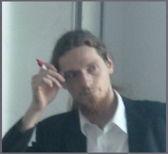
Lecture:
Algorithmic Bioinformatics

Doctoral School, Université Dauphine, 2022

Lecture Overview

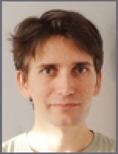
Mathias Weller
mathias.weller@u-pem.fr



- Introduction
  - Molecular Biology
  - Sequencing
  - Assembly
  - Scaffolding
- Phylogenetics

  ...

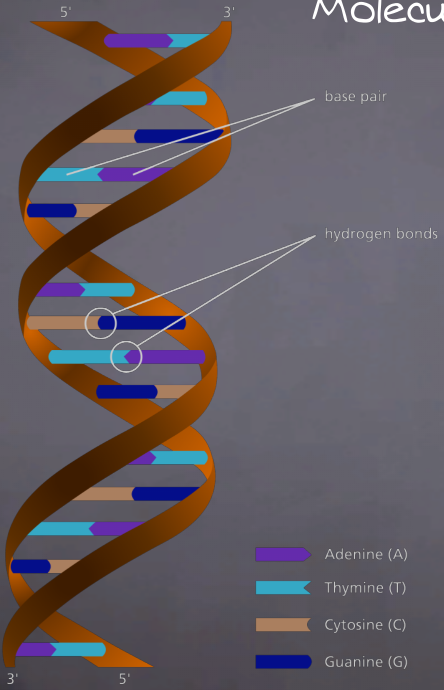
Laurent Bulteau
laurent.bulteau@u-pem.fr



- Genome Rearrangements

  ...
- Scaffold Filling

Molecular Biology Basics



DNA

- double strand
- nucleotides paired: A–T, C–G
- inside nucleus (eucaryotes)

Molecular Biology Basics

DNA

- double strand
- nucleotides paired: A–T, C–G
- inside nucleus (eucaryotes)

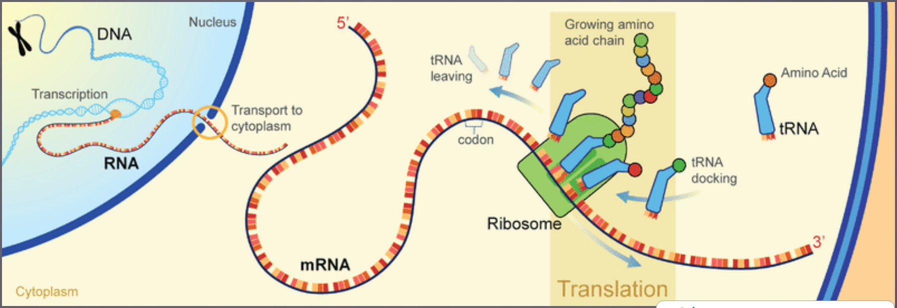
RNA

- single strand
- transported outside nucleus
- translated into actual proteins
- Thymine (T) → Uracil (U)

Molecular Biology Basics

Transcription & Translation

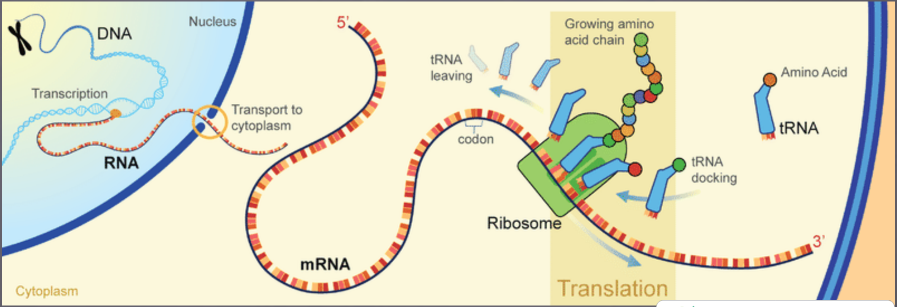
DNA → RNA & RNA → protein



Molecular Biology Basics

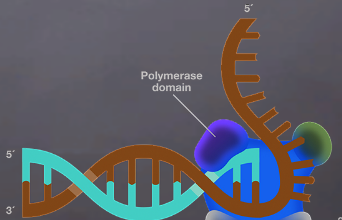
Transcription & Translation

DNA → RNA & RNA → protein



Polymerase

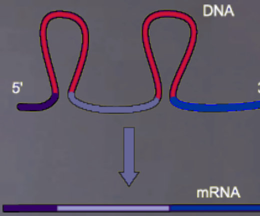
single strand → double strand



Introns & Exons

parts of DNA cut out when forming mRNA ("splicing")

- removed ⇝ "intron"
- not removed ⇝ "exon"

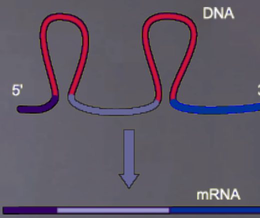


Molecular Biology Basics

Introns & Exons

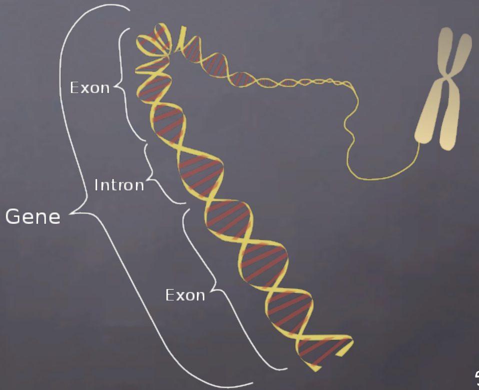
parts of DNA cut out when forming mRNA ("splicing")

- removed ⇝ "intron"
- not removed ⇝ "exon"



Gene

Gene = START...STOP
(including introns)
(10^3–10^5 bp)

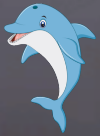


Molecular Biology Basics

Chromosomes

- haploid = 1 set of chromosomes
- diploid = 2 sets of chromosomes (usually one from each parent)
- ... ("polyploid")

- procaryotes $\rightsquigarrow$ one (circular) chromosome, haploid
- eucaryotes $\rightsquigarrow$ set of (linear) chromosomes, polyploid



44

90

16

46

46

Mutation

Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...

Mutation

..AATCGCTAA..
..AATCCTAA..

Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion,

Mutation

```
..AATCCTAA..
..AATCGCTAA..
```

Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion,

Mutation

..AATCGCTAA..
..AATCACTAA..

Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion, substitution

Mutation

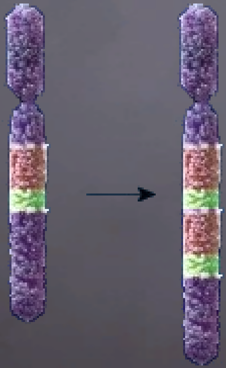
Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion, substitution

Replication Error

- DNA rearrangement
- caused by errors in Meiosis/Mitosis

Mutation



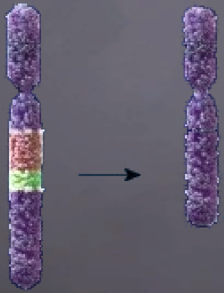
Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion, substitution

Replication Error

- DNA rearrangement
- caused by errors in Meiosis/Mitosis
- duplication,

Mutation



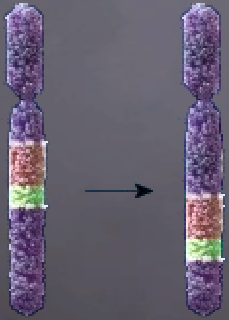
Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion, substitution

Replication Error

- DNA rearrangement
- caused by errors in Meiosis/Mitosis
- duplication, deletion (loss),

Mutation



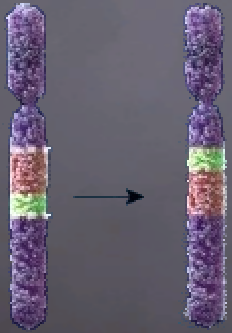
Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion, substitution

Replication Error

- DNA rearrangement
- caused by errors in Meiosis/Mitosis
- duplication, deletion (loss), translocation,

Mutation



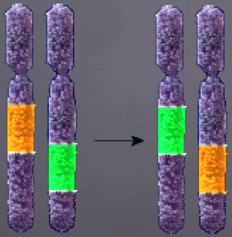
Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion, substitution

Replication Error

- DNA rearrangement
- caused by errors in Meiosis/Mitosis
- duplication, deletion (loss), translocation, inversion,

Mutation



Single-Nucleotide Polymorphism

- DNA damage
- caused by radioactivity, UV light, ...
- insertion, deletion, substitution

Replication Error

- DNA rearrangement
- caused by errors in Meiosis/Mitosis
- duplication, deletion (loss), translocation, inversion, crossover

Sanger Sequencing

[Sanger et al '77]

```
CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGACCTGCCCAGTCTGTACTGTCACCGGGGTTCTAAGTGTTCTAGCATAGAGTTATGTCATTTGCTCGTTA
```

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")

Sanger Sequencing

[Sanger et al '77]

```
CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGACCTGCCCAGTCTGTACTGTCACCGGGGTTCTAAGTGTTCTAGCATAGAGTTATGTCATTTGCTCGTTA
```

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix
3. add polymerase & floating bases: A C G T
4. add a special base: A* (polymerase cannot extend)

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix
3. add polymerase & floating bases: A C G T
4. add a special base: A* (polymerase cannot extend)
5. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGA*

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix
3. add polymerase & floating bases: A C G T
4. add a special base: A* (polymerase cannot extend)
5. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGA*
GGACCTGCCCA*

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix
3. add polymerase & floating bases: A C G T
4. add a special base: A* (polymerase cannot extend)
5. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

```
CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGA*
GGACCTGCCCA*
GGACCTGCCCAGTCTGTA*
```

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix
3. add polymerase & floating bases: A C G T
4. add a special base: A* (polymerase cannot extend)
5. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

```
CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGA*
GGACCTGCCCA*
GGACCTGCCCAGTCTGTA*
```

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix
3. add polymerase & floating bases: A C G T
4. add a special base: A* (polymerase cannot extend)
5. stir & let polymerase act
6. measure the length of each fragment
   ⇝ each length is the position of a T in the template

Sanger Sequencing

[Sanger et al '77]

```
CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGA*
GGACCTGCCCA*
GGACCTGCCCAGTCTGTA*
```

Sanger Sequencing

1. make thousands of copies of target ("amplified genome")
2. split their helix
3. add polymerase & floating bases: A C G T
4. add a special base: A* (polymerase cannot extend)
5. stir & let polymerase act
6. measure the length of each fragment
   ⇝ each length is the position of a T in the template

Problems

- frequency of longer reads decreases drastically
- length-estimate unreliable after a couple hundred bp
  ⇝ chop DNA into pieces and read those
- repeated bases unreliable

Next Generation Sequencing (illumina)

Next Generation Sequencing (illumina)

ACTCA......ACCTC

Preparation

1. chop DNA into smaller pieces (approximate size known)

Next Generation Sequencing (illumina)

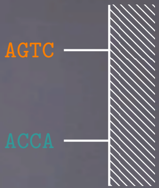
TGGTACTCA......ACCTCTCAG

Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece

Next Generation Sequencing (illumina)

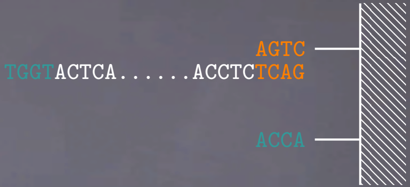
TGGTACTCA......ACCTCTCAG



Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Next Generation Sequencing (illumina)



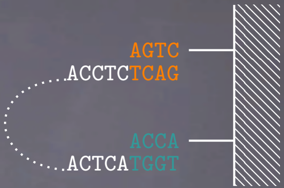
Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Amplification

1. strand anchors its two ends to two anchor places

Next Generation Sequencing (illumina)



Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Amplification

1. strand anchors its two ends to two anchor places

Next Generation Sequencing (illumina)

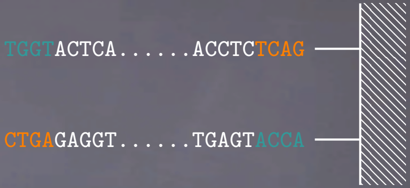
Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Amplification

1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand

Next Generation Sequencing (illumina)



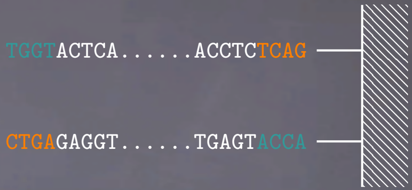
Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Amplification

1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand
3. double strand is cut into single strands

Next Generation Sequencing (illumina)



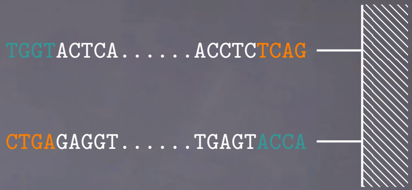
Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Amplification

1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand
3. double strand is cut into single strands
4. rinse, repeat (last 3 steps) until flow chip is "full"

Next Generation Sequencing (illumina)



Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

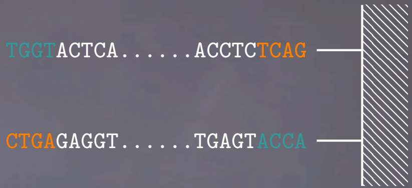
Amplification

1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand
3. double strand is cut into single strands
4. rinse, repeat (last 3 steps) until flow chip is "full"

Sequencing

1. add special (fluorescent, non-extendable) bases + polymerase

Next Generation Sequencing (illumina)



Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

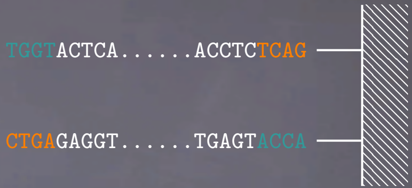
Amplification

1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand
3. double strand is cut into single strands
4. rinse, repeat (last 3 steps) until flow chip is "full"

Sequencing

1. add special (fluorescent, non-extendable) bases + polymerase
2. polymerase attaches one base

Next Generation Sequencing (illumina)



Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

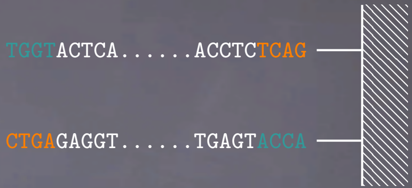
Amplification

1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand
3. double strand is cut into single strands
4. rinse, repeat (last 3 steps) until flow chip is "full"

Sequencing

1. add special (fluorescent, non-extendable) bases + polymerase
2. polymerase attaches one base
3. camera takes picture of the flow cell

Next Generation Sequencing (illumina)



Preparation

1. chop DNA into smaller pieces (approximate size known)
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Amplification

1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand
3. double strand is cut into single strands
4. rinse, repeat (last 3 steps) until flow chip is "full"

Sequencing

1. add special (fluorescent, non-extendable) bases + polymerase
2. polymerase attaches one base
3. camera takes picture of the flow cell
4. rinse, repeat (last 3 steps) until no more bases were added

Next Generation Sequencing (illumina)

"Paired-End reads"

Preparation

1. chop DNA into smaller pieces (approximate size
2. add anchors (and IDs) to each end of each piece
3. "flow cell" containing anchor places

Amplification

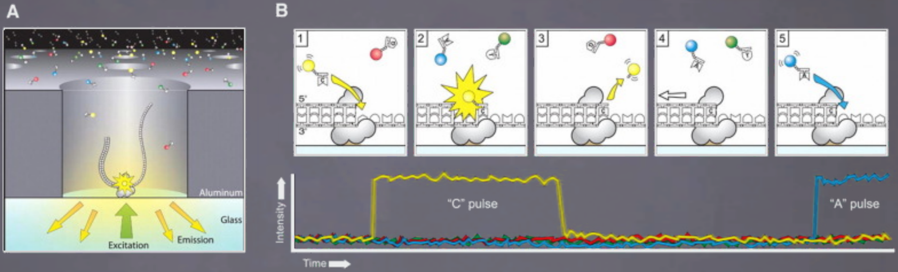
1. strand anchors its two ends to two anchor places
2. polymerase completes the strand into double-strand
3. double strand is cut into single strands
4. rinse, repeat (last 3 steps) until flow chip is "full"

distance between reads = "insert size"

Sequencing

1. add special (fluorescent, non-extendable) bases +
2. polymerase attaches one base
3. camera takes picture of the flow cell
4. rinse, repeat (last 3 steps) until no more bases were added

Third-Gen Sequencing: SMRT

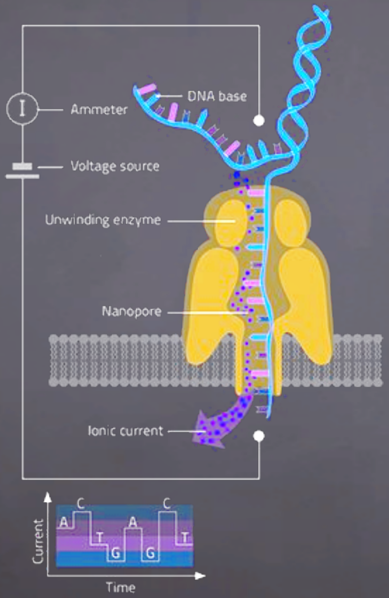


[Rhoads et al, 2015]

Single Molecule Real Time Sequencing

1. fix a polymerase enzyme under a microscope
2. attach fluorescent molecule to each nucleotide
3. polymerase clips off fluorescent molecule when attaching a base
4. observe change in fluorescence ⇝ identify base

Third-Gen Sequencing: PacBio



Nanopores

1. "pore" of diameter 1–20nm
2. only single-strand may pass
3. Base at "bottleneck" hinders current
4. ⇝ "characteristic profile" determines base

Conclusion: Sequencing

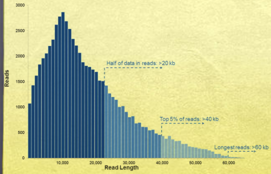
method	read length	% errors	reads/s	$/MBase
Sanger	600–1000	0.001	0.03	500
Illumina HiSeq	2×250	0.1	4000	0.04
SMRT (PacBio)	10^4	13	3.4	0.50
NanoPore (minION)	$5 \cdot 10^3$	38	0.3	11

[Rhoads et al, 2015]

Conclusion: Sequencing

method	read length	% errors	reads/s	$/MBase
Sanger	600–1000	0.001	0.03	500
Illumina HiSeq	2×250	0.1	4000	0.04
SMRT (PacBio)	10^4	13	3.4	0.50
NanoPore (minION)	$5 \cdot 10^3$	38	0.3	11

[Rhoads et al, 2015]



Sequence Assembly: Overview

GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC

Goal: reconstruct sequence
Problem 1: only have (small) reads
Idea: overlap reads to form complete sequence

Sequence Assembly: Overview

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTT                CGACACTCCTTGGGTTTT              CTAGGCCATTGATTGCGGGTC
        ACTTCGC                                        GGTTCTCT                  GGTCCAGGTGCTGTCAACGAC
```

Goal: reconstruct sequence
Problem 1: only have (small) reads
Idea: overlap reads to form complete sequence

Sequence Assembly: Overview

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTT                             CGACACTCCTTGGGTTTT                     CTAGGCCATTGATTGCGGGTC
         ACTTCGC                                                      GGTTCTCT                    GGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTTCGC                          CGACACTCCTTGGGTTTT           GGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
```

Goal: reconstruct sequence
Problem 1: only have (small) reads
Idea: overlap reads to form complete sequence

Sequence Assembly: Overview

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTT                 CGACACTCCTTGGGTTTT               CTAGGCCATTGATTGCGGGTC
        ACTTCGC                                         GGTTCTCT                 GGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTTCGC              CGACACTCCTTGGGTTTT         GGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
```

Goal: reconstruct sequence

Problem 1: only have (small) reads

Idea: overlap reads to form complete sequence

Problem 2: parts of the sequence might not be covered by reads

Sequence Assembly: Overview

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTT                 CGACACTCCTTGGGTTTT         CTAGGCCATTGATTGCGGGTC
        ACTTCGC                                   GGTTCTCT               GGTCCAGGTGCTGTCAACGAC
           TCGCTAGGGTTCTCTAACGA          TTTACGTCGCGG                                     CGAC
```

Goal: reconstruct sequence
Problem 1: only have (small) reads
Idea: overlap reads to form complete sequence

Problem 2: parts of the sequence might not be covered by reads
$\rightsquigarrow$ sequence with "high coverage"

Sequence Assembly: Overview

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTT              CGACACTCCTTGGGTTTT              CTAGGCCATTGATTGCGGGTC
         ACTTCGC                             GGTTCTCT                 GGTCCAGGTGCTGTCAACGAC
            TCGCTAGGGTTCTCTAACGA                 TTTACGTCGCGG                                  CGAC
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
```

Goal: reconstruct sequence
Problem 1: only have (small) reads
Idea: overlap reads to form complete sequence

Problem 2: parts of the sequence might not be covered by reads
$\rightsquigarrow$ sequence with "high coverage"

Sequence Assembly: Overview

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTT                CGACACTCCTTGGGTTTT             CTAGGCCATTGATTGCGGGTC
        ACTTCGC                                      GGTTCTCT               GGTCCAGGTGCTGTCAACGAC
           TCGCTAGGGTTCTCTAACGA            TTTACGTCGCGG                                      CGAC
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
```

Goal: reconstruct sequence
Problem 1: only have (small) reads
Idea: overlap reads to form complete sequence

Problem 3: Shortest Common Superstring is NP-hard
⇝ Heuristic Assembly:

- Overlap-Layout-Consensus
- DeBruijn-Graph

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Naive Overlap

```
    AGGAGTC
GAGTCCA→
```

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Naive Overlap

```
AGGAGTC
  GAGTCCA
```

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Naive Overlap

```
AGGAGTC
  GAGTCCA
```

$\rightsquigarrow$ $O(\#reads^2 \cdot read\text{-}length)$ time worst-case

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇

Overlap-Layout-Consensus Assembly

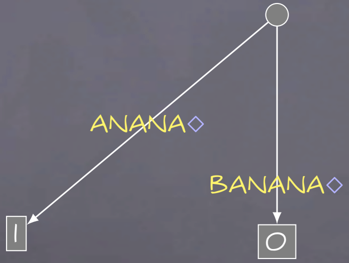
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇



Overlap-Layout-Consensus Assembly

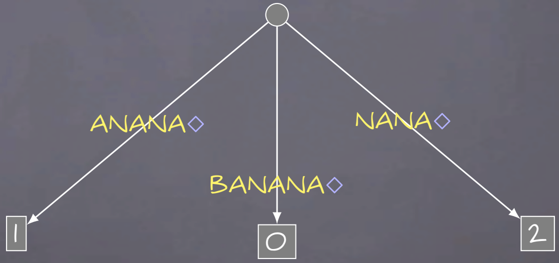
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇



Overlap-Layout-Consensus Assembly

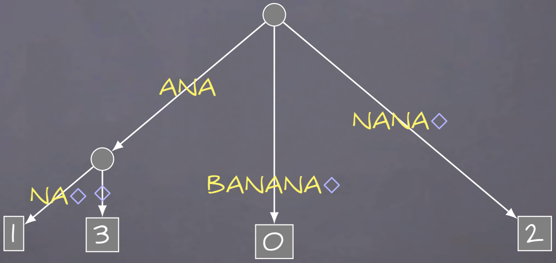
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇



Overlap-Layout-Consensus Assembly

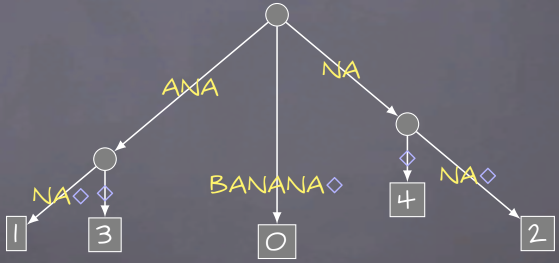
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇



Overlap-Layout-Consensus Assembly

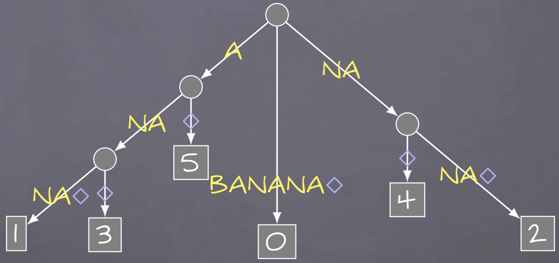
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇



Overlap-Layout-Consensus Assembly

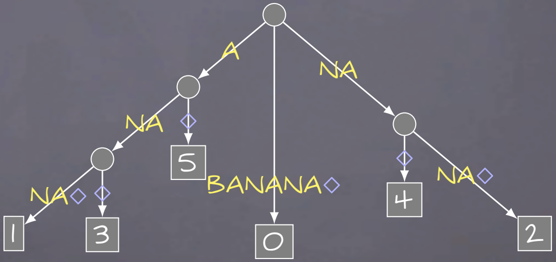
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇



⇝ O(read-length^2) time & space

Overlap-Layout-Consensus Assembly

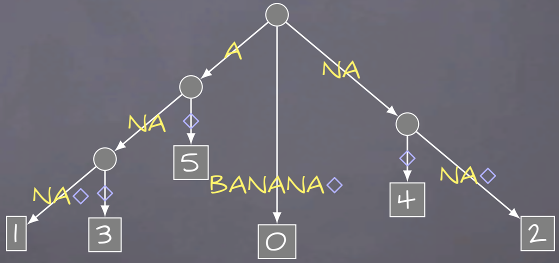
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

example: BANANA◇



⇝ O(read-length^2) time & space

can be improved to linear time & space

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

exercise: joined suffix tree for AGGAGTC◇ and GAGTCCA▷

Overlap-Layout-Consensus Assembly

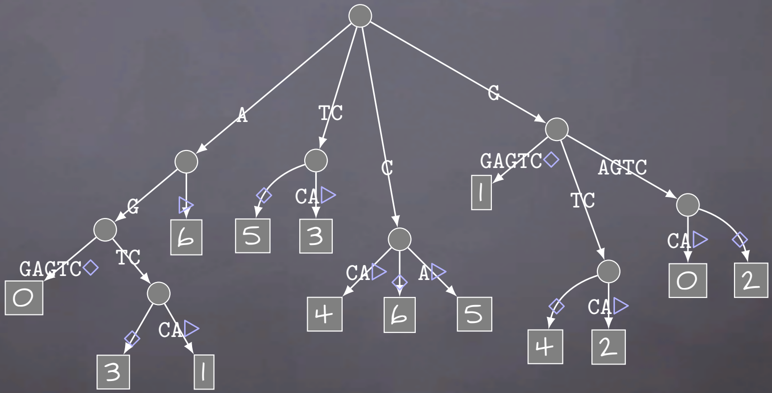
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

exercise: joined suffix tree for AGGAGTC◇ and GAGTCCA▷



Overlap-Layout-Consensus Assembly

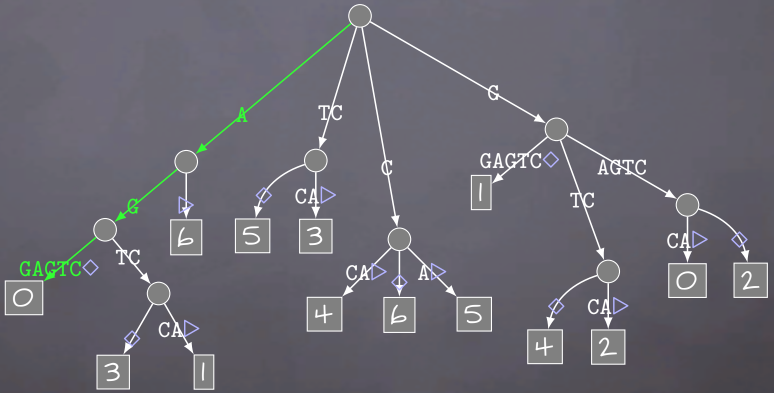
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

exercise: joined suffix tree for AGGAGTC◇ and GAGTCCA▷



Overlap-Layout-Consensus Assembly

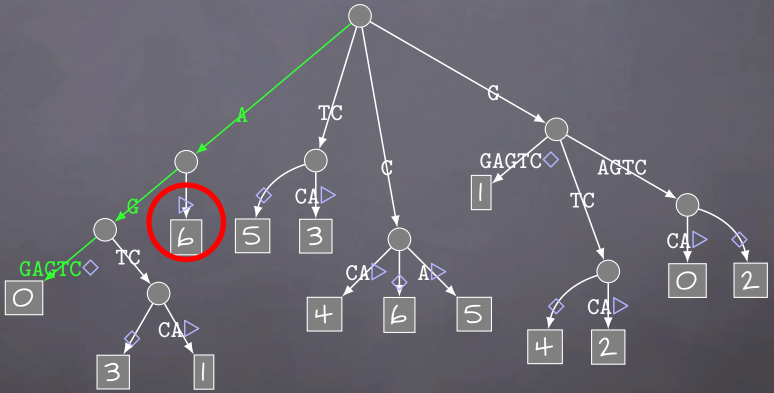
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

exercise: joined suffix tree for AGGAGTC◇ and GAGTCCA▷



Overlap-Layout-Consensus Assembly

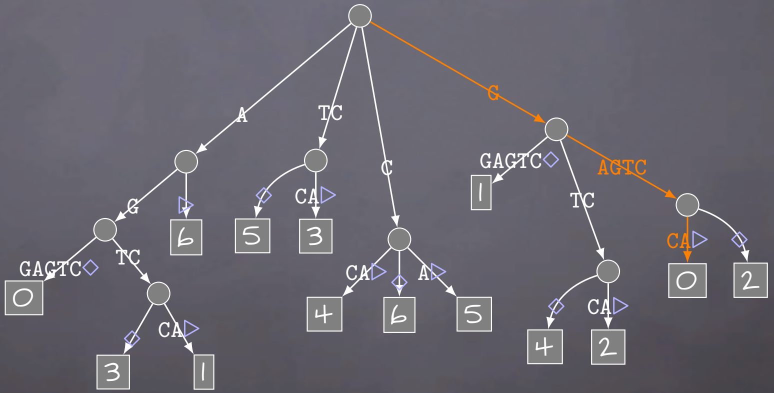
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

exercise: joined suffix tree for AGGAGTC◇ and GAGTCCA▷



Overlap-Layout-Consensus Assembly

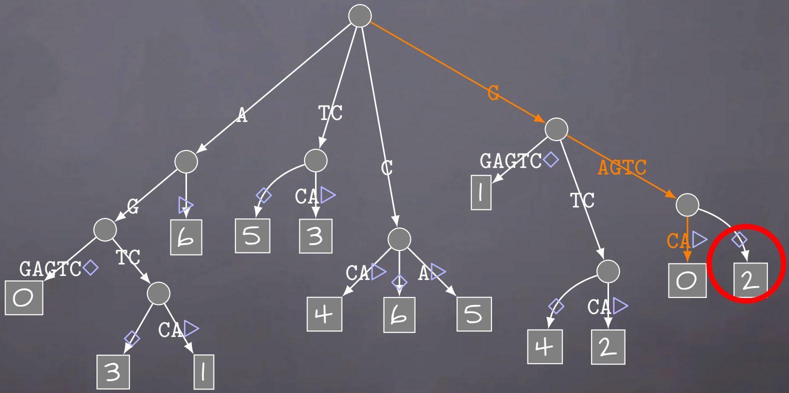
1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

exercise: joined suffix tree for AGGAGTC◇ and GAGTCCA▷



Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

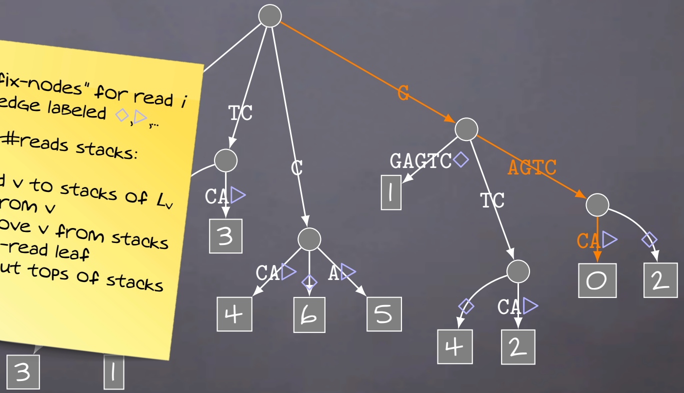
Suffix Trees

annotate branches with strings such that:

1. each root→leaf path is a suffix (leaf labeled with start index)
2. no two siblings have a common prefix

exercise: joined suffix tree for AGGAGTC◇ and GAGTCCA▷

Algo:
L_v := "suffix-nodes" for read i
⇔ has edge labeled ◇, ▷, ...
DFS with #reads stacks:
discover v
⇝ add v to stacks of L_v
backtrack from v
⇝ remove v from stacks
discover full-read leaf
⇝ output tops of stacks



⇝ O(#reads · read-length + #reads²)

[Gusfield et al.'92]

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

```
          AGGAGTC
GGTCTCA→
```

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

```
AGGAGTC
   GGTCTCA
```

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

```
AGGAGTC
  GAGTCTCA
```

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

```
AGGAGTC
  GAGTCTCA
```

edit distance = # of insertions, deletions, and substitutions

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

dynamic programming where

$[i,j]$ = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$

$$= \min\{\boxed{\downarrow} + 1, \boxed{\rightarrow} + 1, \boxed{\searrow} + id_{X_i, Y_j}\}$$

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

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

dynamic programming where

$[i,j]$ = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$

$$= \min\{\boxed{\downarrow}+1, \boxed{\rightarrow}+1, \boxed{\searrow}+id_{X_i,Y_j}\}$$

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

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

dynamic programming where

$[i,j]$ = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$

$$= \min\{\boxed{\downarrow}+1, \boxed{\rightarrow}+1, \boxed{\searrow}+id_{X_i,Y_j}\}$$

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

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

dynamic programming where

$[i,j]$ = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$

$= \min\{\boxed{\downarrow}+1, \boxed{\rightarrow}+1, \boxed{\searrow}+id_{X_i,Y_j}\}$

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

Overlap-Layout-Consensus Asse

1. produce pairwise overlaps (All-Pairs Suffix-P

Fuzzy Overlap – Edit Distance

dynamic programming where

$[i,j]$ = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$

$$= \min\{\boxed{\downarrow}+1, \boxed{\rightarrow}+1, \boxed{\searrow}+id_{X_i,Y_j}\}$$

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

modification: any suffix of GGTCTCA for free

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

dynamic programming where

$[i,j]$ = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$

$$= \min\{\boxed{\downarrow} + 1, \boxed{\rightarrow} + 1, \boxed{\searrow} + id_{X_i, Y_j}\}$$

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

modification: any suffix of GGTCTCA for free

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

dynamic programming where

[i,j] = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$

$$= \min\{\boxed{\downarrow} + 1, \boxed{\rightarrow} + 1, \boxed{\searrow} + id_{X_i, Y_j}\}$$

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

modification: any suffix of GGTCTCA for free

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)

Fuzzy Overlap – Edit Distance

dynamic programming where
$[i,j]$ = edit distance of $X_{i\ldots}$ & $Y_{j\ldots}$
$= \min\{\boxed{\downarrow} + 1, \boxed{\rightarrow} + 1, \boxed{\searrow} + id_{X_i,Y_j}\}$

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

modification: any suffix of GGTCTCA for free
⇝ best overlaps with k errors in $O(\#reads^2 \cdot \text{read-length}^2)$

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps

Overlap Graph

reads = vertices
directed edges = overlaps

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps

Overlap Graph

reads = vertices
directed edges = overlaps

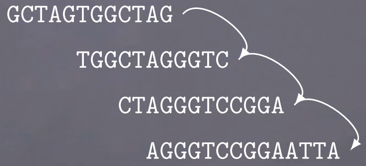
⇝ transitive reduction

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps

Overlap Graph

reads = vertices
directed edges = overlaps



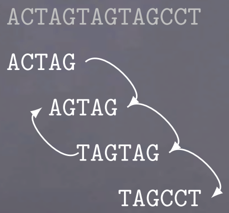
⇝ transitive reduction

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps

Overlap Graph

reads = vertices
directed edges = overlaps

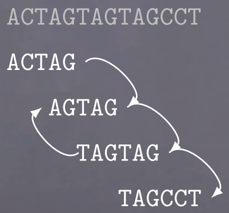


Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps

Overlap Graph

reads = vertices
directed edges = overlaps



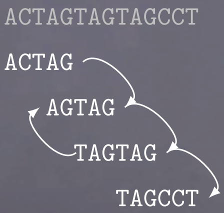
⇝ overlap graph non-linear due to repeats

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps

Overlap Graph

reads = vertices
directed edges = overlaps



⇝ overlap graph non-linear due to repeats
⇝ only return non-branching parts ("contigs"): ACTAGTAG & TAGCCT

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps
3. for each position, compute consensus base

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps
3. for each position, compute consensus base

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTC                CGACACTCCTTGGGTTTT              CTAGGCCATTGATTGCGGGTC
        ACTTCGC   GGTTCTCT                                                   GGTCCAGGTGCTGTCAACGAC
           TCGCTAGGGTTCTCTAACGA                TTTACGTCGCGG                                   CGAC
```

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps
3. for each position, compute consensus base

```
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCCTGAACTC                CGACACTCCTTGGGTTTT              CTAGGCCATTGATTGCGGGTC
        ACTTCGC   GGTTCTCT                                                   GGTCCAGGTGCTGTCAACGAC
           TCGCTAGGGTTCTCTAACGA             TTTACGTCGCGG                                      CGAC
GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTTACGTCGCGG    CTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
```

Overlap-Layout-Consensus Assembly

1. produce pairwise overlaps (All-Pairs Suffix-Prefix)
2. layout the reads according to the overlaps
3. for each position, compute consensus base

Problems

- overlap step too slow in practice:
  10^8 reads $\rightsquigarrow$ 10^{16} read-pairs
  $\rightsquigarrow$ heuristics exclude most of the read-pairs before overlap
- fragmented genome due to repeats

DeBruijn-Graph-Based Assembly

1. chop all reads into "k-mers"
   real genomes: k = 30-50

DeBruijn-Graph-Based Asse

k=5

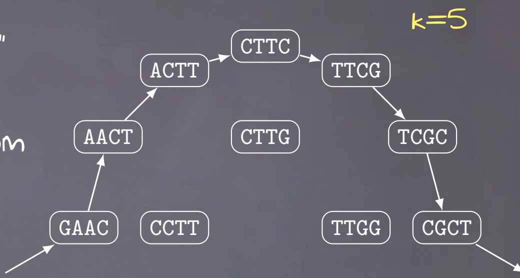
1. chop all reads into "k-mers"
   real genomes: k = 30-50
2. Build "DeBruijn graph":
   for each k-mer add arc from
   left to right k-1 mer

..GAACTTCGCT..

..CCTTGG..

DeBruijn-Graph-Based Assembly

1. chop all reads into "k-mers"
   real genomes: k = 30-50
2. Build "DeBruijn graph":
   for each k-mer add arc from left to right k-1 mer

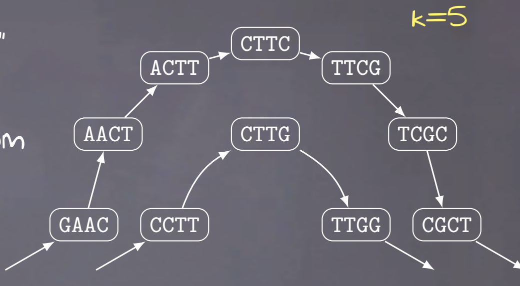



```
..GAACTTCGCT..
 .GAAC
  GAACT
   AACTT
    ACTTC
     CTTCG
      TTCGC
       TCGCT
```

```
..CCTTGG..
 .CCTT
  CCTTG
   TTGG.
```

DeBruijn-Graph-Based Assembly

1. chop all reads into "k-mers"
   real genomes: k = 30-50
2. Build "DeBruijn graph":
   for each k-mer add arc from left to right k-1 mer

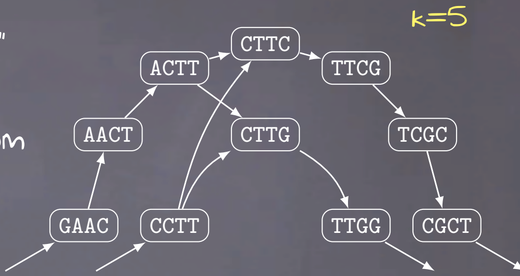



```
..GAACTTCGCT..
 .GAAC
  GAACT
   AACTT
    ACTTC
     CTTCG
      TTCGC
       TCGCT
```

```
..CCTTGG..
 .CCTT
  CCTTG
   TTGG.
```

DeBruijn-Graph-Based Assembly

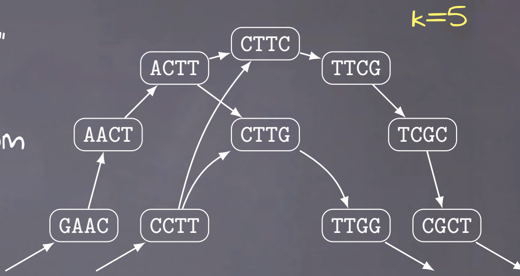
1. chop all reads into "k-mers"
   real genomes: k = 30-50
2. Build "DeBruijn graph":
   for each k-mer add arc from
   left to right k-1 mer




```
..GAACTTCGCT..            ..CCTTGG..
 .GAAC                      .CCTT
  GAACT                      CCTTG
   AACTT                      TTGG.
    ACTTC
     CTTCG                ..CCTTC..
      TTCGC               ..ACTTG..
       TCGCT
```

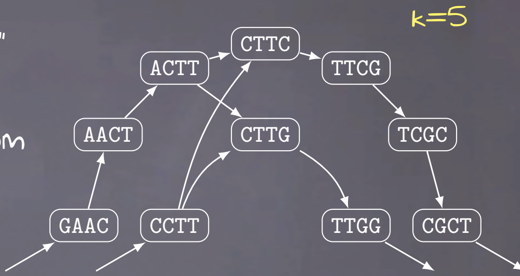
DeBruijn-Graph-Based Assembly

1. chop all reads into "k-mers"
   real genomes: k = 30-50
2. Build "DeBruijn graph":
   for each k-mer add arc from left to right k-1 mer
3. find path using all overlaps



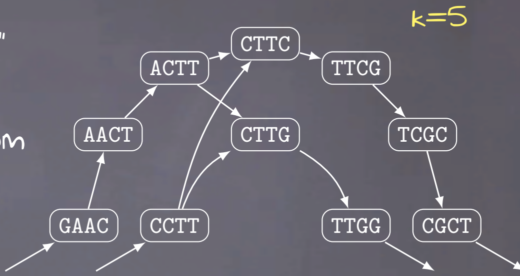
DeBruijn-Graph-Based Assembly

1. chop all reads into "k-mers"
   real genomes: k = 30-50
2. build "DeBruijn graph":
   for each k-mer add arc from left to right k-1 mer
3. find Eulerian walk
   linear time with greedy



DeBruijn-Graph-Based Assembly

1. chop all reads into "k-mers"
   real genomes: k = 30-50
2. Build "DeBruijn graph":
   for each k-mer add arc from left to right k-1 mer
3. find Eulerian walk
   linear time with greedy



Running Time

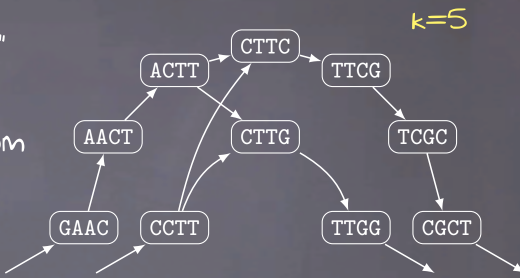
#k-mers = O(#reads · read-length)

1. O(1) per k-mer
2. O(1) per k-mer
3. O(size of graph) = O(#k-mers)

Note: edges can be weighted by #occurances

DeBruijn-Graph-Based Assembly

1. chop all reads into "k-mers"
   real genomes: $k = 30\text{-}50$
2. build "DeBruijn graph":
   for each k-mer add arc from left to right k-1 mer
3. find Eulerian walk
   linear time with greedy



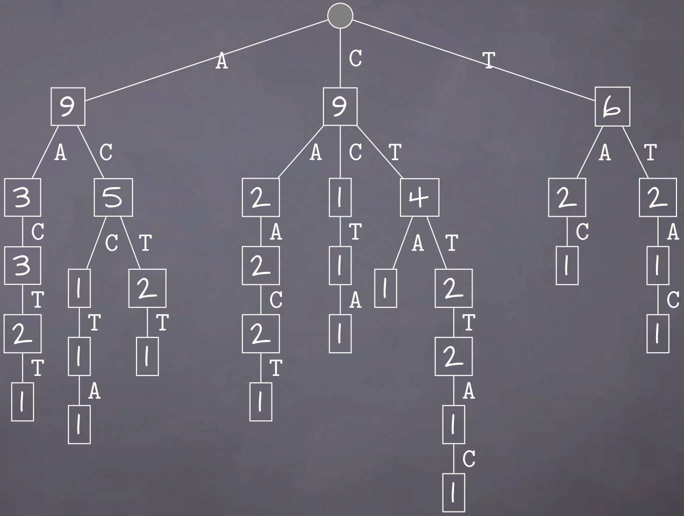
Problems

- choose k well
  - k too small ⇝ small repeats become problems
  - k too big ⇝ miss smaller overlaps
- Eulerian walk not neccessarily unique
- some paths in DeBruijn graph inconsistent with reads
- read-errors problematic
  ⇝ error-correction step before assembling
- same problem with repeats as OLC

Correcting Read Errors in Suffix Trees

Example

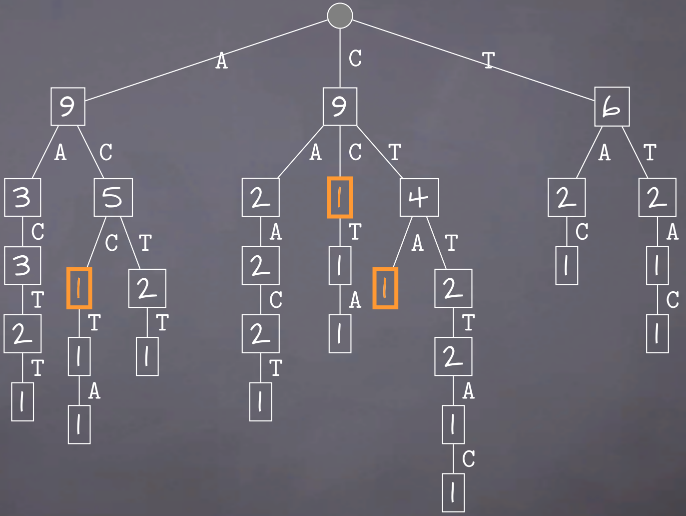
CAACTTAC
CAACT
CAAC
AACTT
ACCTA
CTTAC



Correcting Read Errors in Suffix Trees

Example

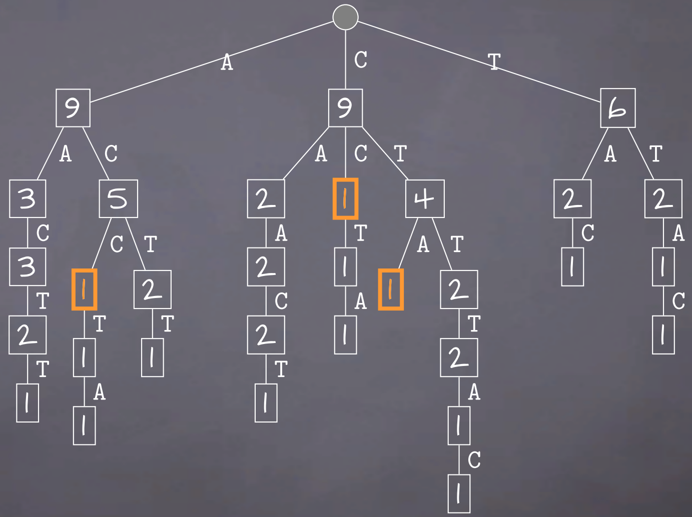
CAACTTAC
CAACT
CAAC
AACTT
ACCTA
CTTAC



Correcting Read Errors in Suffix Trees

Example

CAACTTAC
CAACT
CAAC
AACTT
ACCTA
CTTAC

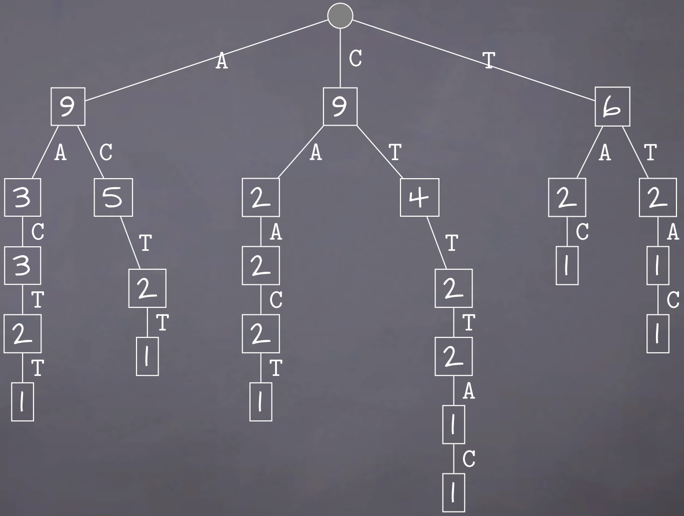


Idea: low freq. node with high freq. parent ⇝ ignore branch

Correcting Read Errors in Suffix Trees

Example

CAACTTAC
CAACT
CAAC
AACTT
ACCTA
CTTAC



Idea: low freq. node with high freq. parent ⇝ ignore branch

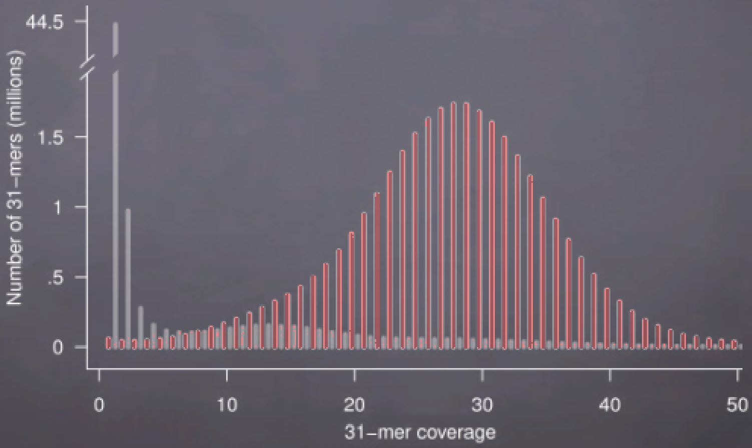
Correcting Read Errors in DBG

Idea

"faulty" k-mers occur less often than correct ones
⇝ build k-mer count histogram

De Bruijn

have to treat errors before building the graph
k=30 & 1% error ⇝ 1/4 faulty



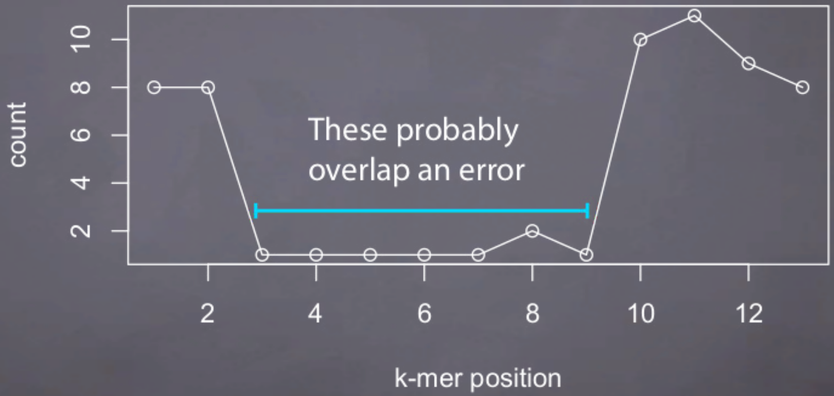
Correcting k-mer Errors

Example

suppose: avg. k-mer count = 10 ⇝ each k-mer occurs about 10x

GCGTATTACGCGTCTGGCCT		GCGTATTACTCGTCTGGCCT	
CGTATT	8x	CGTATT	8x
GTATTA	9x	GTATTA	9x
TATTAC	7x	TATTAC	7x
ATTACG	12x	ATTACT	1x
TTACGC	9x	TTACTC	2x
TACGCG	9x	TACTCG	2x
ACGCGT	10x	ACTCGT	1x
CGCGTC	11x	CTCGTC	1x
GCGTCT	10x	TCGTCT	1x
CGTCTG	9x	CGTCTG	9x
GTCTGG	10x	GTCTGG	10x
TCTGGC	10x	TCTGGC	10x
CTGGCC	11x	CTGGCC	11x
TGGCCT	9x	TGGCCT	9x

Correcting k-mer Errors



Correcting k-mer Errors

Problem

now we have an idea where an error is, but how to fix it?

Idea

errors turn frequent k-mers into infrequent ones
$\rightsquigarrow$ correction should turn infrequent k-mers into frequent ones
$\rightsquigarrow$ replace infrequent k-mer by "frequent neighbor"

Intro: Genome Scaffolding

Recall: repeats (common in DNA) make assembly ambiguous

Intro: Genome Scaffolding

Recall: repeats (common in DNA) make assembly ambiguous
$\leadsto$ end product is a set of "contiguous regions"

Intro: Genome Scaffolding

Recall: repeats (common in DNA) make assembly ambiguous
⇝ end product is a set of "contiguous regions"
Problem: "contig soup" not very useful

Intro: Genome Scaffolding

Recall: repeats (common in DNA) make assembly ambiguous
⇝ end product is a set of "contiguous regions"
Problem: "contig soup" not very useful
But: with NGS, we have paired-end information!
⇝ Scaffolding + Filling

Intro: Genome Scaffolding

Recall: repeats (common in DNA) make assembly ambiguous
⇝ end product is a set of "contiguous regions"
Problem: "contig soup" not very useful
But: with NGS, we have paired-end information!
⇝ Scaffolding + Filling

Scaffolding

Goal: order & orient contigs
Idea: use pairing information on reads to "link" contigs together

Graph-Based Scaffolding

AACGACACTCCTTGGGTTTTTACGTCGCGG

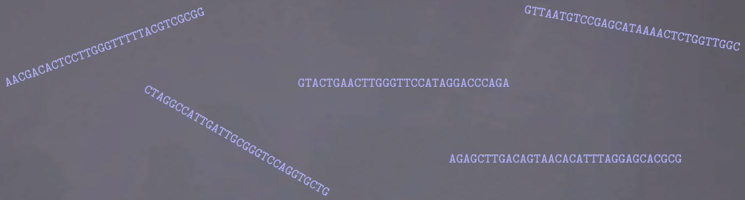
GTTAATGTCCGAGCATAAAACTCTGGTTGGC

GTACTGAACTTGGGTTCCATAGGACCCAGA

CTAGGCCATTGATTGCGGGTCCAGGTGCTG

AGAGCTTGACAGTAACACATTTAGGAGCACGCG

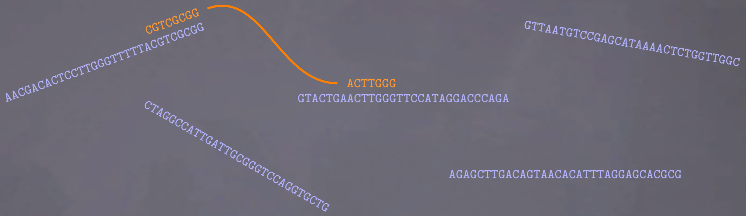
Graph-Based Scaffolding



Strategy

1. map reads into contigs

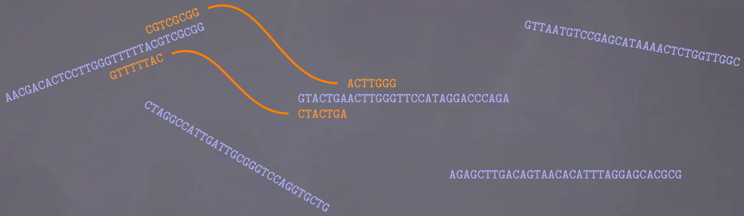
Graph-Based Scaffolding



Strategy

1. map reads into contigs

Graph-Based Scaffolding



Strategy

1. map reads into contigs

Graph-Based Scaffolding

AACGACACTCCTTGGGTTTTTACGTCGCGG

GTTAATGTCCGAGCATAAAACTCTGGTTGGC

GTACTGAACTTGGGTTCCATAGGACCCAGA

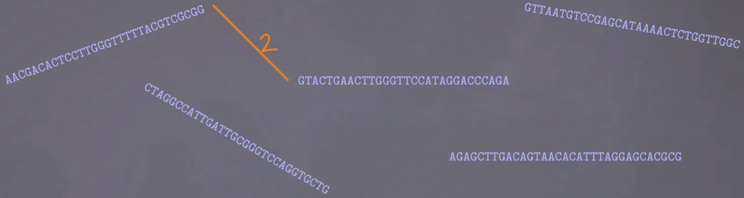
CTAGGCCATTGATTGCGGGTCCAGGTGCTG

AGAGCTTGACAGTAACACATTTAGGAGCACGCG

Strategy

1. map reads into contigs
2. pair contigs according to read-pairing (weighted)

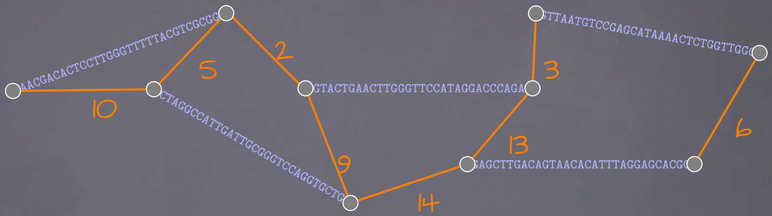
Graph-Based Scaffolding



Strategy

1. map reads into contigs
2. pair contigs according to read-pairing (weighted)

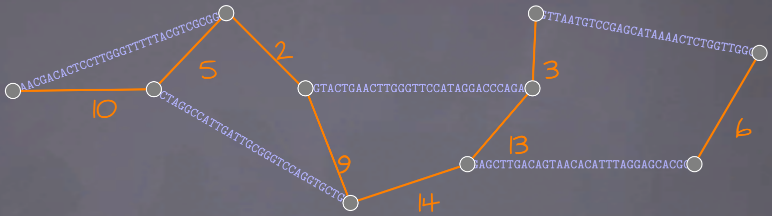
Graph-Based Scaffolding



Strategy

1. map reads into contigs
2. pair contigs according to read-pairing (weighted)

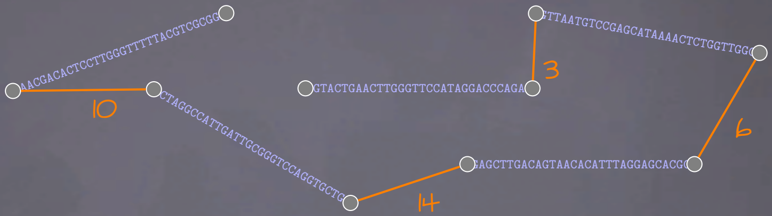
Graph-Based Scaffolding



Strategy

1. map reads into contigs
2. pair contigs according to read-pairing (weighted)
3. cover "scaffold graph" with (heavy) alternating paths
   each path corresponds to a chromosome

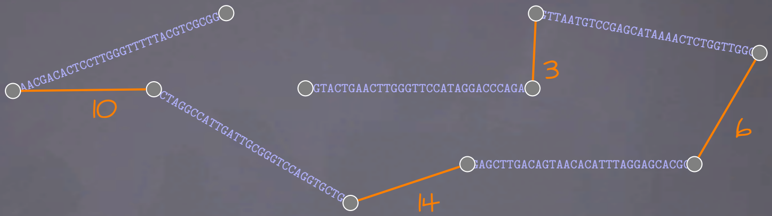
Graph-Based Scaffolding



Strategy

1. map reads into contigs
2. pair contigs according to read-pairing (weighted)
3. cover "scaffold graph" with (heavy) alternating paths
   each path corresponds to a chromosome

Graph-Based Scaffolding



Scaffolding

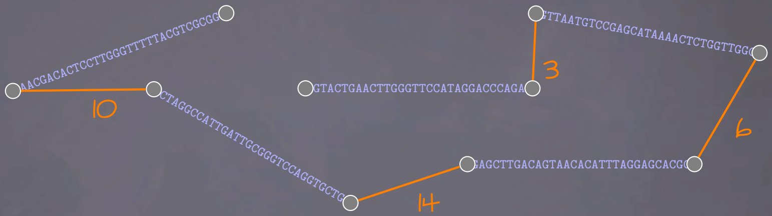
Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by

- $\leq \sigma_p$ alternating paths

of total weight $\geq k$?

Graph-Based Scaffolding



Scaffolding

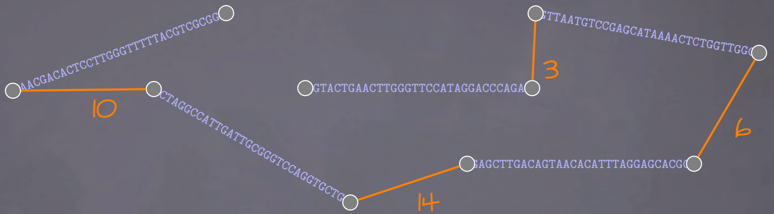
Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by

- $\leq \sigma_p$ alternating paths &
- $\leq \sigma_c$ alternating cycles

of total weight $\geq k$?

Graph-Based Scaffolding



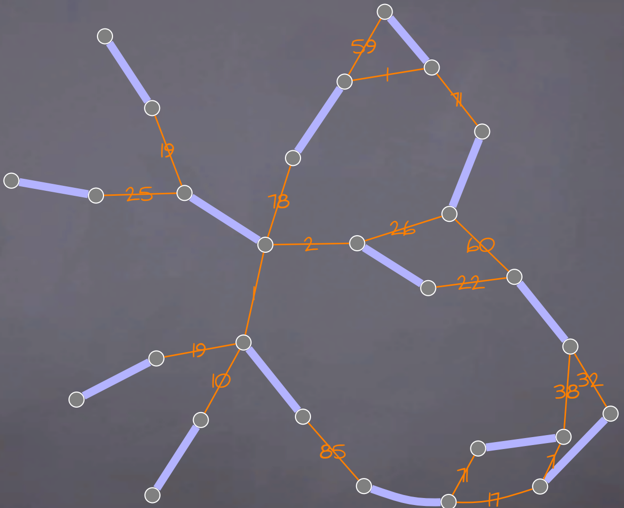
Exact Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

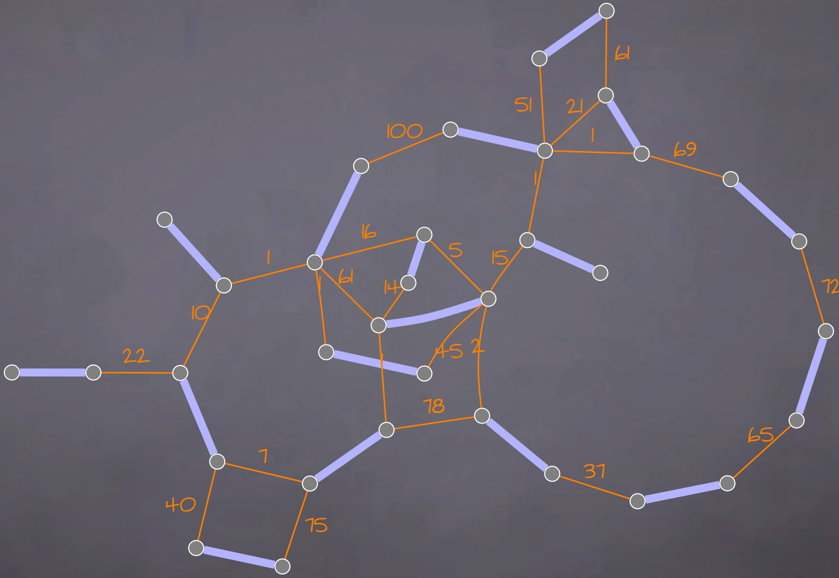
Question: Can $\mathcal{M}$ be covered by

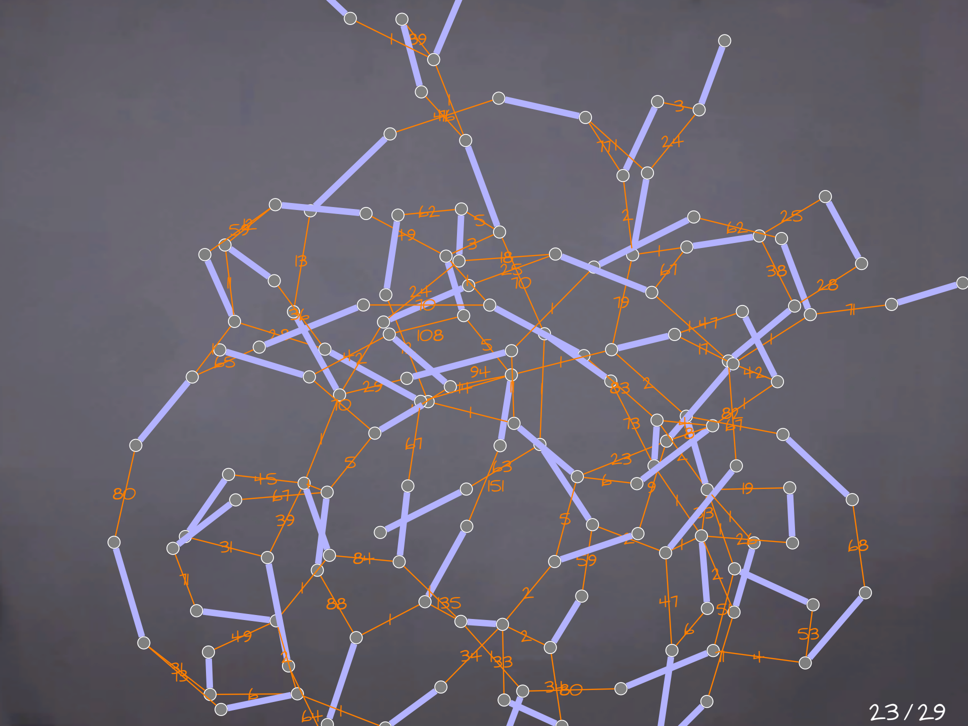
- σ_p alternating paths &
- σ_c alternating cycles

of total weight $\geq k$?



59
1
71
19
25
78
26
2
60
22
1
19
10
85
38
32
71
71
17


61
51
21
1
100
69
1
16
1
5
15
61
1
14
10
72
22
45
2
1
78
7
40
75
31
65

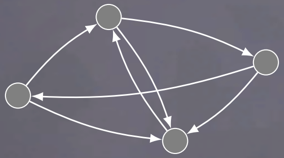


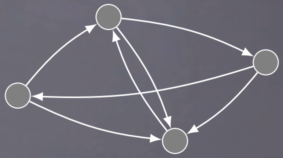
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Construction

Given a directed graph D

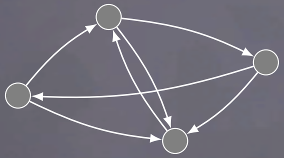
1. make a copy of D

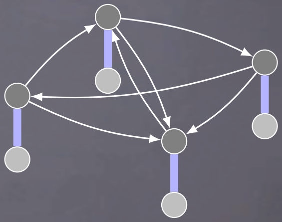
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Construction

Given a directed graph D

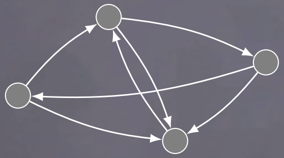
1. make a copy of D
2. duplicate all vertices $\rightsquigarrow \mathcal{M}$

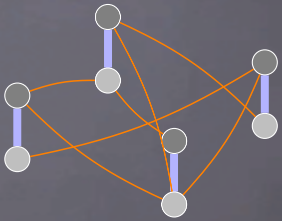
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths &
$\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Construction

Given a directed graph D

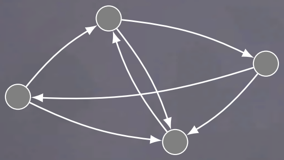
1. make a copy of D
2. duplicate all vertices $\rightsquigarrow \mathcal{M}$
3. "slide" down all arrow tips & ignore directions

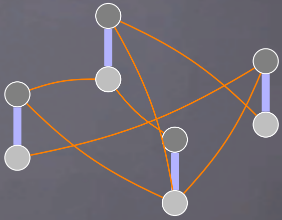
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Lemma

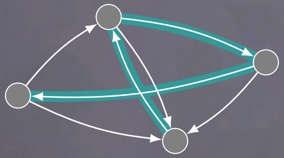
D admits a directed Hamiltonian path $\Leftrightarrow$ $\mathcal{M}$ can be covered with a single alternating path in G

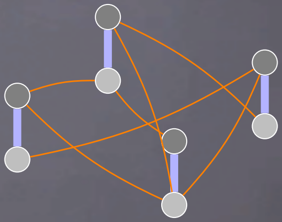
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Lemma

D admits a directed Hamiltonian path $\Leftrightarrow$ $\mathcal{M}$ can be covered with a single alternating path in G

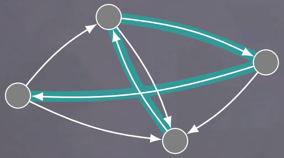
"$\Rightarrow$": replace each v in the Hamiltonian path by $v_{low} \rightarrow v_{high}$

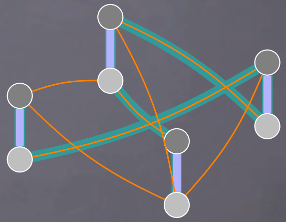
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Lemma

D admits a directed Hamiltonian path $\Leftrightarrow$ $\mathcal{M}$ can be covered with a single alternating path in G

"$\Rightarrow$": replace each v in the Hamiltonian path by $v_{low} \rightarrow v_{high}$

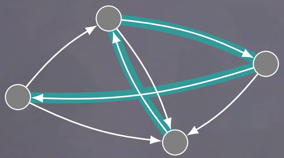
alternating ✓ covers $\mathcal{M}$ ✓

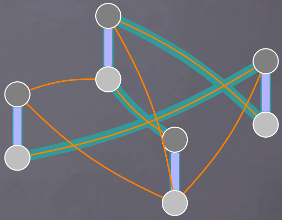
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Lemma

D admits a directed Hamiltonian path $\Leftrightarrow$ $\mathcal{M}$ can be covered with a single alternating path in G

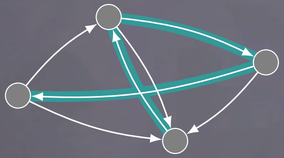
"$\Leftarrow$": contract each matching edge in the covering alternating path

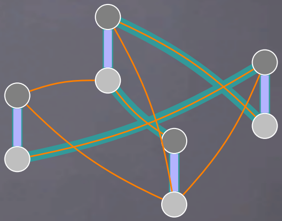
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Lemma

D admits a directed Hamiltonian path $\Leftrightarrow$ $\mathcal{M}$ can be covered with a single alternating path in G

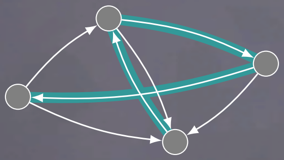
"$\Leftarrow$": contract each matching edge in the covering alternating path
hits all vertices exactly once ✓ is valid directed path ✓

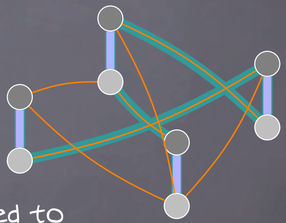
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Theorem

Scaffolding is NP-hard, even restricted to

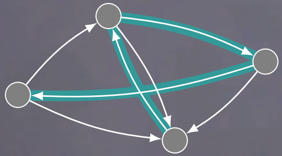
- bipartite graphs
- $(\sigma_p, \sigma_c) \in \{(0,1),(1,0)\}$ and
- $\omega : E \to \{0\}$

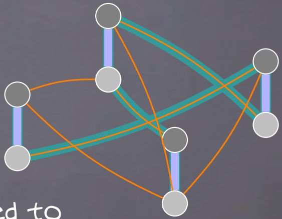
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Theorem

Scaffolding is NP-hard, even restricted to

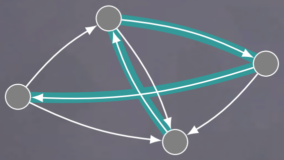
- supergraphs of bipartite graphs
- $(\sigma_p, \sigma_c) \in \{(0,1), (1,0)\}$ and
- $\omega : E \to \{0, 1\}$

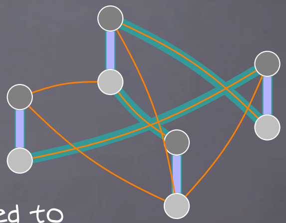
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Theorem

Scaffolding is NP-hard, even restricted to

- supergraphs of bipartite graphs
- $(\sigma_p, \sigma_c) \in \{(0,1),(1,0)\}$ and
- $\omega : E \to \{0, 1\}$

Corollary

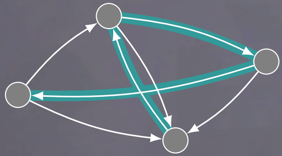
Scaffolding with 2 weights is NP-hard in any sufficiently dense graph class.

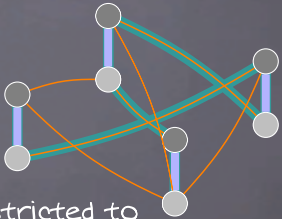
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

Input: Graph G, perfect matching $\mathcal{M}$, weights ω, $k, \sigma_p, \sigma_c \in \mathbb{N}$

Question: Can $\mathcal{M}$ be covered by $\leq \sigma_p$ alternating paths & $\leq \sigma_c$ alternating cycles of total weight $\geq k$?





Theorem

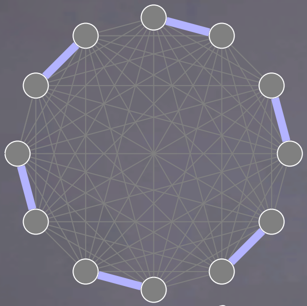
Exact Scaffolding is NP-hard, even restricted to

- supergraphs of bipartite graphs
- $(\sigma_p, \sigma_c) \in \{(0,1),(1,0)\}$ and
- $\omega : E \to \{0, 1\}$

Corollary

Exact Scaffolding with 2 weights is NP-hard in any sufficiently dense graph class.

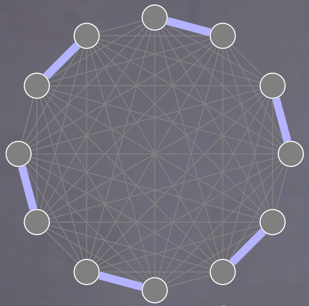
3-Approximation in Dense Graphs



$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

3-Approximation in Dense Graphs

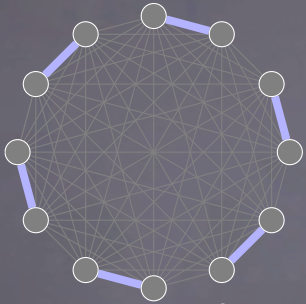


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. sort all edges by weight

3-Approximation in Dense Graphs

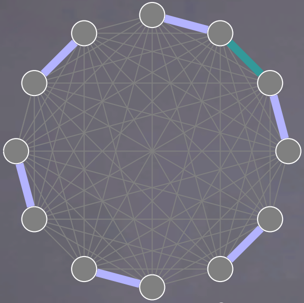


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

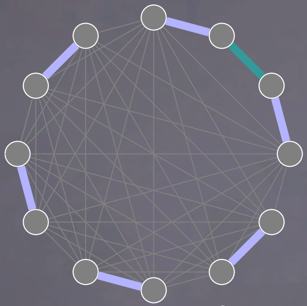


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

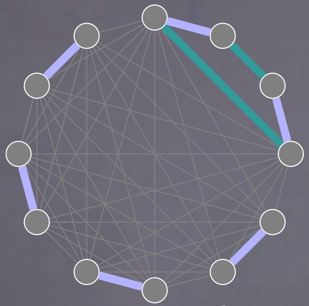


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

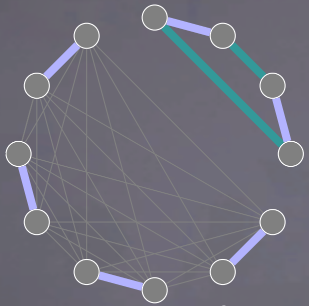


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

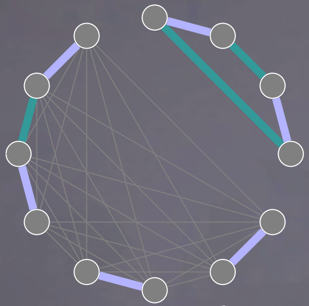


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

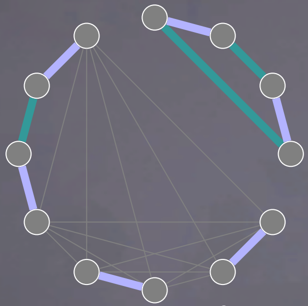


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

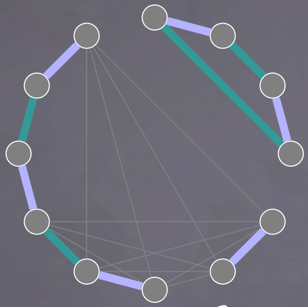


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

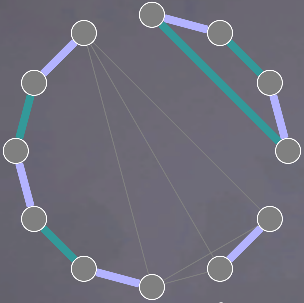


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

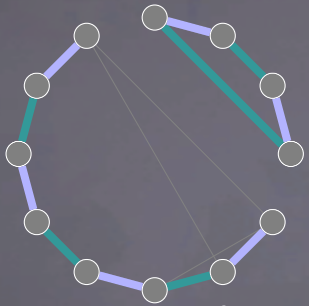


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

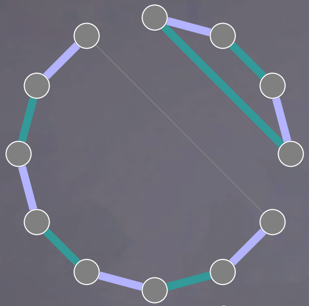


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs

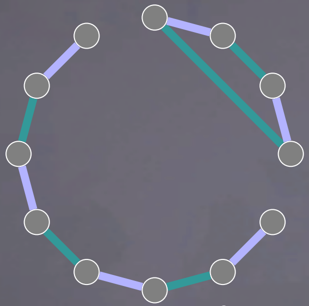


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

3-Approximation in Dense Graphs



$\sigma_p = 1$, $\sigma_c = 1$?

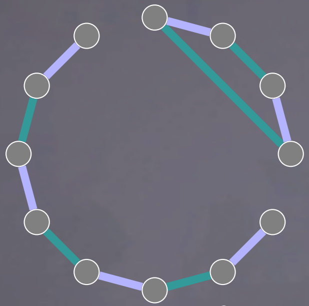
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

Proof

Result S^* is a valid solution ✓

3-Approximation in Dense Graphs



$\sigma_p = 1, \sigma_c = 1?$

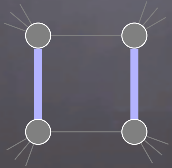
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

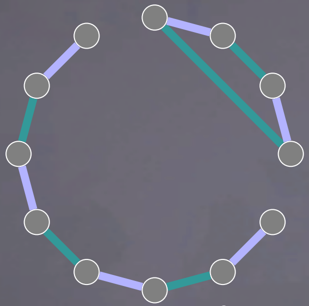
Proof

Result S^* is a valid solution ✓

Note: taking an edge forbids ≤ 3 OPT edges



3-Approximation in Dense Graphs



$\sigma_p = 1, \sigma_c = 1?$

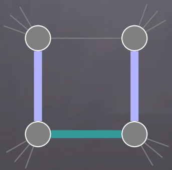
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

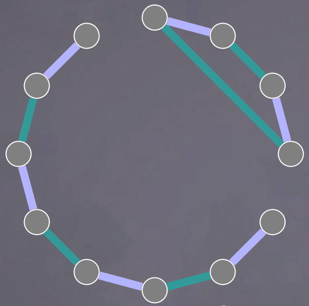
Proof

Result S^* is a valid solution ✓

Note: taking an edge forbids ≤ 3 OPT edges



3-Approximation in Dense Graphs



$\sigma_p = 1, \sigma_c = 1?$

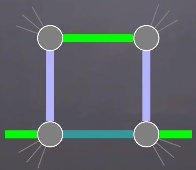
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

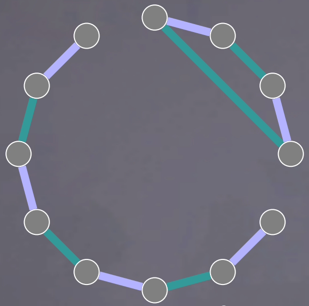
Proof

Result S^* is a valid solution ✓

Note: taking an edge forbids ≤ 3 OPT edges



3-Approximation in Dense Graphs



$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

Proof

Result S^* is a valid solution ✓

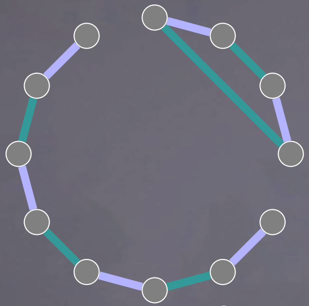
Note: taking an edge forbids ≤ 3 OPT edges

$\rightsquigarrow$ mark the ≤ 3 OPT-edges when taking an edge e

$\rightsquigarrow$ e is heaviest among them

$\rightsquigarrow$ $3\omega(S^*) \geq OPT$

3-Approximation in Dense Graphs



$\sigma_p = 1,\ \sigma_c = 1?$

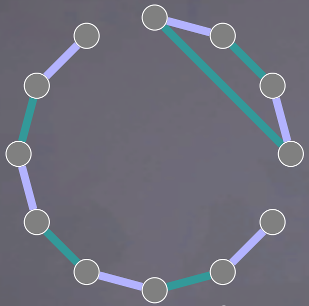
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

Theorem

Scaffolding in complete graphs can be 3-approximated in $O(|V|^2 \log |V|)$ time.

3-Approximation in Dense Graphs



$\sigma_p = 1, \sigma_c = 1?$

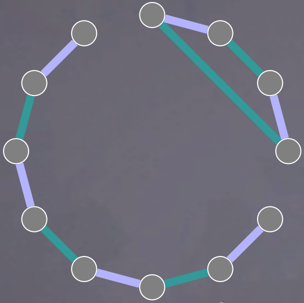
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

Theorem

Scaffolding in complete (bipartite) graphs can be 3-approximated in $O(|V|^2 \log |V|)$ time.

3-Approximation in Dense Graphs



$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest poss. edge

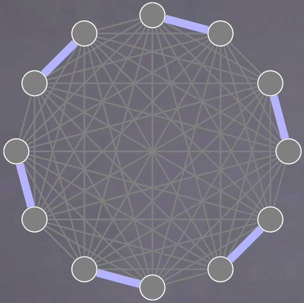
Theorem

Scaffolding in complete (bipartite) graphs can be 3-approximated in $O(|V|^2 \log |V|)$ time.

Remark

For Exact Scaffolding, we have to keep an eye on the number of components too.

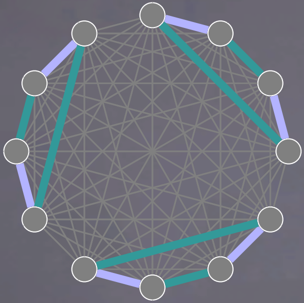
2-Approximation in Dense Graphs



$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

2-Approximation in Dense Graphs

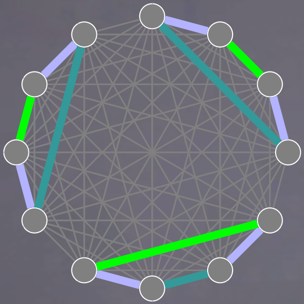


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles

2-Approximation in Dense Graphs

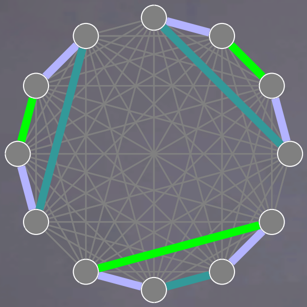


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle

2-Approximation in Dense Graphs

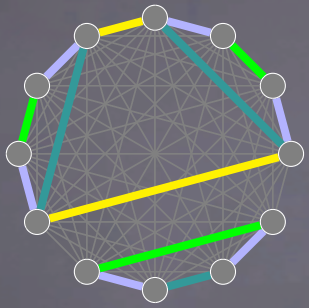


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain

2-Approximation in Dense Graphs

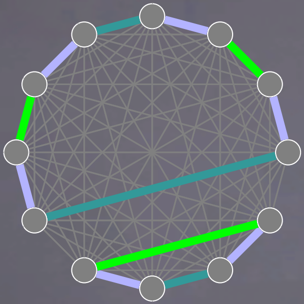


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain

2-Approximation in Dense Graphs

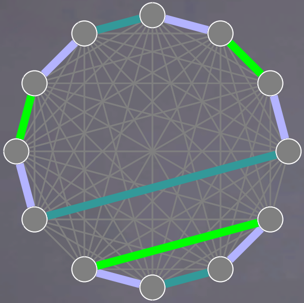


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain

2-Approximation in Dense Graphs

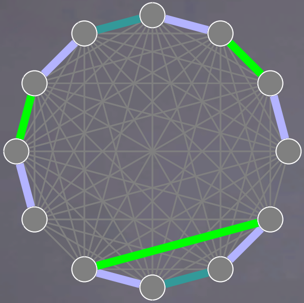


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain
4. repeatedly remove lightest non-fix cycle-edge until at most σ_c cycles remain

2-Approximation in Dense Graphs

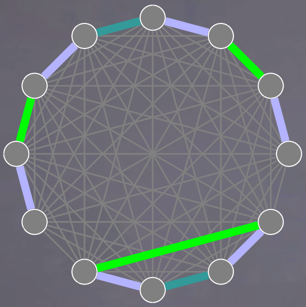


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain
4. repeatedly remove lightest non-fix cycle-edge until at most σ_c cycles remain

2-Approximation in Dense Graphs



$\sigma_p = 1, \sigma_c = 1?$

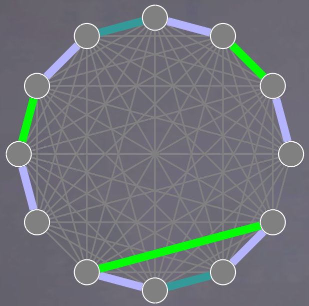
Approximate Scaffolding

1. compute max-weight perfect matching S
   $\leadsto S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain
4. repeatedly remove lightest non-fix cycle-edge until at most σ_c cycles remain

Proof

Result S^* is a valid solution ✓

2-Approximation in Dense Graphs



$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

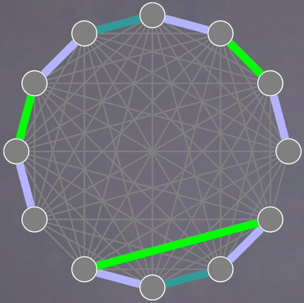
1. compute max-weight perfect matching S
   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain
4. repeatedly remove lightest non-fix cycle-edge until at most σ_c cycles remain

Proof

Result S^* is a valid solution ✓

$\omega(S^*) \geq \omega(fix) \geq \omega(S)/2 \geq OPT/2$

2-Approximation in Dense Graphs



$\sigma_p = 1, \sigma_c = 1?$

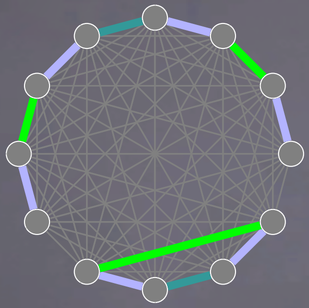
Approximate Scaffolding

1. compute max-weight perfect matching S
   $\rightsquigarrow S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain
4. repeatedly remove lightest non-fix cycle-edge until at most σ_c cycles remain

Theorem

Scaffolding in complete graphs can be 2-approximated in $O(|V|^{2.5})$ time.

2-Approximation in Dense Graphs



$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. compute max-weight perfect matching S

   $\rightsquigarrow$ $S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain
4. repeatedly remove lightest non-fix cycle-edge until at most σ_c cycles remain

Theorem

Scaffolding in complete (bipartite) graphs can be 2-approximated in $O(|V|^{2.5})$ time.

Scaffolding with Multiplicities

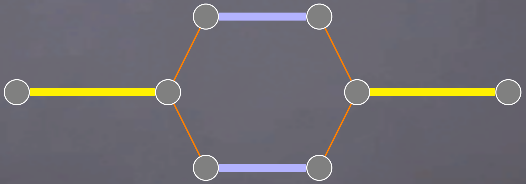
Recall: most eucaryotes are diploid!

GGTGCGAGAGAGGTCATGGATTGCAACGA

GGTGCGAGAGGCCACTCCAATTGCAACGA

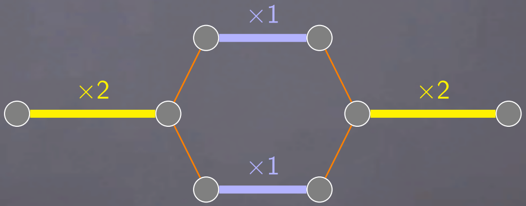
Scaffolding with Multiplicities

Recall: most eucaryotes are diploid!

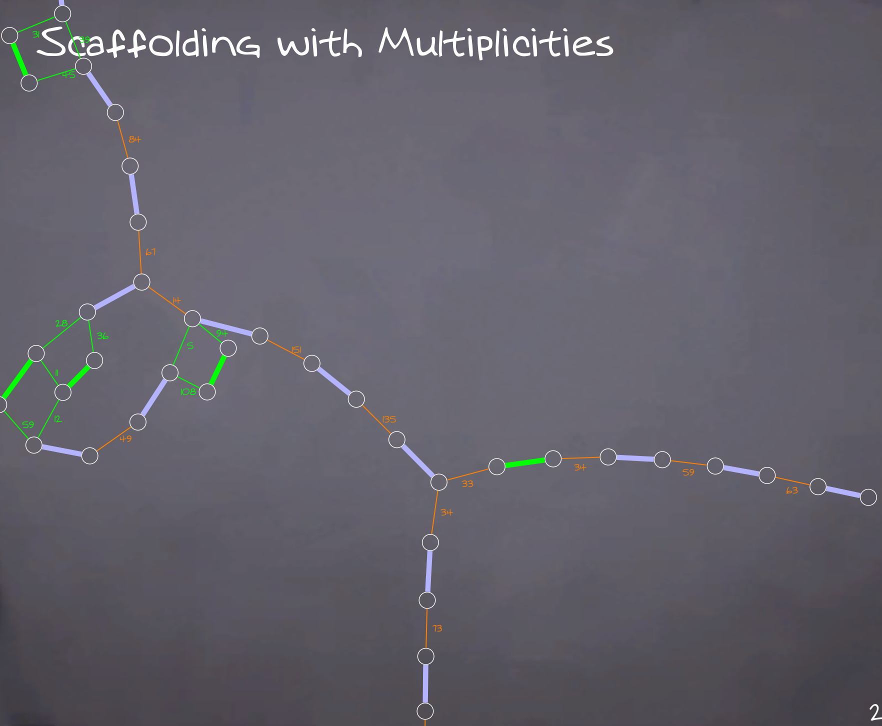


Scaffolding with Multiplicities

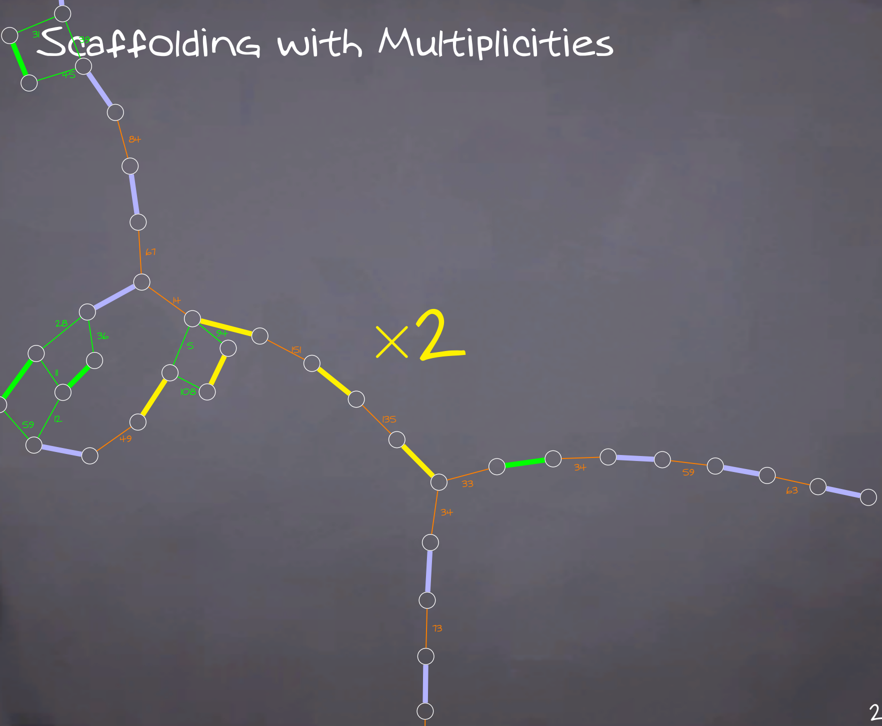
Recall: most eucaryotes are diploid!



Scaffolding with Multiplicities



Scaffolding with Multiplicities



×2

Linearization of Solutions

Problem

no unique chromosome-configuration explaining solution

uniquely linearizable = scaffold graph decomposes uniquely into alternating paths using each edge "the correct" number of times

Linearization of Solutions

Problem

no unique chromosome-configuration explaining solution

uniquely linearizable = scaffold graph decomposes uniquely into alternating paths using each edge "the correct" number of times

Linearization of Solutions

Problem

no unique chromosome-configuration explaining solution

uniquely linearizable = scaffold graph decomposes uniquely into alternating paths using each edge "the correct" number of times

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

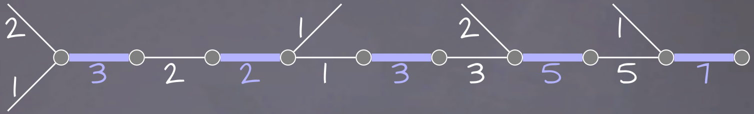
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)



Linearization of Solutions

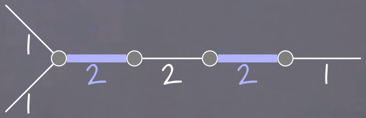
Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Rightarrow$": contraposition; let $p =$ ambigous path



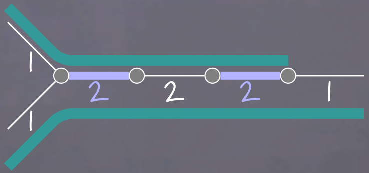
Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Rightarrow$": contraposition; let $p =$ ambigous path



Linearization of Solutions

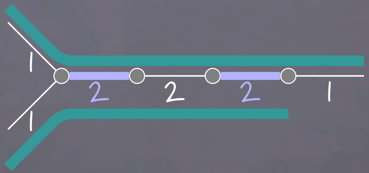
Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Rightarrow$": contraposition; let $p =$ ambigous path



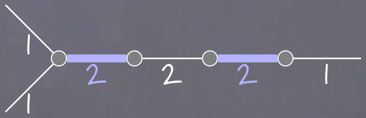
Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Rightarrow$": contraposition; let $p =$ ambigous path



$\rightsquigarrow$ $(G, \mathcal{M}, m)$ not uniquely linearizable

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Leftarrow$": let $(G, \mathcal{M}, m)$ be free of ambigous paths
Reduction (does not decrease number of linearizations):

Linearization of Solutions

Theorem

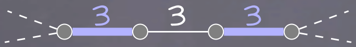
$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Leftarrow$": let $(G, \mathcal{M}, m)$ be free of ambigous paths

Reduction (does not decrease number of linearizations):



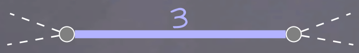
Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Leftarrow$": let $(G, \mathcal{M}, m)$ be free of ambigous paths
Reduction (does not decrease number of linearizations):



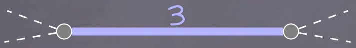
Linearization of Solutions

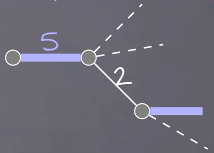
Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Leftarrow$": let $(G, \mathcal{M}, m)$ be free of ambigous paths
Reduction (does not decrease number of linearizations):





Linearization of Solutions

Theorem

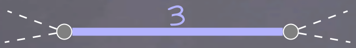
$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

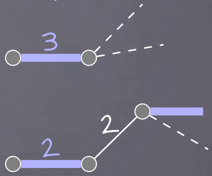
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Leftarrow$": let $(G, \mathcal{M}, m)$ be free of ambigous paths

Reduction (does not decrease number of linearizations):





Linearization of Solutions

Theorem

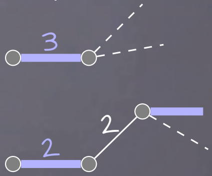
$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

Proof

"$\Leftarrow$": let $(G, \mathcal{M}, m)$ be free of ambigous paths

Reduction (does not decrease number of linearizations):



$\rightsquigarrow$ result is collection of alternating paths & cycles

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

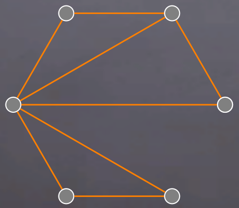
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover



Multiplicities

one &

#non-matching adj. to contig

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"

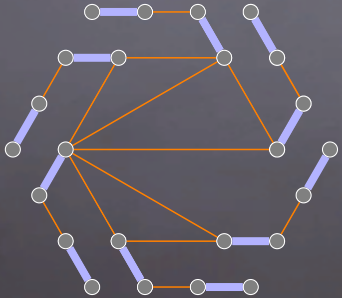
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover



Multiplicities

one &

#non-matching adj. to contig

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable ⇔ no "ambigous paths"

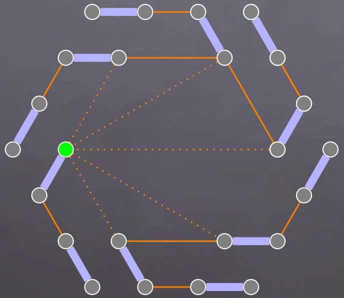
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

⇝ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguous path ⇝ information loss
3. cut as few ends as possible ⇝ as hard as Vertex Cover



Multiplicities

one &

#non-matching adj. to contig

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable ⇔ no "ambigous paths"

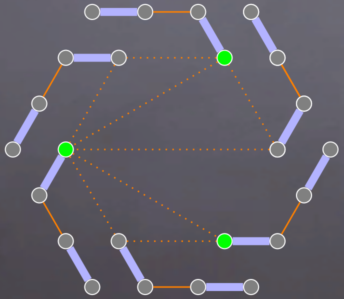
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

⇝ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguous path ⇝ information loss
3. cut as few ends as possible ⇝ as hard as Vertex Cover



Multiplicities

one &

#non-matching adj. to contig

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover
4. cut as few multiplicities as possible

Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover
4. cut as few multiplicities as possible $\rightsquigarrow$ as hard as Trans. Del. (Δ-free)

Linearization of Solutions

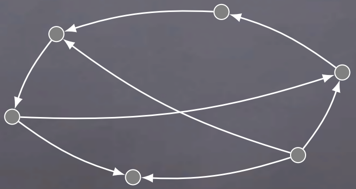
Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover
4. cut as few multiplicities as possible $\rightsquigarrow$ as hard as Trans. Del. (Δ-free)



Linearization of Solutions

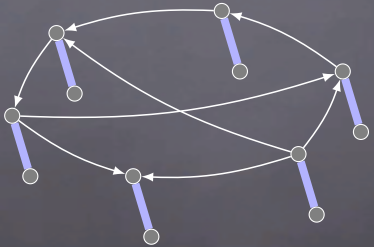
Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover
4. cut as few multiplicities as possible $\rightsquigarrow$ as hard as Trans. Del. (Δ-free)



Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable ⇔ no "ambigous paths"

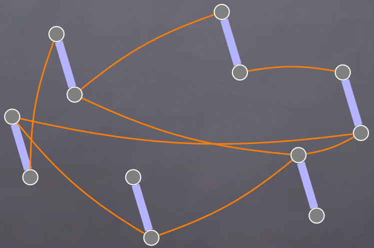
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

⇝ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguous path ⇝ information loss
3. cut as few ends as possible ⇝ as hard as Vertex Cover
4. cut as few multiplicities as possible ⇝ as hard as Trans. Del. (Δ-free)



Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable ⇔ no "ambigous paths"

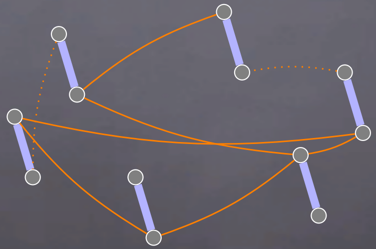
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

⇝ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguous path ⇝ information loss
3. cut as few ends as possible ⇝ as hard as Vertex Cover
4. cut as few multiplicities as possible ⇝ as hard as Trans. Del. (Δ-free)



Linearization of Solutions

Theorem

$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)
$\rightsquigarrow$ must destroy ambiguous paths

Idea: remove non-matching edges at their endpoints; strategy?

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguous path $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ as hard as Vertex Cover
4. cut as few multiplicities as possible $\rightsquigarrow$ as hard as Trans. Del. (Δ-free)