Alignment & Assembly

Michael Schatz

Bioinformatics Lecture 3
Quantitative Biology 2011

Exact Matching Review

Where is GATTACA in the human genome?
E=183,105

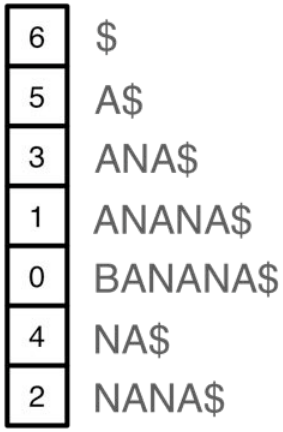
Brute Force (3 GB)

```
BANANA
BAN
  ANA
    NAN
      ANA
```

Naive

Slow & Easy

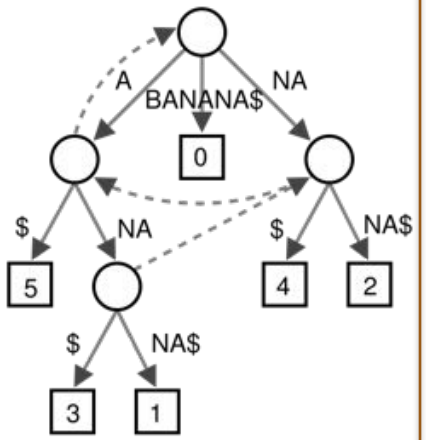
Suffix Array (>15 GB)



Vmatch, PacBio Aligner

Binary Search

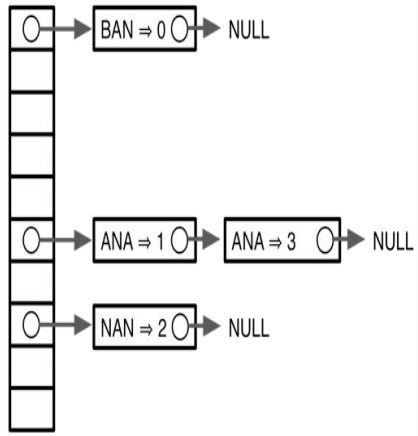
Suffix Tree (>51 GB)



MUMmer, MUMmerGPU

Tree Searching

Hash Table (>15 GB)



BLAST, MAQ, ZOOM, RMAP, CloudBurst

Seed-and-extend

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

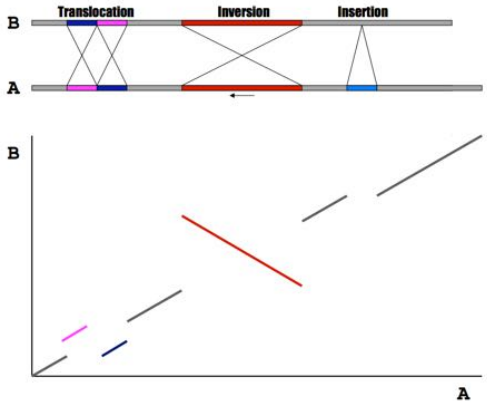
BLAST

Very Similar Sequences

```
Query: HBA_HUMAN Hemoglobin alpha subunit
Sbjct: HBB_HUMAN Hemoglobin beta subunit

Score =  114 bits (285),  Expect = 1e-26
Identities = 61/145 (42%), Positives = 86/145 (59%), Gaps = 8/145 (5%)

Query  2   LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF------DLSHGSAQV 55
           L+P +K+ V A WGKV  +  E G EAL R+ + +P T+ +F  F      D   G+ +V
Sbjct  3   LTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKV 60

Query  56  KGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPA 115
           K HGKKV  A ++ +AH+D++    + LS+LH  KL VDP NF+LL + L+  LA H
Sbjct  61  KAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGK 120

Query  116 EFTPAVHASLDKFLASVSTVLTSKY 140
           EFTP V A+  K +A V+  L  KY
Sbjct  121 EFTPPVQAAYQKVVAGVANALAHKY 145
```

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

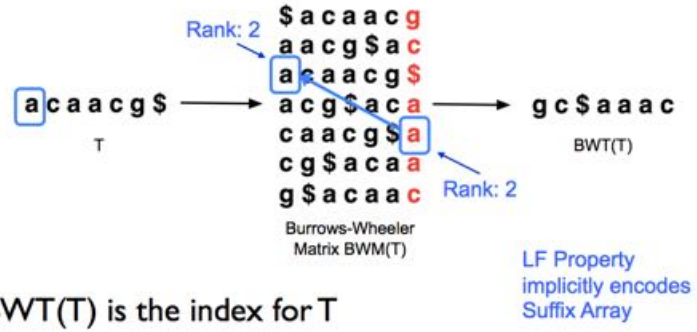
MUMmer



Whole Genome Alignment w/ Suffix Tree

Bowtie

- Reversible permutation of the characters in a text



- BWT(T) is the index for T

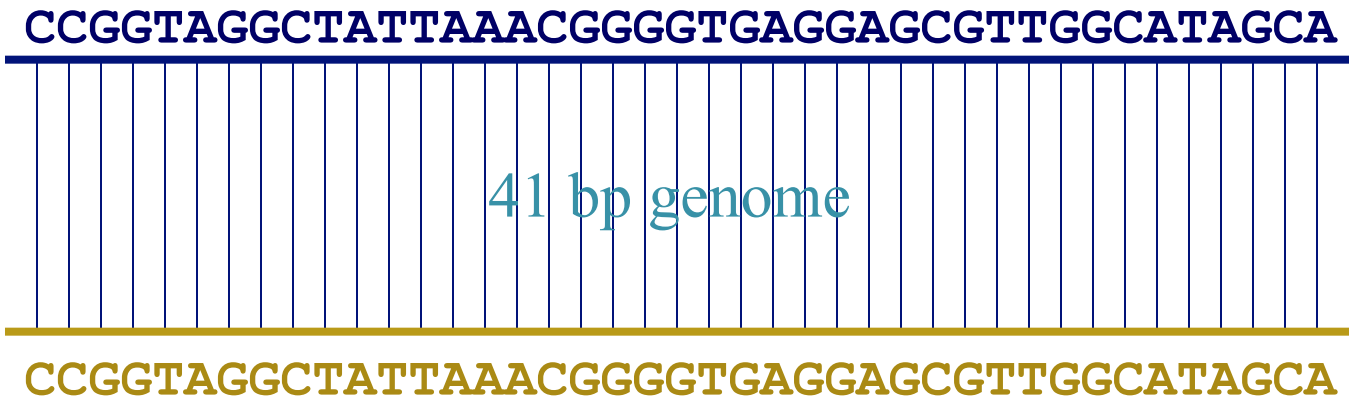
Fast searching for short read mapping

Whole Genome Alignment with MUMmer

Slides Courtesy of Adam M. Phillippy
amp@umics.umd.edu

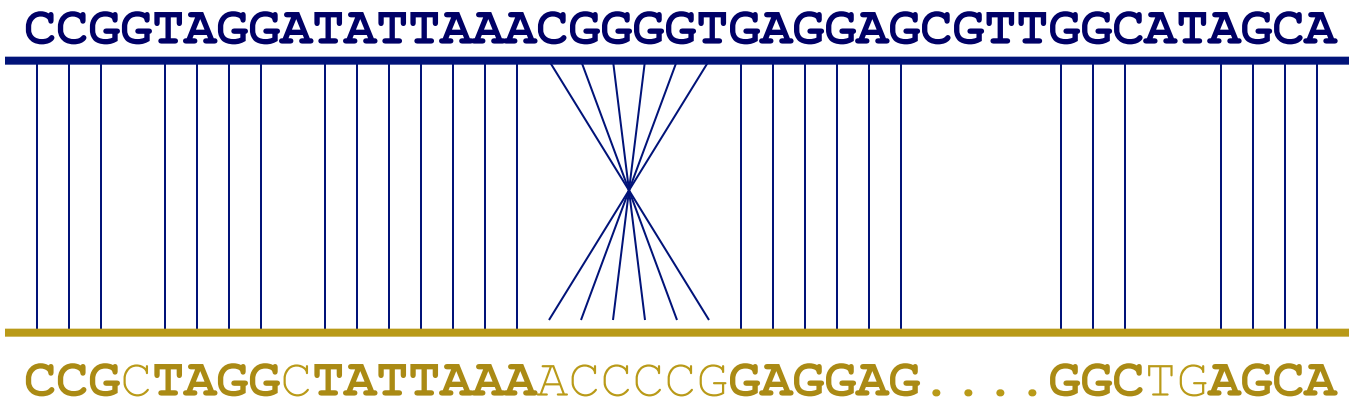
Goal of WGA

- For two genomes, *A* and *B*, find a mapping from each position in *A* to its corresponding position in *B*



Not so fast...

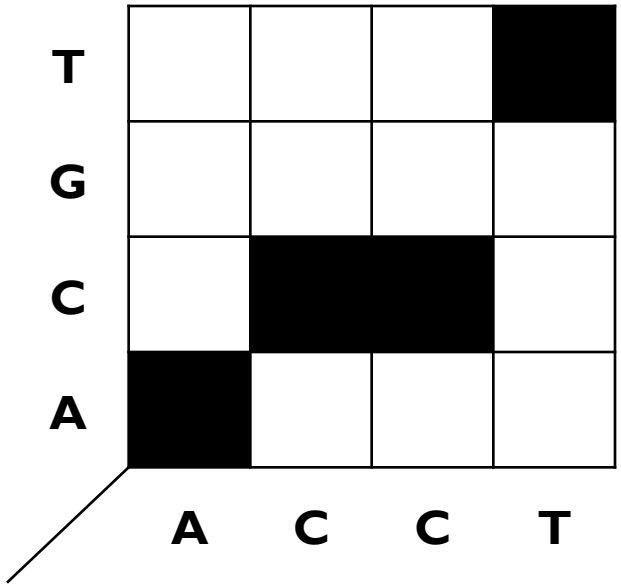
- Genome *A* may have insertions, deletions, translocations, inversions, duplications or SNPs with respect to *B* (sometimes all of the above)



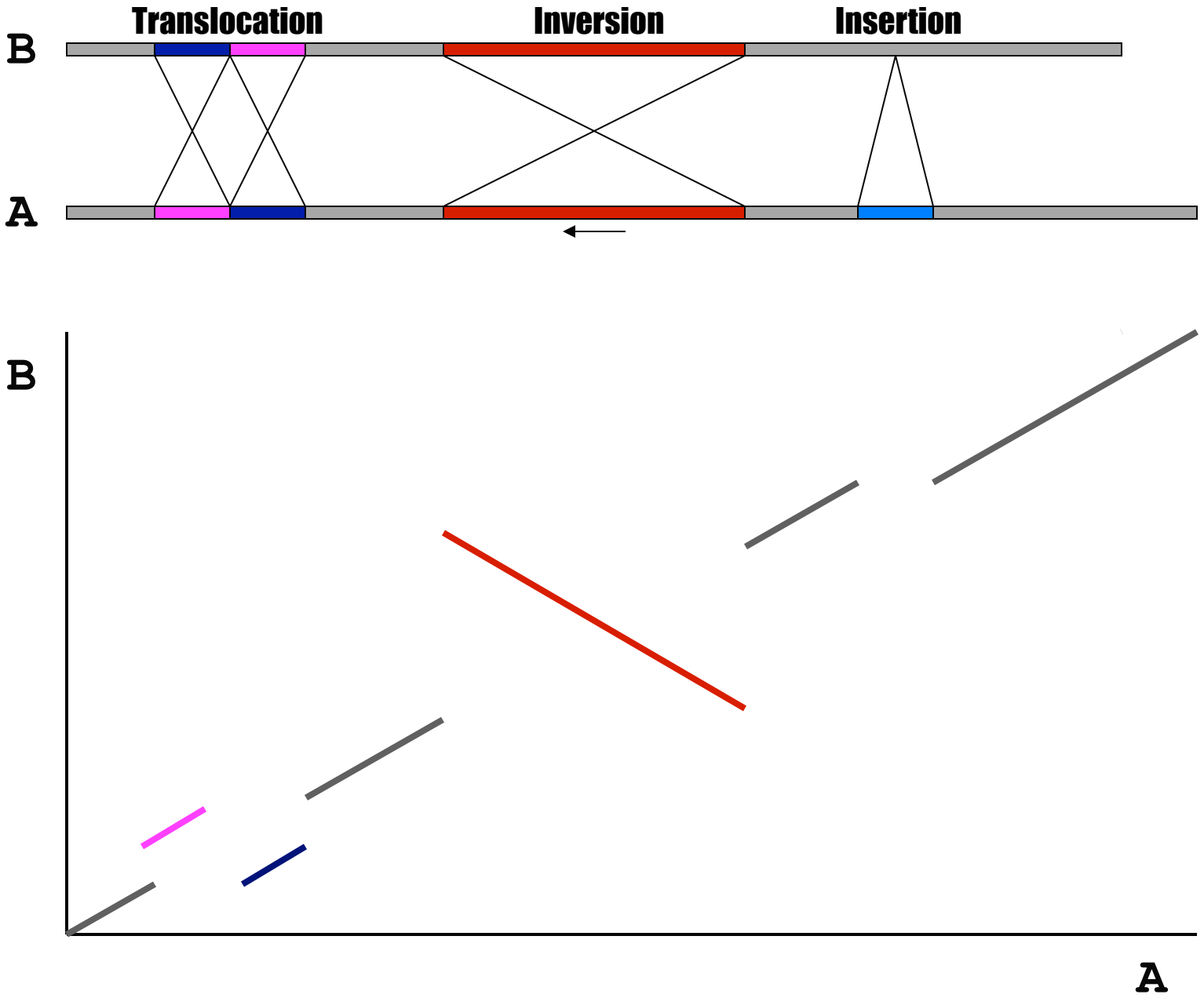
WGA visualization

- How can we visualize *whole* genome alignments?

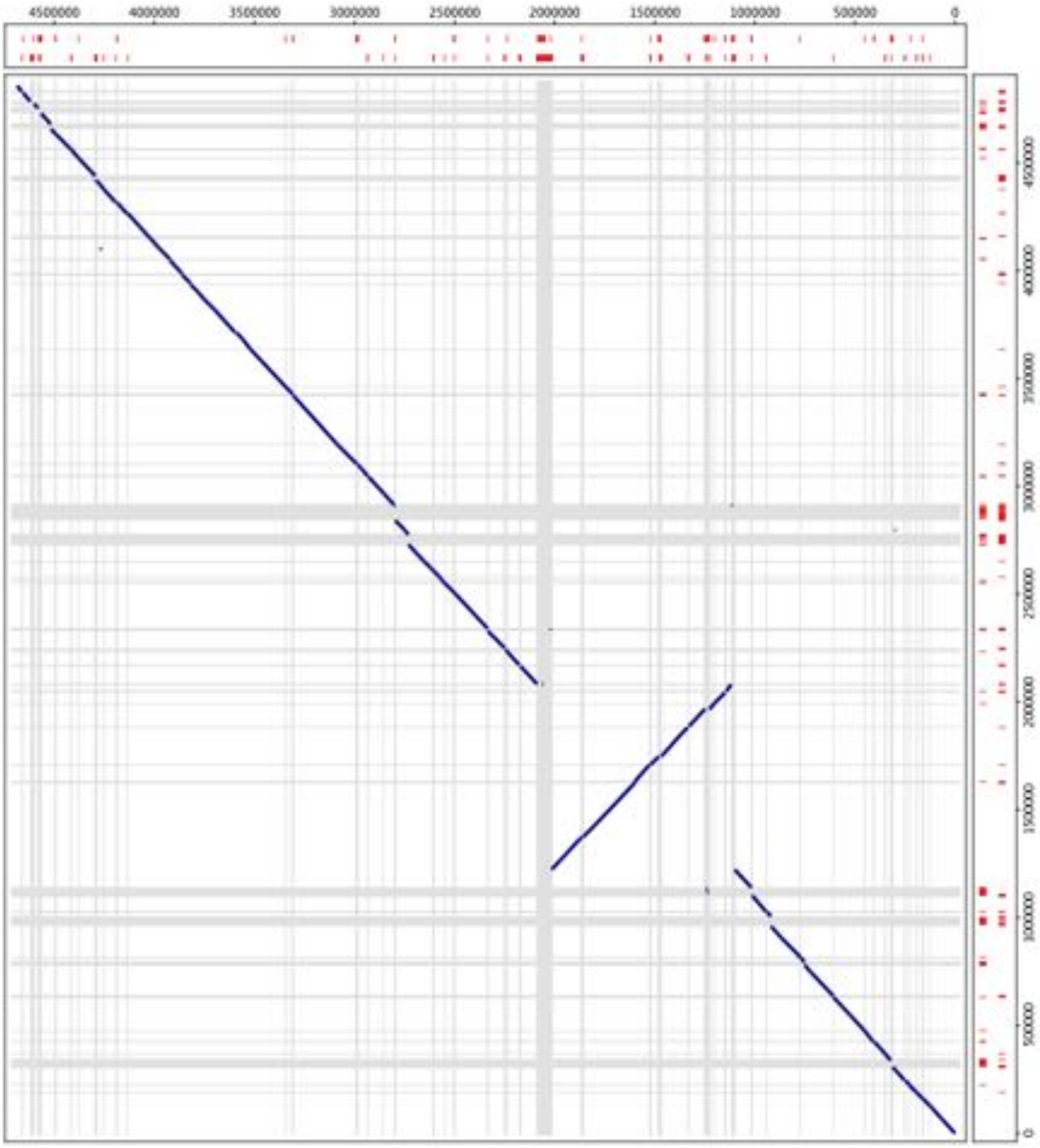
- With an alignment dot plot
  - *N* x *M* matrix
    - Let *i* = position in genome *A*
    - Let *j* = position in genome *B*
    - Fill cell *(i,j)* if A_i shows similarity to B_j



  - A perfect alignment between *A* and *B* would completely fill the positive diagonal



http://mummer.sourceforge.net/manual/AlignmentTypes.pdf


4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0

MUMmer

- Maximal Unique Matcher (MUM)
  - match
    - exact match of a minimum length
  - maximal
    - cannot be extended in either direction without a mismatch
  - *unique*
    - occurs only once in both sequences (MUM)
    - occurs only once in a single sequence (MAM)
    - occurs one or more times in either sequence (MEM)

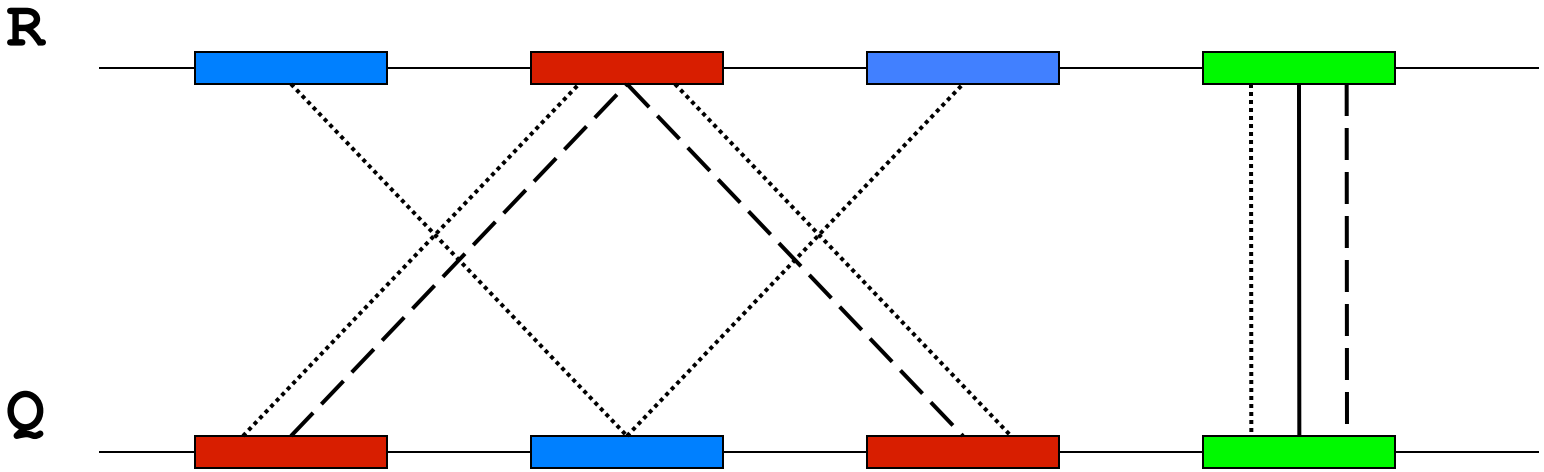
Fee Fi Fo Fum,

is it a MAM, MEM or MUM?

MUM : maximal unique match

MAM : maximal almost-unique match

MEM : maximal exact match



Seed and Extend

- How can we make MUMs **BIGGER?**

1. Find MUMs
   - using a suffix tree
2. Cluster MUMs
   - using size, gap and distance parameters
3. Extend clusters
   - using modified Smith-Waterman algorithm

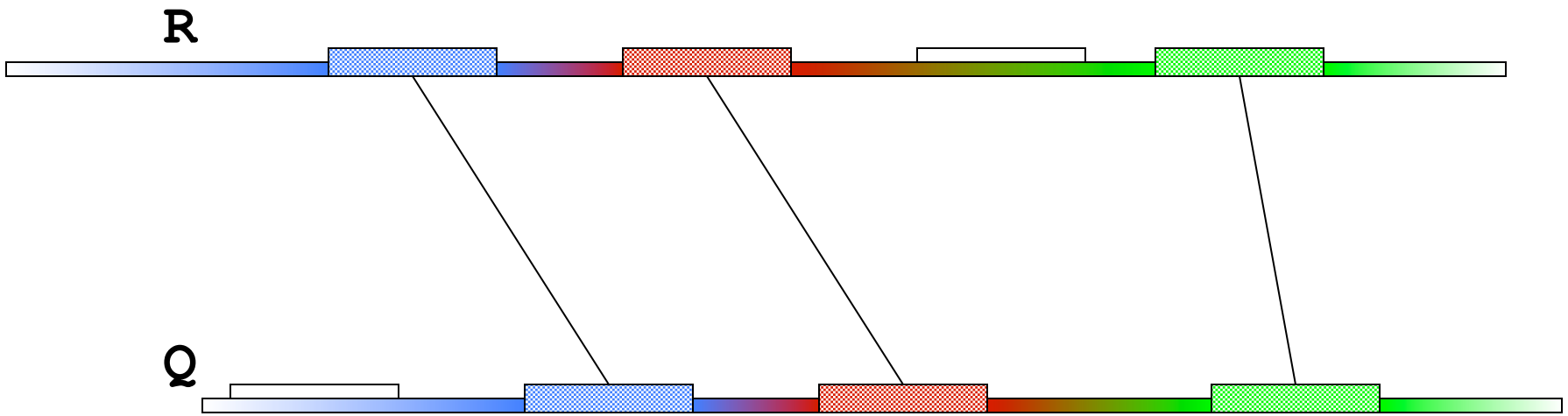
Seed and Extend

visualization

FIND all MUMs

CLUSTER consistent MUMs

EXTEND alignments



WGA example with **nucmer**

- *Yersina pestis CO92* vs. *Yersina pestis KIM*
  - High nucleotide similarity, 99.86%
    - Two strains of the same species
  - Extensive genome shuffling
    - Global alignment will not work
  - Highly repetitive
    - Many local alignments

WGA Alignment

nucmer –maxmatch CO92.fasta KIM.fasta

```
-maxmatch    Find maximal exact matches (MEMs)
```

delta-filter –m out.delta > out.filter.m

```
-m  Many-to-many mapping
```

show-coords -r out.delta.m > out.coords

```
-r  Sort alignments by reference position
```

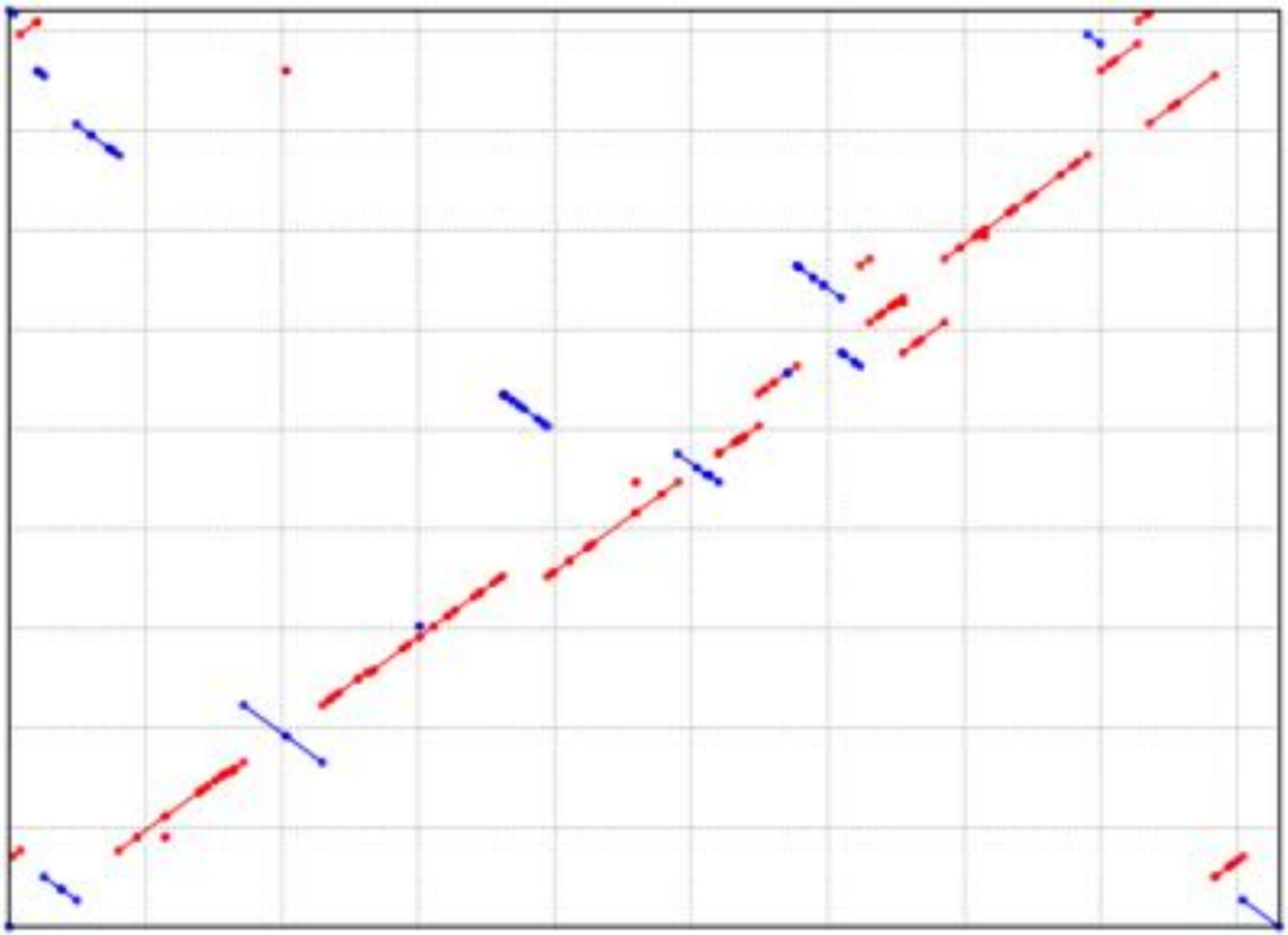
dnadiff out.delta.m

```
Construct catalog of sequence variations
```

mummerplot --large --layout out.delta.m

```
--large      Large plot
--layout Nice layout for multi-fasta files
--x11        Default, draw using x11 (--postscript, --png)
*requires gnuplot
```

See manual at http://
mummer.sourceforge.net/manual



References

- Documentation
  - http://mummer.sourceforge.net
    - » publication listing
  - http://mummer.sourceforge.net/manual
    - » documentation
  - http://mummer.sourceforge.net/examples
    - » walkthroughs

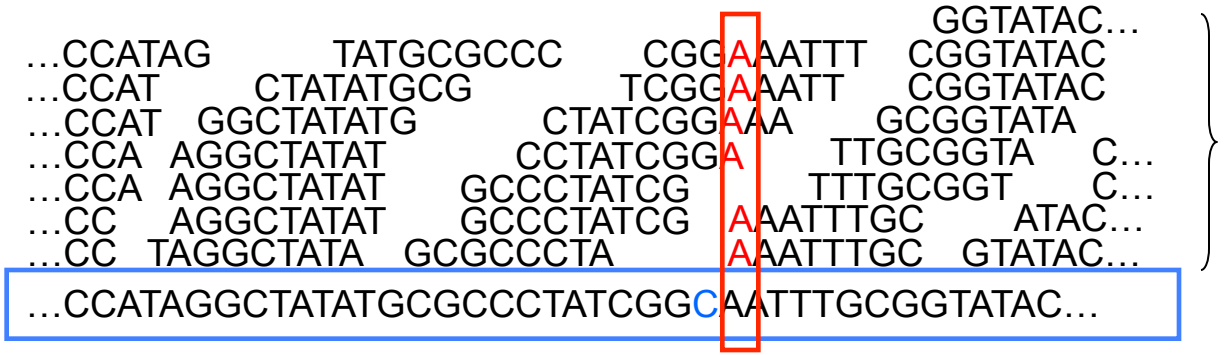
- Email
  - mummer-help@lists.sourceforge.net
  - amp@umiacs.umd.edu

Bowtie: Ultrafast and memory efficient alignment of short DNA sequences to the human genome

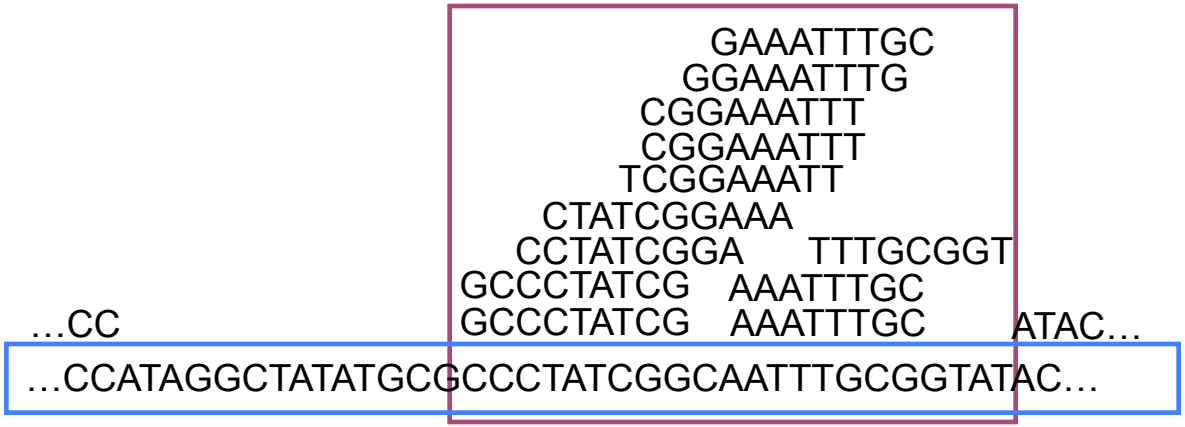
Slides Courtesy of Ben Langmead
(langmead@umiacs.umd.edu)

Short Read Applications

- Genotyping: Identify Variations



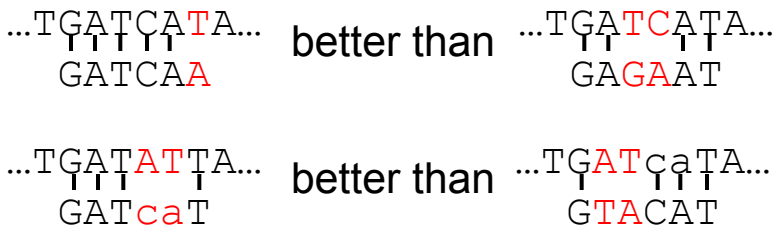
- *-seq: Classify & measure significant peaks



Short Read Alignment

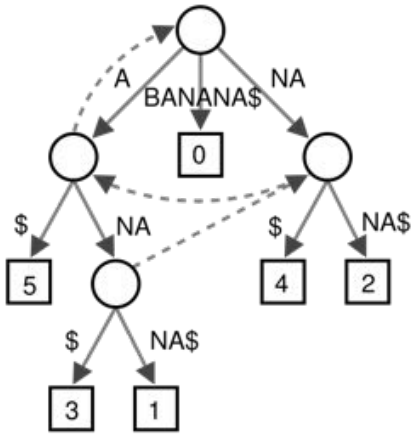
- Given a reference and a set of reads, report at least one "good" local alignment for each read if one exists
  - Approximate answer to: where in genome did read originate?

- What is "good"? For now, we concentrate on:
  - Fewer mismatches is better
  - Failing to align a low-quality base is better than failing to align a high-quality base

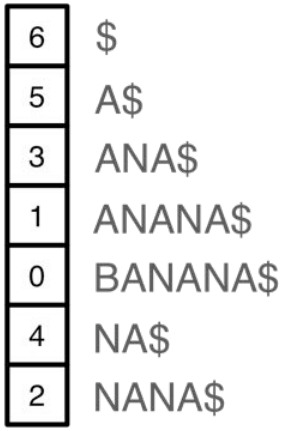


Indexing

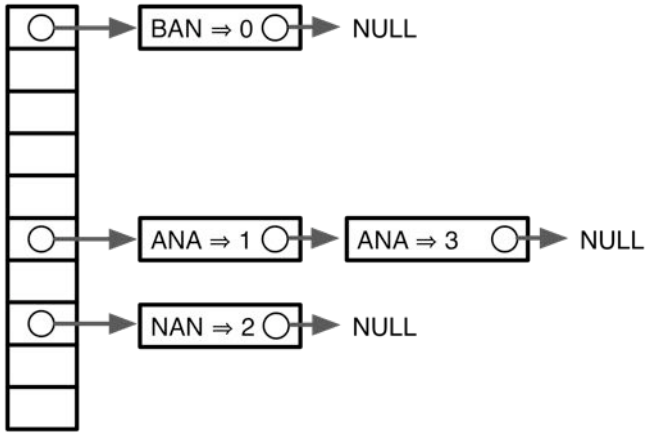
- Genomes and reads are too large for direct approaches like dynamic programming
  - Genome indices can be big. For human:



> 35 GBs



> 12 GBs

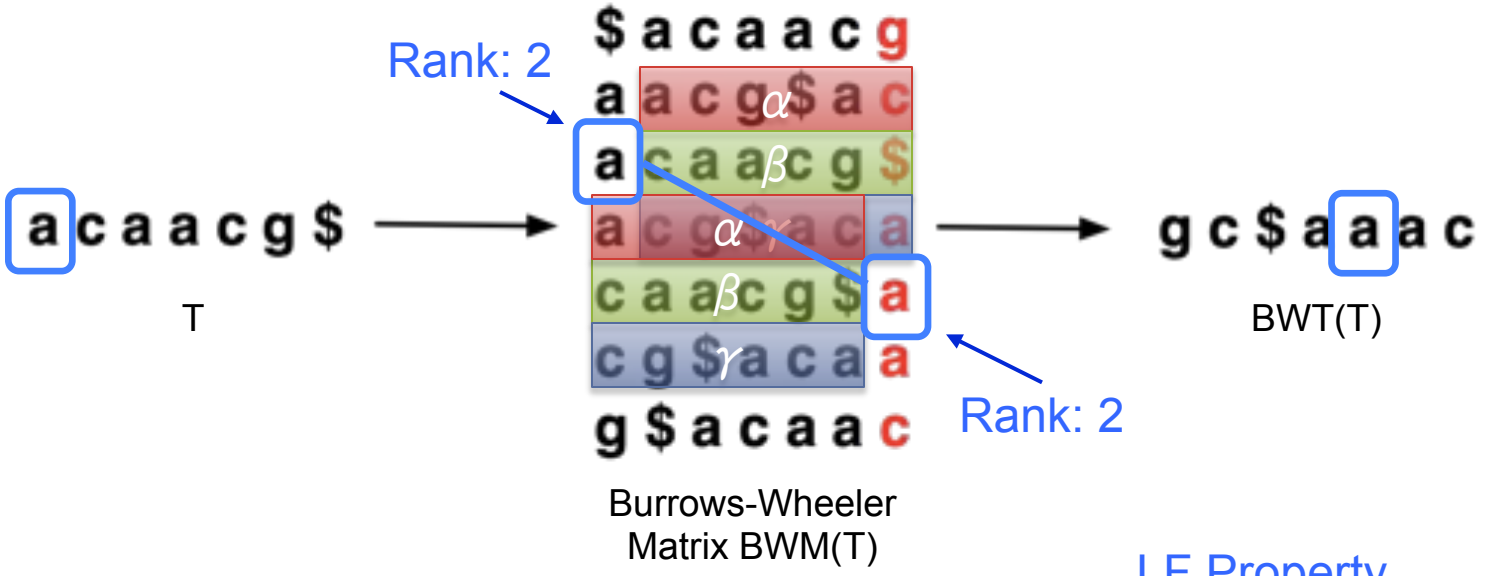


> 12 GBs

- Large indices necessitate painful compromises
  1. Require big-memory machine
  2. Use secondary storage
  3. Build new index each run
  4. Subindex and do multiple passes

Burrows-Wheeler Transform

- Reversible permutation of the characters in a text



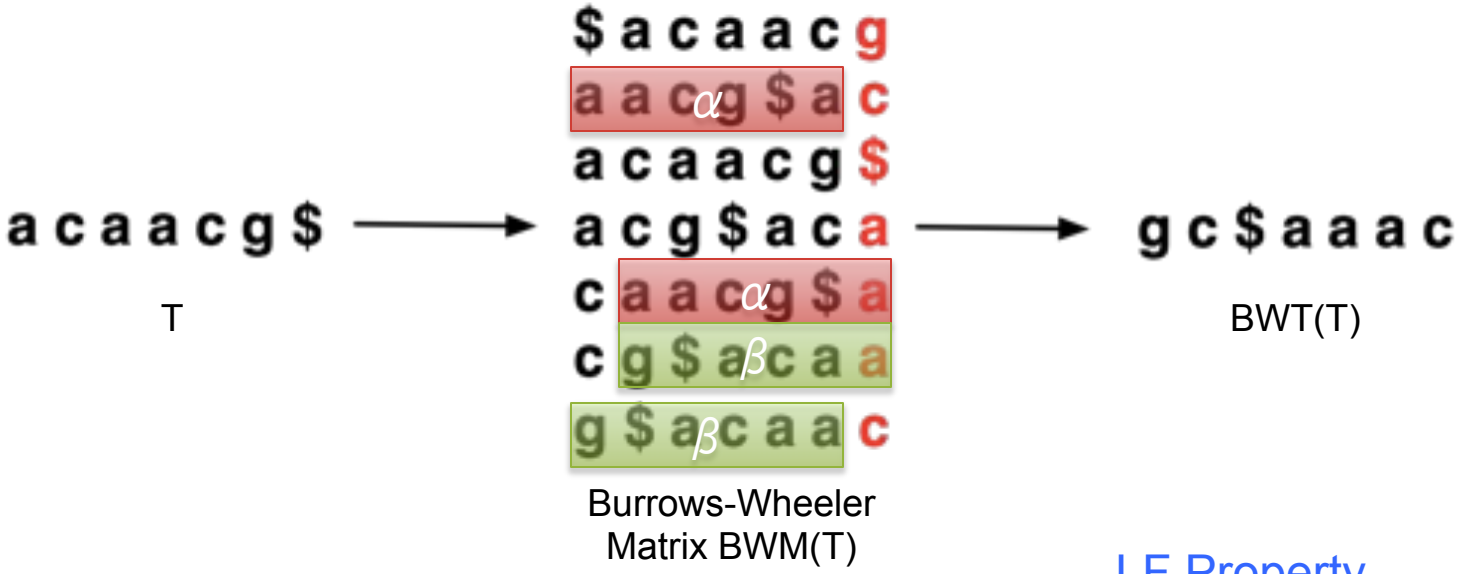
LF Property implicitly encodes Suffix Array

- BWT(T) is the index for T

A block sorting lossless data compression algorithm.
Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation*. Technical Report 124

Burrows-Wheeler Transform

- Reversible permutation of the characters in a text



- BWT(T) is the index for T

LF Property implicitly encodes Suffix Array

A block sorting lossless data compression algorithm.
Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation.* Technical Report 124

Burrows-Wheeler Transform

- Recreating T from BWT(T)
  - Start in the first row and apply **LF** repeatedly, accumulating predecessors along the way

Bowtie algorithm

Reference

BWT(Reference)

Query:
A A T G A T A C G G C G A C C A C C G A G A T C T A

Bowtie algorithm

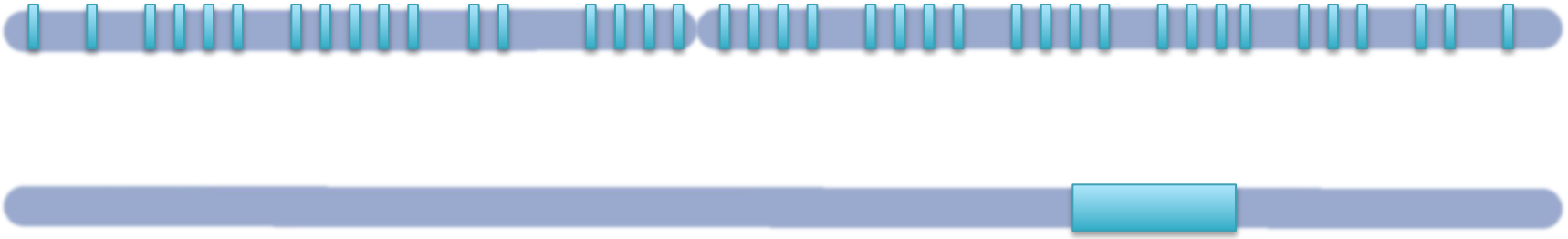
Reference

BWT(Reference)

Query:
A A T G A T A C G G C G A C C A C C G A G A T C T A

Bowtie algorithm

Reference

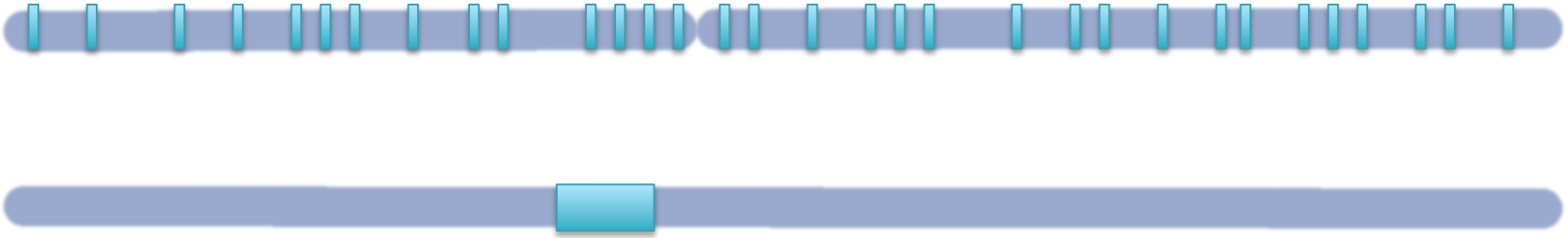


BWT(Reference)

Query:
A A T G A T A C G G C G A C C A C C G A G A T C T A

Bowtie algorithm

Reference

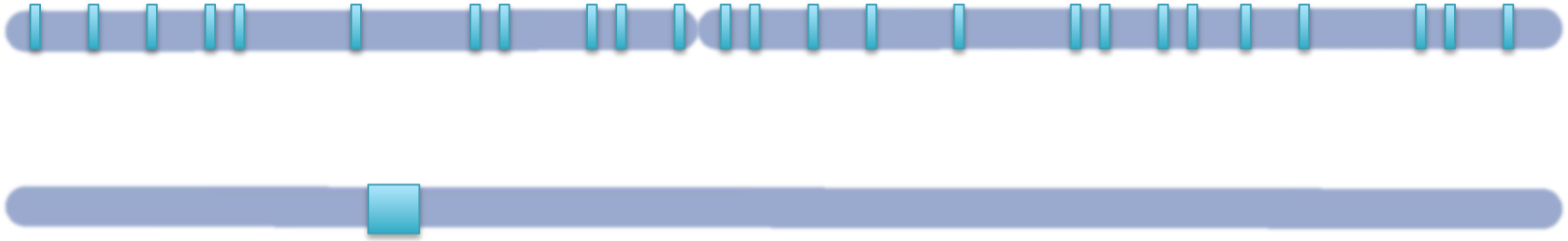


BWT(Reference)

Query:
AATGATACGGCGACCACCGAGATCTA

Bowtie algorithm

Reference



BWT(Reference)

Query:
AATGATACGGCGACCACCGAGATCTA

Bowtie algorithm

Reference

BWT(Reference)

Query:
A A T G A T A C G G C G A C C A C C G A G A T C T A

Bowtie algorithm

Reference

BWT(Reference)

Query:
AATGATACGGCGACCACCGAGATCTA

Bowtie algorithm

Reference

BWT(Reference)

Query:

A A T G T T A C G G C G A C C A C C G A G A T C T A

Bowtie algorithm

Reference

BWT(Reference)

Query:

A A T G T T A C G G C G A C C A C C G A G A T C T A

BWT Short Read Mapping

1. Trim off very low quality bases & adapters from ends of sequences

2. Execute depth-first-search of the implicit suffix tree represented by the BWT
    1. If we fail to reach the end, back-track and resume search
    2. BWT enables searching for good end-to-end matches entirely in RAM
        1. 100s of times faster than competing approaches

3. Report the "best" n alignments
    1. Best = fewest mismatches/edit distance, possibly weighted by QV
    2. Some reads will have millions of equally good mapping positions
    3. If reads are paired, try to find mapping that satisfies both

Mapping Applications

- Mapping Algorithms
  - Bowtie: (BWT) Fastest, No indels => moderate sensitivity
  - BWA: (BWT) Fast, small indels => good sensitivity
  - Novoalign: (Hash Table) Slow, RAM intensive, big indels => high sensitivity

- Variation Detection
  - SNPs
    - SAMTools: Bayesian model incorporating depth, quality values, also indels
    - SOAPsnp: SAMTools + known SNPs, nucleotide specific errors, no indels
  - Structural Variations
    - Hydra: Very sensitive alignment, scan for discordant pairs
    - Large indels: Open Research Problem to assembly their sequence
  - Copy number changes
    - RDexplorer: Scan alignments for statistically significant coverage pileup
  - Microsatellite variations
    - See Mitch!

Sequence Alignment Summary

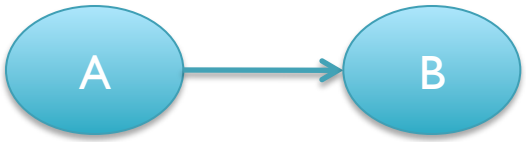
- Distance metrics:
  - Hamming: How many substitutions?
  - Edit Distance: How many substitutions or indels?
  - Sequence Similarity: How similar (under this model of similarity)?

- Techniques
  - Seed-and-extend: Anchor the search for in-exact using exact only
  - Dynamic Programming: Find a global optimal as a function of its parts
  - BWT Search: implicit DFS of SA/ST

- Sequence Alignment Algorithms: Pick the right tool for the job
  - Smith-Waterman: DP Local sequence alignment
  - BLAST: Homology Searching
  - MUMmer: Whole genome alignment, short read mapping (with care)
  - Bowtie/BWA/Novoalign: short read mapping

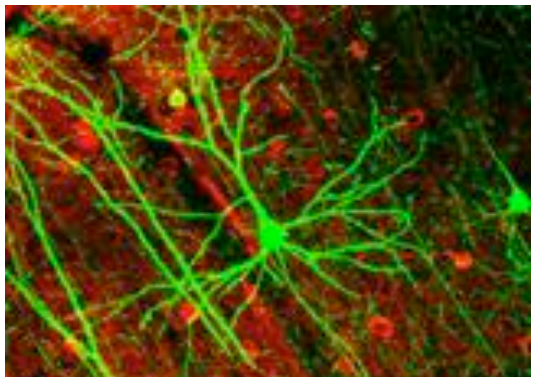
Break

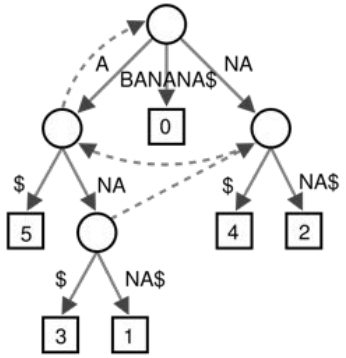
Graphs



- 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
  - …

Biological Networks





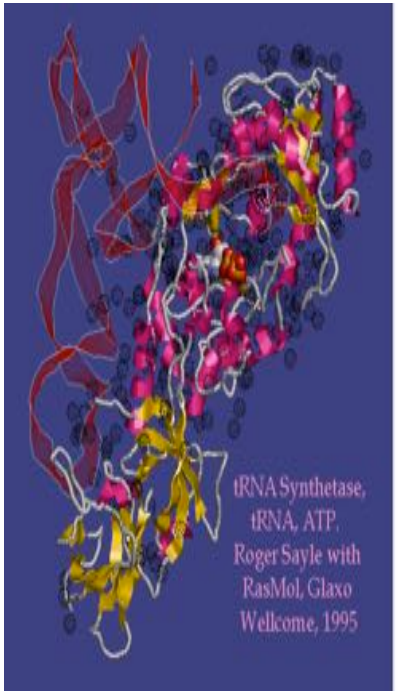
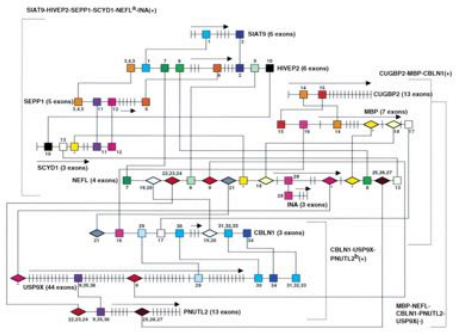
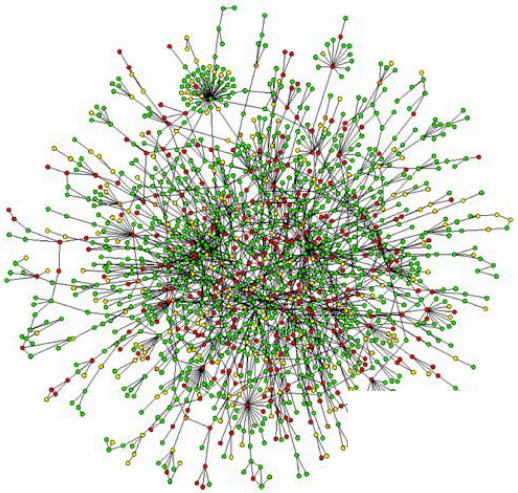


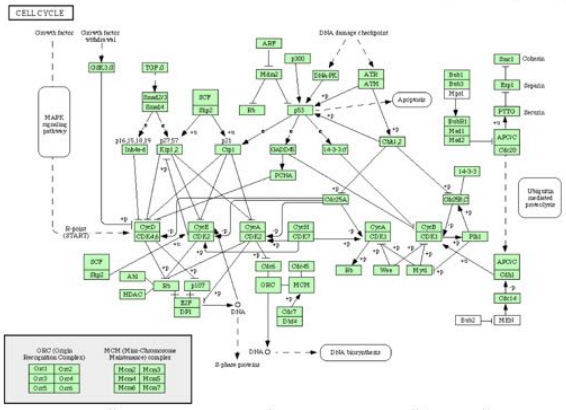
Figure 5 Putative regulatory elements shared between groups of correlated and anticorrelated genes

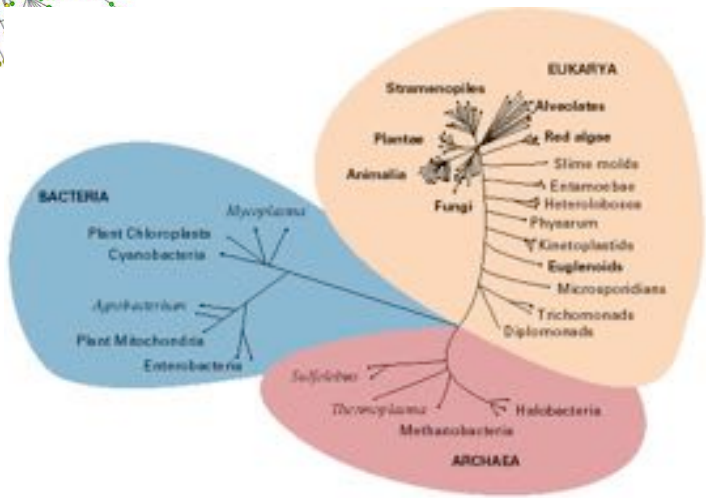


Vanessa M. Brown et al. Genome Res. 2002; 12: 868-884

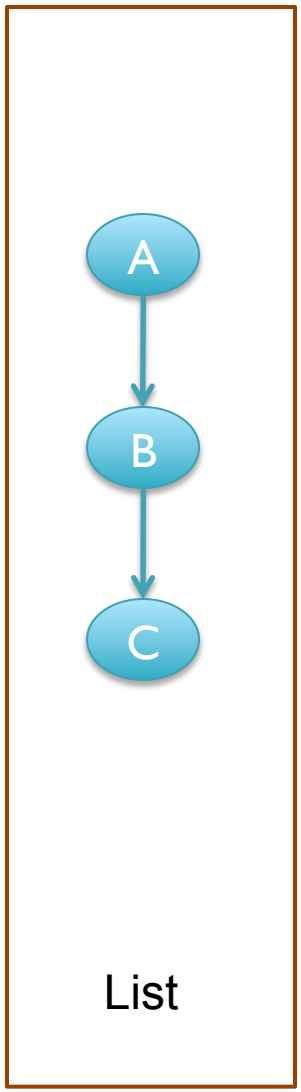
Cold Spring Harbor Laboratory Press

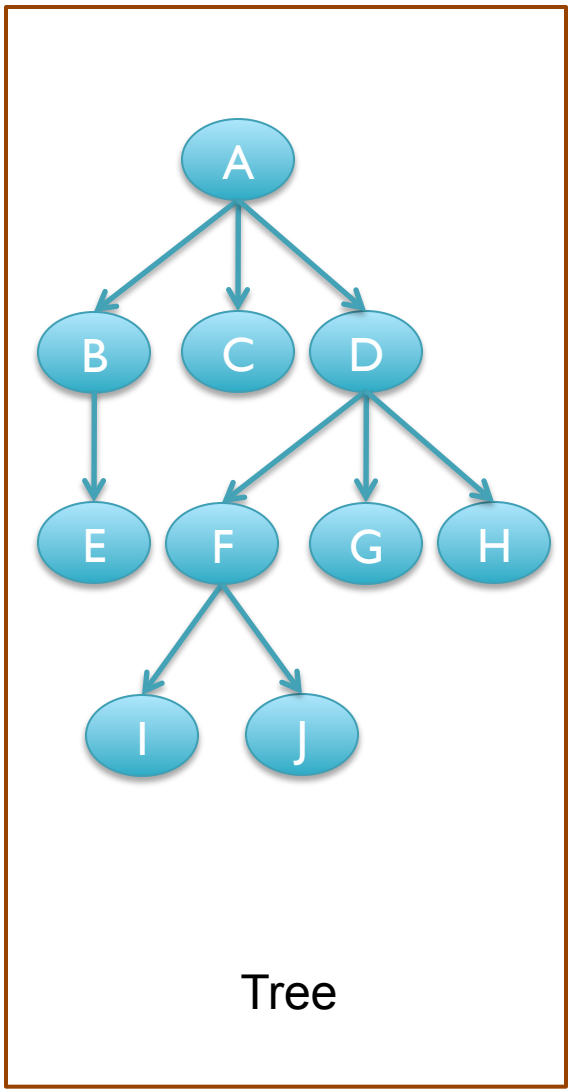


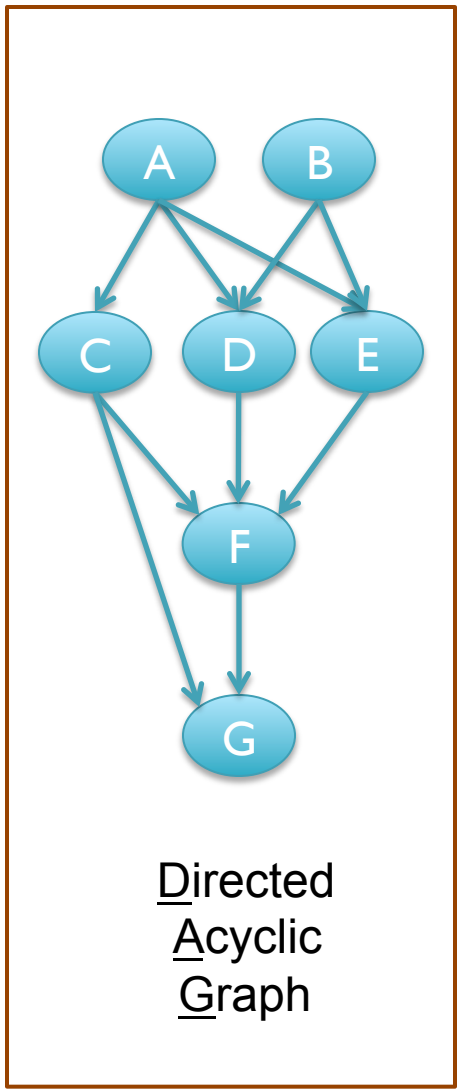


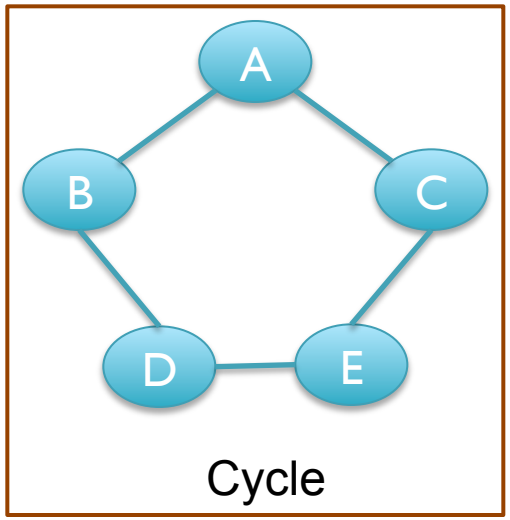


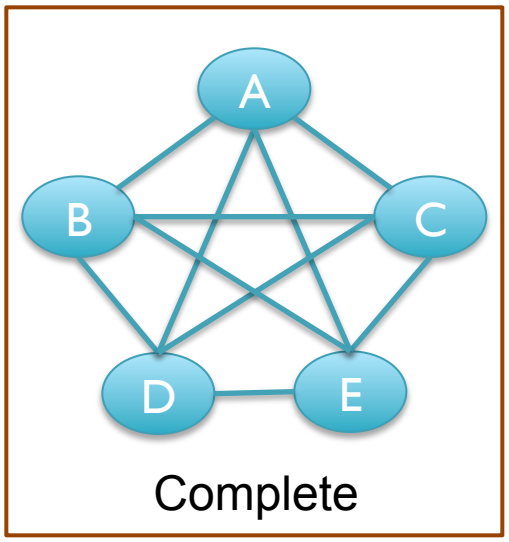
Graph Types


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

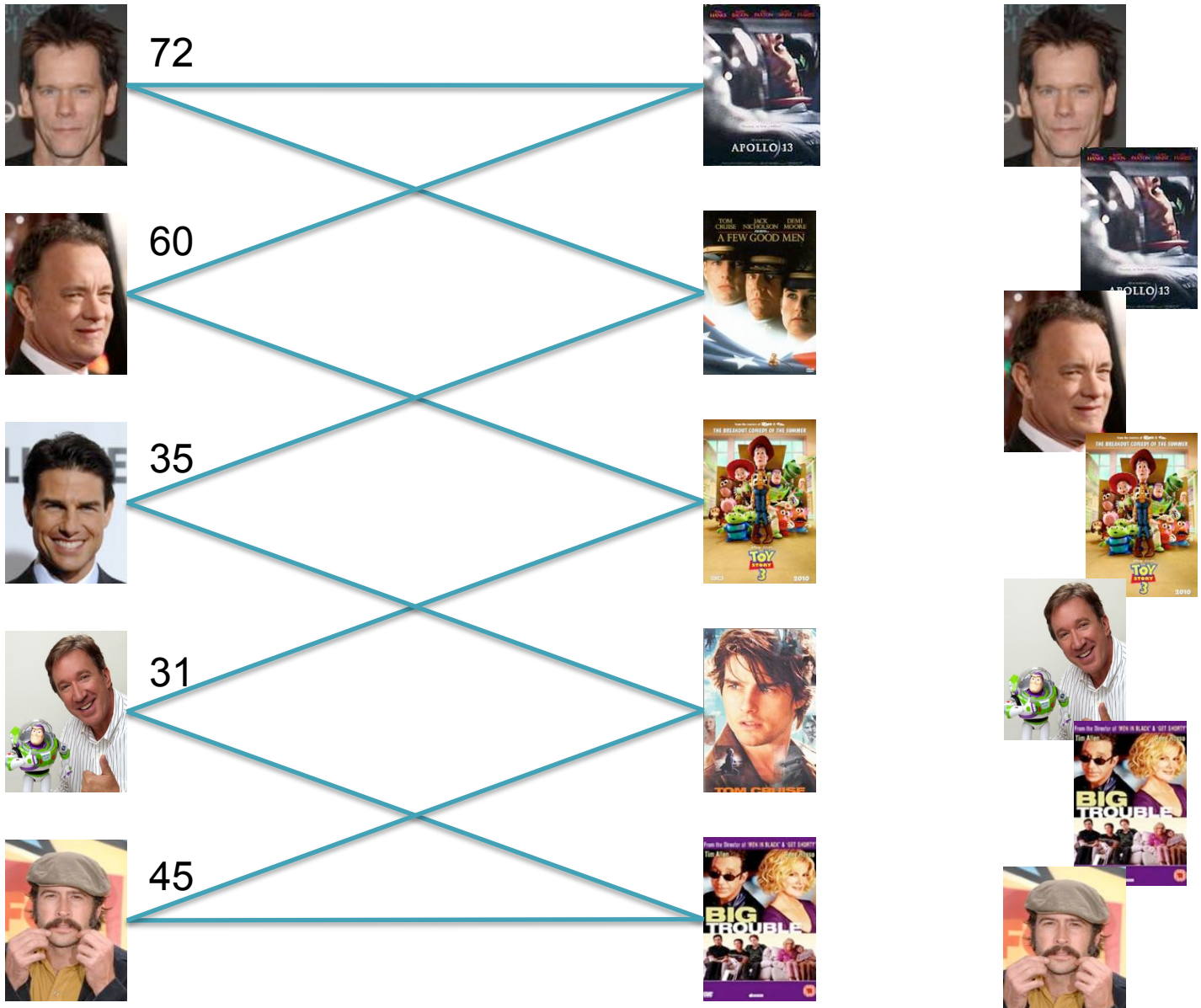

A
B
C
D
E
Complete

Kevin Bacon and Bipartite Graphs

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

Depth First Search:
6 hops

Bacon Distance:
3

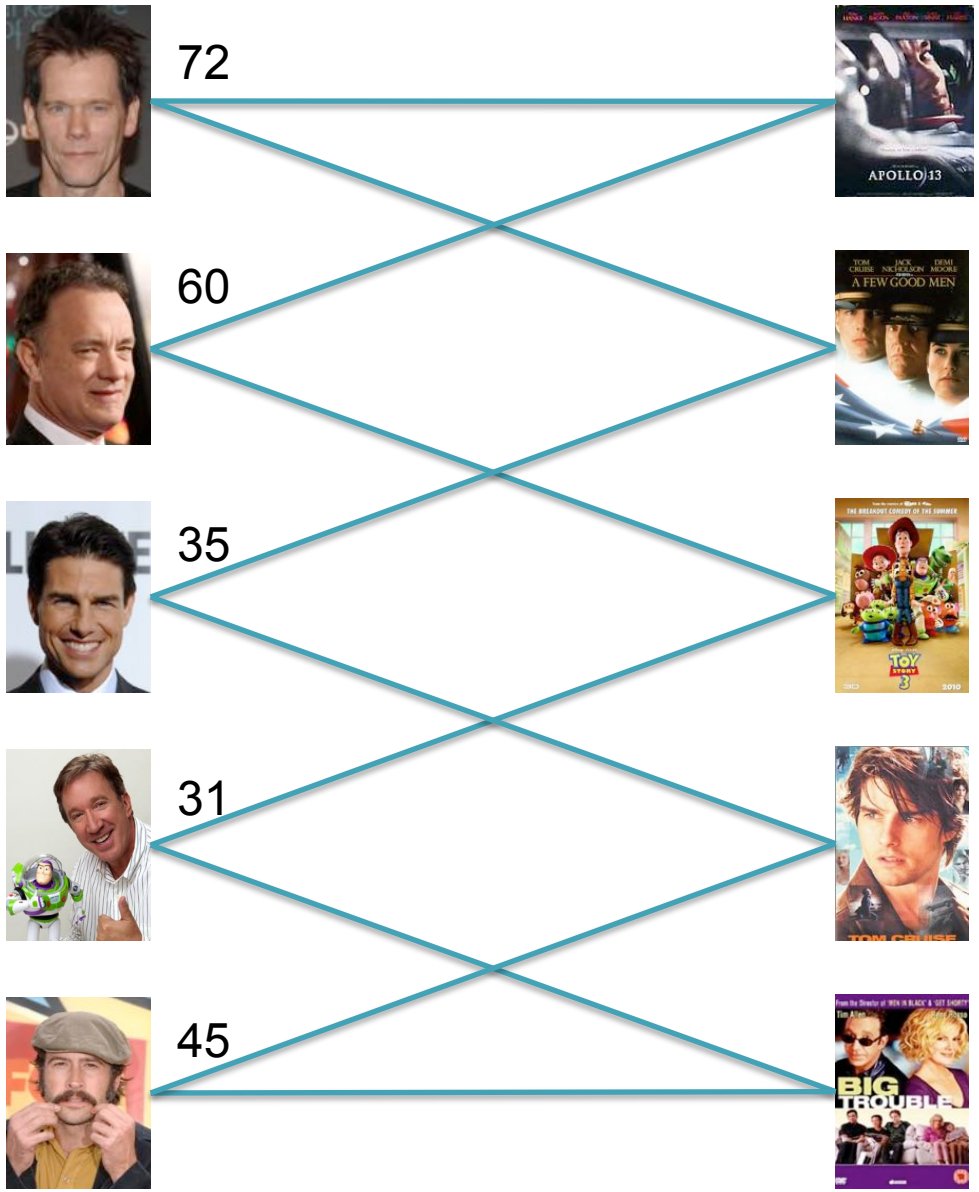


Kevin Bacon and Bipartite Graphs

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

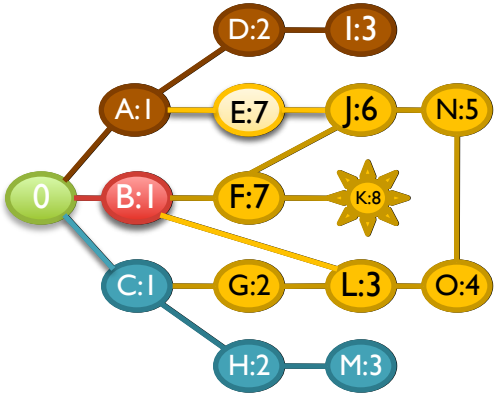
Breadth First Search:
4 hops

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?]

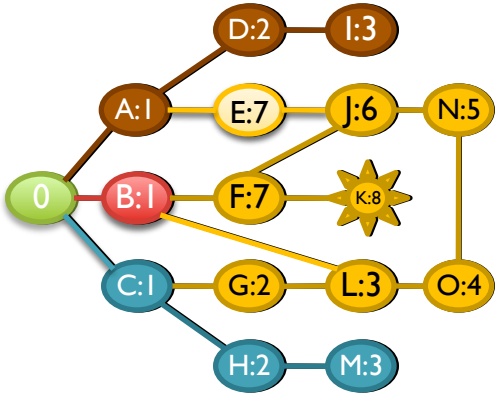
[What's the running time?]

[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)
```

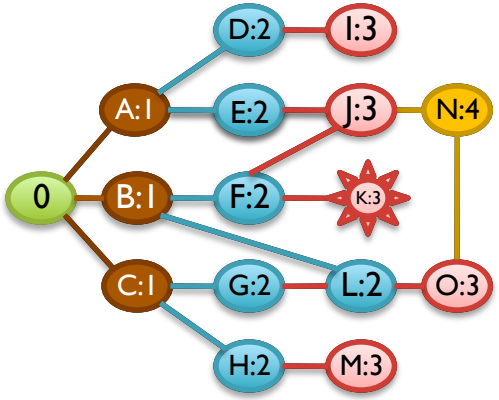
0
A,B,C
A,B,G,H
A,B,G,M
A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E
A,B
A
D
I



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

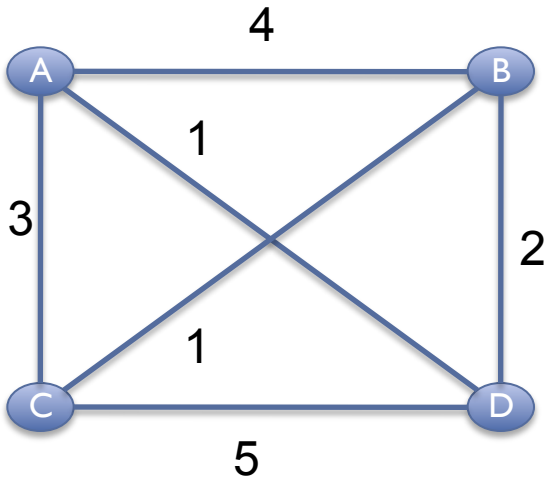
ACDBA: 3+5+2+4 = 14*

ABCDA: 4+1+5+1 = 11

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

ACBDA: 3+1+2+1 = 7

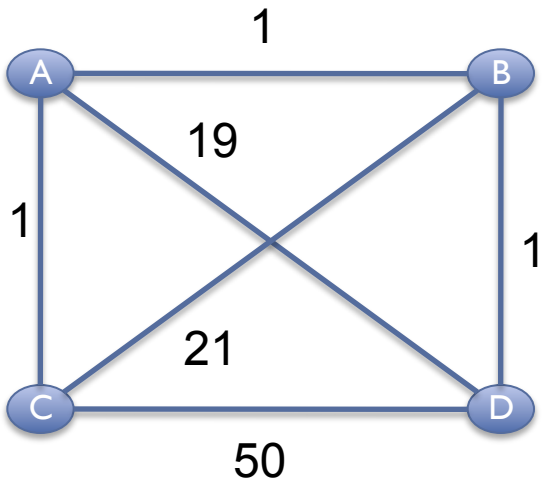
ADBCA: 1+2+1+3= 7 *



Greedy Search

Greedy Search

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



Greedy: ABDCA = 1+1+1+50= 53

Optimal: ACBDA = 1+19+1+21 = 42

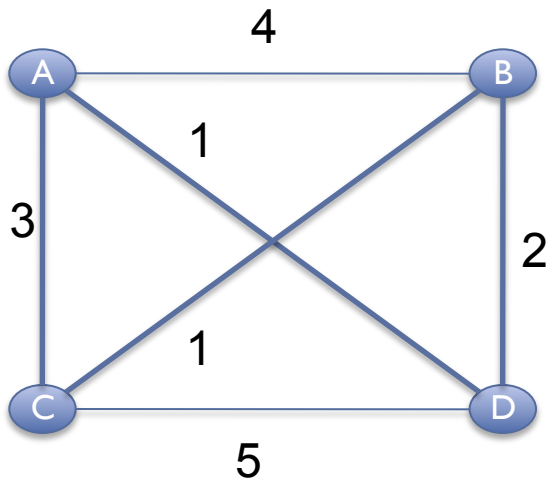
Greedy finds the global optimum only when

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

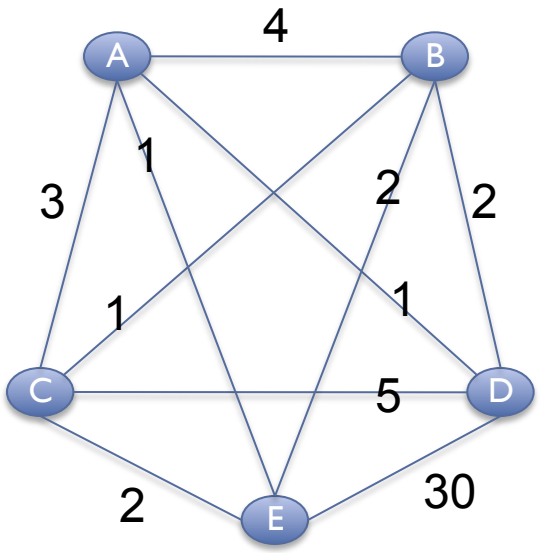
TSP Hardness

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



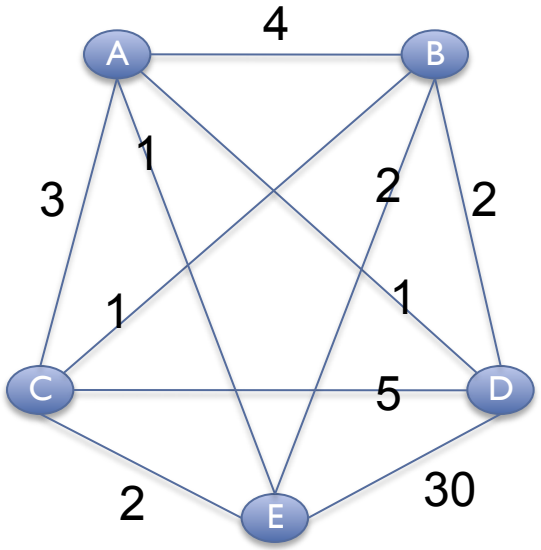
[How many possible tours for n cities?]

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



Branch-and-Bound

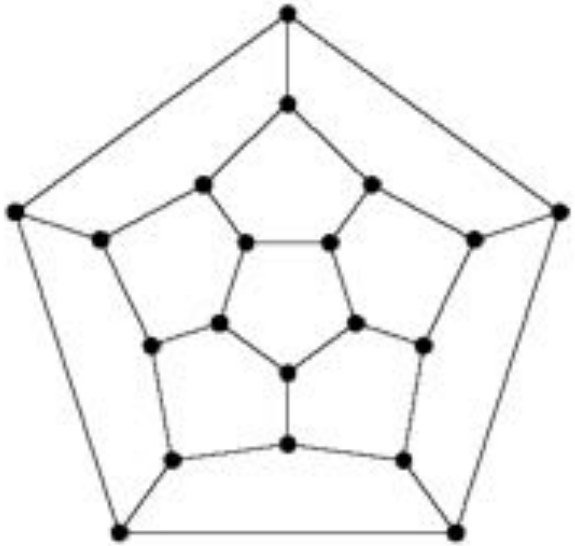
- Abort on suboptimal solutions as soon as possible
  - ADBECA = 1+2+2+2+3 = 10
  - ABDE = 4+2+30 > 10
  - ADE = 1+30 > 10
  - AED = 1+30 > 10
  - …



- 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 = CACCCACCGGGTGC 14

s_2,s_1,s_3 = CCGGGTGCACCCACC 15

s_2,s_3,s_1 = CCGGGTGCCACCC 13

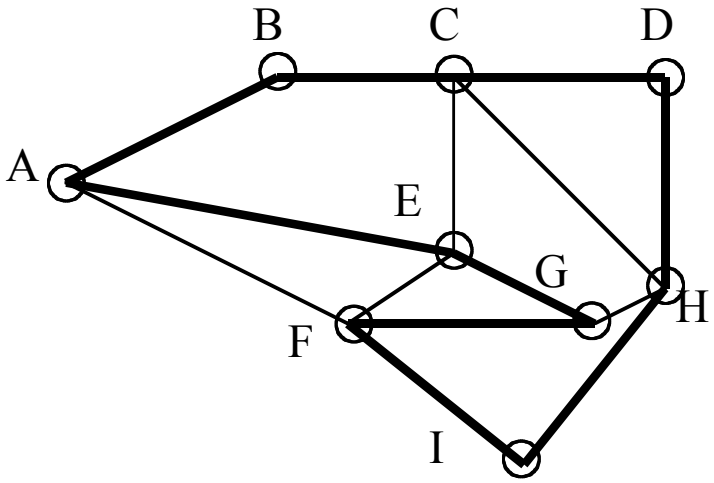
s_3,s_1,s_2 = CCACCCGGGTGC 12

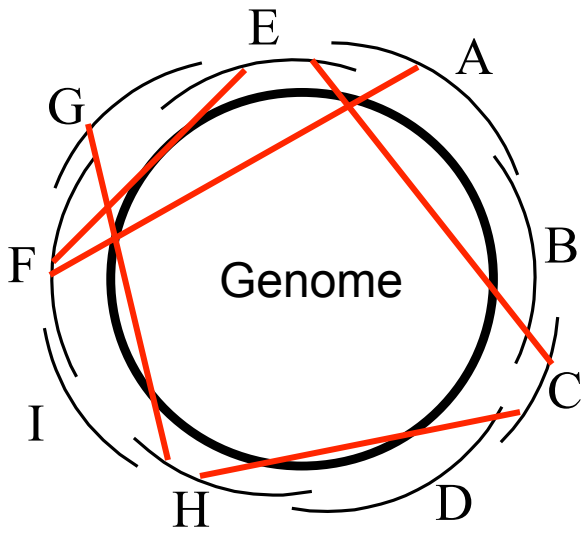
s_3,s_2,s_1 = CCACCGGGTGCACCC 15

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

Paths through graphs and assembly

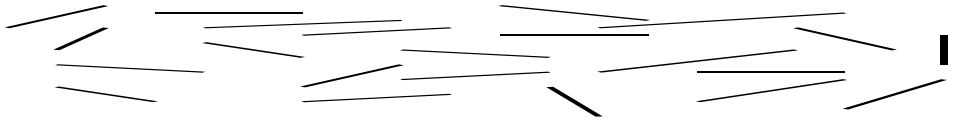
- Hamiltonian circuit: visit each node (read) exactly once, returning to the start
  - If we could do this fast, we could exactly assemble genomes as the shortest common superstring





Assembling a Genome

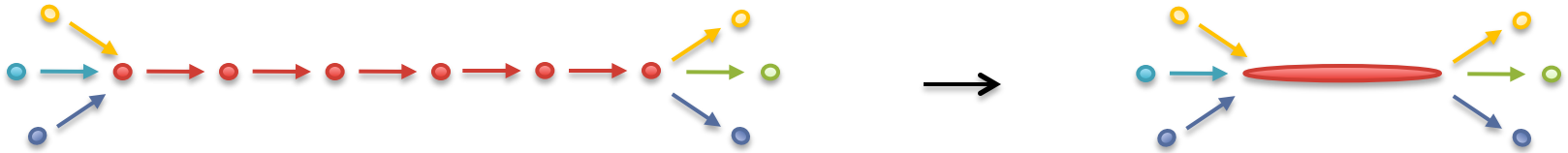
1. Shear & Sequence DNA



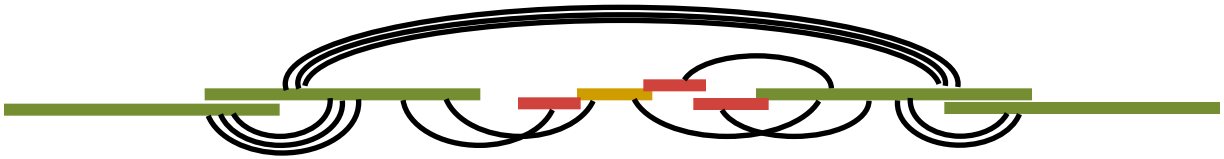
2. Construct assembly graph from overlapping reads

...AGCCTAGACCTACAGGATGCGCGACACGT
GGATGCGCGACACGTCGCATATCCGGT...

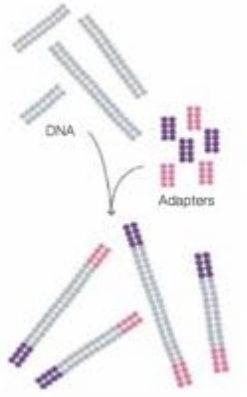
3. Simplify assembly graph



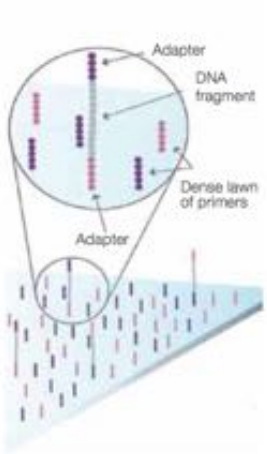
4. Detangle graph with long reads, mates, and other links



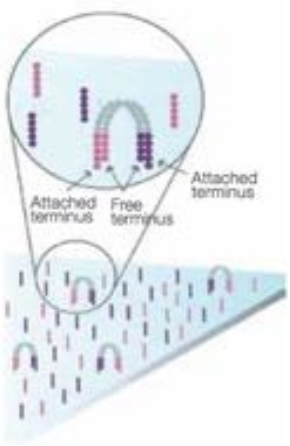
Illumina Sequencing by Synthesis



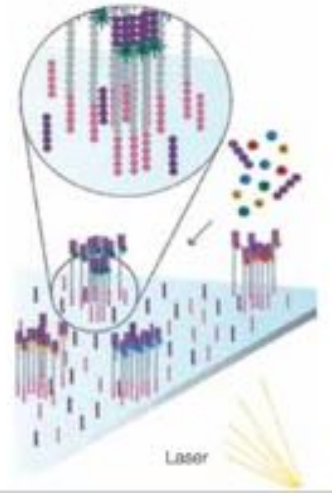
1. Prepare



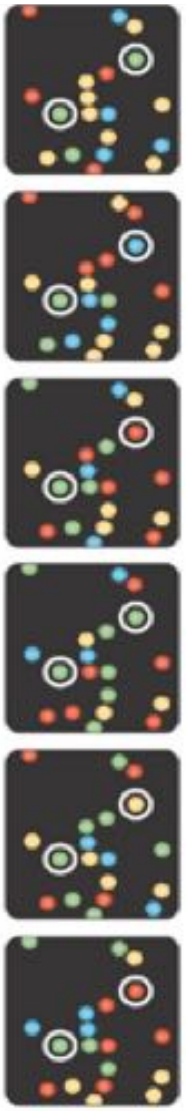
2. Attach



3. Amplify



4. Image



5. Basecall

Metzker (2010) Nature Reviews Genetics 11:31-46

http://www.illumina.com/documents/products/techspotlights/techspotlight_sequencing.pdf

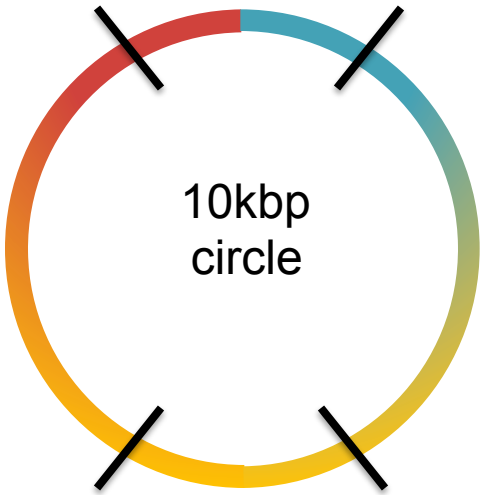
Paired-end and Mate-pairs

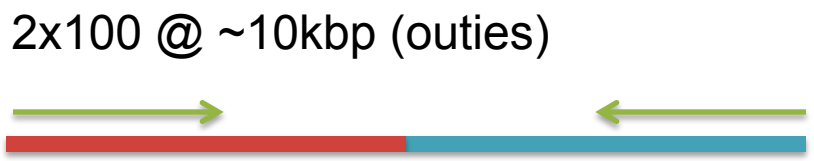
Paired-end sequencing

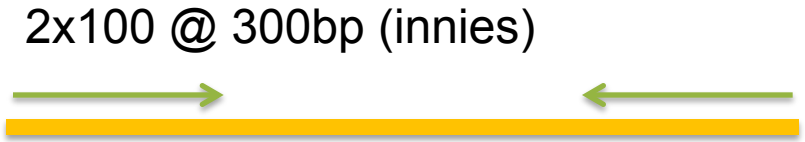
- Read one end of the molecule, flip, and read the other end
- Generate pair of reads separated by up to 500bp with inward orientation

Mate-pair sequencing

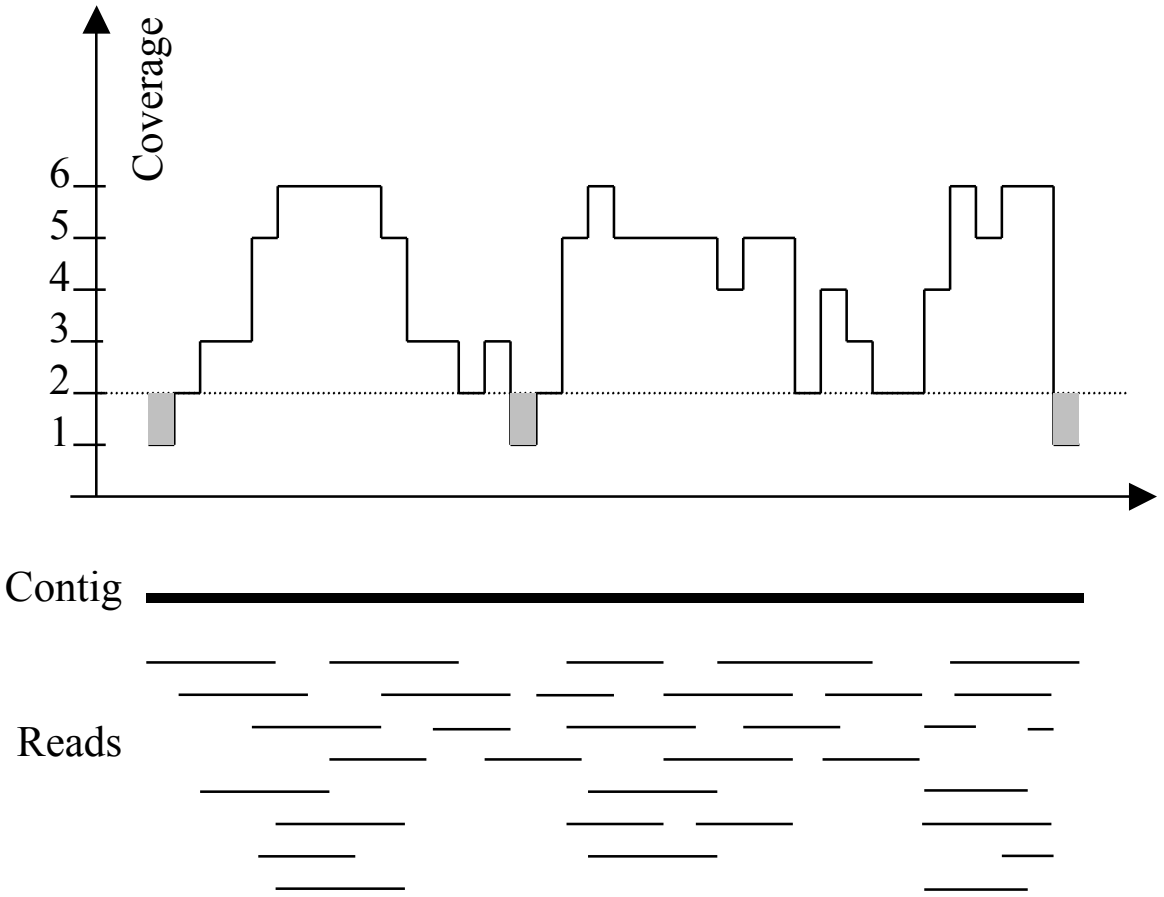
- Circularize long molecules (1-10kbp), shear into fragments, & sequence
- Mate failures create short paired-end reads





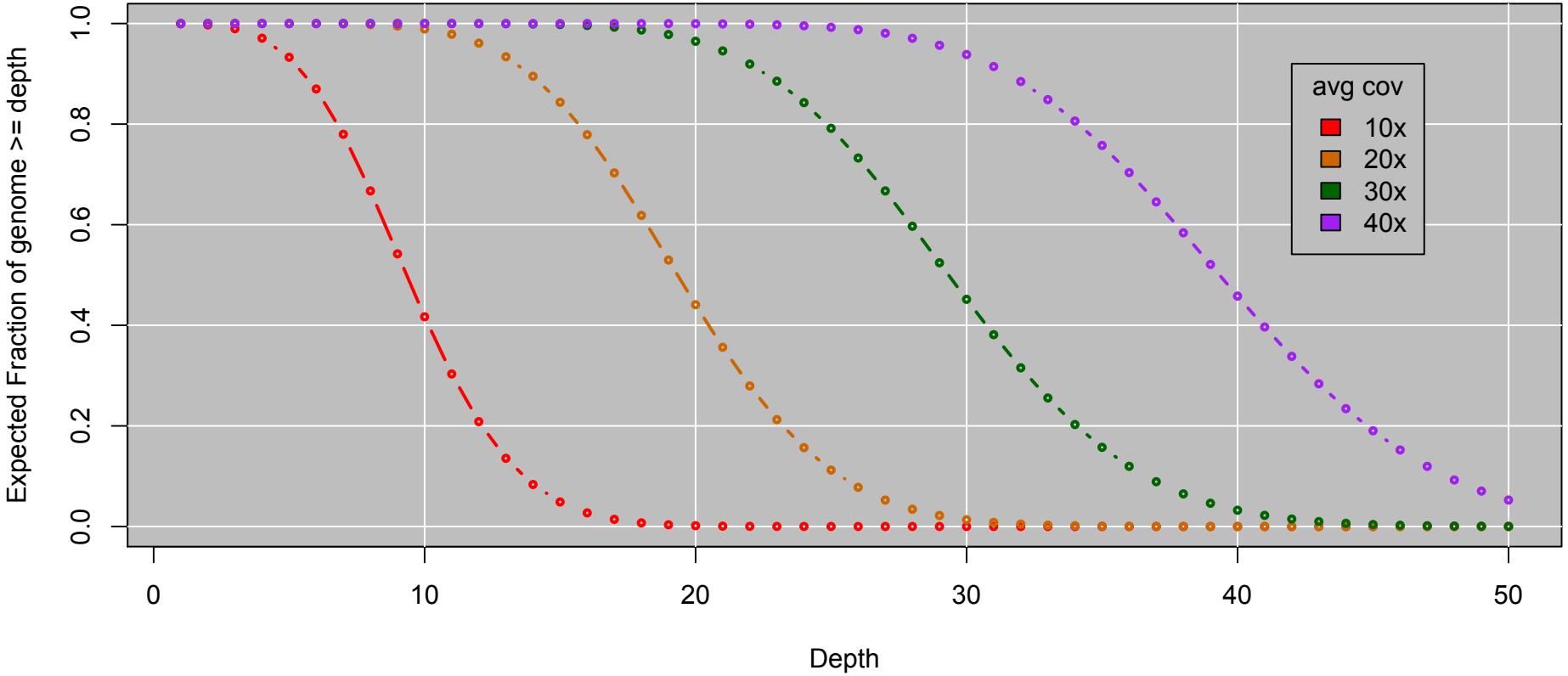


Typical contig coverage



Imagine raindrops on a sidewalk

Genome Coverage Distribution

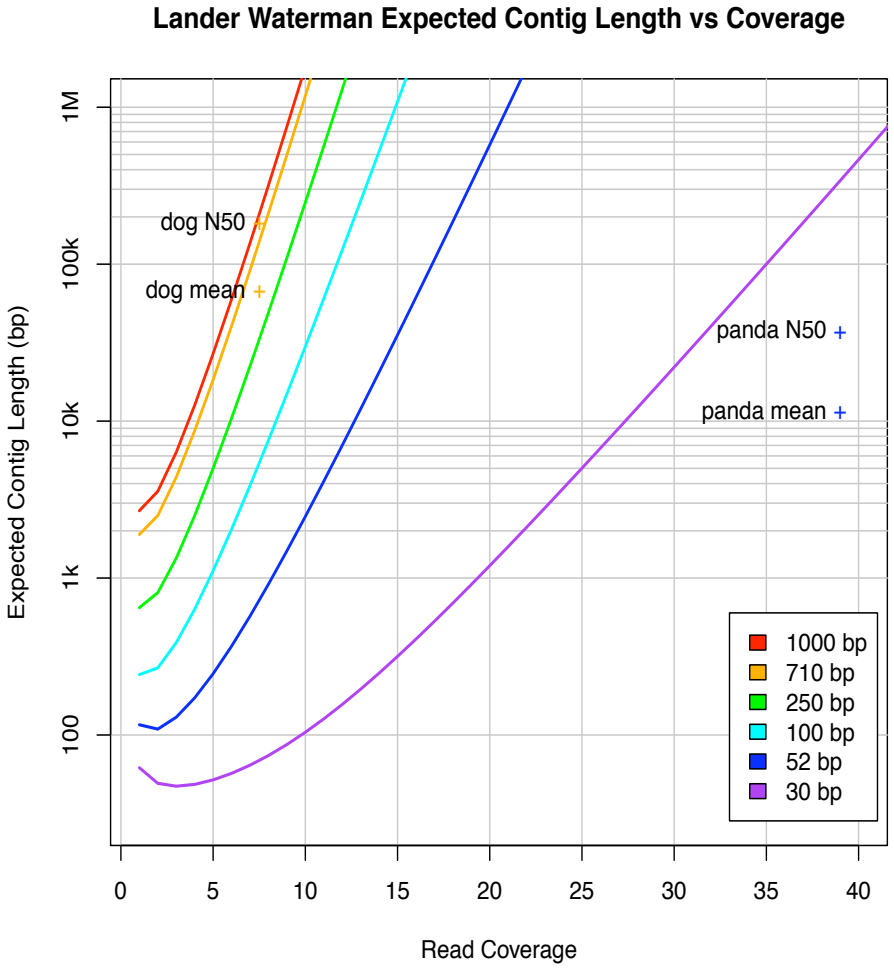


This is the mathematically model => reality may be much worse

Coverage and Read Length

Idealized Lander-Waterman model

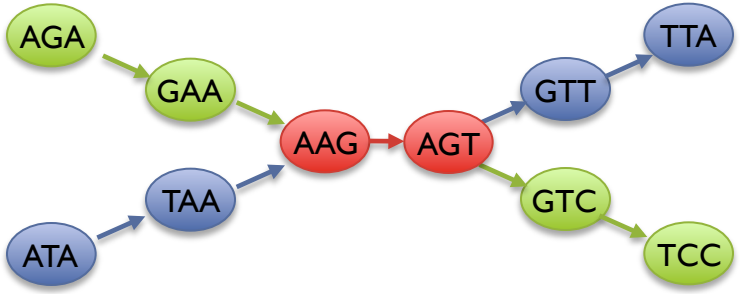
- Reads start at perfectly random positions
- Poisson distribution in coverage
  - Contigs end when there are no overlapping reads
- Contig length is a function of coverage and read length
  - Effective coverage reduced by *o/l*
  - Short reads require much higher coverage to reach same expected contig length



Assembly of Large Genomes using Second Generation Sequencing
Schatz MC, Delcher AL, Salzberg SL (2010) *Genome Research.* 20:1165-1173.

Two Paradigms for Assembly

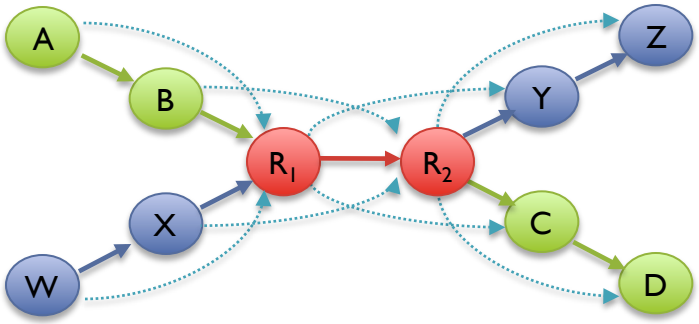
de Bruijn Graph



Short read assemblers

- Repeats depends on word length
- Read coherency, placements lost
- Robust to high coverage

Overlap Graph



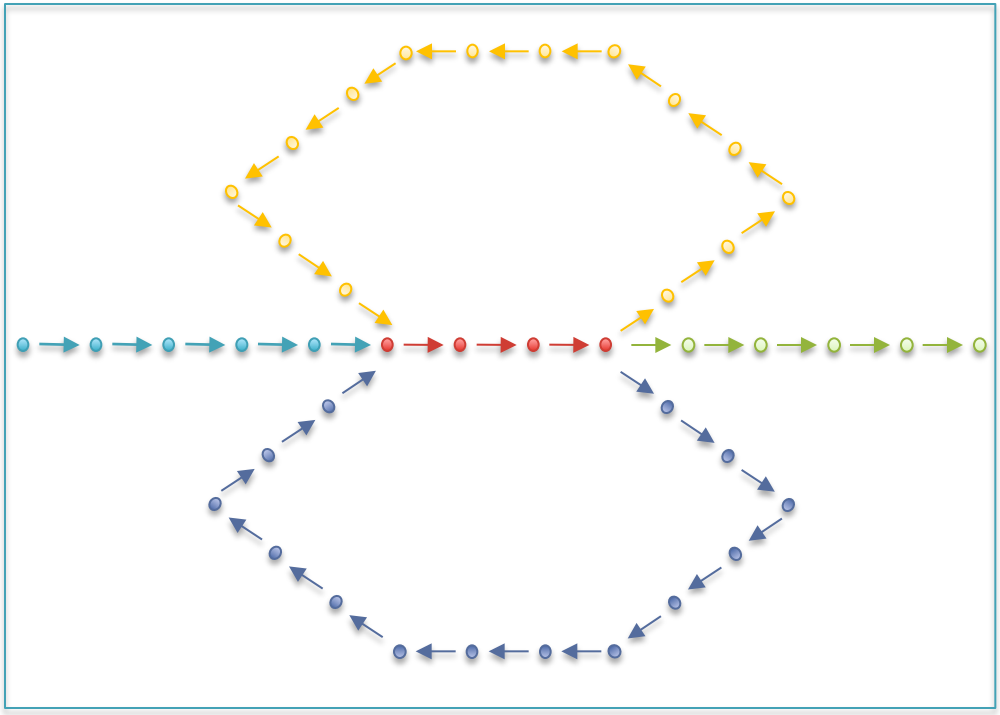
Long read assemblers

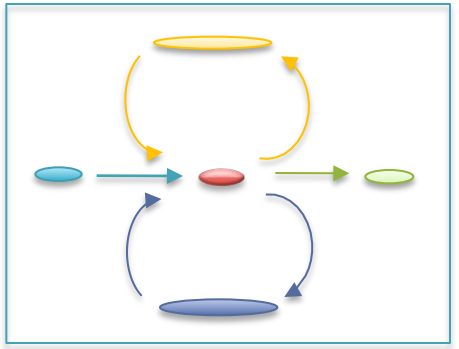
- Repeats depends on read length
- Read coherency, placements kept
- Tangled by high coverage

Assembly of Large Genomes using Second Generation Sequencing
Schatz MC, Delcher AL, Salzberg SL (2010) *Genome Research.* 20:1165-1173.

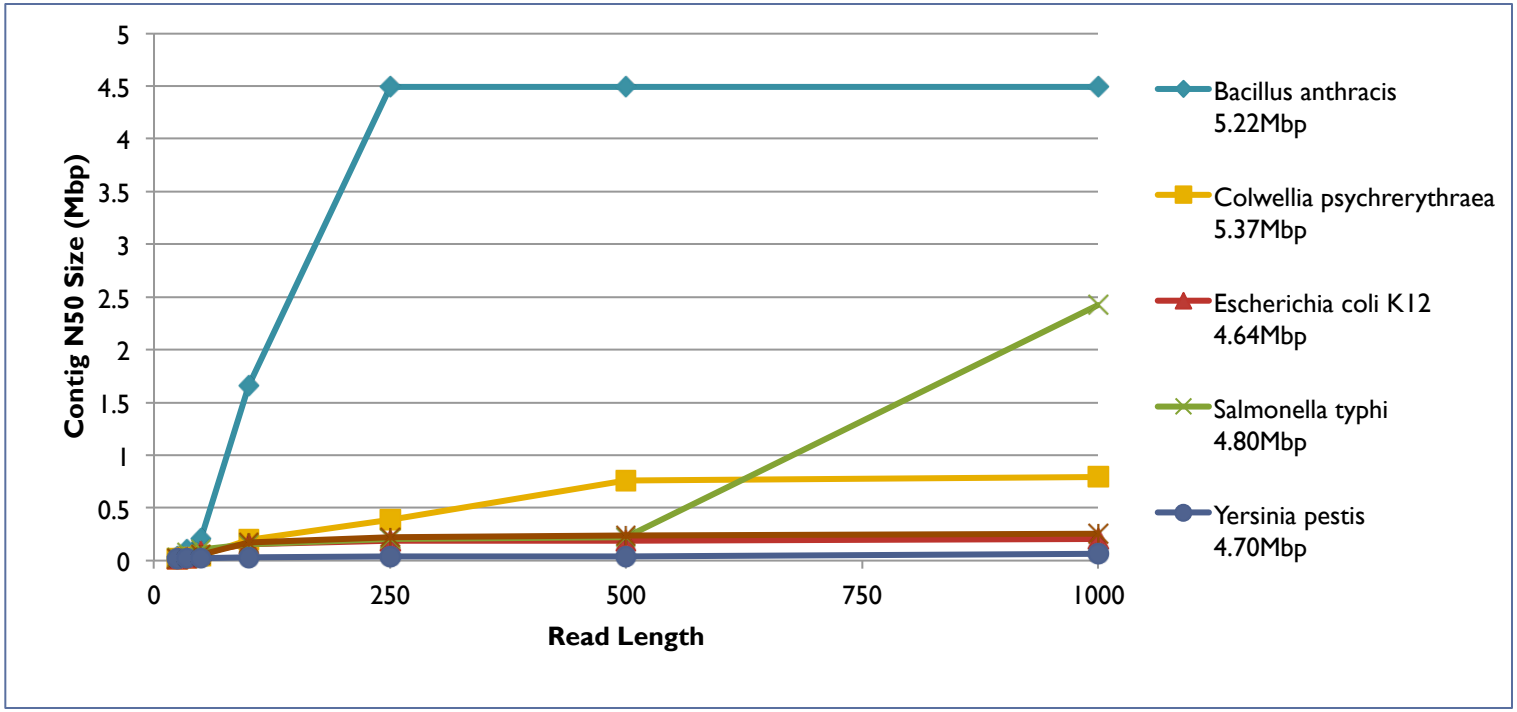
Initial Contigs

- After simplification and correction, compress graph down to its non-branching initial contigs
  - Aka "unitigs", "unipaths"





Repeats and Read Length



- Explore the relationship between read length and contig N50 size
  - Idealized assembly of read lengths: 25, 35, 50, 100, 250, 500, 1000
  - Contig/Read length relationship depends on specific repeat composition

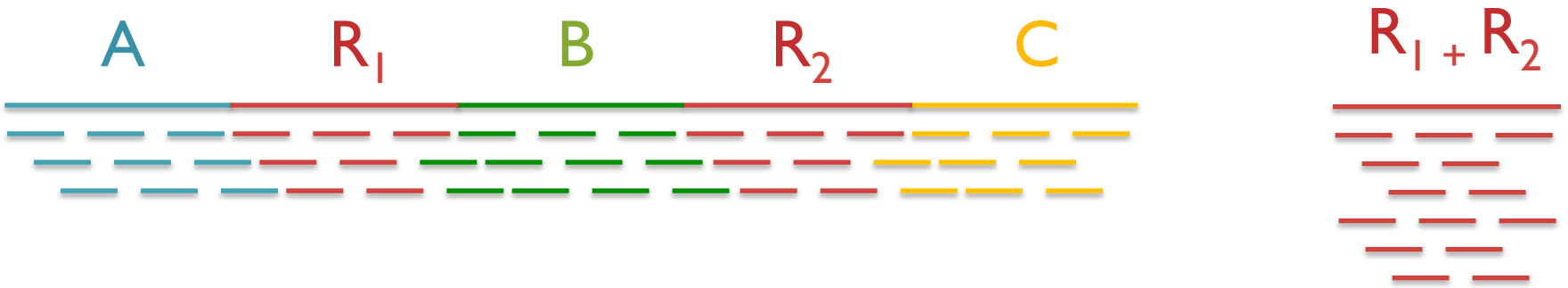
Assembly Complexity of Prokaryotic Genomes using Short Reads.
Kingsford C, Schatz MC, Pop M (2010) *BMC Bioinformatics*. 11:21.

Repetitive regions

- Over 50% of the human genome is repetitive

Repeat Type	Definition / Example	Prevalence
Low-complexity DNA / Microsatellites	$(b_1b_2\ldots b_k)^N$ where $1 \leq k \leq 6$ CACACACACACACACACACA	2%
SINEs (Short Interspersed Nuclear Elements)	*Alu* sequence (~280 bp) Mariner elements (~80 bp)	13%
LINEs (Long Interspersed Nuclear Elements)	~500 – 5,000 bp	21%
LTR (long terminal repeat) retrotransposons	Ty1-copia, Ty3-gypsy, Pao-BEL (~100 – 5,000 bp)	8%
Other DNA transposons		3%
Gene families & segmental duplications		4%

Repeats and Coverage Statistics



- 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 > A) , it is likely to be a collapsed repeat
  - Requires an accurate genome size estimate

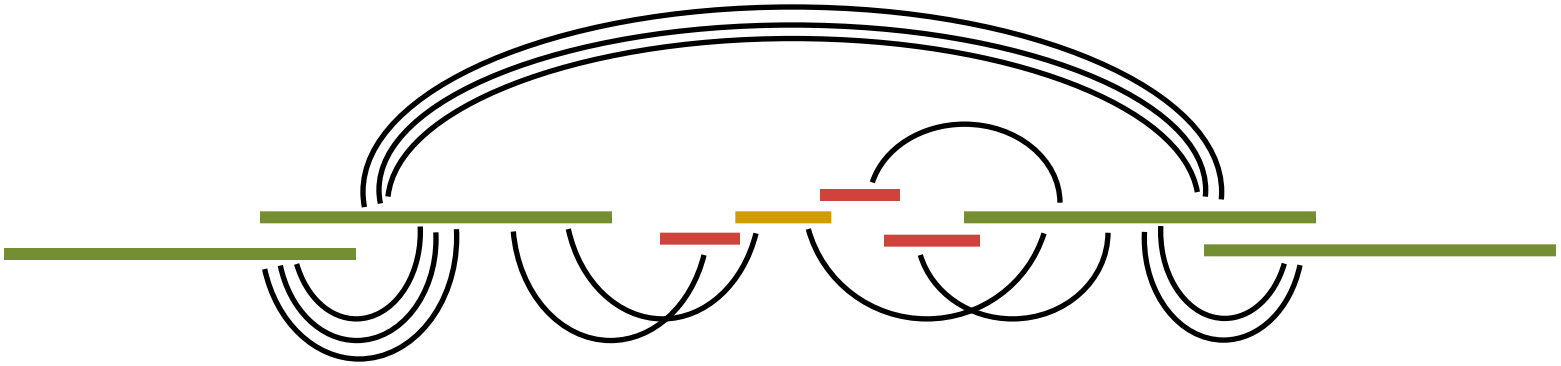
$$\Pr(X-copy)=\binom{n}{k}\left(\frac{X\Delta}{G}\right)^{k}\left(\frac{G-X\Delta}{G}\right)^{n-k}$$

$$A(\Delta,k)=\ln\left(\frac{\Pr(1-copy)}{\Pr(2-copy)}\right)=\ln\left(\frac{\frac{(\Delta n/G)^{k}}{k!}e^{\frac{-\Delta n}{G}}}{\frac{(2\Delta n/G)^{k}}{k!}e^{\frac{-2\Delta n}{G}}}\right)=\frac{n\Delta}{G}-k\ln 2$$

Scaffolding

- Initial contigs (*aka* unipaths, unitigs) terminate at
  - *Coverage gaps*: especially extreme GC regions
  - *Conflicts*: sequencing errors, repeat boundaries

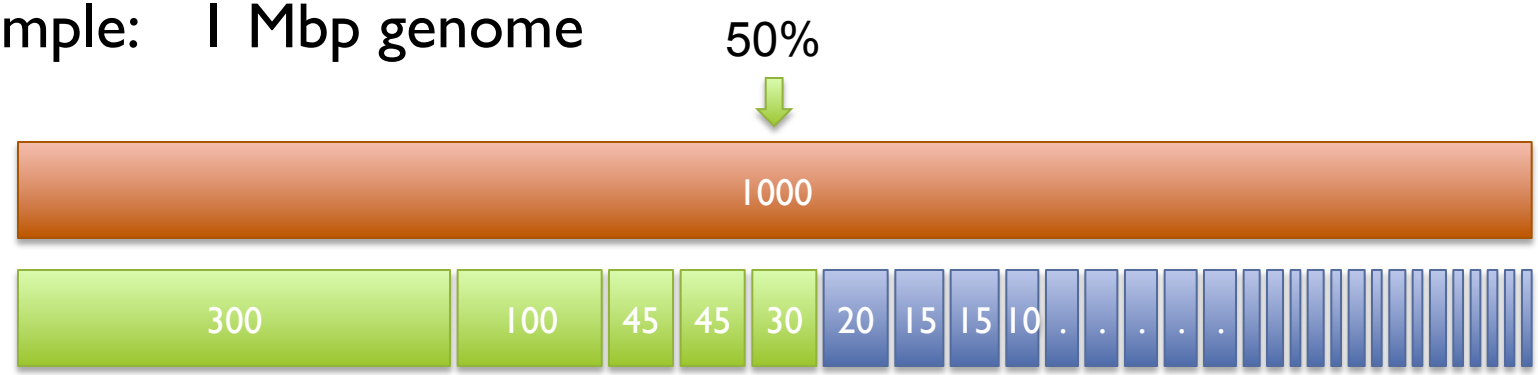
- Iteratively resolve longest, 'most unique' contigs
  - Both overlap graph and de Bruijn assemblers initially collapse repeats into single copies
  - Uniqueness measured by a statistical test on coverage



N50 size

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

Example: 1 Mbp genome



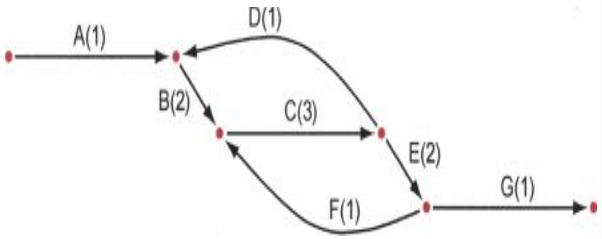
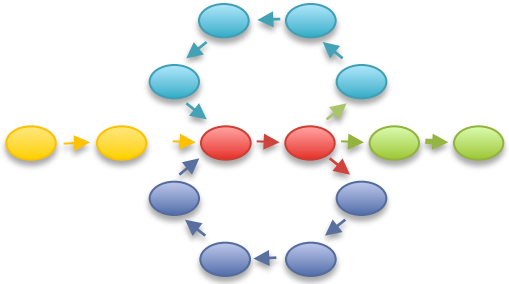
N50 size = 30 kbp

(300k+100k+45k+45k+30k = 520k >= 500kbp)

Note:

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

Assembly Algorithms

ALLPATHS-LG	SOAPdenovo	Celera Assembler
		
Broad's assembler (Gnerre et al. 2011)	BGI's assembler (Li et al. 2010)	JCVI's assembler (Miller et al. 2008)
De bruijn graph Short + PacBio (patching)	De bruijn graph Short reads	Overlap graph Medium + Long reads
Easy to run if you have compatible libraries	Most flexible, but requires a lot of tuning	Supports Illumina/454/PacBio Hybrid assemblies
http://www.broadinstitute.org/software/allpaths-lg/blog/	http://soap.genomics.org.cn/soapdenovo.html	http://wgs-assembler.sf.net

THE ASSEMBLATHON

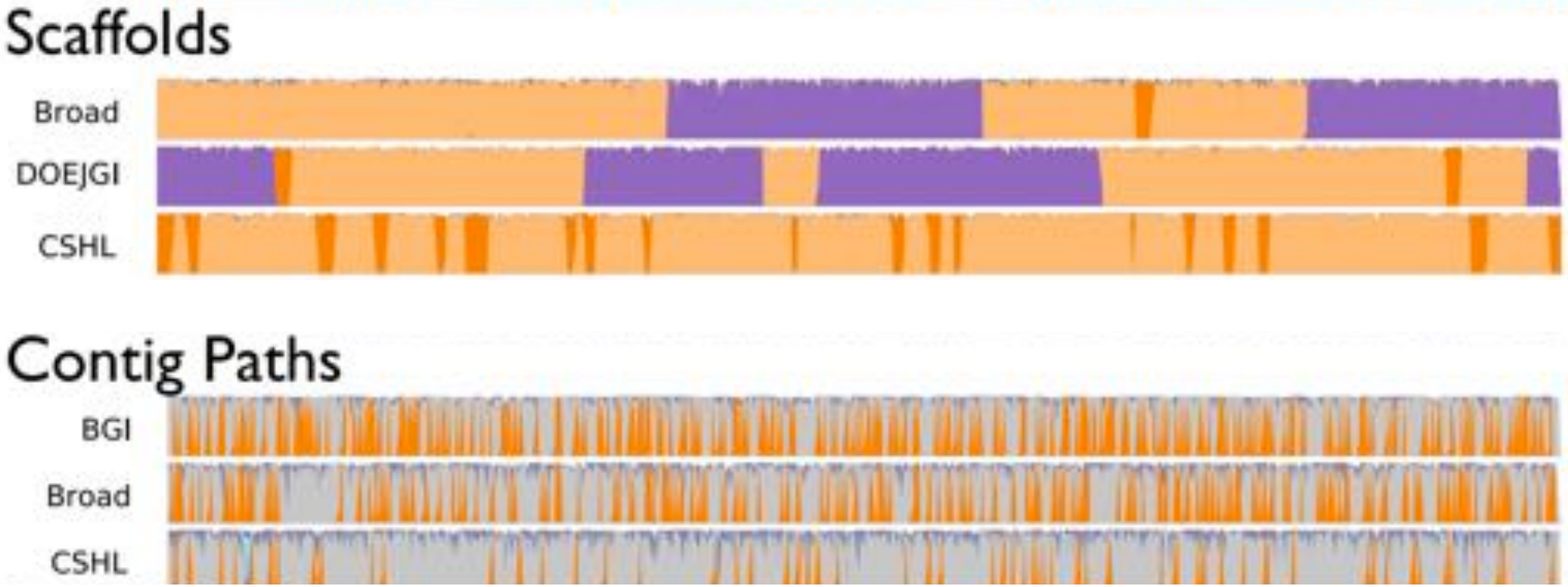
- Attempt to answer the question:
  "What makes a good assembly?"

- Organizers provided simulated sequence data
  - Simulated 100 base pair Illumina reads from simulated diploid organism

- 41 submissions from 17 groups

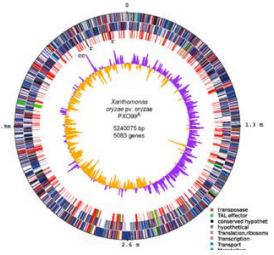
- Results demonstrate trade-offs assemblers must make

Assembly Results



- No assembler was perfect!
  - See tomorrow's in house for details

Summary



Graphs are ubiquitous in the world

- Pairwise searching is easy, finding features is hard

Assembly quality depends on

1. ***Coverage***: low coverage is mathematically hopeless
2. ***Repeat composition***: high repeat content is challenging
3. ***Read length***: longer reads help resolve repeats
4. ***Error rate***: errors reduce coverage, obscure true overlaps

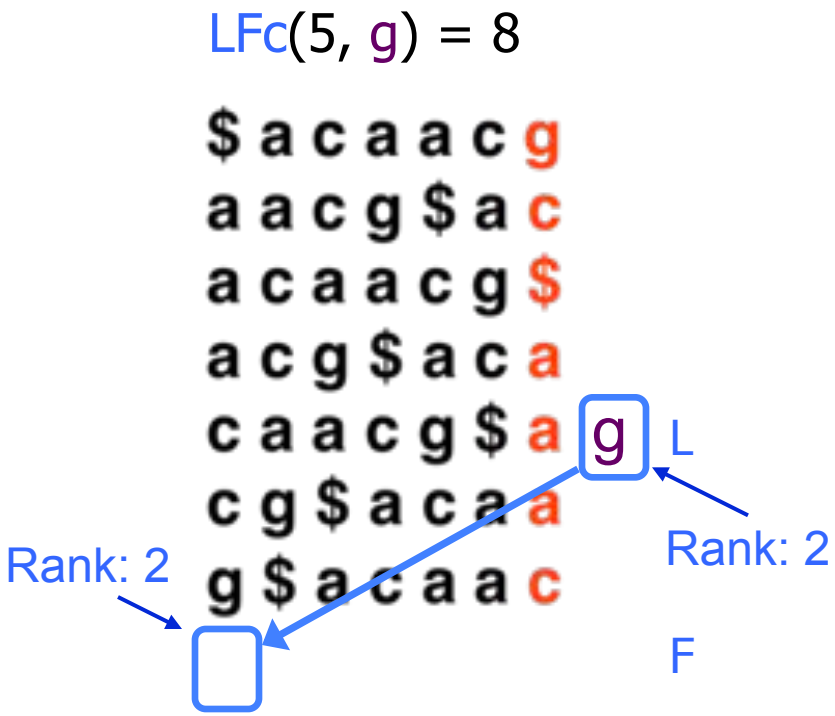
Assembly is a hierarchical, starting from individual reads, build high confidence contigs/unitigs, incorporate the mates to build scaffolds

- Extensive error correction is the key to getting the best assembly possible from a given data set

Supplemental

BWT Exact Matching

- **LFc**(r, c) does the same thing as **LF**(r) but it ignores r's actual final character and "pretends" it's c:



BWT Exact Matching

- Start with a range, (**top**, **bot**) encompassing all rows and repeatedly apply **LFc**:

  top = **LFc**(**top**, **qc**); **bot** = **LFc**(**bot**, **qc**)

  qc = the next character to the left in the query

Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*

BWT Exact Matching

- If range becomes empty (**top** = **bot**) the query suffix (and therefore the query as a whole) does not occur