Assembling Genomes

BCH364C/391L Systems Biology / Bioinformatics – Spring 2015

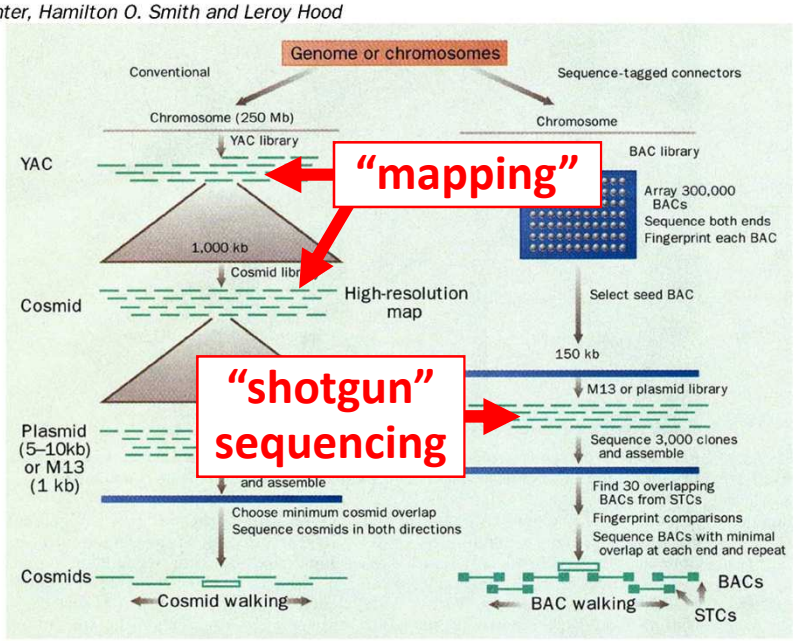
Edward Marcotte, Univ of Texas at Austin

http://www.triazzle.com; The image from http://www.dangilbert.com/port_fun.html

Reference: Jones NC, Pevzner PA, Introduction to Bioinformatics Algorithms, MIT press

A new strategy for genome sequencing

J. Craig Venter, Hamilton O. Smith and Leroy Hood



NATURE · VOL 381 · 30 MAY 1996

(Translating the cloning jargon)

CLONE LIBRARIES USED FOR GENOME MAPPING AND SEQUENCING

Vector	Human-DNA insert size range	Number of clones required to cover the human genome
Yeast artificial chromosome (YAC)	100–2,000 kb	3,000 (1,000 kb)
Bacterial artificial chromosome (BAC)	80–350 kb	20,000 (150 kb)
Cosmid	30–45 kb	75,000 (40 kb)
Plasmid	3–10 kb	600,000 (5 kb)
M13 phage	1 kb	3,000,000 (1 kb)

NATURE · VOL 381 · 30 MAY 1996

Thinking about the basic shotgun concept

- Start with a very large set of random sequencing reads
- How might we match up the overlapping sequences?
- How can we assemble the overlapping reads together in order to derive the genome?

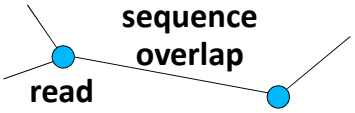
Thinking about the basic shotgun concept

- At a high level, the first genomes were sequenced by comparing pairs of reads to find overlapping reads
- Then, building a graph (*i.e.*, a network) to represent those relationships
- The genome sequence is a “walk” across that graph

The "Overlap-Layout-Consensus" method

Overlap: Compare all pairs of reads
(allow some low level of mismatches)

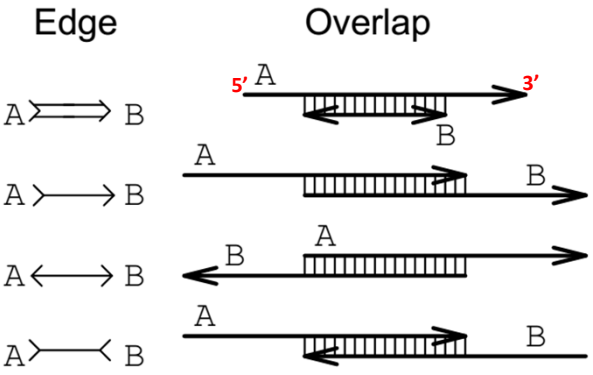
Layout: Construct a graph describing the overlaps



Simplify the graph
Find the simplest path through the graph

Consensus: Reconcile errors among reads along that path to find the consensus sequence

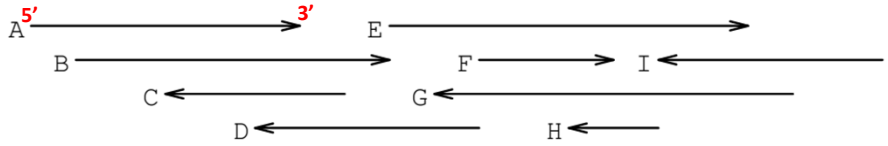
Building an overlap graph



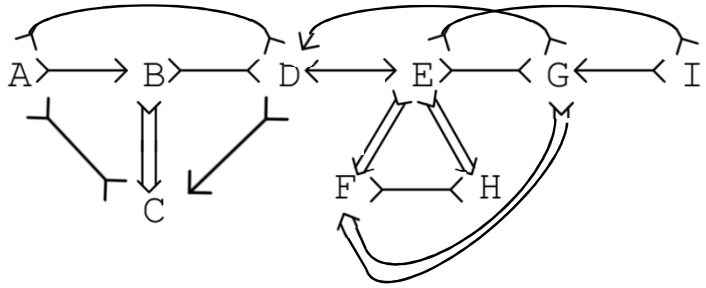
EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290

Building an overlap graph

Reads

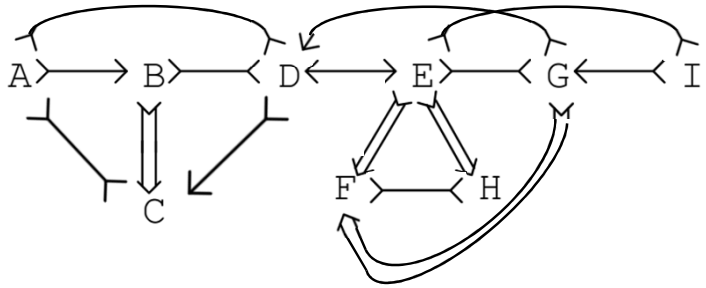


Overlap graph



EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

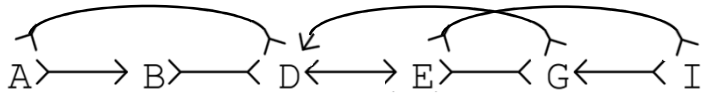
Simplifying an overlap graph



1. Remove all contained nodes & edges going to them

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

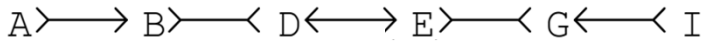
Simplifying an overlap graph



2. Transitive edge removal:
   Given A – B – C and A – C , remove A – C

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

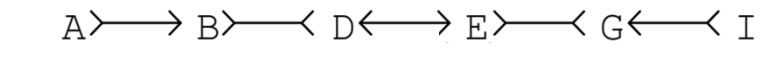
Simplifying an overlap graph

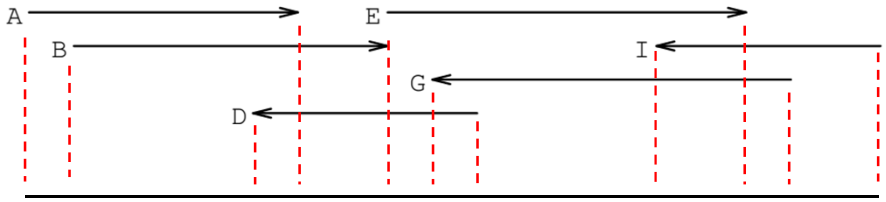


3. If un-branched, calculate consensus sequence
   If branched, assemble un-branched bits and then decide how they fit together

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

Simplifying an overlap graph



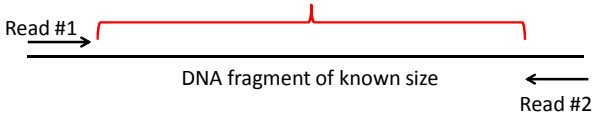


"contig" (assembled contiguous sequence)

EUGENE W. MYERS. *Journal of Computational Biology*. Summer 1995, 2(2): 275-290 (more or less)

This basic strategy was used for most of the early genomes. Also useful: "mate pairs"

2 reads separated by a known distance

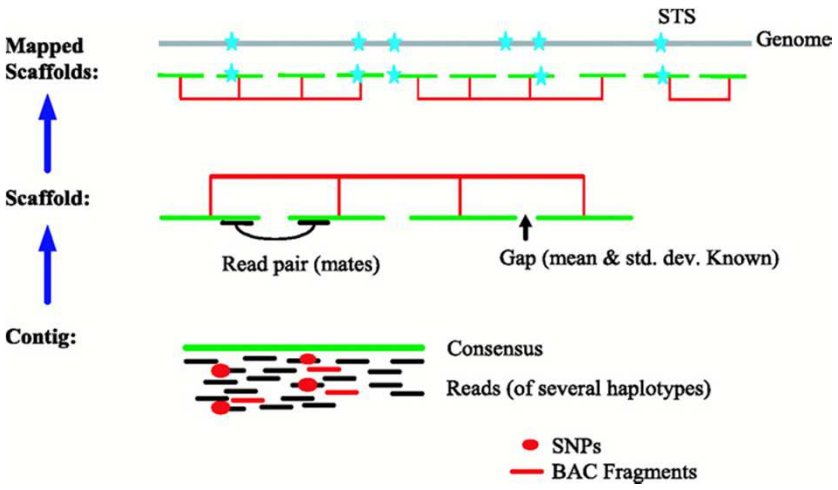


Contigs can be ordered using these paired reads

GigAssembler (used to assemble the public human genome project sequence)

Jim Kent David Haussler

Whole genome Assembly: big picture



http://www.nature.com/scitable/content/anatomy-of-whole-genome-assembly-20429

GigAssembler – Preprocessing

1. Decontaminating & Repeat Masking.
2. Aligning of mRNAs, ESTs, BAC ends & paired reads against initial sequence contigs.
   - psLayout → BLAT
3. Creating an input directory (folder) structure.

```
chr1/
chr1/contig1.e
chr1/contig1.a
chr1/contig1.c
chr1/contig1.b
chr1/contig1.d
chr3/
chr2/
chr2/contig2.d
chr2/contig2.b
chr2/contig2.a
chr2/contig2.c
```

RepBase + RepeatMasker

```
taejoon@fourierseq:~/RepBase/RepBase15.05.fasta$ ls -a
.           dcotrep.ref  mamsub.ref   rodsub.ref
..          diarep.ref   mcotrep.ref  simple.ref
angrep.ref  drorep.ref   mousub.ref   spurep.ref
appendix    fngrep.ref   nemrep.ref   synrep.ref
athrep.ref  fugrep.ref   oryrep.ref   tmplanrep.ref
bctrep.ref  grasrep.ref  plnrep.ref   tmpnemrep.ref
cbrrep.ref  humrep.ref   prirep.ref   tmpxenrep.ref
celrep.ref  humsub.ref   prisub.ref   version
chlrep.ref  invrep.ref   pseudo.ref   vrtrep.ref
cinrep.ref  invsub.ref   ratsub.ref   zebrep.ref
cinunc.ref  mamrep.ref   rodrep.ref
```

```
>MER51D ERV1  Homo sapiens
tgaggcaggagaaaatagcagagggaattggaagttggataaagggagaatgagtaaaagcangagagca
gaagcaaggtaaagaggcgggtgagcaagaagcaagataagaagcagaagttgagcagccaaaacaaaag
taagatnanaaagaagtgagtaaggagcccacatggctggctagatccagaccaaaccagtaaggggcag
ctcctcagagatgggcatgtacattagagagaaaaagtatccttaaaatgaccccgtatgataatcagct
cattaaagctcatgcatatggactgcatatcatgcatgtacttaaaattatgggatggaggtgacgcgca
agawgtcacaagcacacaggggccatagkattaagtaactaagcaacccacctatcaatcaaaaggcaga
tgctggctagagattaggcagccttgggaagagaagaaaaaaaaaacacataaaaagacccaaagtacac
caaactgacgctgatctcatttcgcagaggtcagccccactctcccctctctgagagtgtaatactgtgct
taataaacttttgctgctttgctatctgtgtgtgtcttgtccaattctttgtttgggacaccaagagcct
ggaactgcacrgcaccakctggtaaca
>MIRb SINE2/tRNA  Mammalia
cagaggggcagcgtggtgcagtggaaagagcacgggctttggagtcaggcagacctgggttcgaatcctg
gctctgccacttactagctgtgtgaccttgggcaagtcacttaacctctctgagcctcagtttcctcatc
tgtaaaatggggataataatacctacctcgcagggttgttgtgaggattaaatgagataatgcatgtaaa
gcgcttagcacagtgcctggcacacagtaagcgctcaataaatggtagctctattatt
>LTR45  ERV1  Homo sapiens
tgtaaccgcgggaccagcccaaactgggcctactctgttgataacaaaatgtcaagttaccttgtaggta
taacagagcccaaaactgcaagtcatgtagcccgggcatgtgcaatagaaaaagctttgacctctaacaa
cacccagaaccaatgattcctcccctcggaaccaagaagaccgggacatgaccggaacctgaatgccgga
actctttcagaagcaaaggggtccgttggcccggaagatctggggctaaaatctgcctcaacatacctta
ccgtaaatggtcaaatttgaagccctccaatcagaccctgccaagccaacattcctaaatccttttccctt
gccctctgatcccttaaaacttgccccagaccccaaatcgggggagacagatttgagcccacctcctgtct
ccttgctggccggttttgcaataaagcctttctttttctcaaaagctggtgccatagttattggcttctgt
gtgcatcaggcagcaagcccatttgctcgataaca
>MER80B hAT Homo sapiens
cagggcttcttaaccagaggtccatggatgggcttcaggaggtctgtgaaccctctgaaattatatacaca
aaatgttgtgtatatgtgcatatatgtatttttctggggagagggttcatagctttcatcagattctcaa
aggggtctatgatctmaaaaaggttaagaagccctg
```

GigAssembler: Build merged sequence contigs (“rafts”)

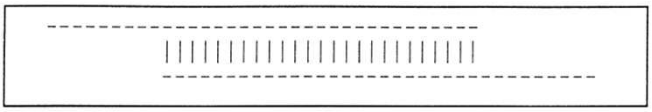
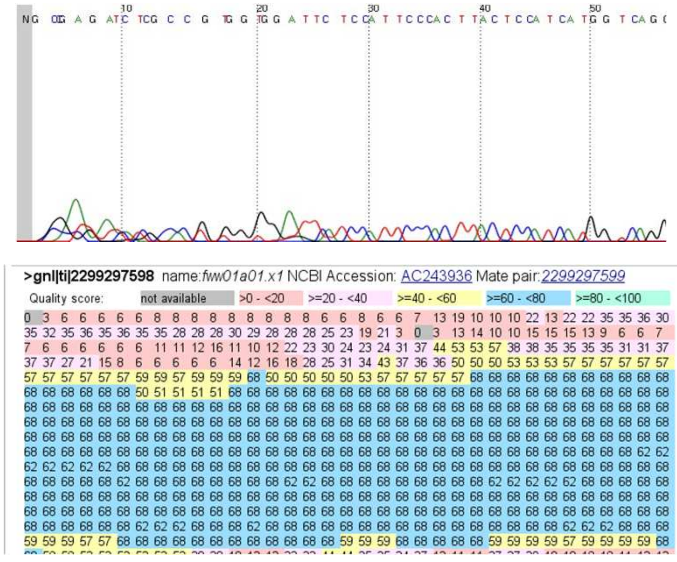


Figure 1 Two sequences overlapping end to end. The sequences are represented as dashes. The aligning regions are joined by vertical bars. End-to-end overlap is an extremely strong indication that two sequences should be joined into a contig.

Sequencing quality (Phred Score)



Sequencing quality (Phred Score)

$$Q = -10 \log_{10} P$$

Base-calling Error Probability

or

$$P = 10^{\frac{-Q}{10}}$$

Phred quality scores are logarithmically linked to error probabilities

Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90 %
20	1 in 100	99 %
30	1 in 1000	99.9 %
40	1 in 10000	99.99 %
50	1 in 100000	99.999 %

http://en.wikipedia.org/wiki/Phred_quality_score

GigAssembler: Build merged sequence contigs ("rafts")

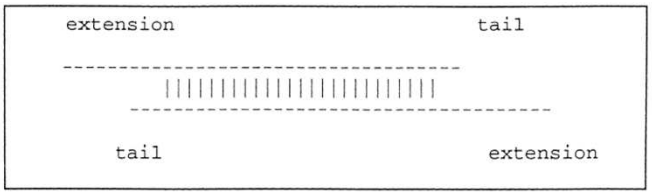


Figure 2 Two sequences with tails. The nonaligning regions on either side can be classified into 'extensions' and 'tails.' Short tails are fairly common even when two sequences should be joined into a contig because of poor quality sequence near the ends and occasional chimeric reads. Long tails, however, are generally a sign that the alignment is merely due to the sequences sharing a repeating element.

GigAssembler: Build merged sequence contigs (“rafts”)

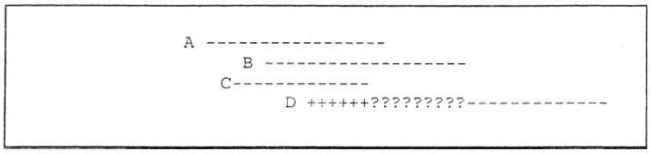


Figure 3 Merging into a raft. A contig (‘raft’) of three sequences: A, B, and C has already been constructed by `GigAssembler`. The program now examines an alignment between sequence C and a new sequence, D, to see whether D should also be added to the raft. The parts of D marked with +s are compatible with the raft because of the C/D alignment. The program must also check that the parts of D marked with ?s are compatable with the raft by examining other alignments.

GigAssembler: Build sequenced clone contigs (“barges”)

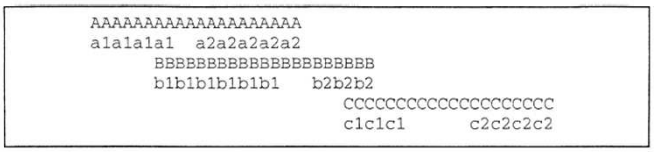


Figure 4 Three overlapping draft clones: A, B, and C. Each clone has two initial sequence contigs. Note that initial sequence contigs a1, b1, and a2 overlap as do b2 and c1.

GigAssembler: Build a “raft-ordering” graph

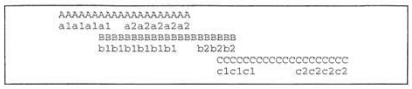


Figure 4 Three overlapping draft clones: A, B, and C. Each clone has two initial sequence contigs. Note that initial sequence contigs a1, b1, and a2 overlap as do b2 and c1.

Figure 5 Ordering graph of clone starts and ends. This represents the same clones as in Fig. 4. (As) The start of clone A; (Ae) the end of clone A. Similarly Bs, Be, Cs, and Ce represent the starts and ends of clones B and C.

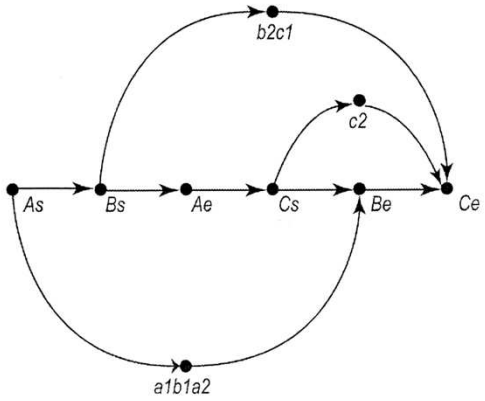


Figure 6 Ordering graph after adding in rafts. The initial sequence contigs shown in Fig. 4 are merged into rafts where they overlap. This forms three rafts: a1b1a2, b2c1, and c2. These rafts are constrained to lie between the relevant clone ends by the addition of additional ordering edges to the graph shown in Fig. 5.

GigAssembler: Build a “raft-ordering” graph

- Add information from mRNAs, ESTs, paired plasmid reads, BAC end pairs: building a “bridge”
  - Different weight to different data type: (mRNA ~ highest)
  - Conflicts with the graph as constructed so far are rejected.
- Build a sequence path through each raft.
- Fill the gap with N’s.
  - 100: between rafts
  - 50,000: between bridged barges

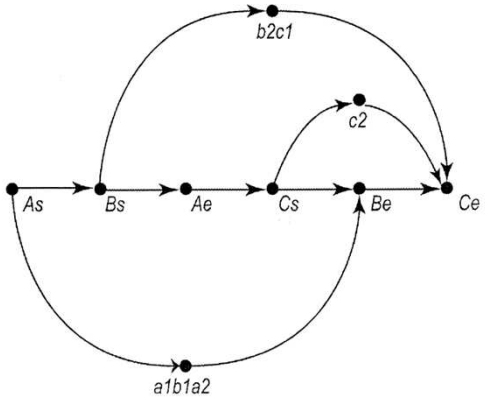


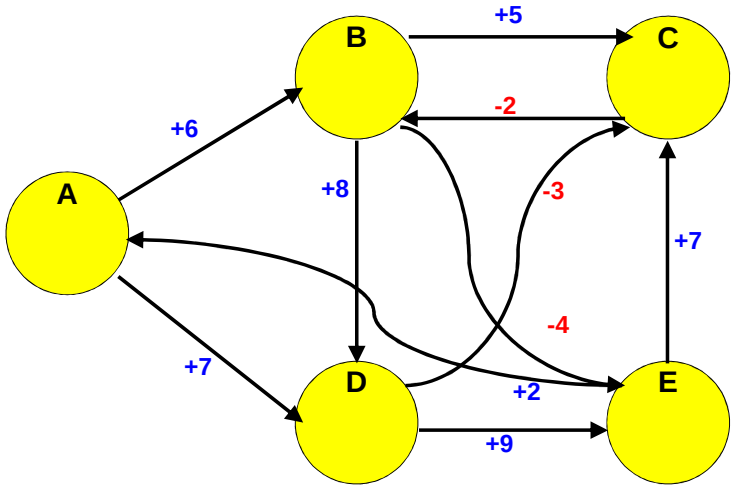
Figure 6 Ordering graph after adding in rafts. The initial sequence contigs shown in Fig. 4 are merged into rafts where they overlap. This forms three rafts: a1b1a2, b2c1, and c2. These rafts are constrained to lie between the relevant clone ends by the addition of additional ordering edges to the graph shown in Fig. 5.

Finding the shortest path across the ordering graph using the Bellman-Ford algorithm

http://compprog.wordpress.com/2007/11/29/one-source-shortest-path-the-bellman-ford-algorithm/

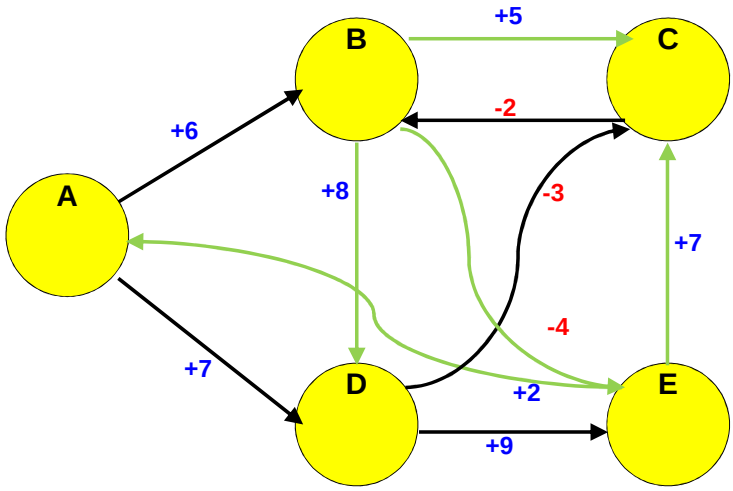
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)



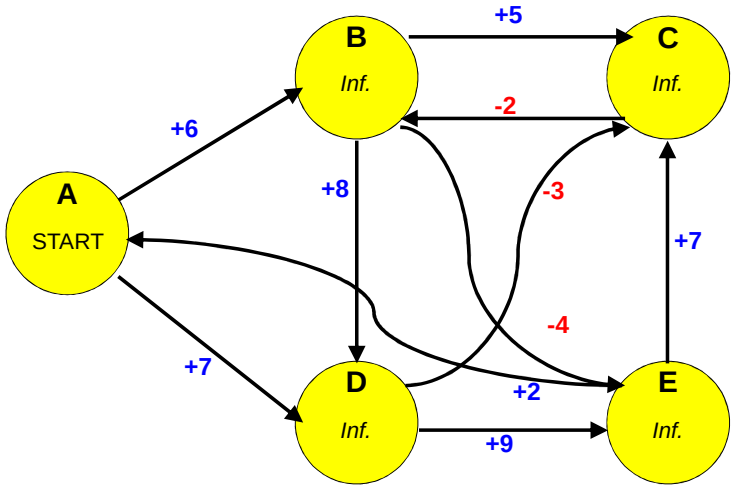
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)



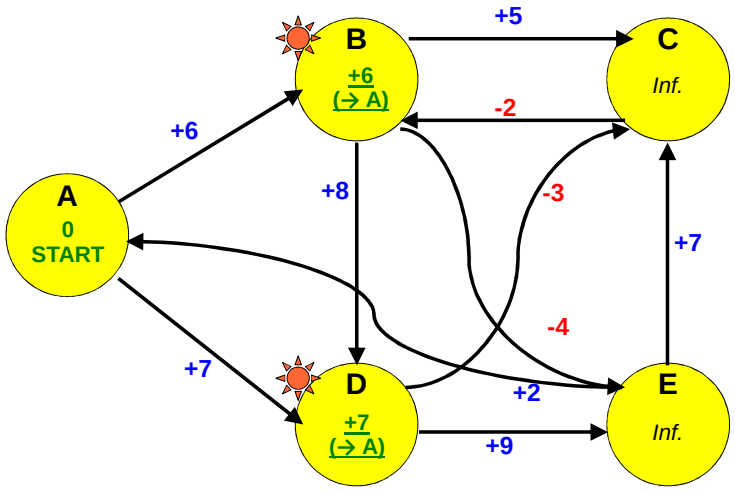
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)



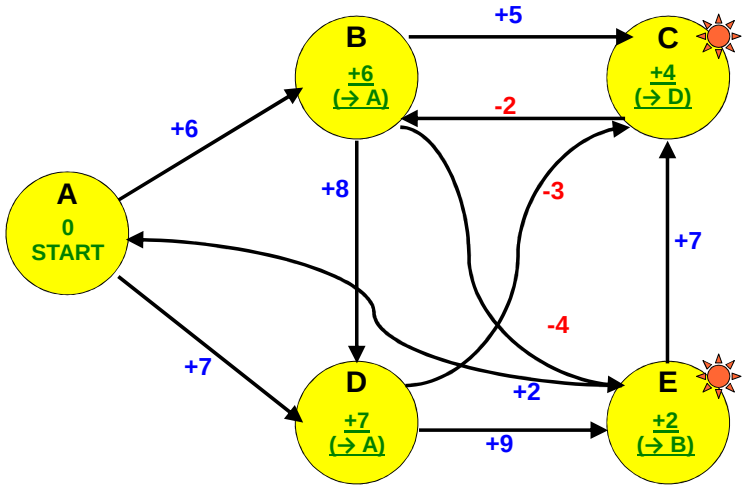
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)



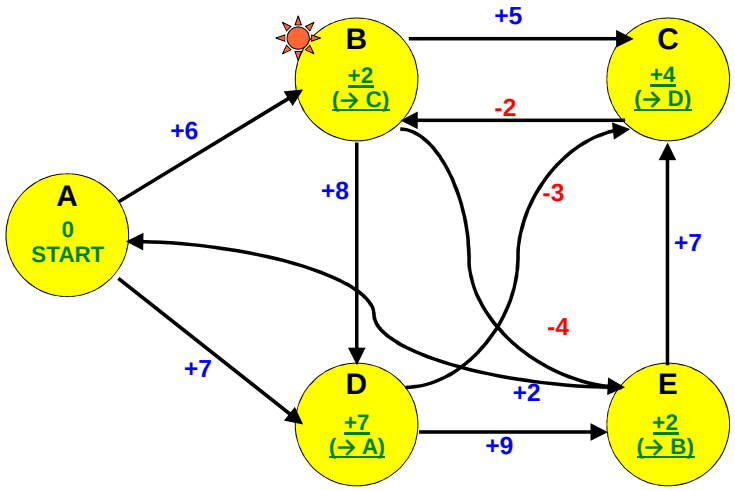
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)



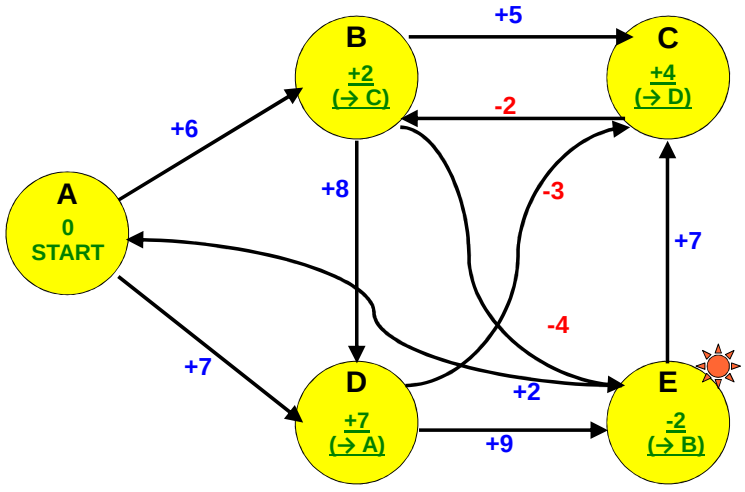
Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)

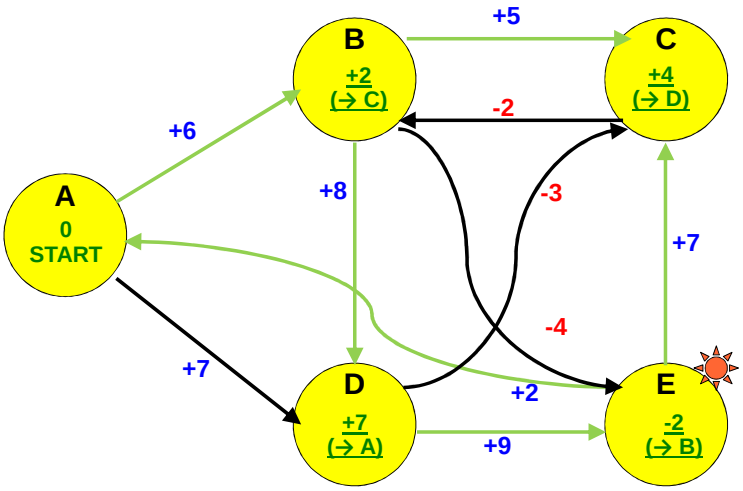


Find the shortest path to all nodes.

Take every edge and try to relax it (N – 1 times where N is the count of nodes)

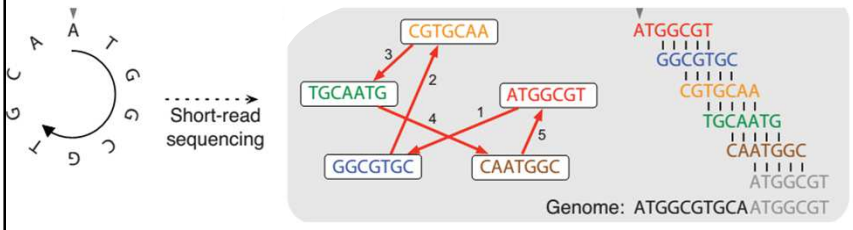


Answer: A-D-C-B-E



Modern assemblers now work a bit differently, using so-called DeBruijn graphs:

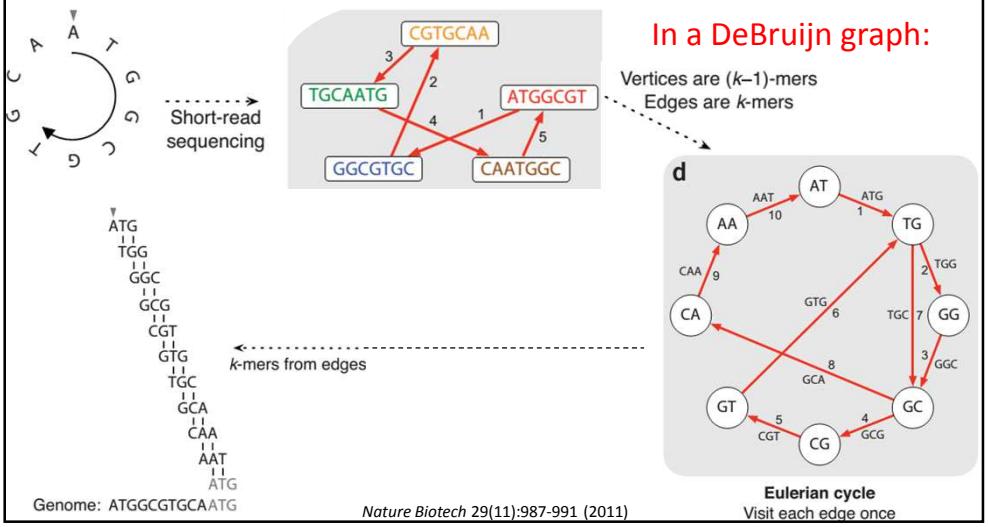
Here's what we saw before:



In Overlap-Layout-Consensus:
Nodes are reads
Edges are overlaps

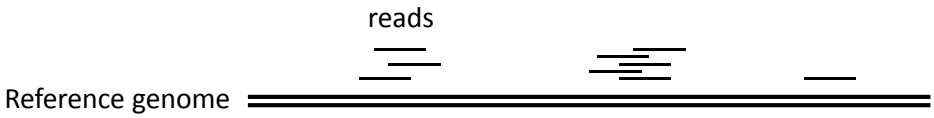
Nature Biotech 29(11):987-991 (2011)

Modern assemblers now work a bit differently, using so-called DeBruijn graphs:



Nature Biotech 29(11):987-991 (2011)

Once a reference genome is assembled, new sequencing data can 'simply' be mapped to the reference.



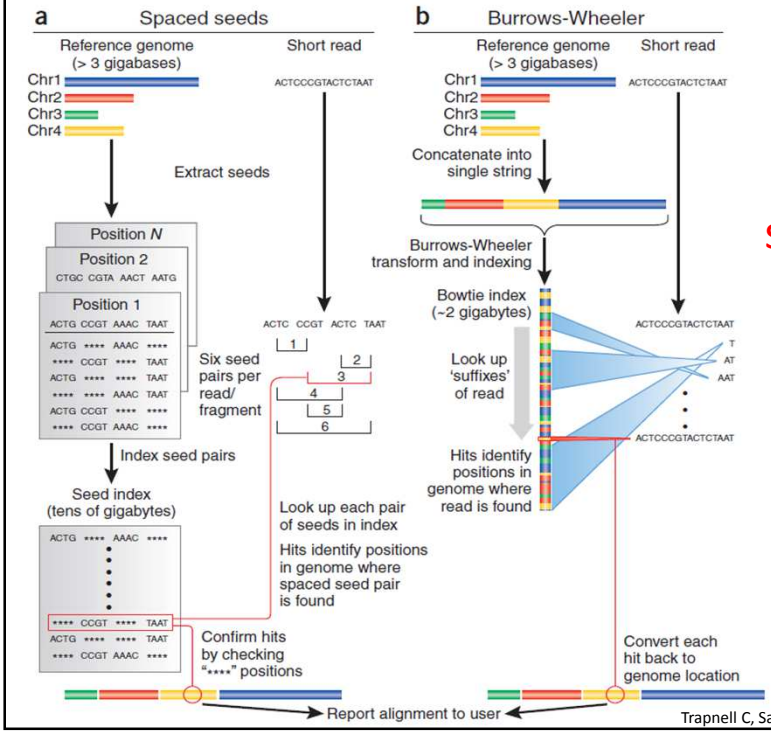
Mapping reads to assembled genomes

Table 1 A selection of short-read analysis software

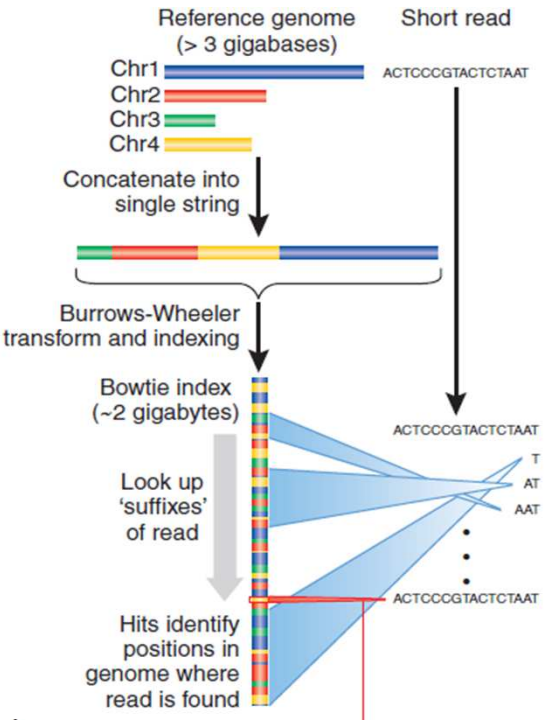
Program	Website	Open source?	Handles ABI color space?	Maximum read length
Bowtie	http://bowtie.cbcb.umd.edu	Yes	No	None
BWA	http://maq.sourceforge.net/bwa-man.shtml	Yes	Yes	None
Maq	http://maq.sourceforge.net	Yes	Yes	127
Mosaik	http://bioinformatics.bc.edu/marthlab/Mosaik	No	Yes	None
Novoalign	http://www.novocraft.com	No	No	None
SOAP2	http://soap.genomics.org.cn	No	No	60
ZOOM	http://www.bioinfor.com	No	Yes	240

Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009

Mapping strategies



Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009



Burroughs Wheeler indexing

Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009

Burroughs-Wheeler transform indexing

BWT is often used for file compression (like bzip2),
here used to make a fast 'lookup' index in a genome

BWT = 'reversible block-sorting'

Input	SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES
	↓ **Forward BWT** (This sequence is more compressible)
Output	TEXYDST.E.IXIXIXXSSMPPS.B..E.S.EUSFXDIIOIIIT
	↓ **Reverse BWT**
Recovered input	SIX.MIXED.PIXIES.SIFT.SIXTY.PIXIE.DUST.BOXES

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Input
^BANANA|

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

All Rotations
^BANANA|
|^BANANA
A|^BANAN
NA|^BANA
ANA|^BAN
NANA|^BA
ANANA|^B
BANANA|^

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Sorting All Rows in Alphabetical Order

```
ANANA|^B
ANA|^BAN
A|^BANAN
BANANA|^
NANA|^BA
NA|^BANA
^BANANA|
|^BANANA
```

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Taking Last Column

```
ANANA|^B
ANA|^BAN
A|^BANAN
BANANA|^
NANA|^BA
NA|^BANA
^BANANA|
|^BANANA
```

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Output Last Column
BNN^AA|A

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Transformation				
Input	All Rotations	Sorting All Rows in Alphabetical Order	Taking Last Column	Output Last Column
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	**A**NANA|^B **A**NA|^BAN **A**|^BANAN **B**ANANA|^ **N**ANA|^BA **N**A|^BANA ^BANANA| |^BANANA	ANANA|^**B** ANA|^BA**N** A|^BANA**N** BANANA|**^** NANA|^B**A** NA|^BAN**A** ^BANANA| |^BANAN**A**	BNN^AA|A

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

BWT is remarkable because it is *reversible.*

Any ideas as how you might reverse it?

Burroughs-Wheeler transform indexing

Input
BNN^AA|A

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Add 1	Sort 1	Add 2	Sort 2
B	A	BA	AN
N	A	NA	AN
N	A	NA	A|
^	B	^B	BA
A	N	AN	NA
A	N	AN	NA
|	^	|^	^B
A	|	A|	|^
Write the sequence as the last column	Sort it…	Add the columns…	Sort those…

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Add 3	Sort 3	Add 4	Sort 4
BAN	ANA	BANA	ANAN
NAN	ANA	NANA	ANA|
NA|	A|^	NA|^	A|^B
^BA	BAN	^BAN	BANA
ANA	NAN	ANAN	NANA
ANA	NA|	ANA|	NA|^
|^B	^BA	|^BA	^BAN
A|^	|^B	A|^B	|^BA
Add the columns…	Sort those…	Add the columns…	Sort those…

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Add 5	Sort 5	Add 6	Sort 6
BANAN NANA| NA|^B ^BANA ANANA ANA|^ |^BAN A|^BA	ANANA ANA|^ A|^BA BANAN NANA| NA|^B ^BANA |^BAN	BANANA NANA|^ NA|^BA ^BANAN ANANA| ANA|^B |^BANA A|^BAN	ANANA| ANA|^B A|^BAN BANANA NANA|^ NA|^BA ^BANAN |^BANA
Add the columns…	Sort those…	Add the columns…	Sort those…

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs-Wheeler transform indexing

Add 7	Sort 7	Add 8
BANANA| NANA|^B NA|^BAN ^BANANA ANANA|^ ANA|^BA |^BANAN A|^BANA	ANANA|^ ANA|^BA A|^BANA BANANA| NANA|^B NA|^BAN ^BANANA |^BANAN	BANANA|^ NANA|^BA NA|^BANA ^BANANA| ANANA|^B ANA|^BAN |^BANANA A|^BANAN
Add the columns…	Sort those…	Add the columns…

The row with the "end of file" character at the end is the original text

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

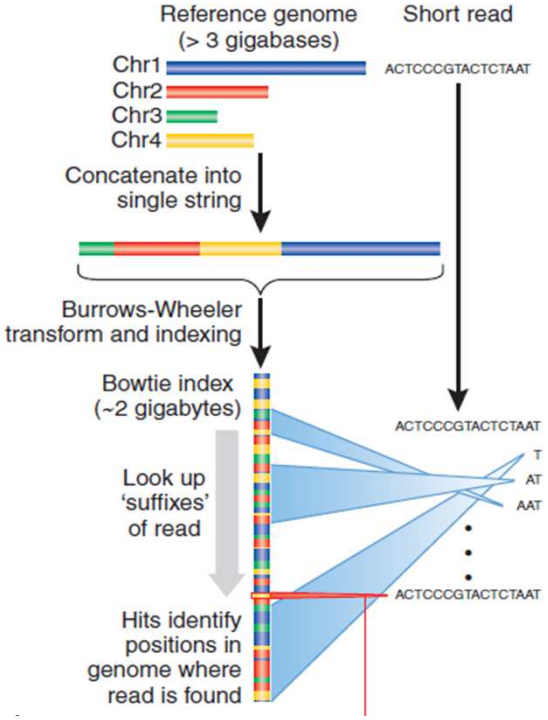
Burroughs-Wheeler transform indexing

Output
^BANANA|

The row with the "end of file" character at the end is the original text

http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Burroughs Wheeler indexing



Trapnell C, Salzberg SL, *Nat. Biotech.*, 2009