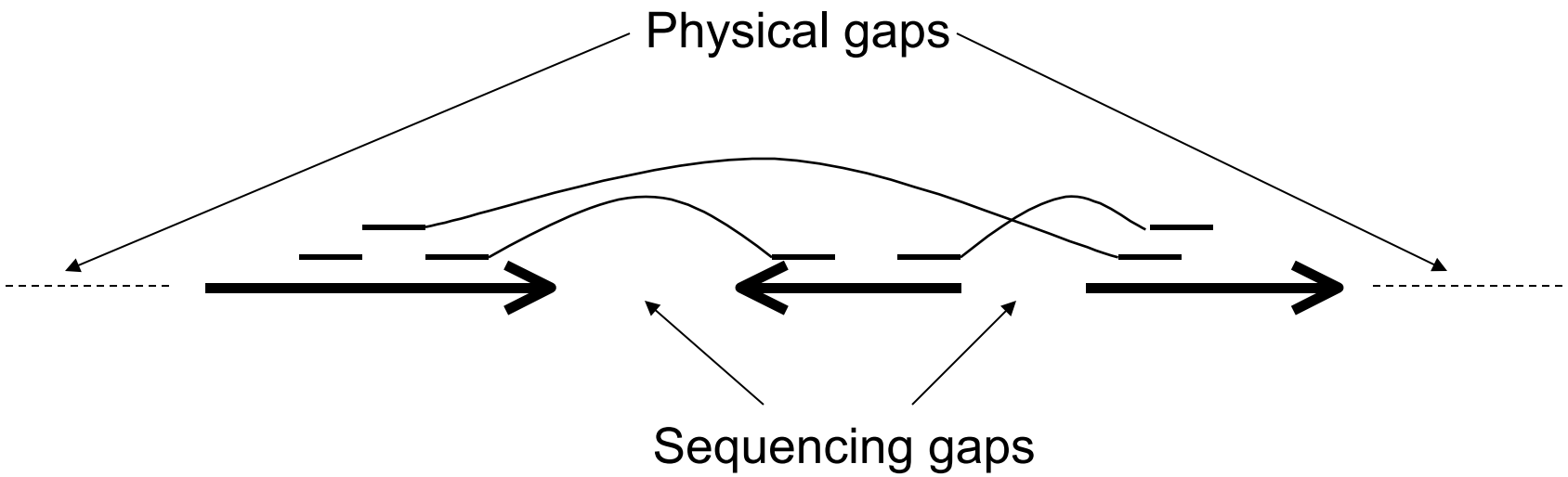


Contigs

Scaffolding



Assembly gaps



sequencing gap - we know the order and orientation of the contigs and have at least one clone spanning the gap

physical gap - no information known about the adjacent contigs, nor about the DNA spanning the gap

N50 size

Def: 50% of the genome is in contigs larger than N50

Example:

1 Mbp genome
Contigs: 300k, 100k, 50k, 45k, 30k, 20k, 15k, 15k, 10k, ...

N50 size = 30 kbp
(300k+100k+50k+45k+30k = 525k >= 500kbp)

Note:

N50 values are only meaningful to compare when base genome size is the same in all cases

Recent Large Assemblies

Table 1. *De novo assemblies of second-generation sequencing projects.*

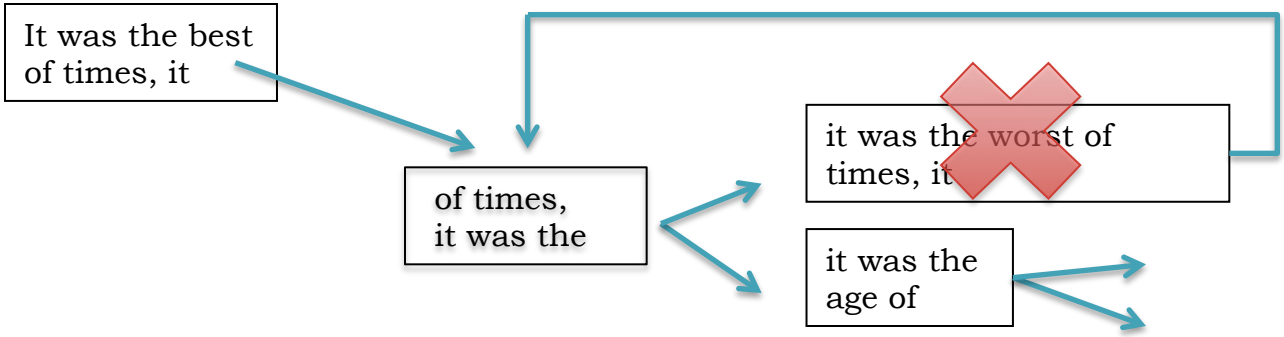
Organism/ Genome Size	Assembler/ Status	Input sequence: Type	Input sequence: Pair Size	Input sequence: Avg Read(bp)	Input sequence: # Reads	Input sequence: Read Cov	Input sequence: Pair Cov	Assembly: Contigs: #	Assembly: Contigs: N50	Assembly: Contigs: Max	Assembly: Contigs: Total	Assembly: Scaffolds: #	Assembly: Scaffolds: N50	Assembly: Scaffolds: Max	Assembly: Scaffolds: Total	Ref
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	Sanger	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
		Sanger	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		Sanger	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
[illegible]	[illegible]	Sanger	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
		Sanger	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		Sanger	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
[illegible]	[illegible]	454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
		454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SOLiD	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
[illegible]	[illegible]	454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
		454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		454	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									
		SA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]									

Assembly Validation

Automatically scan an assembly to locate misassembly signatures for further analysis and correction

Assembly-validation pipeline

1. Evaluate Mate Pairs & Libraries
2. Evaluate Read Alignments
3. Evaluate Read Breakpoints
4. Analyze Depth of Coverage



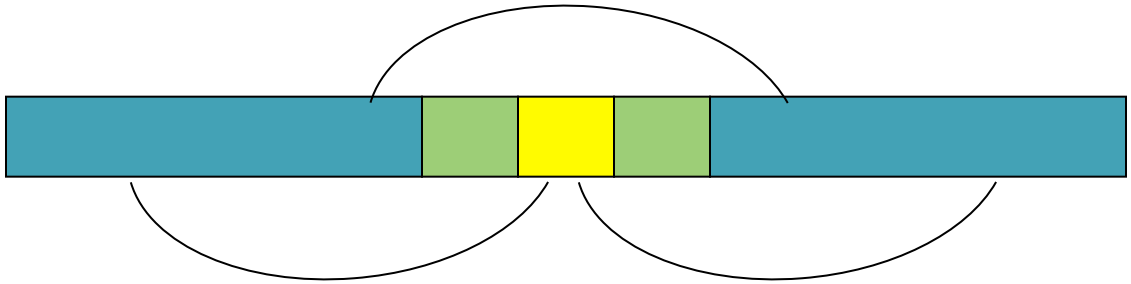
Genome Assembly forensics: finding the elusive mis-assembly.
Phillippy, AM, Schatz, MC, Pop, M. (2008) *Genome Biology* 9:R55.

Mate-Happiness: asmQC

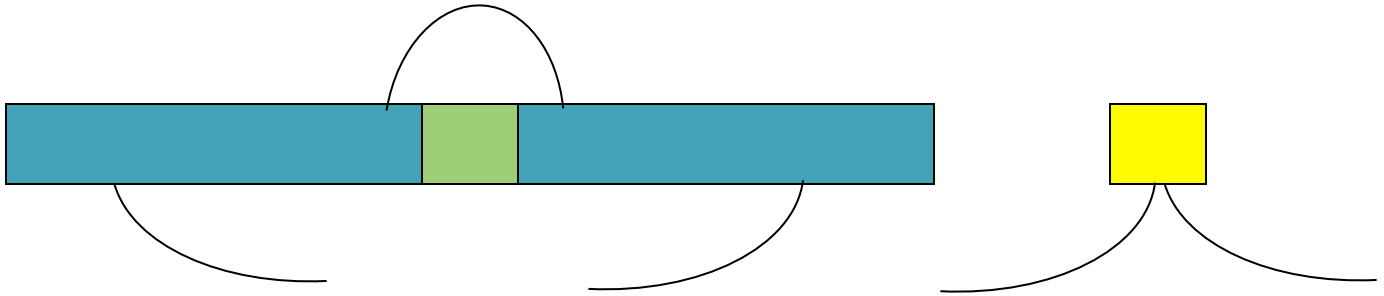
- Evaluate mate "happiness" across assembly
  - Happy = Correct orientation and distance
- Finds regions with multiple:
  - Compressed Mates
  - Expanded Mates
  - Invalid same orientation (→ →)
  - Invalid outie orientation (← →)
  - Missing Mates
    - Linking mates (mate in a different scaffold)
    - Singleton mates (mate is not in any contig)
- Regions with high C/E statistic

Mate-Happiness: asmQC

- Excision: Skip reads between flanking repeats
  - Truth

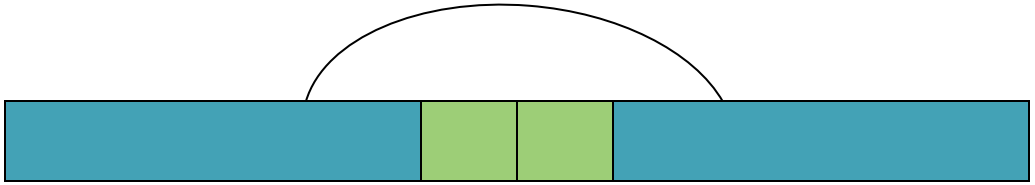


  - Misassembly: Compressed Mates, Missing Mates

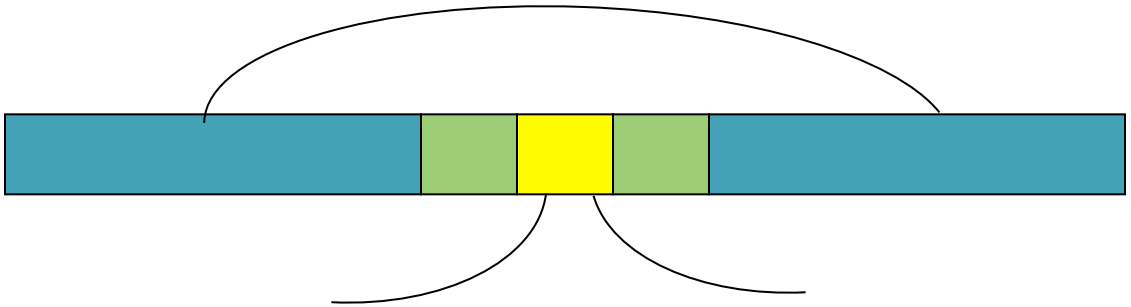


Mate-Happiness: asmQC

- Insertion: Additional reads between flanking repeats
  - Truth

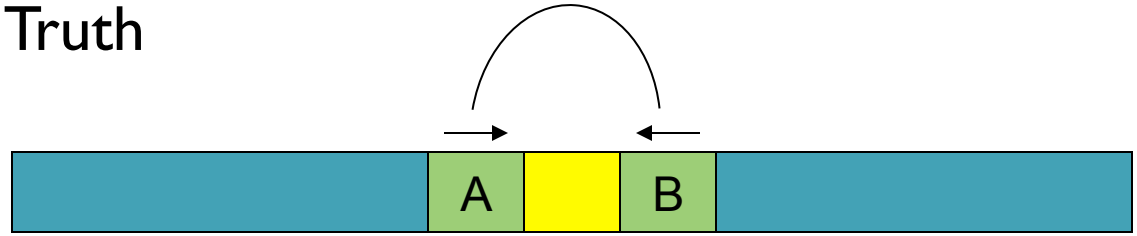


  - Misassembly: Expanded Mates, Missing Mates

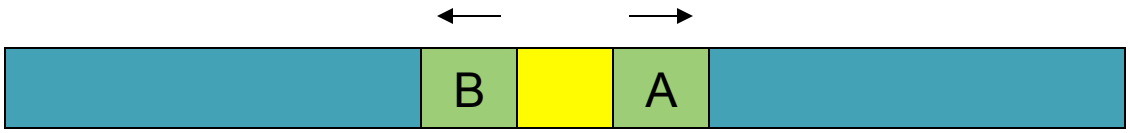


Mate-Happiness: asmQC

- Rearrangement: Reordering of reads
  - Truth



  - Misassembly: Misoriented Mates



Note: Unhappy mates may also occur for biological or technical reasons.

C/E Statistic

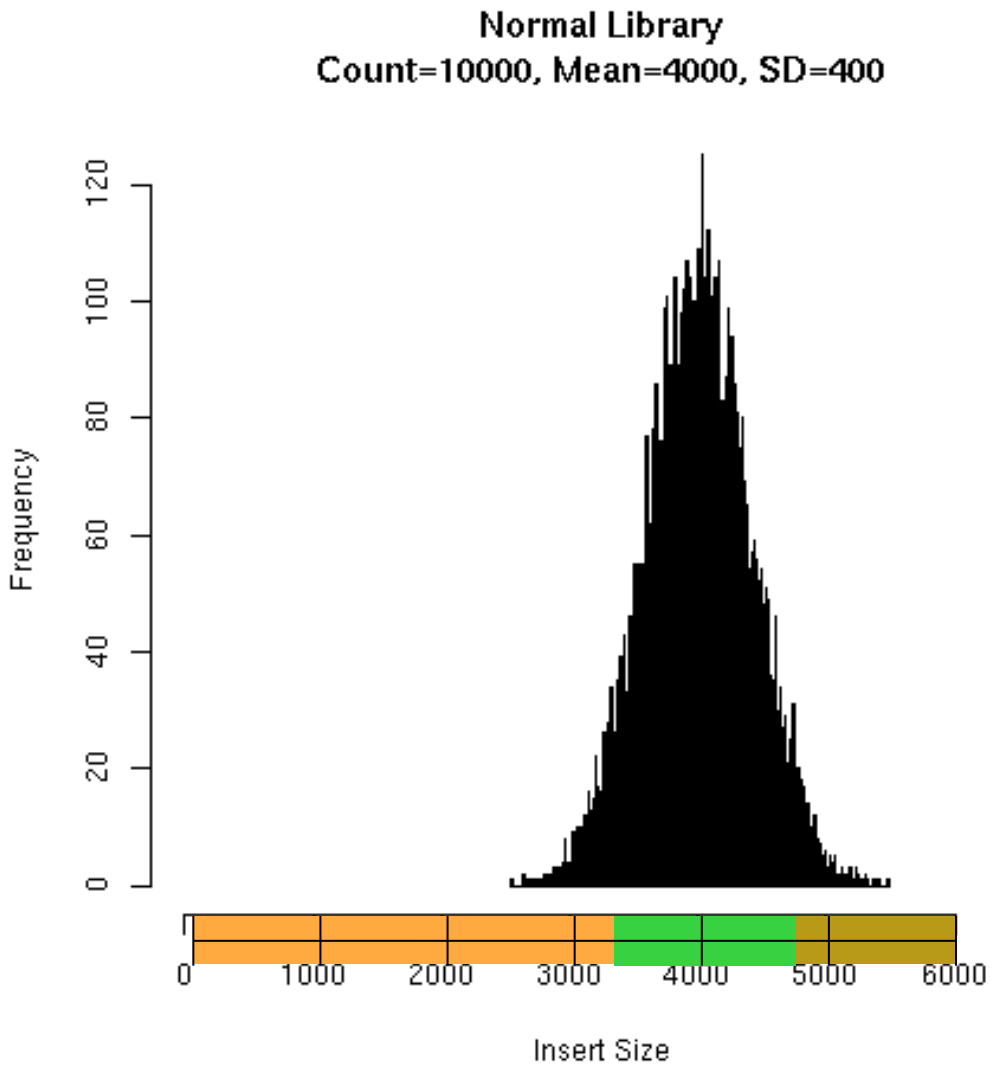
- The presence of individual compressed or expanded mates is rare but expected.

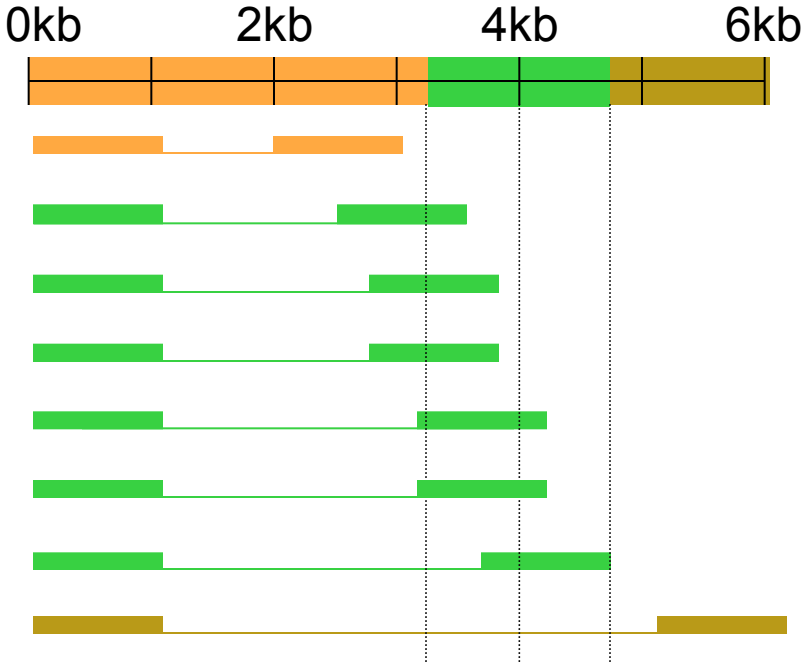
- Do the inserts spanning a given position differ from the rest of the library?
  - Flag large differences as potential misassemblies
  - Even if each individual mate is "happy"

- Compute the statistic at all positions
  - (Local Mean – Global Mean) / Scaling Factor

- Introduced by Jim Yorke's group at UMD

Sampling the Genome





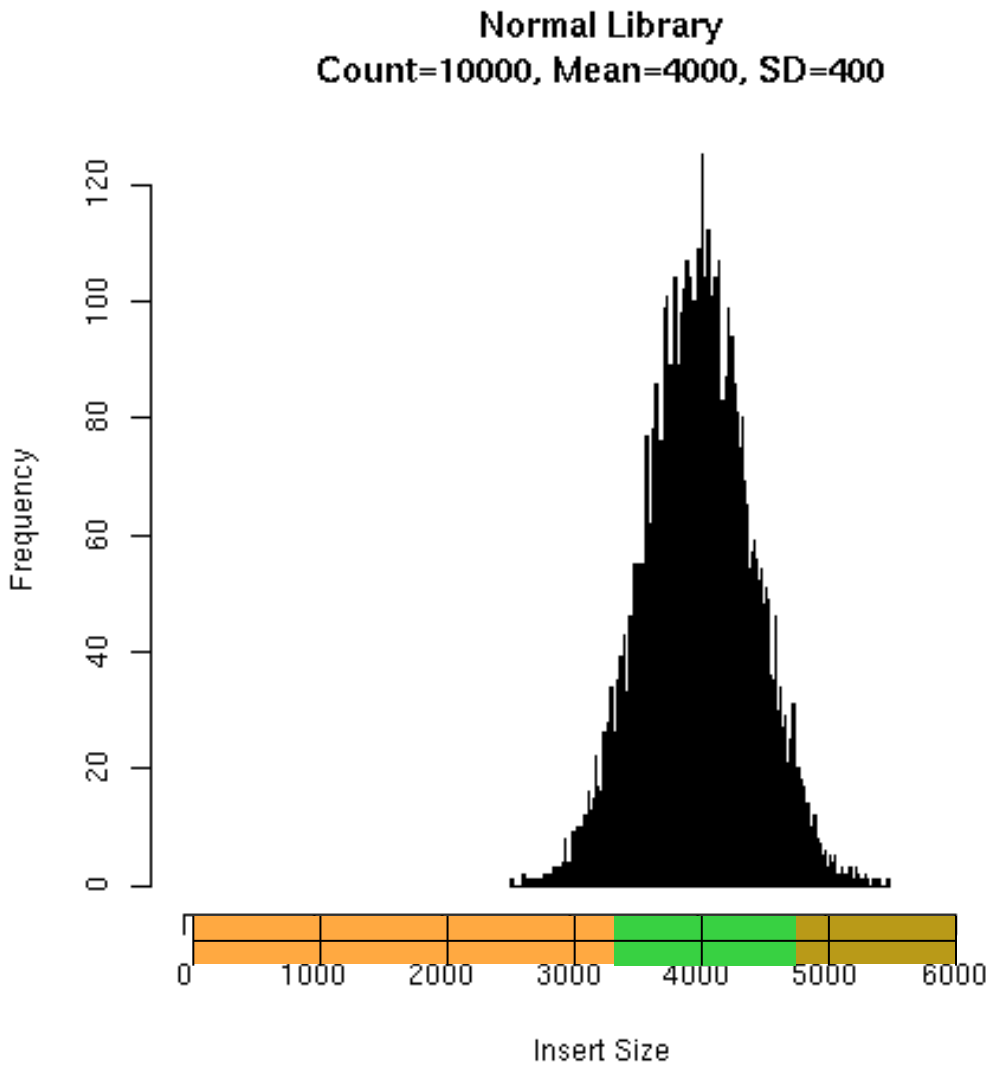
8 inserts: 3kb-6kb

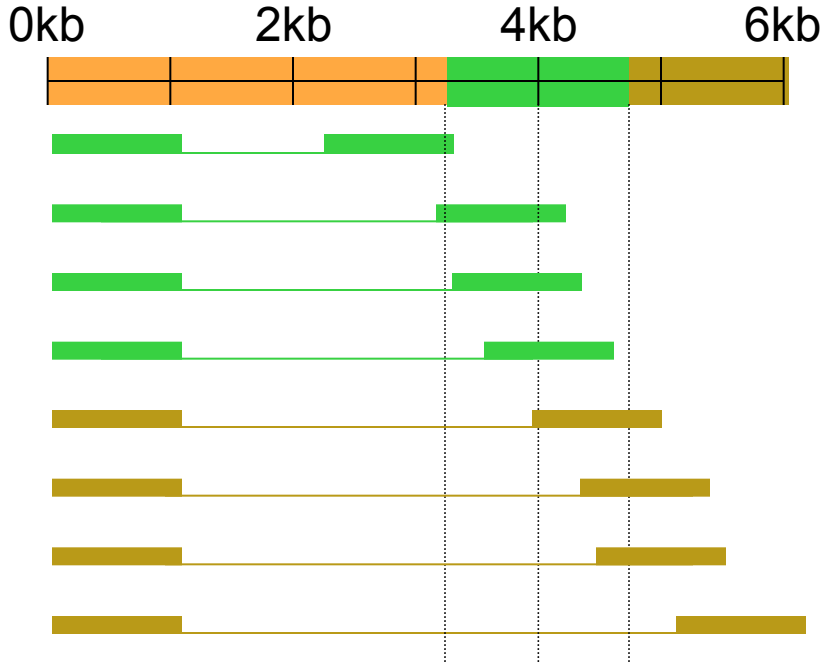
Local Mean: 4048

C/E Stat: $\frac{(4048-4000)}{(400 / \sqrt{8})}$ = +0.33

Near 0 indicates overall happiness

C/E-Statistic: Expansion





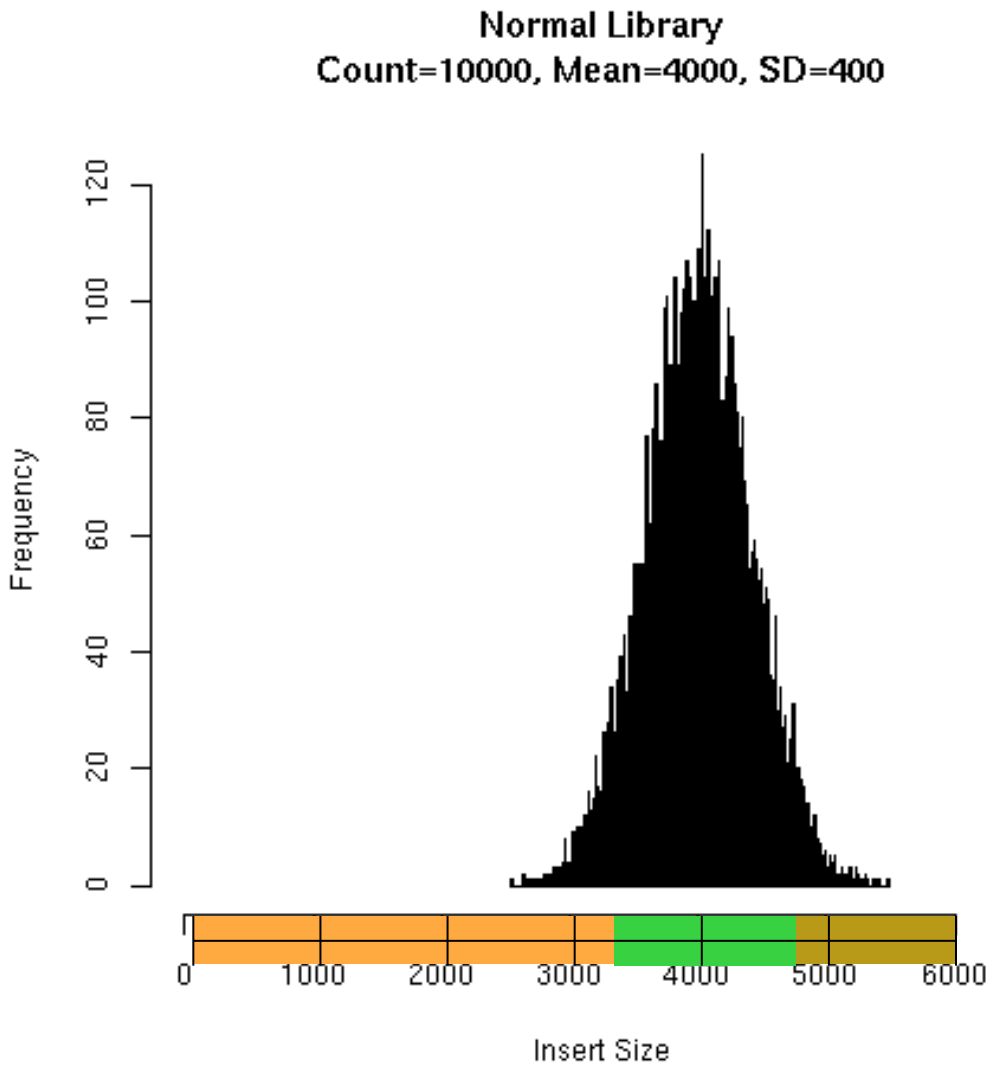
8 inserts: 3.2kb-6kb

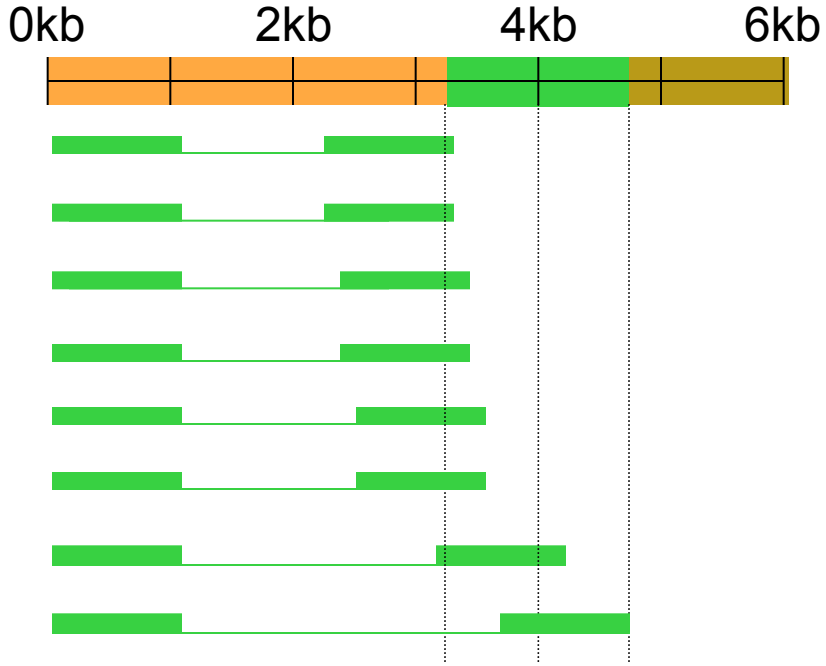
Local Mean: 4461

C/E Stat: $\frac{(4461-4000)}{(400 / \sqrt{8})}$ = +3.26

C/E Stat ≥ 3.0 indicates Expansion

C/E-Statistic: Compression





8 inserts: 3.2 kb-4.8kb

Local Mean: 3488

C/E Stat: $\frac{(3488-4000)}{(400 / \sqrt{8})} = -3.62$

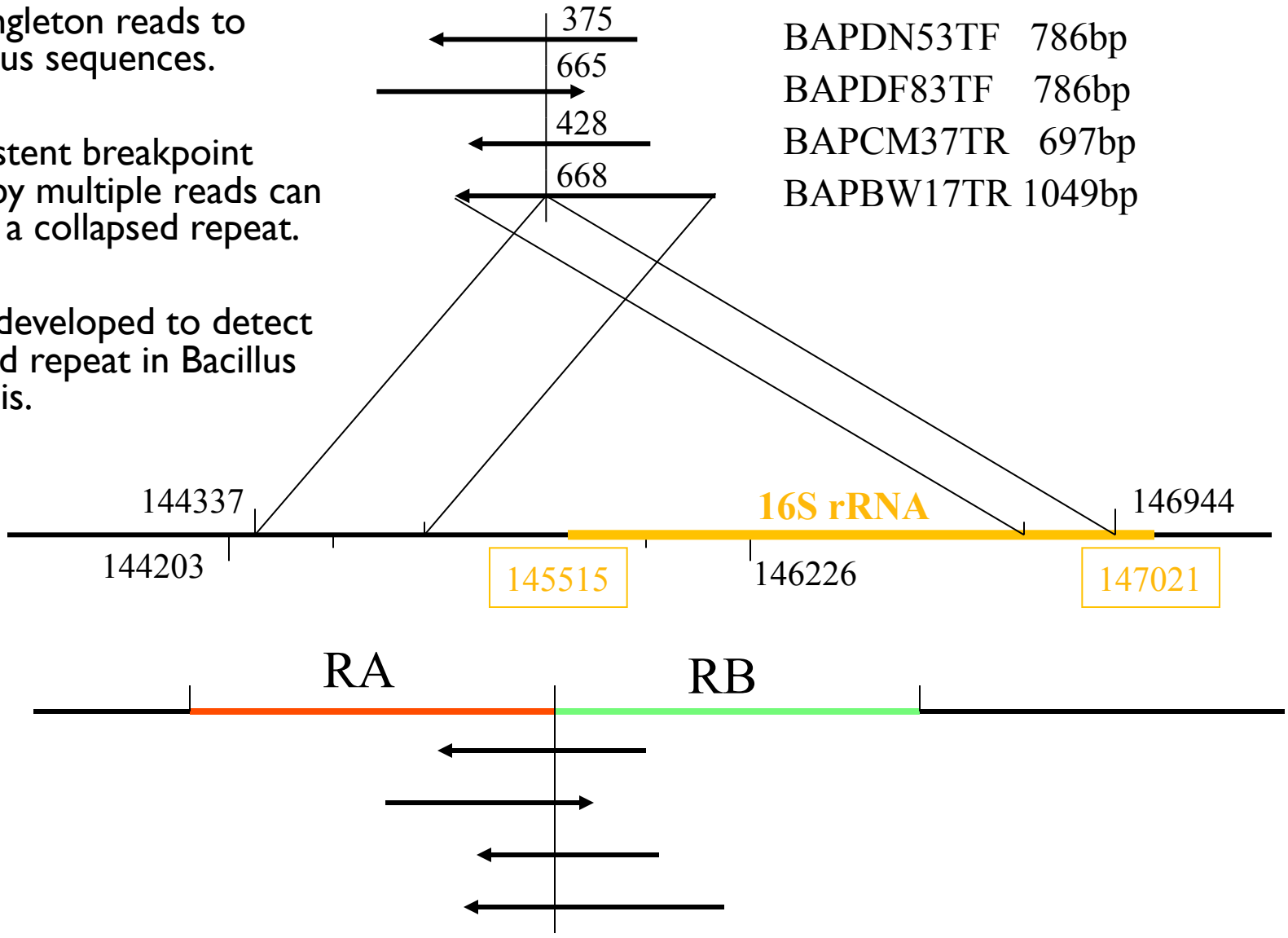
C/E Stat ≤ -3.0 indicates Compression

Read Alignment

- Multiple reads with same conflicting base are unlikely
  - 1x QV 30: 1/1000 base calling error
  - 2x QV 30: 1/1,000,000 base calling error
  - 3x QV 30: 1/1,000,000,000 base calling error
- Regions of correlated SNPs are likely to be assembly errors or interesting biological events
  - Highly specific metric
- AMOS Tools: analyzeSNPs & clusterSNPs
  - Locate regions with high rate of correlated SNPs
  - Parameterized thresholds:
    - Multiple positions within 100bp sliding window
    - 2+ conflicting reads
    - Cumulative QV >= 40 (1/10000 base calling error)

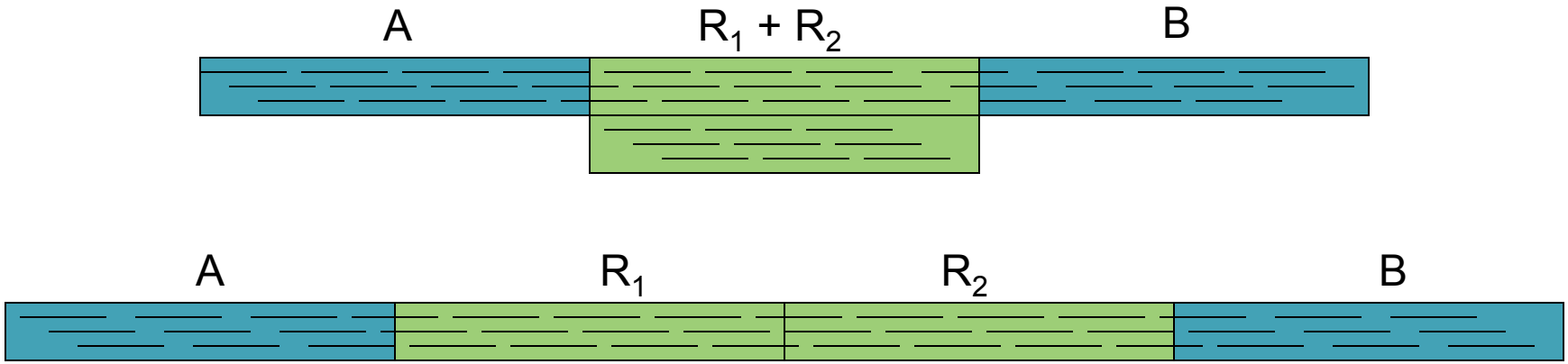
Read Breakpoints

- Align singleton reads to consensus sequences.
- A consistent breakpoint shared by multiple reads can indicate a collapsed repeat.
- Initially developed to detect collapsed repeat in Bacillus Anthracis.

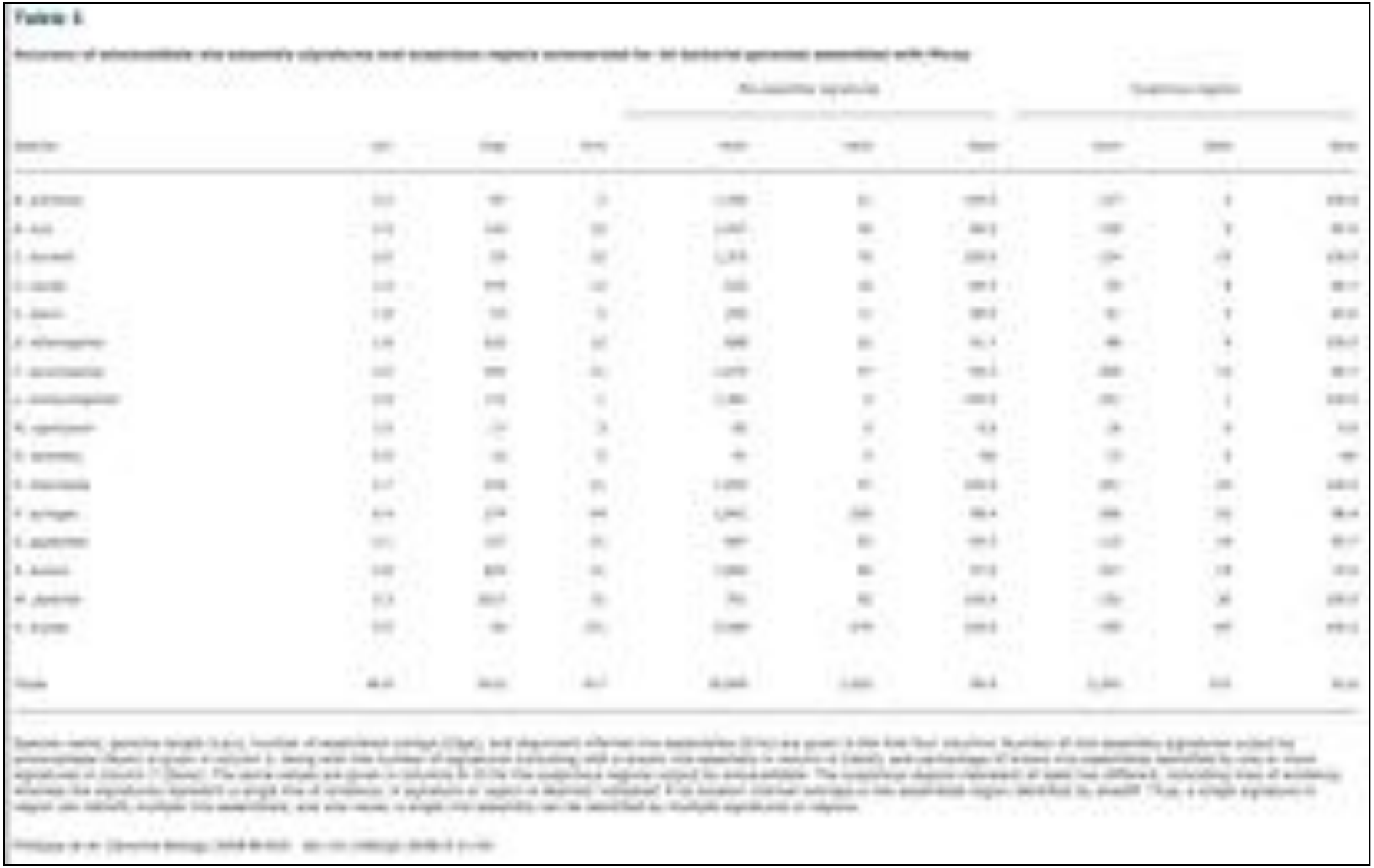


Read Coverage

- Find regions of contigs where the depth of coverage is unusually high
- Collapsed Repeat Signature
  - Can detect collapse of 100% identical repeats
- AMOS Tool: analyzeReadDepth
  - 2.5x mean coverage



Validation Accuracy



Summary

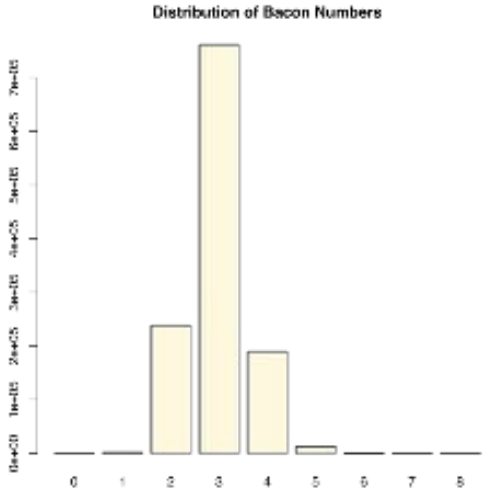
- Graphs are ubiquitous in the world
  - Pairwise searching is easy, finding features is hard

- Assembly is challenging because of repeats
  - The repetitive content depends on the read length
    => Shorter reads are harder to assemble
  - Assembly is a hierarchical, starting from individual reads, build high confidence contigs/unitigs, incorporate the mates to build scaffolds
  - Watch out for collapsed repeats & other misassemblies
    - Globally/Locally reassemble data from scratch with better parameters & stitch the 2 assemblies together

Supplemental

IMDB Movie Graph

- Bipartite Graph
  - 1.5 M people
  - 1.2 M shows
- Small world graph
  - KB has 2350 direct collaborators
  - 1.2 M within 8 hops
  - 83% within 3 hops

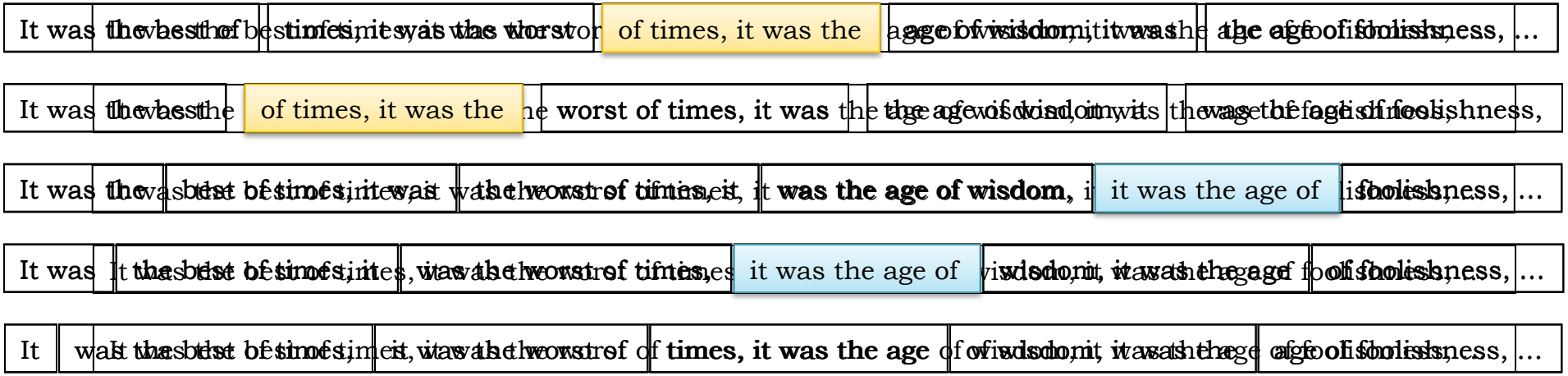


Average Bacon Number: 2.981

Oracle of Bacon
http://oracleofbacon.org

Shredded Book Reconstruction

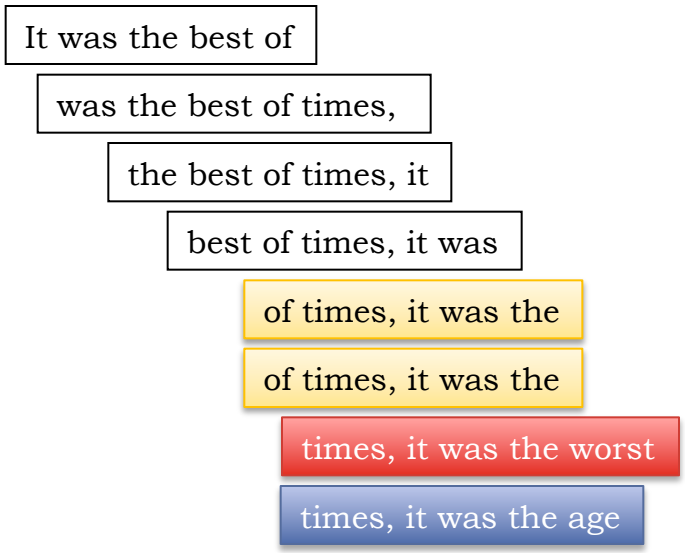
- Dickens accidentally shreds the first printing of A Tale of Two Cities
  - Text printed on 5 long spools



- How can he reconstruct the text?
  - 5 copies x 138, 656 words / 5 words per fragment = 138k fragments
  - The short fragments from every copy are mixed together
  - Some fragments are identical

Greedy Reconstruction

It was the best of
age of wisdom, it was
best of times, it was
it was the age of
it was the age of
it was the worst of
of times, it was the
of times, it was the
of wisdom, it was the
the age of wisdom, it
the best of times, it
the worst of times, it
times, it was the age
times, it was the worst
was the age of wisdom,
was the age of foolishness,
was the best of times,
was the worst of times,
wisdom, it was the age
worst of times, it was



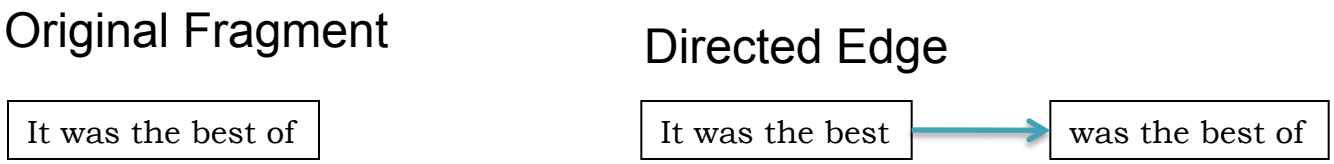
The repeated sequence make the correct reconstruction ambiguous

- It was the best of times, it was the [worst/age]

Model sequence reconstruction as a graph problem.

de Bruijn Graph Construction

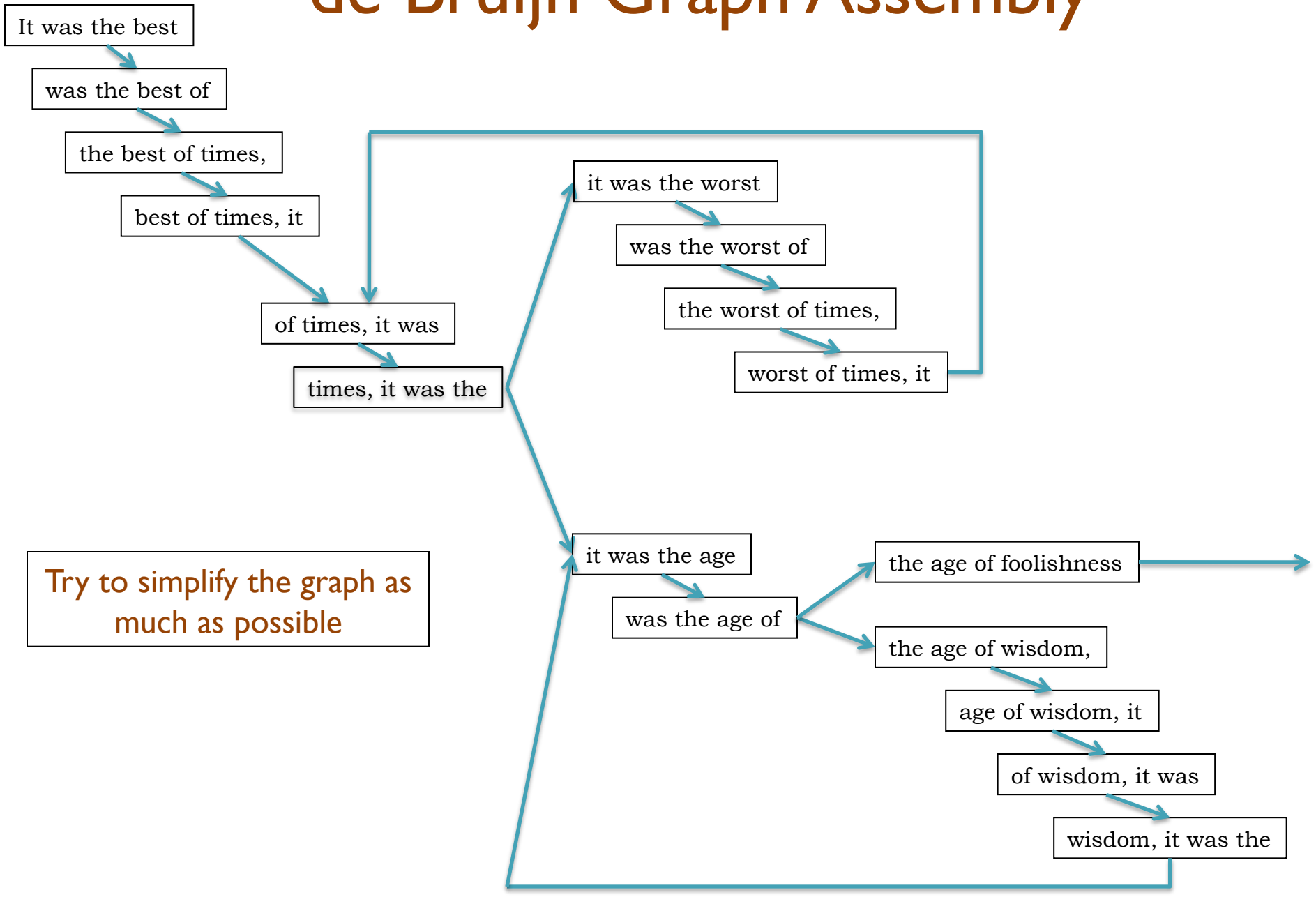
- $D_k = (V,E)$
  - V = All length-k subfragments (k < l)
  - E = Directed edges between consecutive subfragments
    - Nodes overlap by k-1 words



- Locally constructed graph reveals the global sequence structure
  - Overlaps between sequences implicitly computed

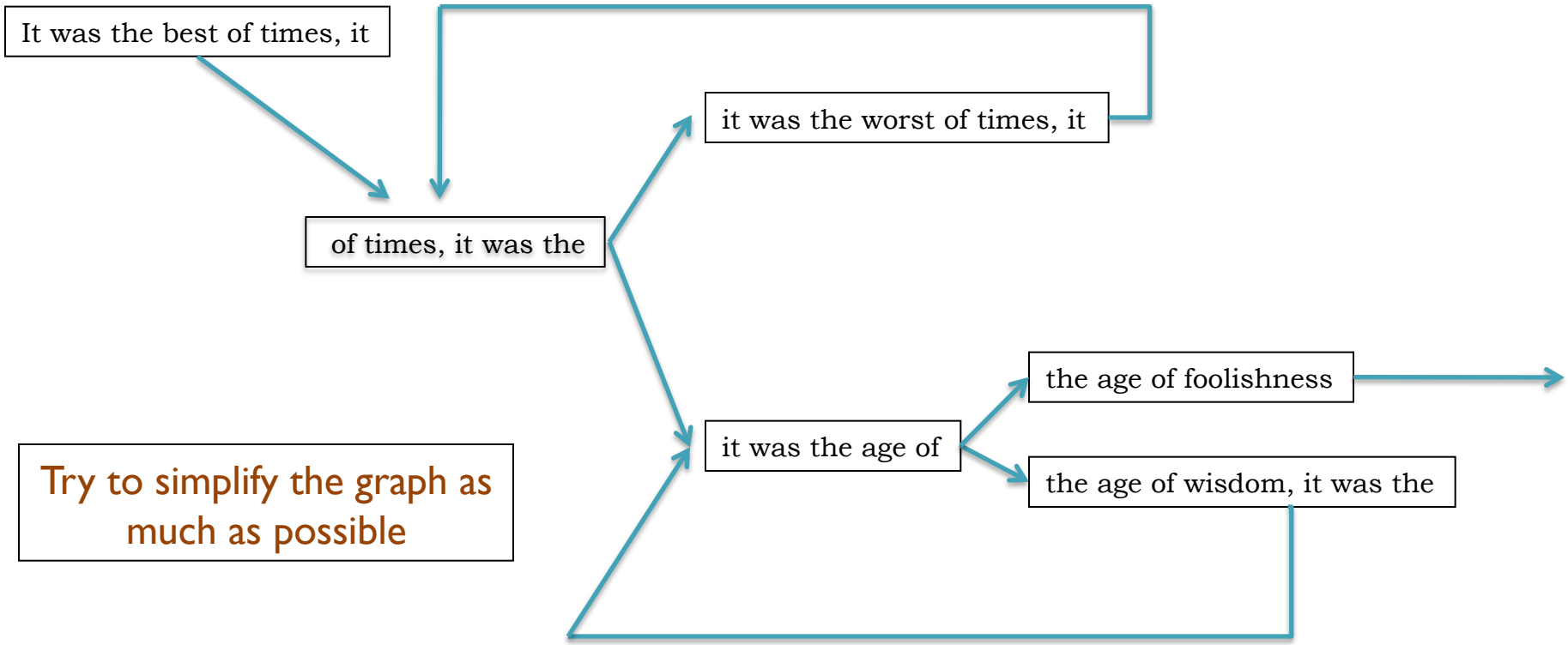
de Bruijn, 1946
Idury and Waterman, 1995
Pevzner, Tang, Waterman, 2001

de Bruijn Graph Assembly



Try to simplify the graph as much as possible

de Bruijn Graph Assembly



Try to simplify the graph as much as possible