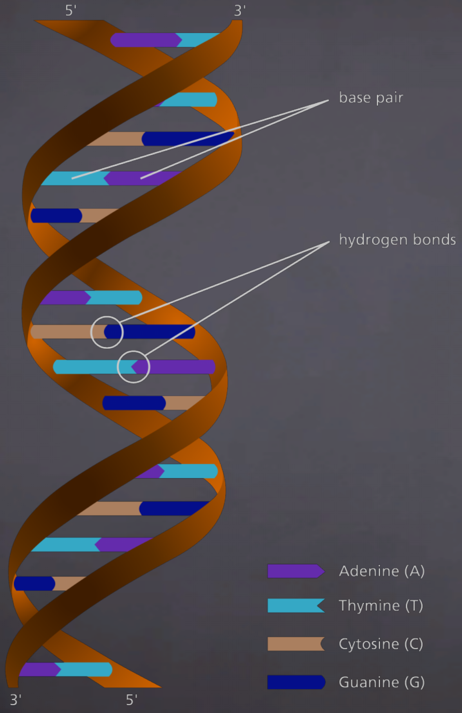
Lecture: Graph-Based Genome Scaffolding

Mathias Weller
mathias.weller@univ-mlv.fr

Montpellier, 2017

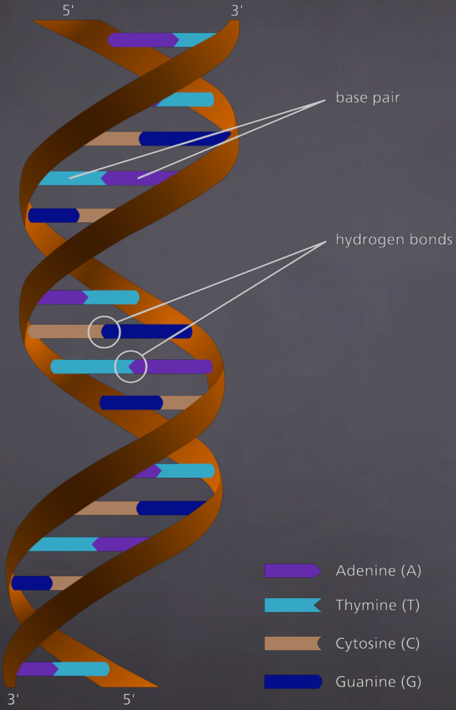


DNA

- double strand
- inside nucleus (safe)

RNA

- single strand
- outside nucleus
- transfers genetic code
- Thymine (T) → Uracil (U)



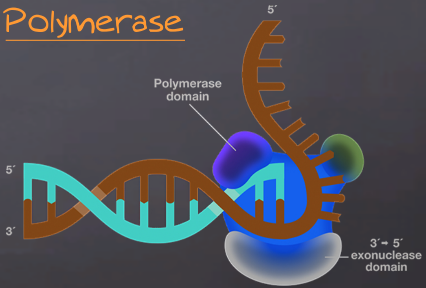
DNA

- double strand
- inside nucleus (safe)

RNA

- single strand
- outside nucleus
- transfers genetic code
- Thymine (T) → Uracil (U)

Polymerase



Sanger Sequencing

[Sanger et al '77]

```
CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGACCTGCCCAGTCTGTACTGTCACCGGGGTTCTAAGTGTTCTAGCATAGAGTTATGTCATTTGCTCGTTA
```

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGACCTGCCCAGTCTGTACTGTCACCGGGGTTCTAAGTGTTCTAGCATAGAGTTATGTCATTTGCTCGTTA

Sanger Sequencing

1. split helix & create thousands of copies

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT

Sanger Sequencing

1. split helix & create thousands of copies

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT

Sanger Sequencing

1. split helix & create thousands of copies
2. add polymerase & floating bases: A C G T
3. add a special base: A* (polymerase cannot extend)

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT

Sanger Sequencing

1. split helix & create thousands of copies
2. add polymerase & floating bases: A C G T
3. add a special base: A* (polymerase cannot extend)
4. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGA*

Sanger Sequencing

1. split helix & create thousands of copies
2. add polymerase & floating bases: A C G T
3. add a special base: A* (polymerase cannot extend)
4. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

CCTGGACGGGTCAGACATGACAGTGGCCCCAAGATTCACAAGATCGTATCTCAATACAGTAAACGAGCAAT
GGA*
GGACCTGCCCA*

Sanger Sequencing

1. split helix & create thousands of copies
2. add polymerase & floating bases: A C G T
3. add a special base: A* (polymerase cannot extend)
4. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

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

Sanger Sequencing

1. split helix & create thousands of copies
2. add polymerase & floating bases: A C G T
3. add a special base: A* (polymerase cannot extend)
4. stir & let polymerase act

Sanger Sequencing

[Sanger et al '77]

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

Sanger Sequencing

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

Problem

unreliable after a couple hundred bp
⇝ chop up DNA into pieces and read those

Next Generation Sequencing (illumina)

Next Generation Sequencing (illumina)

ACTCA......ACCTC

1. chop DNA into smaller pieces

Next Generation Sequencing (illumina)

TGGTACTCA......ACCTCTCAG

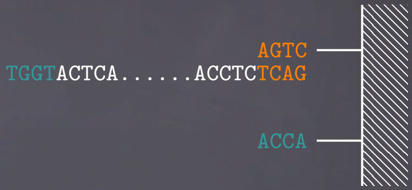
1. chop DNA into smaller pieces
2. add anchors to each end of each piece

Next Generation Sequencing (illumina)

TGGTACTCA......ACCTCTCAG

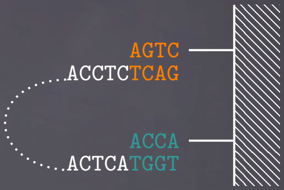
1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places

Next Generation Sequencing (illumina)



1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places
4. strand anchors its two ends to two anchor places

Next Generation Sequencing (illumina)

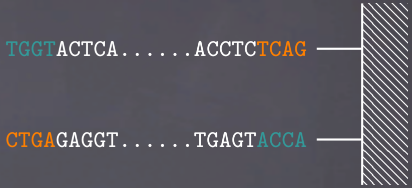


1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places
4. strand anchors its two ends to two anchor places

Next Generation Sequencing (illumina)

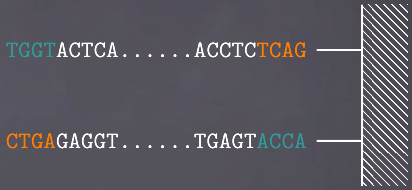
1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places
4. strand anchors its two ends to two anchor places
5. polymerase completes the strand into double-strand

Next Generation Sequencing (illumina)



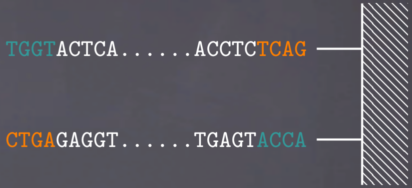
1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places
4. strand anchors its two ends to two anchor places
5. polymerase completes the strand into double-strand
6. double strand is denaturized into single strands

Next Generation Sequencing (illumina)



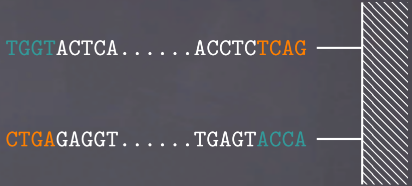
1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places
4. strand anchors its two ends to two anchor places
5. polymerase completes the strand into double-strand
6. double strand is denaturized into single strands
7. rinse, repeat (last 3 steps) until flow chip is "full"

Next Generation Sequencing (illumina)



1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places
4. strand anchors its two ends to two anchor places
5. polymerase completes the strand into double-strand
6. double strand is denaturized into single strands
7. rinse, repeat (last 3 steps) until flow chip is "full"
8. read all strands from their anchor points outwards

Next Generation Sequencing (illumina)



1. chop DNA into smaller pieces
2. add anchors to each end of each piece
3. "flow cell" containing anchor places
4. strand anchors its two ends to two anchor places
5. polymerase completes the strand into double-strand
6. double strand is denaturized into single strands
7. rinse, repeat (last 3 steps) until flow chip is "full"
8. read all strands from their anchor points outwards

⇝ Paired-End reads (distance between reads = "insert size")

Sequence Assembly: Overview

GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC

Goal: reconstruct sequence

Idea: overlap reads

Sequence Assembly: Overview

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

Goal: reconstruct sequence

Idea: overlap reads

Sequence Assembly: Overview

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

Goal: reconstruct sequence

Idea: overlap reads

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 1: parts of the sequence might not be covered by reads

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 1: 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

Idea: overlap reads

Problem 1: 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
Idea: overlap reads

Problem 2: Shortest Common Superstring is NP-hard
$\rightsquigarrow$ "Overlap-Layout-Consensus" assemblers

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 2: Shortest Common Superstring is NP-hard
⇝ "Overlap-Layout-Consensus" assemblers
1. produce best pairwise overlaps
2. layout the reads according to the overlaps
3. for each position, compute consensus base

Sequence Assembly: Overview

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

Goal: reconstruct sequence

Idea: overlap reads

Problem 2: Shortest Common Superstring is NP-hard

⇝ "Overlap-Layout-Consensus" assemblers

1. produce best pairwise overlaps
2. layout the reads according to the overlaps
3. for each position, compute consensus base

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 2: Shortest Common Superstring is NP-hard
$\rightsquigarrow$ "Overlap-Layout-Consensus" assemblers
Problem: $\Theta(n^2)$ too slow in practice
$\rightsquigarrow$ DeBruijn-graph based assembly

Sequence Assembly: Overview

```
GCCCTGAACTTCGCTAGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
GCCCTGAACTT              CGACACTCCTTGGGTTTT              CTAGGCCATTGATTGCGGGTC
      ACTTCGC                                     GGTTCTCT                GGTCCAGGTGCTGTCAACGAC
         TCGCTAGGTTCTCTAACGA            TTTACGTCGCGG                                       CGAC
GCCCTGAACTTCGCTAGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC
```

Goal: reconstruct sequence

Idea: overlap reads

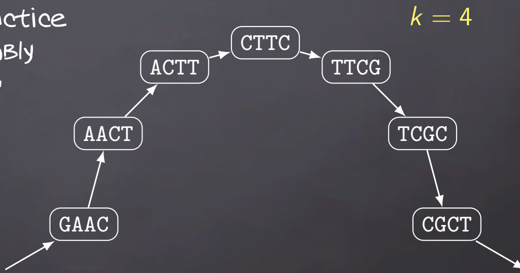
Problem 2: Shortest Common Superstring is NP-hard

⇝ "Overlap-Layout-Consensus" assemblers

Problem: $\Theta(n^2)$ too slow in practice

⇝ DeBruijn-graph based assembly

1. chop all reads into "k-mers"
2. builds overlap graph ("DeBruijn graph")
3. find path using all overlaps



Sequence Assembly: Overview

Goal: reconstruct sequence

Idea: overlap reads

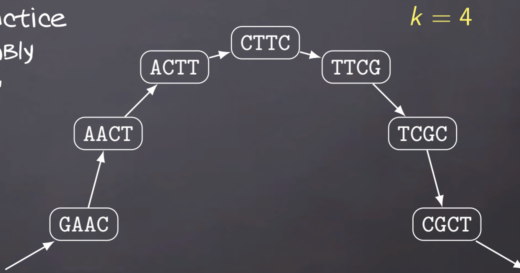
Problem 2: Shortest Common Superstring is NP-hard

⇝ "Overlap-Layout-Consensus" assemblers

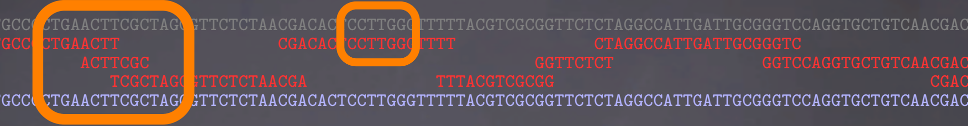
Problem: $\Theta(n^2)$ too slow in practice

⇝ DeBruijn-graph based assembly

1. chop all reads into "k-mers"
2. builds overlap graph ("DeBruijn graph")
3. find Eulerian path



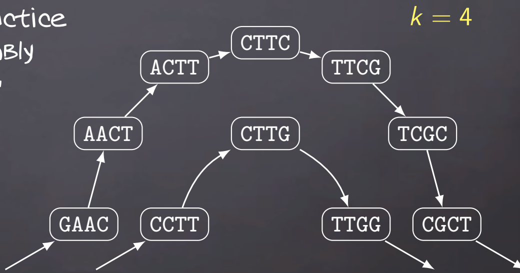
Sequence Assembly: Overview



Goal: reconstruct sequence
Idea: overlap reads

Problem 2: Shortest Common Superstring is NP-hard
⇝ "Overlap-Layout-Consensus" assemblers
Problem: $\Theta(n^2)$ too slow in practice
⇝ DeBruijn-graph based assembly

1. chop all reads into "k-mers"
2. builds overlap graph ("DeBruijn graph")
3. find Eulerian path

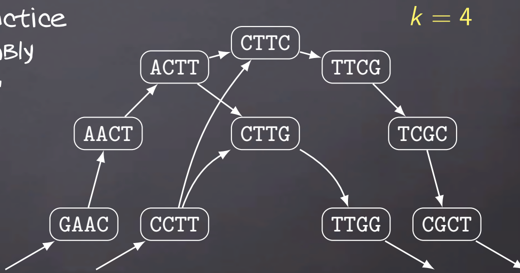


Sequence Assembly: Overview

Goal: reconstruct sequence
Idea: overlap reads

Problem 2: Shortest Common Superstring is NP-hard
⇝ "Overlap-Layout-Consensus" assemblers
Problem: $\Theta(n^2)$ too slow in practice
⇝ DeBruijn-graph based assembly

1. chop all reads into "k-mers"
2. builds overlap graph ("DeBruijn graph")
3. find Eulerian path



Sequence Assembly: Overview

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

Goal: reconstruct sequence

Idea: overlap reads

Problem 3: repeats (common in DNA) make assembly ambiguous

Sequence Assembly: Overview

GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC

GCCCCTGAACTT CGACACTCCTTGGGTTTT CTAGGCCATTGATTGCGGGTC

ACTTCGC GGTTCTCT GGTCCAGGTGCTGTCAACGAC

TCGCTAGGGTTCTCTAACGA TTTACGTCGCGG CGAC

GCCCCTGAACTTCGCTAGGGTTCTCTAACGACACTCCTTGGGTTTTACGTCGCGGTTCTCTAGGCCATTGATTGCGGGTCCAGGTGCTGTCAACGAC

Goal: reconstruct sequence

Idea: overlap reads

Problem 3: repeats (common in DNA) make assembly ambiguous

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 3: repeats (common in DNA) make assembly ambiguous

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 3: repeats (common in DNA) make assembly ambiguous
⇝ end product is a set of "contiguous regions"

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 3: repeats (common in DNA) make assembly ambiguous
⇝ end product is a set of "contiguous regions"
Problem: "contig soup" not very useful

Sequence Assembly: Overview

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

Goal: reconstruct sequence

Idea: overlap reads

Problem 3: repeats (common in DNA) make assembly ambiguous

⇝ end product is a set of "contiguous regions"

Problem: "contig soup" not very useful

But: we have paired-end information!

Sequence Assembly: Overview

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

Goal: reconstruct sequence
Idea: overlap reads

Problem 3: repeats (common in DNA) make assembly ambiguous
⇝ end product is a set of "contiguous regions"
Problem: "contig soup" not very useful
But: we have paired-end information!

Genome Scaffolding: Previous Work

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

- SOPRA [Dayarian, Michael, Sengupta, BMC Bioinf. 11, '10]
  - removes reads in high-coverage area (likely repeats)
  - orientation step (heuristic) + ordering step (heuristic)
  - coded in Pearl (!!!)
  - (observed sparse contig graph)

Genome Scaffolding: Previous Work

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

- SOPRA [Dayarian, Michael, Sengupta, BMC Bioinf. 11, '10]
- SSPACE [Boetzer & al., Bioinf. 27(4), '11]
  - heuristic contig extension
  - "reasonable time"

Genome Scaffolding: Previous Work

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

- SOPRA [Dayarian, Michael, Sengupta, BMC Bioinf. 11, '10]
- SSPACE [Boetzer & al., Bioinf. 27(4), '11]
- OPERA [Gao, Sung, Ngaraja, JCB. 18(11), '11]
  - $n^{p+O(1)}$ time ($p =$#edge-deletions ($\geq$ feedback edge set))
  - most work done by a heuristic "graph contraction"

Genome Scaffolding: Previous Work

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

- SOPRA [Dayarian, Michael, Sengupta, BMC Bioinf. 11, '10]
- SSPACE [Boetzer & al., Bioinf. 27(4), '11]
- OPERA [Gao, Sung, Ngaraja, JCB. 18(11), '11]
- GRASS [Gritsenko & al., Bioinf. 28(11), '12]
  - Mixed-Integer Quadratic Programming
  - deals with uncertain data (slack variables)

    ⇝ "intractable even for small # of contigs"
  - heuristic workaround:
    - solve relaxed formulation & use slack values ⇝ ILP

Genome Scaffolding: Previous Work

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

- SOPRA [Dayarian, Michael, Sengupta, BMC Bioinf. 11, '10]
- SSPACE [Boetzer & al., Bioinf. 27(4), '11]
- OPERA [Gao, Sung, Ngaraja, JCB. 18(11), '11]
- GRASS [Gritsenko & al., Bioinf. 28(11), '12]
- SCARPA [Donmez, Brudno, Bioinf. 29(4), '13]
  - orientation step: use FPT algo for Odd Cycle Transversal
  - ordering step: heuristic

Genome Scaffolding: Previous Work

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

- SOPRA [Dayarian, Michael, Sengupta, BMC Bioinf. 11, '10]
- SSPACE [Boetzer & al., Bioinf. 27(4), '11]
- OPERA [Gao, Sung, Ngaraja, JCB. 18(11), '11]
- GRASS [Gritsenko & al., Bioinf. 28(11), '12]
- SCARPA [Donmez, Brudno, Bioinf. 29(4), '13]
- ... [Huson & al., JACM, '02][Nieuwerburgh & al., NAR, '12]

Graph-Based Scaffolding

AACGACACTCCTTGGGTTTTTACGTCGCGG

GTTAATGTCCGAGCATAAAACTCTGGTTGGC

GTACTGAACTTGGGTTCCATAGGACCCAGA

CTAGGCCATTGATTGCGGGTCCAGGTGCTG

AGAGCTTGACAGTAACACATTTAGGAGCACGCG

Graph-Based Scaffolding

GTTAATGTCCGAGCATAAAACTCTGGTTGGC

AACGACACTCCTTGGGTTTTTACGTCGCGG

GTACTGAACTTGGGTTCCATAGGACCCAGA

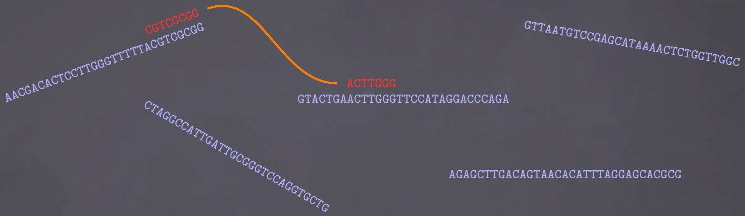
CTAGGCCATTGATTGCGGGTCCAGGTGCTG

AGAGCTTGACAGTAACACATTTAGGAGCACGCG

Strategy

1. map reads into contigs

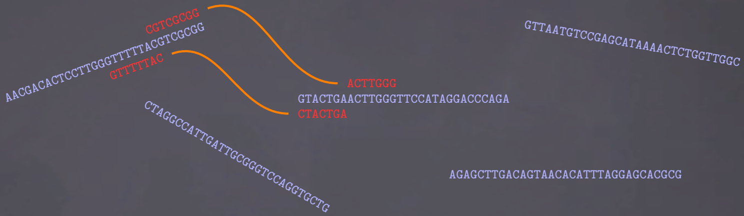
Graph-Based Scaffolding



Strategy

1. map reads into contigs

Graph-Based Scaffolding



Strategy

1. map reads into contigs

Graph-Based Scaffolding

AACGACACTCCTTGGGTTTTTACGTCGCGG

GTTAATGTCCGAGCATAAAACTCTGGTTGGC

CTAGGCCATTGATTGCGGGTCCAGGTGCTG

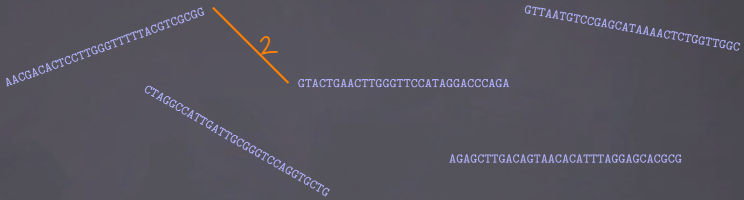
GTACTGAACTTGGGTTCCATAGGACCCAGA

AGAGCTTGACAGTAACACATTTAGGAGCACGCG

Strategy

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

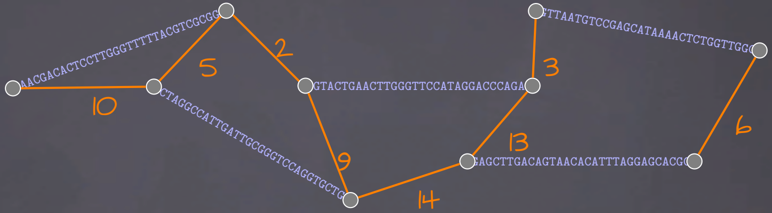
Graph-Based Scaffolding



Strategy

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

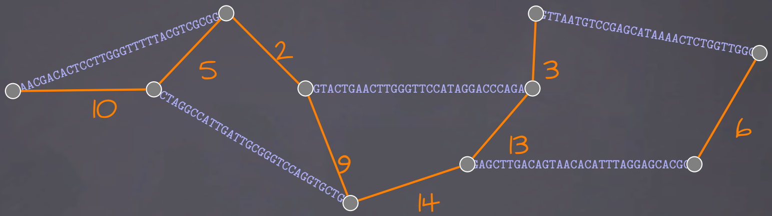
Graph-Based Scaffolding



Strategy

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

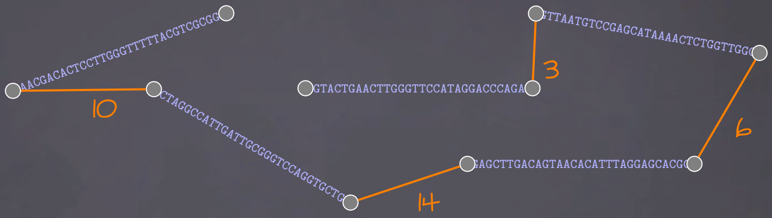
Graph-Based Scaffolding



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

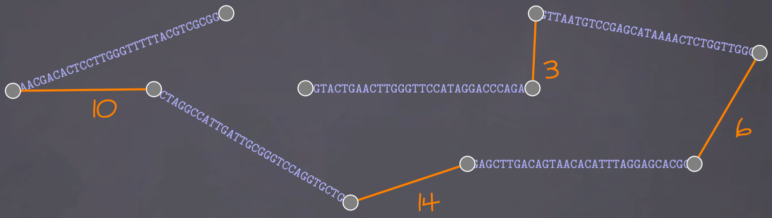
Graph-Based Scaffolding



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

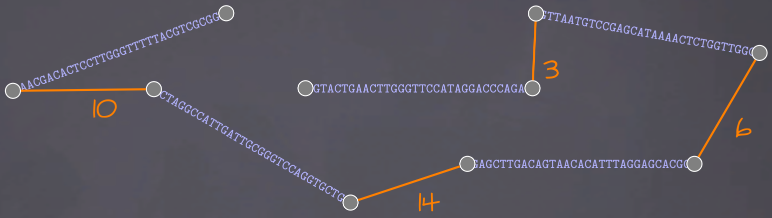


Scaffolding

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

Question: Can G 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, $k, \sigma_p, \sigma_c \in \mathbb{N}$

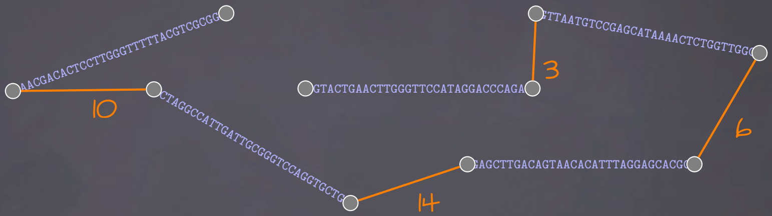
Question: Can G be covered by

$\leq \sigma_p$ alternating paths &

$\leq \sigma_c$ alternating cycles

of total weight $\geq k$?

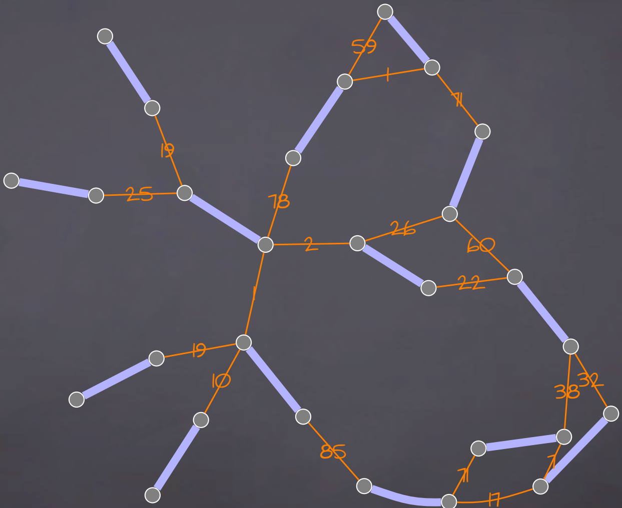
Graph-Based Scaffolding

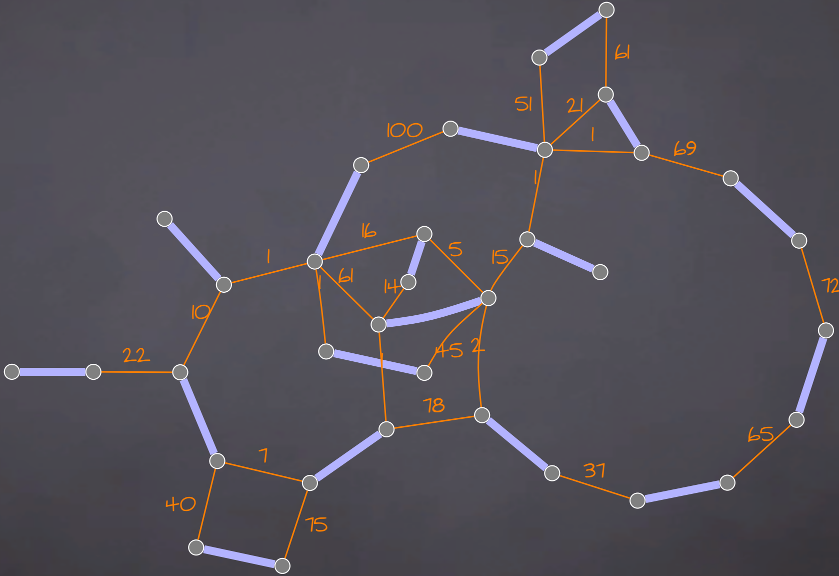


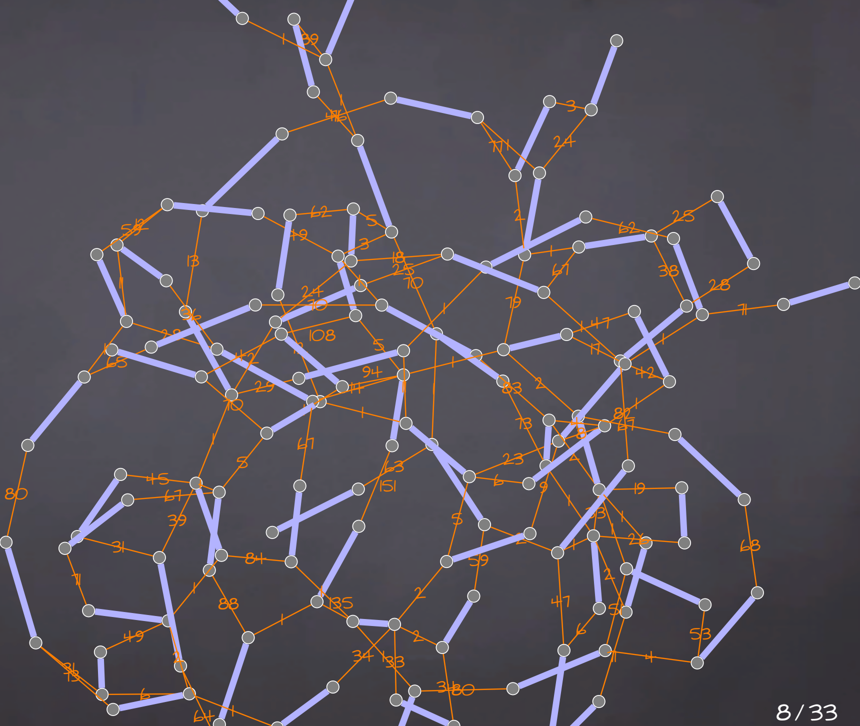
Exact Scaffolding

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

Question: Can G 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
38
32
85
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
65
40
31
75



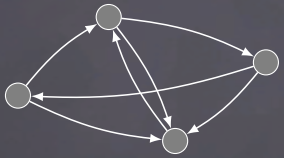
39
3
24
2
62
25
13
62
49
5
3
18
25
70
67
38
28
71
24
10
108
36
28
65
42
5
94
29
10
147
17
42
83
2
73
82
67
8
23
63
151
45
67
39
31
5
6
9
19
33
26
68
2
59
84
88
135
2
47
6
53
49
6
64
34
33
80
80

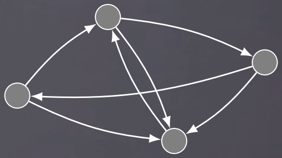
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

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

Construction

Given a directed graph D.

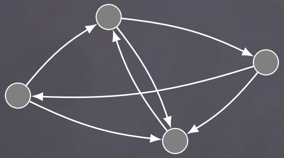
1. make a copy of D
2. duplicate all vertices $\rightsquigarrow M$

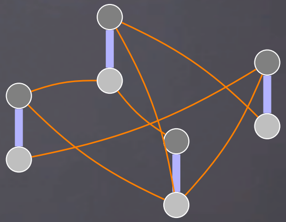
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

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





Construction

Given a directed graph D.

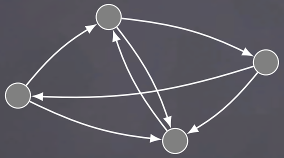
1. make a copy of D
2. duplicate all vertices $\rightsquigarrow M$
3. "slide" down all arrow tips & ignore directions

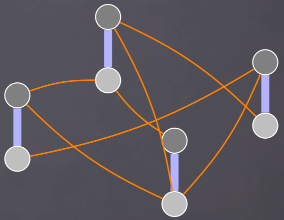
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

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





Lemma

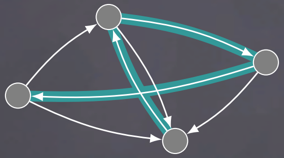
D admits a directed Hamiltonian path $\Leftrightarrow$ M can be covered with a single alternating path in G.

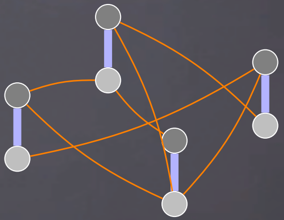
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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 $\Leftrightarrow$ M can be covered with a single alternating path in G.

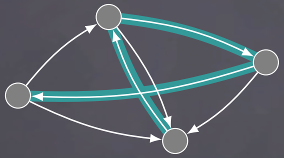
"$\Rightarrow$": replace each v in the Hamiltonian path by $v_{up} \to v_{low}$.

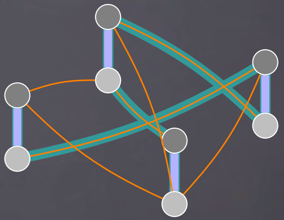
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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 $\Leftrightarrow$ M can be covered with a single alternating path in G.

"$\Rightarrow$": replace each v in the Hamiltonian path by $v_{up} \to v_{low}$.

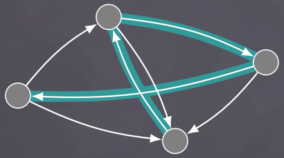
alternating ✓ covers M ✓

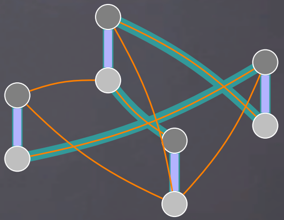
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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 $\Leftrightarrow$ M can be covered with a single alternating path in G.

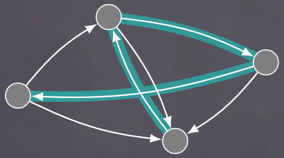
"$\Leftarrow$": contract each matching edge in the covering alternating path.

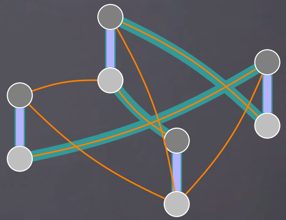
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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 $\Leftrightarrow$ M can be covered with a single alternating path in G.

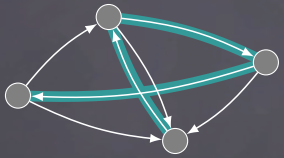
"$\Leftarrow$": contract each matching edge in the covering alternating path.
hits all vertices exactly once ✓ is valid directed path ✓

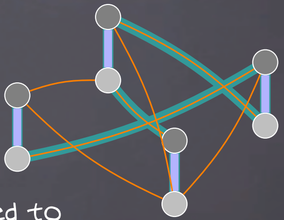
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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

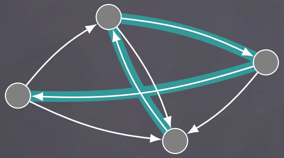
- bipartite graphs
- $(\sigma_p, \sigma_c) \in \{(0,1),(1,0)\}$ and
- $\omega : E \to \{0\}$.

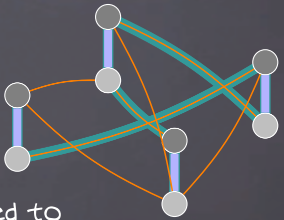
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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

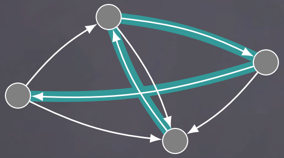
- supergraphs of bipartite graphs
- $(\sigma_p, \sigma_c) \in \{(0,1),(1,0)\}$ and
- $\omega : E \to \{0, 1\}$.

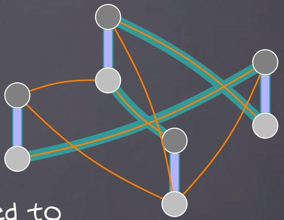
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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

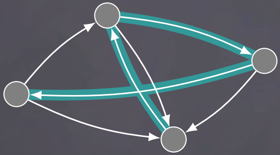
Scaffolding with 2 weights is NP-hard in any sufficiently dense graph class.

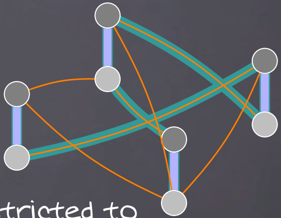
Hardness Warm up: Hamiltonian Path

Recall: Scaffolding

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

Question: Can G 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.

Scaffolding in Co-Bipartites

Wait, what?

Scaffolding in Co-Bipartites

Wait, what?

Recap: Corollary

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

Scaffolding in Co-Bipartites

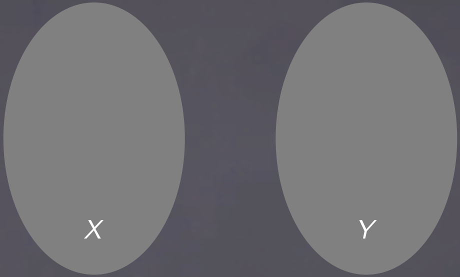
Wait, what?

Recap: Corollary

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

⇝ Unweighted!

Scaffolding in Unweighted Co-Bipartites



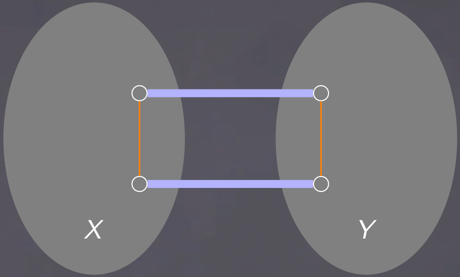
Observation

no edges between X & Y $\rightsquigarrow$ need 2 objects (paths/cycles)
otherwise $\rightsquigarrow$ can always cover G with 1 path

TODO

decide if we can cover with 1 cycle

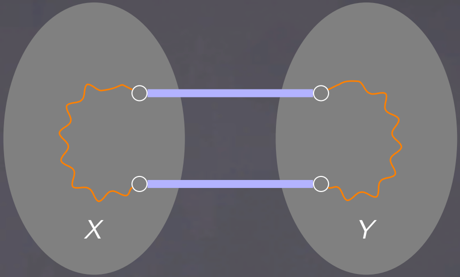
Scaffolding in Unweighted Co-Bipartites



Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

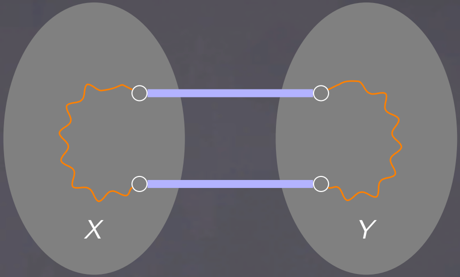
Scaffolding in Unweighted Co-Bipartites



Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Scaffolding in Unweighted Co-Bipartites



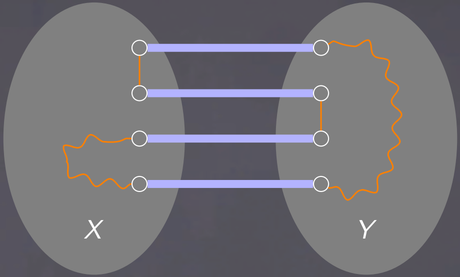
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓

Scaffolding in Unweighted Co-Bipartites



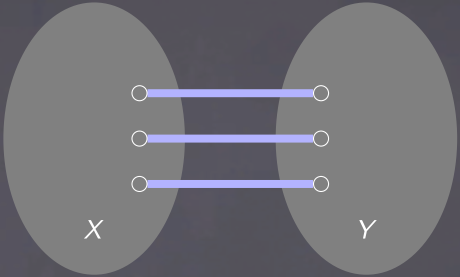
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓

Scaffolding in Unweighted Co-Bipartites



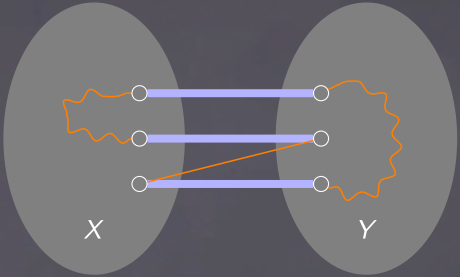
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd

Scaffolding in Unweighted Co-Bipartites



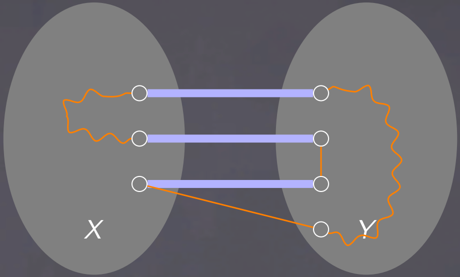
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y

Scaffolding in Unweighted Co-Bipartites



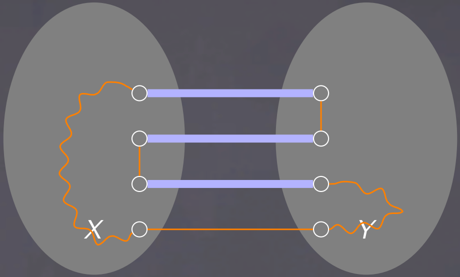
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y

Scaffolding in Unweighted Co-Bipartites



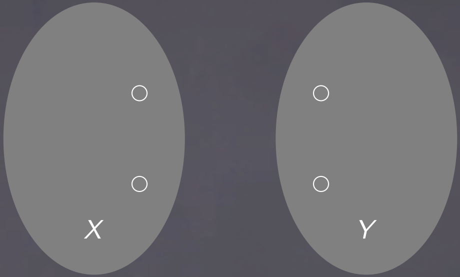
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y

Scaffolding in Unweighted Co-Bipartites



Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y
#matching edges between X & Y is 0

Scaffolding in Unweighted Co-Bipartites

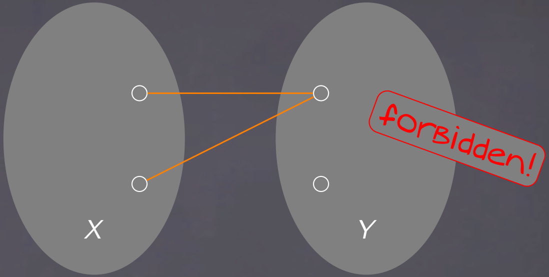
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y
#matching edges between X & Y is 0

Scaffolding in Unweighted Co-Bipartites



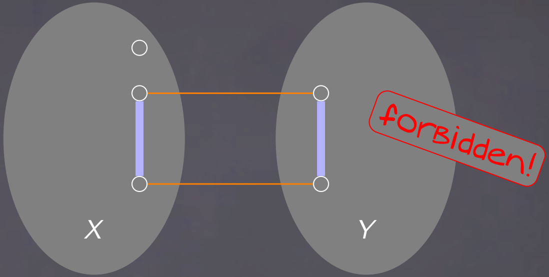
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y
#matching edges between X & Y is 0

Scaffolding in Unweighted Co-Bipartites



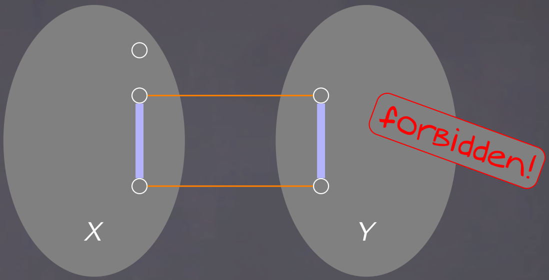
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y
#matching edges between X & Y is 0

Scaffolding in Unweighted Co-Bipartites



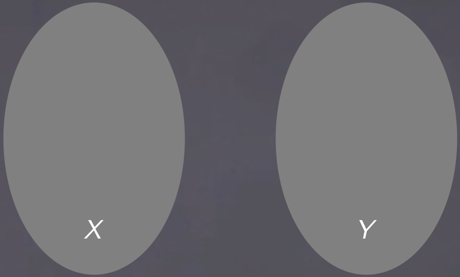
Observation

$\exists$ alternating cycle with non-matching edge X
$\rightsquigarrow$ extend to cover all $\mathcal{M}$ in $G[X]$

Observation

#matching edges between X & Y even (and > 0) $\rightsquigarrow$ ✓
#matching edges between X & Y odd
$\rightsquigarrow$ find any non-matching edge between X & Y
#matching edges between X & Y is 0
all other cases are ✓ (tedious case analysis)

Scaffolding in Unweighted Co-Bipartites



Theorem

Scaffolding can be solved in $O(n+m)$ time on co-bipartite graphs

Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ

Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched

Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ

$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched

Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child

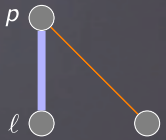
Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child



Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child

Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

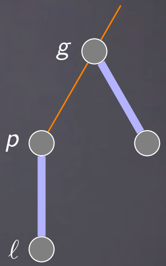
consider a lowest leaf ℓ

$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched

parent p of ℓ has only 1 child

Case 1

parent g of p is matched "below"



Scaffolding in Unweighted Trees

Observation

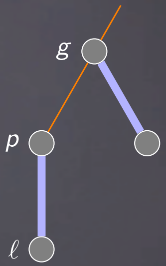
no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child

Case 1

parent g of p is matched "below"
$\rightsquigarrow$ g is matched to a leaf ℓ'



Scaffolding in Unweighted Trees

Observation

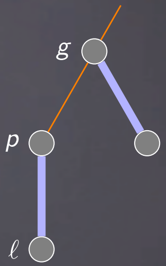
no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child

Case 1

parent g of p is matched "below"
$\rightsquigarrow$ g is matched to a leaf ℓ'
$\rightsquigarrow$ always take $\ell - p - g - \ell'$



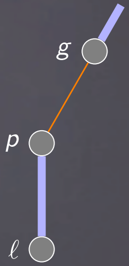
Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\leadsto$ ℓ matched
parent p of ℓ has only 1 child



Case 1

parent g of p is matched "below"
$\leadsto$ g is matched to a leaf ℓ'
$\leadsto$ always take $\ell - p - g - \ell'$

Case 2

parent g of p is matched "above"

Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ

$\mathcal{M}$ is perfect $\leadsto$ ℓ matched

parent p of ℓ has only 1 child

Case 1

parent g of p is matched "below"

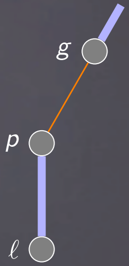
$\leadsto$ g is matched to a leaf ℓ'

$\leadsto$ always take $\ell - p - g - \ell'$

Case 2

parent g of p is matched "above"

either p is the only child of g



Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child

Case 1

parent g of p is matched "below"
$\rightsquigarrow$ g is matched to a leaf ℓ'
$\rightsquigarrow$ always take $\ell - p - g - \ell'$

Case 2

parent g of p is matched "above"
either p is the only child of g $\rightsquigarrow$ delete ℓ & g and reduce k

Scaffolding in Unweighted Trees

Observation

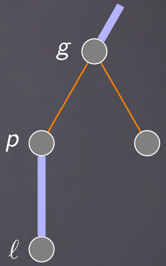
no alternating cycles in a tree

Observation

consider a lowest leaf ℓ

$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched

parent p of ℓ has only 1 child



Case 1

parent g of p is matched "below"

$\rightsquigarrow$ g is matched to a leaf ℓ'

$\rightsquigarrow$ always take $\ell - p - g - \ell'$

Case 2

parent g of p is matched "above"

either p is the only child of g $\rightsquigarrow$ delete ℓ & g and reduce k

or g has another child u

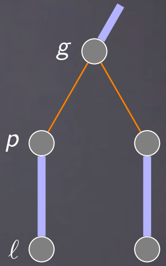
Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\leadsto$ ℓ matched
parent p of ℓ has only 1 child



Case 1

parent g of p is matched "below"
$\leadsto$ g is matched to a leaf ℓ'
$\leadsto$ always take $\ell - p - g - \ell'$

Case 2

parent g of p is matched "above"
either p is the only child of g $\leadsto$ delete ℓ & g and reduce k
or g has another child u $\leadsto$ u matched "below"

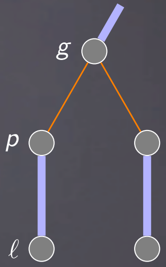
Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child



Case 1

parent g of p is matched "below"
$\rightsquigarrow$ g is matched to a leaf ℓ'
$\rightsquigarrow$ always take $\ell - p - g - \ell'$

Case 2

parent g of p is matched "above"
either p is the only child of g $\rightsquigarrow$ delete ℓ & g and reduce k
or g has another child u $\rightsquigarrow$ u matched "below" $\rightsquigarrow$ $\exists$ "clone" of $g - p - \ell$

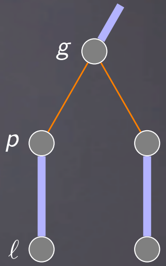
Scaffolding in Unweighted Trees

Observation

no alternating cycles in a tree

Observation

consider a lowest leaf ℓ
$\mathcal{M}$ is perfect $\rightsquigarrow$ ℓ matched
parent p of ℓ has only 1 child



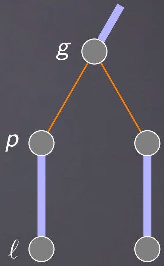
Case 1

parent g of p is matched "below"
$\rightsquigarrow$ g is matched to a leaf ℓ'
$\rightsquigarrow$ always take $\ell - p - g - \ell'$

Case 2

parent g of p is matched "above"
either p is the only child of g $\rightsquigarrow$ delete ℓ & g and reduce k
or g has another child u $\rightsquigarrow$ u matched "below" $\rightsquigarrow$ $\exists$ "clone" of $g - p - \ell$
$\rightsquigarrow$ take $p - \ell$

Scaffolding in Unweighted Trees



Theorem

Scaffolding can be solved in $O(n)$ time on unweighted trees

Scaffolding in Weighted Trees

Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Scaffolding in Weighted Trees

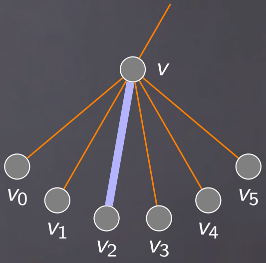
Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[p, x]_v$ = max. weight collected below v with p finished paths "under x"



Scaffolding in Weighted Trees

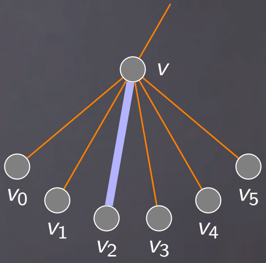
Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[p, x]_v$ = max. weight collected below v with p finished paths "under x"



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$$[p, \text{✗}]_v := \max_{\substack{p_1, p_2, \ldots, p_c \\ \sum p_i = p}} \sum_{1 \le i \le c} \max_{x \in \{\text{✓}, \text{✗}\}} [p_i, x]_{v_i}$$

Scaffolding in Weighted Trees

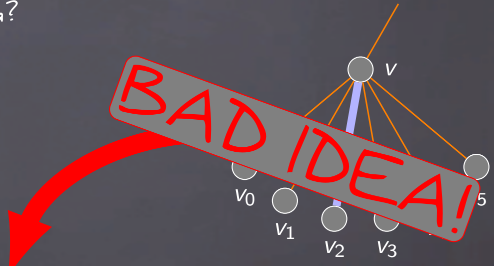
Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[p, x]_v$ = max. weight collected below v with p finished paths "under x"



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$$[p, \times]_v := \max_{\substack{p_1, p_2, \ldots, p_c \\ \sum p_i = p}} \sum_{1 \le i \le c} \max_{x \in \{\checkmark, \times\}} [p_i, x]_{v_i}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

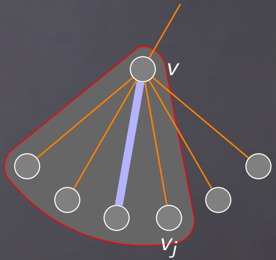
Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)

Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.



Scaffolding in Weighted Trees

Dynamic Programming Idea

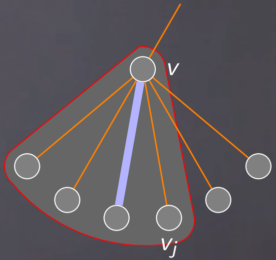
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, \cancel{x}]_v := 0$

$$[j, p, x]_v := \max_{p_j \leq p} \left\{ \right.$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

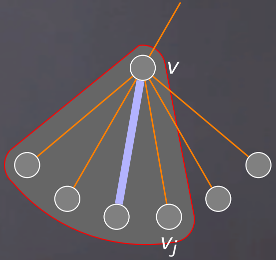
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow$ $[p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$$[0, 0, \times]_v := 0$$

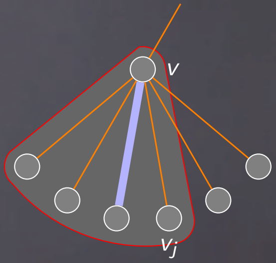
$$[j, p, x]_v := \max_{p_j \leq p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \times]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?



Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j
(abbrev: last child $\rightsquigarrow$ $[p, x]_v$)

Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, \text{✗}]_v := 0$

$$[j, p, x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \text{✗}]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, \text{✗}]_{v_j} + [j-1, p-p_j, \text{✗}]_v & \text{if } x = \checkmark \,\&\, vv_j \notin \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

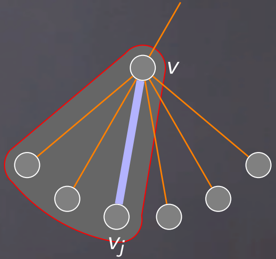
bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, ✗]_v := 0$

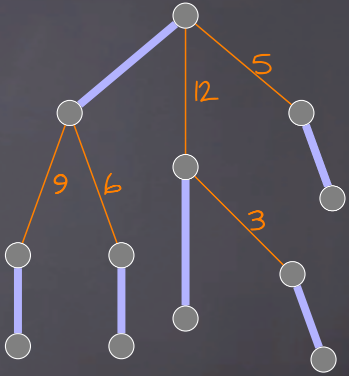
$$[j, p, x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, ✓]_{v_j}, [p_j, ✗]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, ✗]_{v_j} + [j-1, p-p_j, ✗]_v & \text{if } x = ✓ \ \& \ vv_j \notin \mathcal{M} \\ \left\{\begin{matrix} [p_j - 1, ✗]_{v_j} \\ [p_j - 1, ✓]_{v_j} \end{matrix}\right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?



Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j
(abbrev: last child $\rightsquigarrow [p, x]_v$)

Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$$[0, 0, \times]_v := 0$$

$$[j, p, x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \times]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, \times]_{v_j} + [j-1, p-p_j, \times]_v & \text{if } x = \checkmark \text{ \& } vv_j \notin \mathcal{M} \\ \left\{\begin{matrix} [p_j - 1, \times]_{v_j} \\ [p_j - 1, \checkmark]_{v_j} \end{matrix}\right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

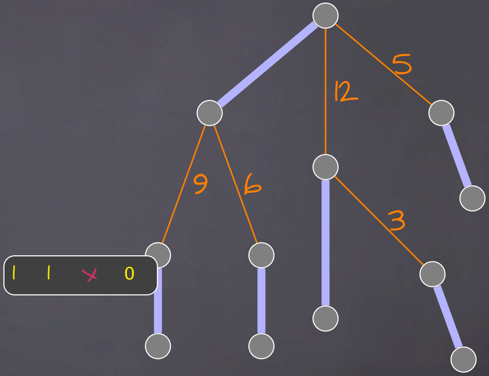
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, \times]_v := 0$

$$[j, p, x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \times]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, \times]_{v_j} + [j-1, p-p_j, \times]_v & \text{if } x = \checkmark \wedge vv_j \notin \mathcal{M} \\ \left\{ \begin{matrix} [p_j - 1, \times]_{v_j} \\ [p_j - 1, \checkmark]_{v_j} \end{matrix} \right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

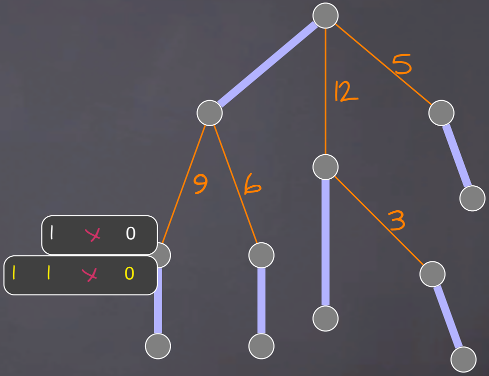
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, \times]_v := 0$

$$[j,p,x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \times]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, \times]_{v_j} + [j-1, p-p_j, \times]_v & \text{if } x = \checkmark \;\&\; vv_j \notin \mathcal{M} \\ \left\{ \begin{matrix} [p_j - 1, \times]_{v_j} \\ [p_j - 1, \checkmark]_{v_j} \end{matrix} \right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

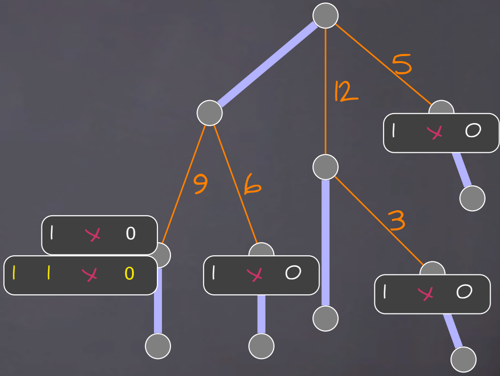
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow$ $[p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, ✗]_v := 0$

$$[j,p,x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, ✗]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j+1, ✗]_{v_j} + [j-1, p-p_j, ✗]_v & \text{if } x = \checkmark \nLeftarrow vv_j \notin \mathcal{M} \\ \left\{\begin{matrix} [p_j-1, ✗]_{v_j} \\ [p_j-1, \checkmark]_{v_j} \end{matrix}\right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

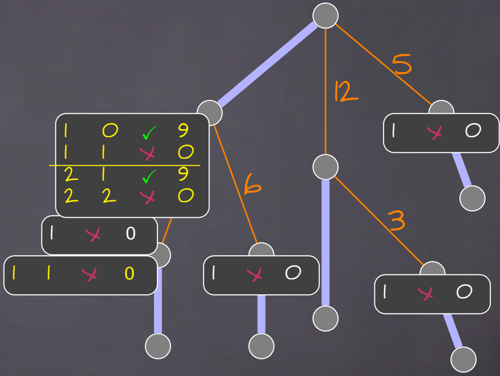
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, \times]_v := 0$

$$[j, p, x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \times]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, \times]_{v_j} + [j-1, p-p_j, \times]_v & \text{if } x = \checkmark \wedge vv_j \notin \mathcal{M} \\ \begin{Bmatrix} [p_j - 1, \times]_{v_j} \\ [p_j - 1, \checkmark]_{v_j} \end{Bmatrix} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

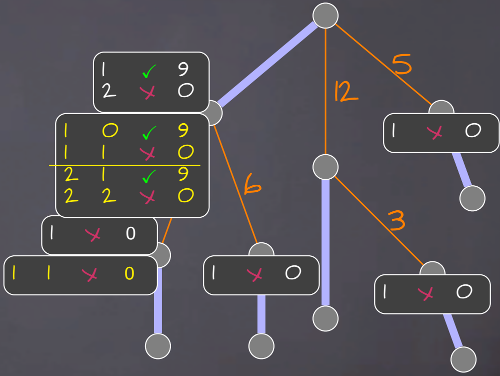
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow$ $[p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, ✗]_v := 0$

$$[j, p, x]_v := \max_{p_j \leq p} \begin{cases} \max\{[p_j, ✓]_{v_j}, [p_j, ✗]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, ✗]_{v_j} + [j-1, p-p_j, ✗]_v & \text{if } x = ✓ \;\&\; vv_j \notin \mathcal{M} \\ \left\{ \begin{matrix} [p_j - 1, ✗]_{v_j} \\ [p_j - 1, ✓]_{v_j} \end{matrix} \right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

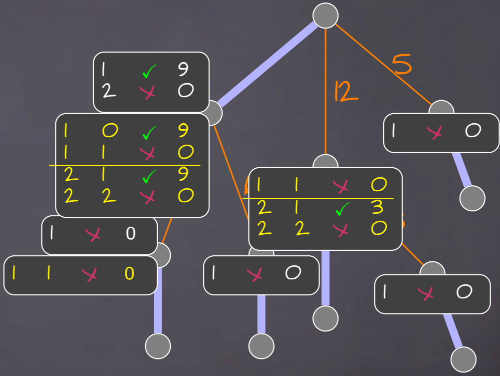
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)



Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, \text{✗}]_v := 0$

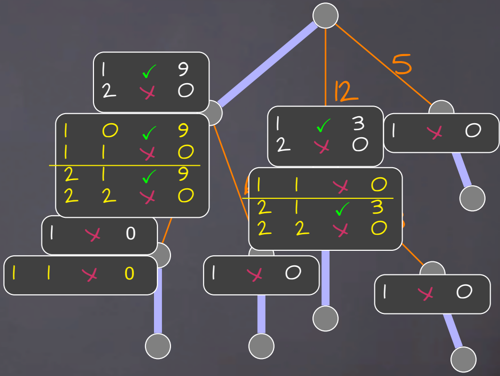
$$[j, p, x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \text{✗}]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, \text{✗}]_{v_j} + [j-1, p-p_j, \text{✗}]_v & \text{if } x = \checkmark \Leftarrow vv_j \notin \mathcal{M} \\ \left\{ \begin{matrix} [p_j - 1, \text{✗}]_{v_j} \\ [p_j - 1, \checkmark]_{v_j} \end{matrix} \right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?



Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)

Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, \times]_v := 0$

$$[j,p,x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, \times]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j+1, \times]_{v_j} + [j-1, p-p_j, \times]_v & \text{if } x = \checkmark \,\&\, vv_j \notin \mathcal{M} \\ \left\{ \begin{matrix} [p_j - 1, \times]_{v_j} \\ [p_j - 1, \checkmark]_{v_j} \end{matrix} \right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees

Dynamic Programming Idea

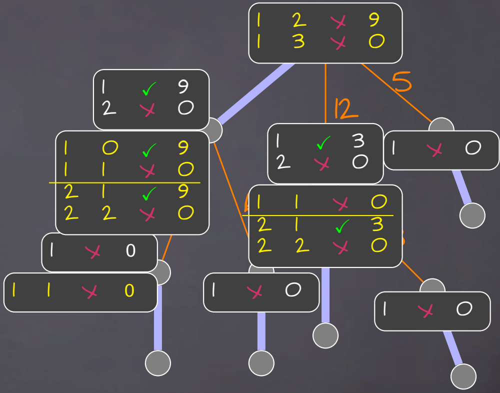
Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow$ $[p, x]_v$)



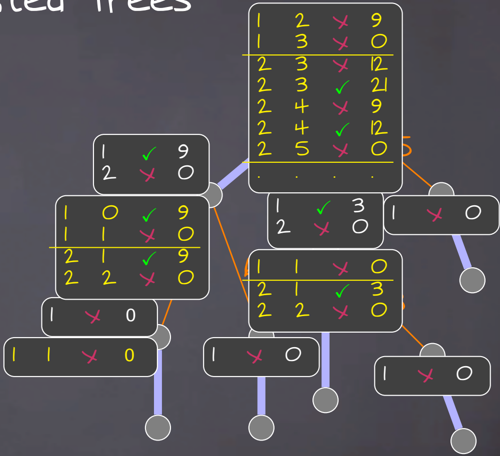
Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, ✗]_v := 0$

$$[j, p, x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, \checkmark]_{v_j}, [p_j, ✗]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j + 1, ✗]_{v_j} + [j-1, p-p_j, ✗]_v & \text{if } x = \checkmark \notin vv_j \notin \mathcal{M} \\ \left\{ \begin{matrix} [p_j - 1, ✗]_{v_j} \\ [p_j - 1, \checkmark]_{v_j} \end{matrix} \right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

Scaffolding in Weighted Trees



Dynamic Programming Idea

Bottom-up traversal; in each vertex v, need to remember:

- #paths used below v
- v incident with non-matching?

Semantics

$[j, p, x]_v$ = max. weight collected below v with p finished paths "under x" up to v_j

(abbrev: last child $\rightsquigarrow [p, x]_v$)

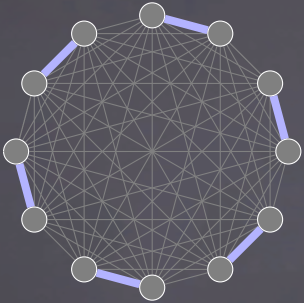
Recurrence

Let $v_1, v_2, \ldots, v_c$ be the children of v.

$[0, 0, ✗]_v := 0$

$$[j,p,x]_v := \max_{p_j \le p} \begin{cases} \max\{[p_j, ✓]_{v_j}, [p_j, ✗]_{v_j}\} + [j-1, p-p_j, x]_v & \text{if } vv_j \notin \mathcal{M} \\ \omega(vv_j) + [p_j+1, ✗]_{v_j} + [j-1, p-p_j, ✗]_v & \text{if } x = ✓ \ \&\ vv_j \notin \mathcal{M} \\ \left\{ \begin{matrix} [p_j-1, ✗]_{v_j} \\ [p_j-1, ✓]_{v_j} \end{matrix} \right\} + [j-1, p-p_j, x]_v & \text{if } vv_j \in \mathcal{M} \end{cases}$$

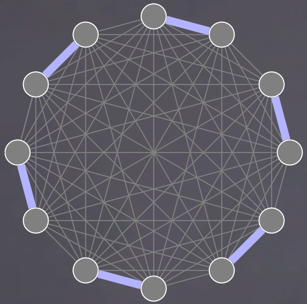
3-Approximation in Dense Graphs



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

Approximate Scaffolding

3-Approximation in Dense Graphs

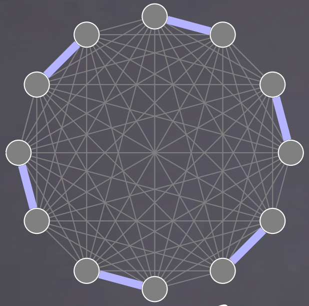


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight

3-Approximation in Dense Graphs

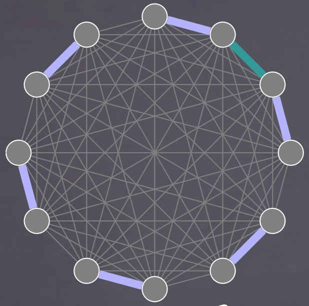


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

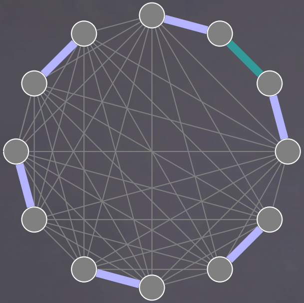


$\sigma_p = 1$, $\sigma_c = 1?$

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

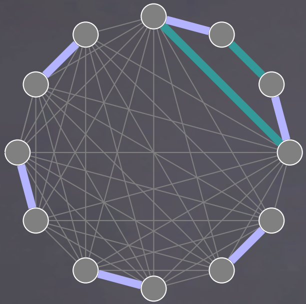


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

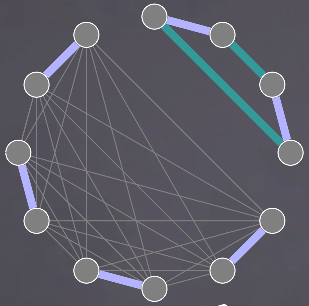


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

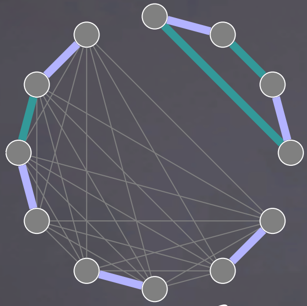


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

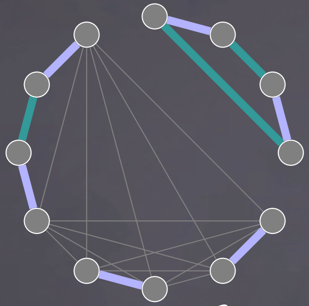


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

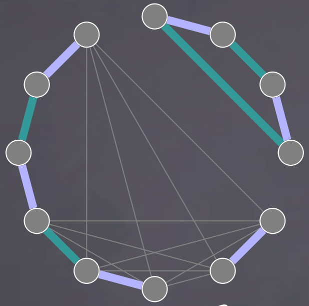
3-Approximation in Dense Graphs



Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

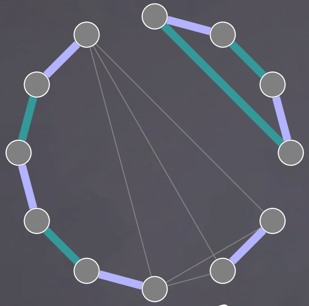


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

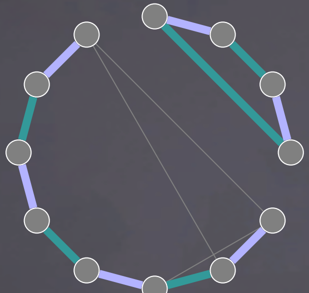
3-Approximation in Dense Graphs



Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

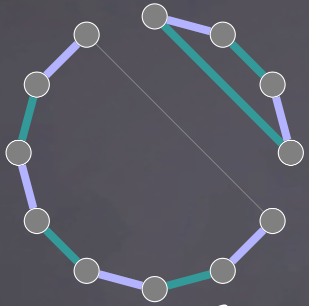


$\sigma_p = 1$, $\sigma_c = 1$?

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs

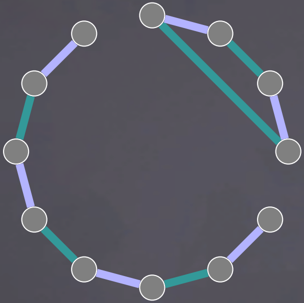


$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

3-Approximation in Dense Graphs



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

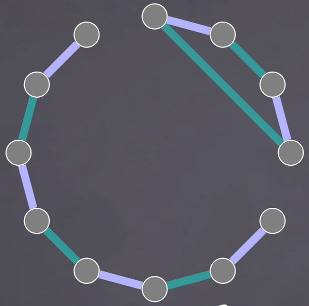
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

Proof

Result S^* is a valid solution ✓

3-Approximation in Dense Graphs



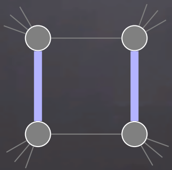
$\sigma_p = 1, \sigma_c = 1?$

Approximate Scaffolding

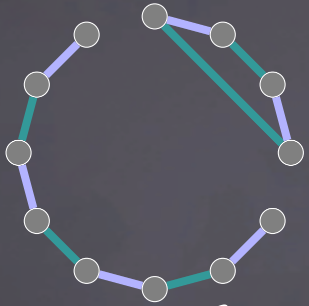
1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

Proof

Result S^* is a valid solution ✓
Note: taking an edge forbids ≤ 3 OPT edges



3-Approximation in Dense Graphs



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

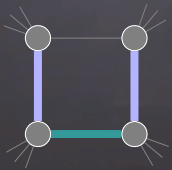
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

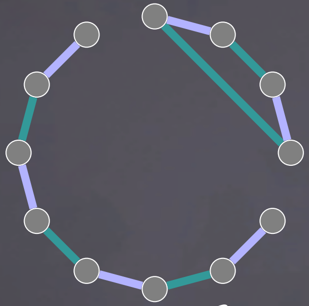
Proof

Result S^* is a valid solution ✓

Note: taking an edge forbids ≤ 3 OPT edges



3-Approximation in Dense Graphs



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

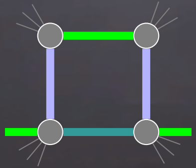
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

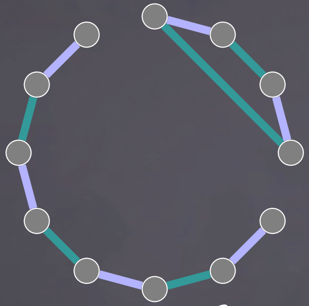
Proof

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 edge, if possible

Proof

Result S^* is a valid solution ✓

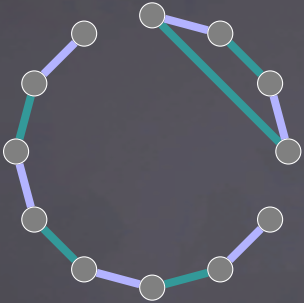
Note: taking an edge forbids ≤ 3 OPT edges

$\rightsquigarrow$ 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



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

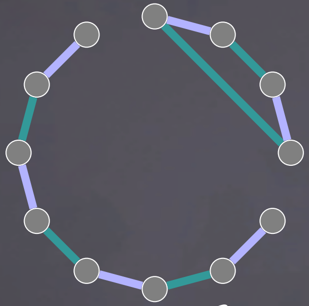
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

Theorem

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

3-Approximation in Dense Graphs



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

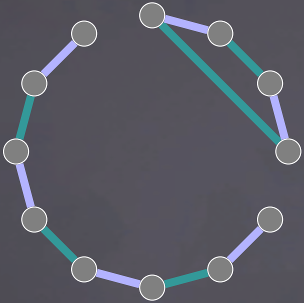
Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

Theorem

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

3-Approximation in Dense Graphs



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

Approximate Scaffolding

1. sort all edges by weight
2. repeatedly take heaviest edge, if possible

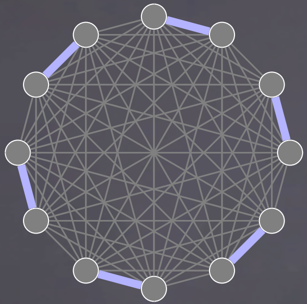
Theorem

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

Remark

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

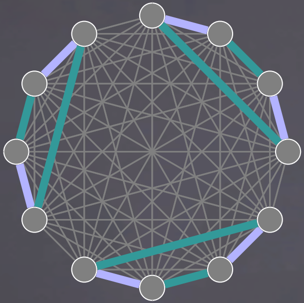
2-Approximation in Dense Graphs



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

Approximate Exact Scaffolding

2-Approximation in Dense Graphs

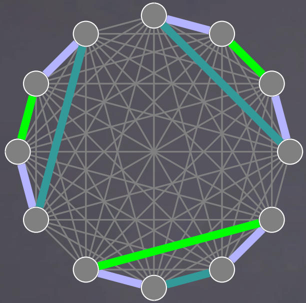


$\sigma_p = 1, \sigma_c = 1?$

Approximate Exact Scaffolding

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

2-Approximation in Dense Graphs

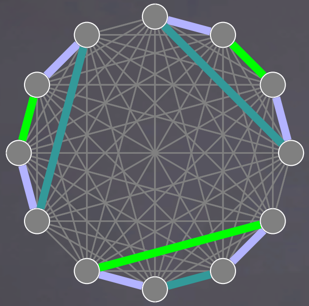


$\sigma_p = 1, \sigma_c = 1?$

Approximate Exact 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

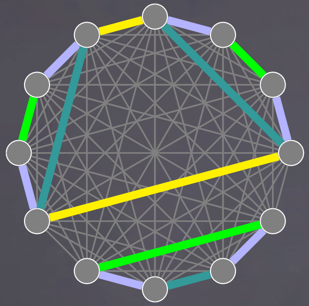


$\sigma_p = 1, \sigma_c = 1?$

Approximate Exact 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

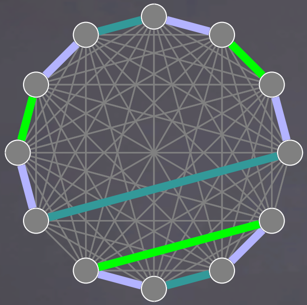


$\sigma_p = 1, \sigma_c = 1?$

Approximate Exact 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

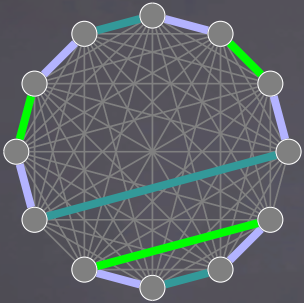


$\sigma_p = 1, \sigma_c = 1?$

Approximate Exact 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

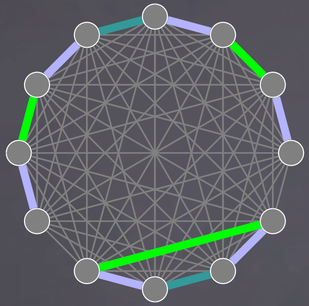


$\sigma_p = 1, \sigma_c = 1?$

Approximate Exact 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

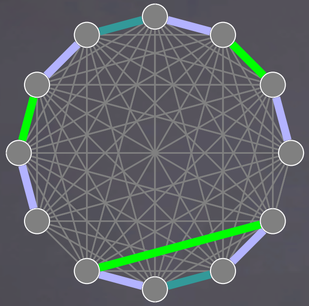


$\sigma_p = 1, \sigma_c = 1?$

Approximate Exact 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



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

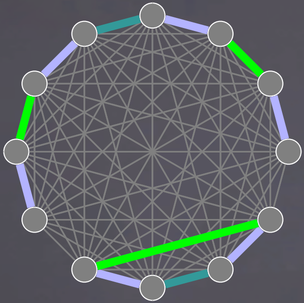
Approximate Exact Scaffolding

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 ✓

2-Approximation in Dense Graphs



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

Approximate Exact Scaffolding

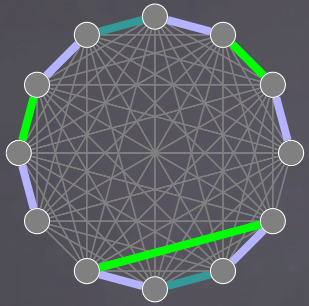
1. compute max.-weight perfect matching S
   $\rightsquigarrow S \cup \mathcal{M}$ is collection of cycles
2. "fix" all but lightest edge per cycle
3. repeatedly flip any lightest non-fix 4-cycle intersecting 2 cycles until at most $\sigma_c + \sigma_p$ cycles remain
4. repeatedly remove lightest non-fix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

Proof

Result S^* is a valid solution ✓

$\omega(S^*) \geq \omega(\mathit{fix}) \geq {}^{\omega(S)}/_2 \geq {}^{OPT}/_2$

2-Approximation in Dense Graphs



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

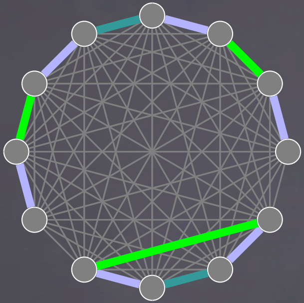
Approximate Exact Scaffolding

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

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

2-Approximation in Dense Graphs



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

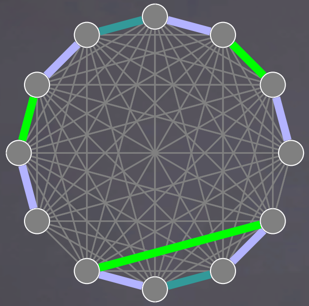
Approximate Exact Scaffolding

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

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

2-Approximation in Dense Graphs



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

Approximate Exact 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

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

Remark

For Scaffolding, replace Step 3 by either merging cycles or removing lightest edge, whatever looses less weight

Exact Algorithms I: Brute Force

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

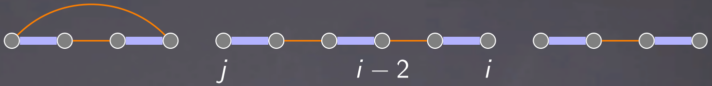
Exact Algorithms I: Brute Force

$[p, c, j]_i :=$ max. weight collectible before v_i with p & c paths/cycles plus one path starting at v_j

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

Exact Algorithms I: Brute Force



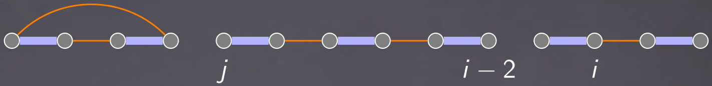
$[p, c, j]_i :=$ max. weight collectible before v_i with p & c paths/cycles plus one path starting at v_j

$$[p, c, j]_i = [p, c, j]_{i-2} + \omega(v_{i-2}v_{i-1}) \quad \text{if } j < i-2 \text{ \& } v_{i-2}v_{i-1} \in E$$

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

Exact Algorithms I: Brute Force



$[p, c, j]_i :=$ max. weight collectible before v_i with p & c paths/cycles plus one path starting at v_j

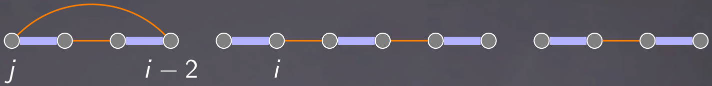
$$[p, c, j]_i = [p, c, j]_{i-2} + \omega(v_{i-2} v_{i-1}) \quad \text{if } j < i-2 \;\&\; v_{i-2} v_{i-1} \in E$$

$$[p, c, i-1]_i = \max_{\substack{j < i-2 \\ j \text{ even}}} \begin{cases} [p-1, c, j]_{i-2} \\ \end{cases}$$

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

Exact Algorithms I: Brute Force



$[p, c, j]_i :=$ max. weight collectible before v_i with p & c paths/cycles plus one path starting at v_j

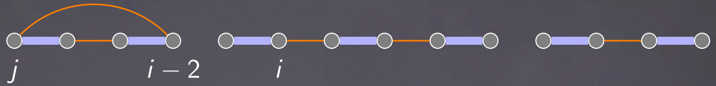
$$[p, c, j]_i = [p, c, j]_{i-2} + \omega(v_{i-2}v_{i-1}) \quad \text{if } j < i-2 \text{ \& } v_{i-2}v_{i-1} \in E$$

$$[p, c, i-1]_i = \max_{\substack{j<i-2 \\ j \text{ even}}} \begin{cases} [p-1, c, j]_{i-2} \\ [p, c-1, j]_{i-2} + \omega(v_j v_{i-2}) & \text{if } v_j v_{i-2} \in E \end{cases}$$

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

Exact Algorithms I: Brute Force



$[p,c,j]_i :=$ max. weight collectible before v_i with p & c paths/cycles plus one path starting at v_j

$$[p,c,j]_i = [p,c,j]_{i-2} + \omega(v_{i-2}v_{i-1}) \quad \text{if } j < i-2 \text{ \& } v_{i-2}v_{i-1} \in E$$

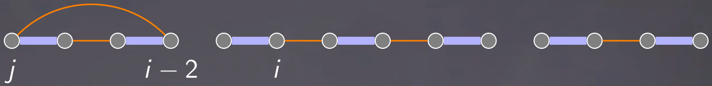
$$[p,c,i-1]_i = \max_{\substack{j<i-2 \\ j \text{ even}}} \begin{cases} [p-1,c,j]_{i-2} \\ [p,c-1,j]_{i-2} + \omega(v_j v_{i-2}) & \text{if } v_j v_{i-2} \in E \end{cases}$$

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

$\rightsquigarrow$ try all $O(n!)$ certificates

Exact Algorithms I: Brute Force



$[p, c, j]_i :=$ max. weight collectible before v_i with p & c paths/cycles plus one path starting at v_j

$$[p, c, j]_i = [p, c, j]_{i-2} + \omega(v_{i-2}v_{i-1}) \quad \text{if } j < i-2 \text{ \& } v_{i-2}v_{i-1} \in E$$

$$[p, c, i-1]_i = \max_{\substack{j < i-2 \\ j \text{ even}}} \begin{cases} [p-1, c, j]_{i-2} \\ [p, c-1, j]_{i-2} + \omega(v_j v_{i-2}) & \text{if } v_j v_{i-2} \in E \end{cases}$$

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

$\rightsquigarrow$ try all $O(n!)$ certificates

contigs force every other vertex $\rightsquigarrow$ $O(n!!)$

Exact Algorithms I: Brute Force

$[p,c,j]_i :=$ max. weight collectible before v_i with p & c paths/cycles plus one path starting at v_j

$$[p,c,j]_i = [p,c,j]_{i-2} + \omega(v_{i-2}v_{i-1}) \quad \text{if } j < i-2 \text{ \& } v_{i-2}v_{i-1} \in E$$

$$[p,c,i-1]_i = \max_{\substack{j<i-2 \\ j \text{ even}}} \begin{cases} [p-1,c,j]_{i-2} \\ [p,c-1,j]_{i-2} + \omega(v_j v_{i-2}) & \text{if } v_j v_{i-2} \in E \end{cases}$$

Observation

An ordering of $V(G)$ certifies YES-instances of Scaffolding.

$\rightsquigarrow$ try all $O(n!)$ certificates

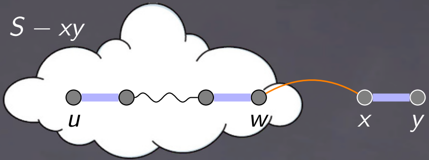
contigs force every other vertex $\rightsquigarrow O(\sqrt{2}^n \cdot n/2!)$

Exact Algorithms II: Dynamic Programming

Semantics

$[S, p, c, u, v]$ = max. weight collectible in $G[S]$ by p alt. paths, c alt. cycles and an alt. path starting at u & ending at v

Exact Algorithms II: Dynamic Programming



Semantics

$[S, p, c, u, v]$ = max. weight collectible in $G[S]$ by p alt. paths, c alt. cycles and an alt. path starting at u & ending at v

Computation

Let $xy \in \mathcal{M}$. Then, $[\{xy\}, 0, 0, x, y] := 0$ and

$$[S, p, c, u, y] := \max_{\substack{w \in G[S-xy] \\ u \neq w}} [S - xy, p, c, u, w] + \omega(wx)$$

Exact Algorithms II: Dynamic Programming

Semantics

$[S, p, c, u, v]$ = max. weight collectible in $G[S]$ by p alt. paths, c alt. cycles and an alt. path starting at u & ending at v

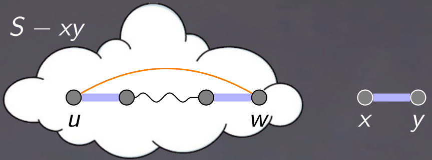
Computation

Let $xy \in \mathcal{M}$. Then, $[\{xy\}, 0, 0, x, y] := 0$ and

$$[S, p, c, u, y] := \max_{\substack{w \in G[S-xy] \\ u \neq w}} [S - xy, p, c, u, w] + \omega(wx)$$

$$[S, p, c, x, y] := \max_{u,w \in G[S-xy]} \begin{cases} [S - xy, p - 1, c, u, w] \\ \end{cases}$$

Exact Algorithms II: Dynamic Programming



Semantics

$[S, p, c, u, v]$ = max. weight collectible in $G[S]$ by p alt. paths, c alt. cycles and an alt. path starting at u & ending at v

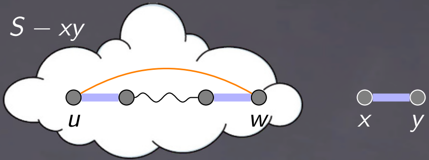
Computation

Let $xy \in \mathcal{M}$. Then, $[\{xy\}, 0, 0, x, y] := 0$ and

$$[S, p, c, u, y] := \max_{\substack{w \in G[S-xy] \\ u \neq w}} [S - xy, p, c, u, w] + \omega(wx)$$

$$[S, p, c, x, y] := \max_{u,w \in G[S-xy]} \begin{cases} [S - xy, p - 1, c, u, w] \\ [S - xy, p, c - 1, u, w] + \omega(wu) & \text{if } wu \in E(G) \setminus \mathcal{M} \end{cases}$$

Exact Algorithms II: Dynamic Programming



Semantics

$[S, p, c, u, v]$ = max. weight collectible in $G[S]$ by p alt. paths, c alt. cycles and an alt. path starting at u & ending at v

Theorem

Scaffolding can be solved in $O(\sqrt{2}^n n^3 \sigma_p \sigma_c)$ time.

Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

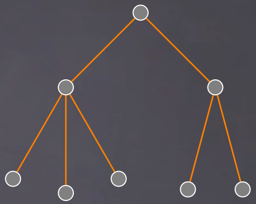
Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

A Shot at Sparsity

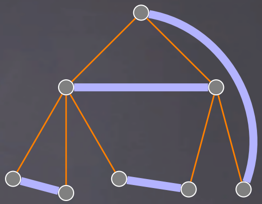
G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest



Sparse Graphs: Quasi-Forest

Recall

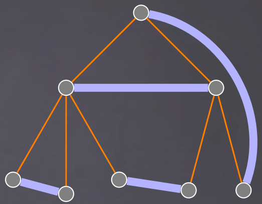
- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees



A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Sparse Graphs: Quasi-Forest



Recall

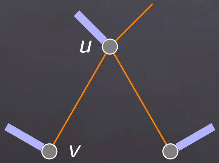
- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

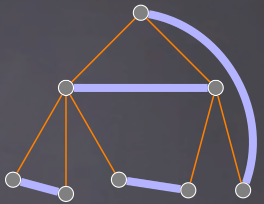
Each leaf v of $G - \mathcal{M}$ has degree 2 in G
$\rightsquigarrow$ if unweighted, can we take both?



Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

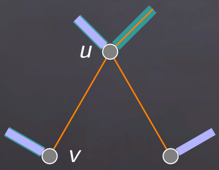


A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

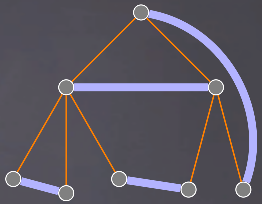
Each leaf v of $G - \mathcal{M}$ has degree 2 in G
$\rightsquigarrow$ if unweighted, can we take both?



Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

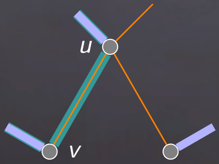


A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

Each leaf v of $G - \mathcal{M}$ has degree 2 in G
$\rightsquigarrow$ if unweighted, can we take both?



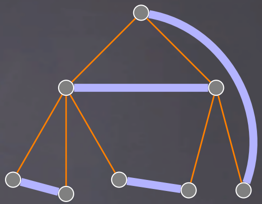
Observation

- v in path & u in cycle $\rightsquigarrow$ 1 path ✓
- v in path & u in path $\rightsquigarrow$ 2 paths ✓

Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

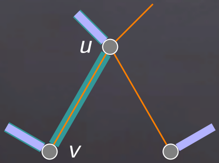


A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

Each leaf v of $G - \mathcal{M}$ has degree 2 in G
$\rightsquigarrow$ if unweighted, can we take both?



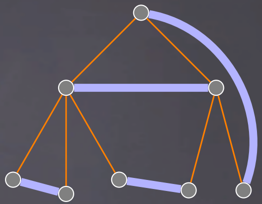
Observation

- v in path & u in cycle $\rightsquigarrow$ 1 path ✓
- v in path & u in path $\rightsquigarrow$ 2 paths ✓
  unless it's the same path!

Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

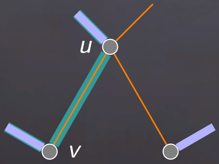


A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

Each leaf v of $G - \mathcal{M}$ has degree 2 in G
$\rightsquigarrow$ if unweighted, can we take both? ✗



Observation

- v in path & u in cycle $\rightsquigarrow$ 1 path ✓
- v in path & u in path $\rightsquigarrow$ 2 paths ✓

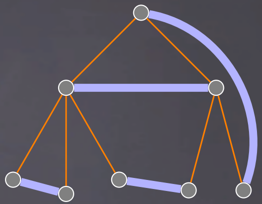
Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

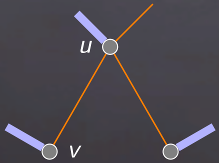
A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

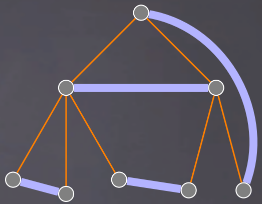


Observation

Each leaf v of $G - \mathcal{M}$ has degree 2 in G
$\rightsquigarrow$ if $\sigma_p = 0$, we have to take both!



Sparse Graphs: Quasi-Forest



Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

A Shot at Sparsity

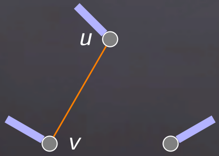
G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

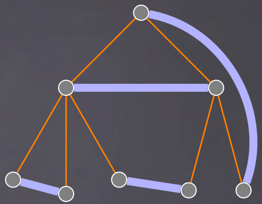
Each leaf v of $G - \mathcal{M}$ has degree 2 in G

$\rightsquigarrow$ if $\sigma_p = 0$, we have to take both!

$\rightsquigarrow$ remove all non-matching edges from parent u, except uv



Sparse Graphs: Quasi-Forest



Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

Each leaf v of $G - \mathcal{M}$ has degree 2 in G

$\rightsquigarrow$ if $\sigma_p = 0$, we have to take both!

$\rightsquigarrow$ remove all non-matching edges from parent u, except uv

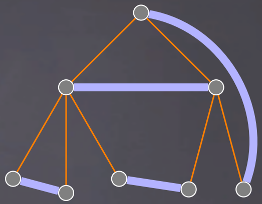
Corollary

Scaffolding can be solved in $O(n)$ on quasi-forests if $\sigma_p = 0$.

Sparse Graphs: Quasi-Forest

Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees



A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

Each leaf v of $G - \mathcal{M}$ has degree 2 in G

$\rightsquigarrow$ if $\sigma_p = 0$, we have to take both!

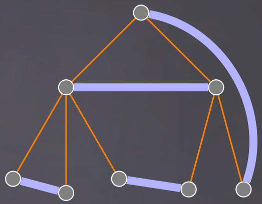
$\rightsquigarrow$ remove all non-matching edges from parent u, except uv

Corollary

Scaffolding can be solved in $O(n)$ on quasi-forests if $\sigma_p = 0$.

Scaffolding can be solved in $O(n^{2\sigma_p+1})$ in quasi-forests.

Sparse Graphs: Quasi-Forest



Recall

- Scaffolding is hard in any sufficiently dense graph class
- Scaffolding is easy in trees

A Shot at Sparsity

G is Quasi-forest $\Leftrightarrow$ $G - \mathcal{M}$ is forest

Observation

Each leaf v of $G - \mathcal{M}$ has degree 2 in G

$\rightsquigarrow$ if $\sigma_p = 0$, we have to take both!

$\rightsquigarrow$ remove all non-matching edges from parent u, except uv

Corollary

Scaffolding can be solved in $O(n)$ on quasi-forests if $\sigma_p = 0$.

Scaffolding can be solved in $O(n^{2\sigma_p+1})$ in quasi-forests.

But is it even NP-hard?

Sparse Graphs: Quasi-Forest

Weighted 2-SAT

Input: φ on X in 2-CNF, weights $\omega : X \times \{0,1\} \to \mathbb{N}$, $k \in \mathbb{N}$

Question: is there a satisfying assignment for φ of weight $\leq k$?

Sparse Graphs: Quasi-Forest

Weighted 2-SAT

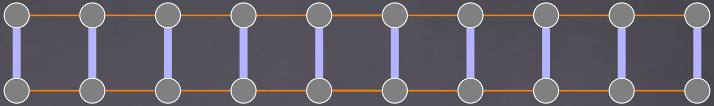
Input: φ on X in 2-CNF, weights $\omega : X \times \{0,1\} \to \mathbb{N}$, $k \in \mathbb{N}$

Question: is there a satisfying assignment for φ of weight $\leq k$?

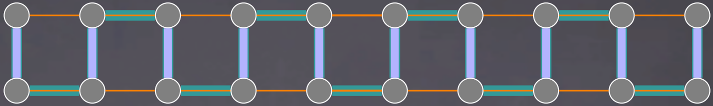
Remark

Independent Set is special case of Weighted 2-SAT

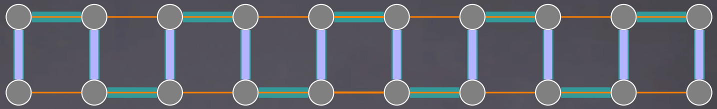
Sparse Graphs: Quasi-Forest



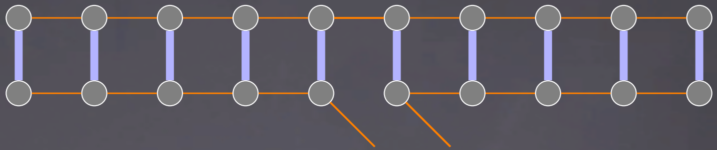
Sparse Graphs: Quasi-Forest



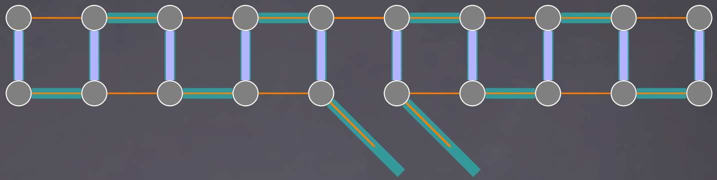
Sparse Graphs: Quasi-Forest



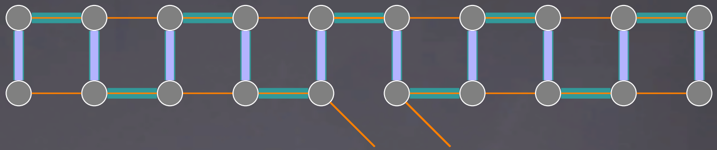
Sparse Graphs: Quasi-Forest



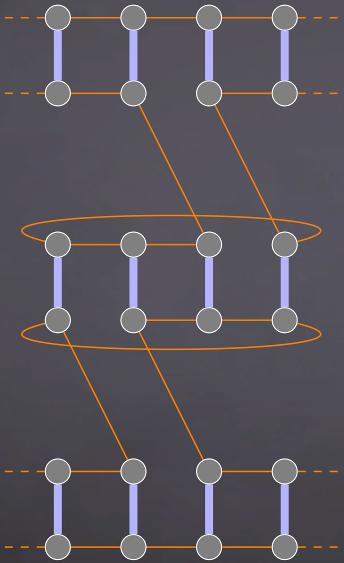
Sparse Graphs: Quasi-Forest



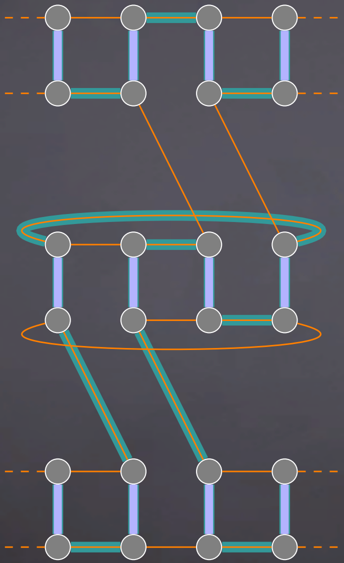
Sparse Graphs: Quasi-Forest



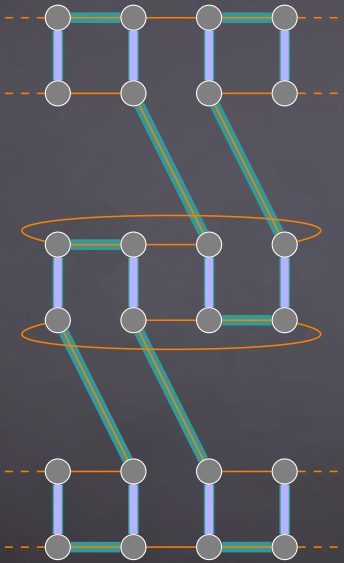
Sparse Graphs: Quasi-Forest



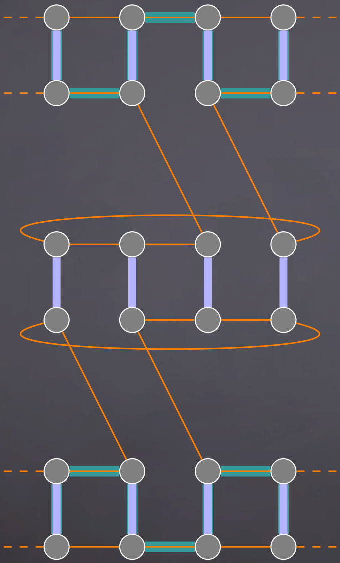
Sparse Graphs: Quasi-Forest



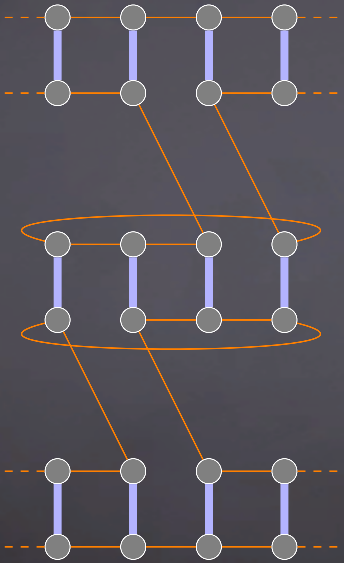
Sparse Graphs: Quasi-Forest



Sparse Graphs: Quasi-Forest



Sparse Graphs: Quasi-Forest



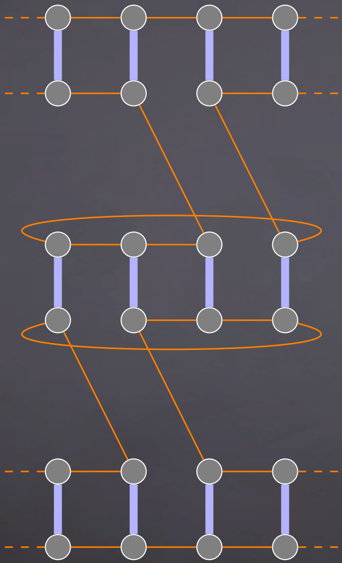
Observation

$\exists$ weight-k satisfying assignment

$\Leftrightarrow$

$\exists$ weight-k cover with $\leq n$ alternating paths

Sparse Graphs: Quasi-Forest



Observation

$\exists$ weight-k satisfying assignment

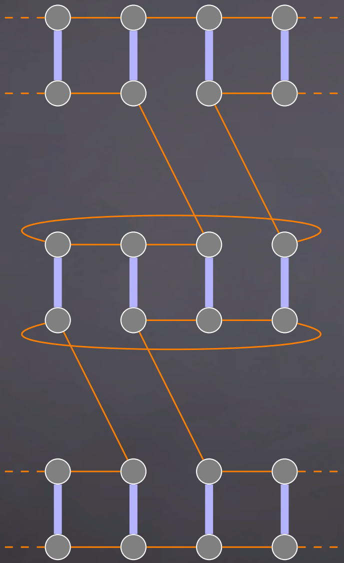
$\Leftrightarrow$

$\exists$ weight-k cover with $\leq n$ alternating paths

Theorem

Scaffolding is NP-hard even if $G - \mathcal{M}$ is a collection of paths with weights 0/1

Sparse Graphs: Quasi-Forest



Observation

$\exists$ weight-k satisfying assignment

$\Leftrightarrow$

$\exists$ weight-k cover with $\leq n$ alternating paths

Theorem

Scaffolding is NP-hard even if $G - \mathcal{M}$ is a collection of paths with weights 0/1

Corollary

no $2^{o(n+m)}$-time algorithm (ETH)
no $n^{o(k)}$-time algorithm (FPT$\neq$W[t])

Other Forms of Tree-Likeness

Tree Decompositions

tree T, each vertex i associated to some $X_i \subseteq V(G)$ s.t.

1. $\forall\, e \in E(G)$, there is some $i \in V(T)$ with $e \in X_i$
2. $\forall\, v \in V(G)$, bags containing v induce a connected subtree

treewidth $\mathrm{tw} =$ size of largest bag - 1

Other Forms of Tree-Likeness

Tree Decompositions

tree T, each vertex i associated to some $X_i \subseteq V(G)$ s.t.

1. $\forall e \in E(G)$, there is some $i \in V(T)$ with $e \in X_i$
2. $\forall v \in V(G)$, bags containing v induce a connected subtree

treewidth $\text{tw} =$ size of largest bag - 1

Hope

Practical instances of Scaffolding have low treewidth (they originate from linear structure)

Other Forms of Tree-Likeness

Tree Decompositions

tree T, each vertex i associated to some $X_i \subseteq V(G)$ s.t.

1. $\forall\, e \in E(G)$, there is some $i \in V(T)$ with $e \in X_i$
2. $\forall\, v \in V(G)$, bags containing v induce a connected subtree

treewidth $\mathrm{tw} =$ size of largest bag - 1

Hope

Practical instances of Scaffolding have low treewidth (they originate from linear structure)

Nice Decompositions

Leaf: $X = \varnothing$

Introduce v: i has single child j and $X_i \setminus X_j = \{v\}$

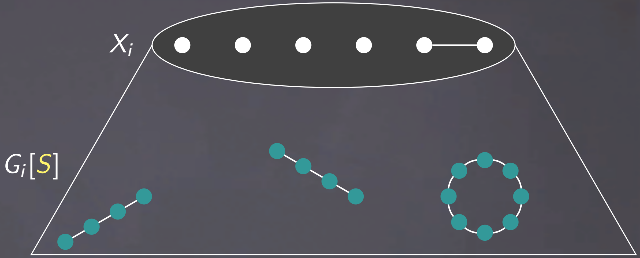
Forget v: i has single child j and $X_j \setminus X_i = \{v\}$

Introduce uv: i has single child j and $uv \subseteq X_i = X_j$

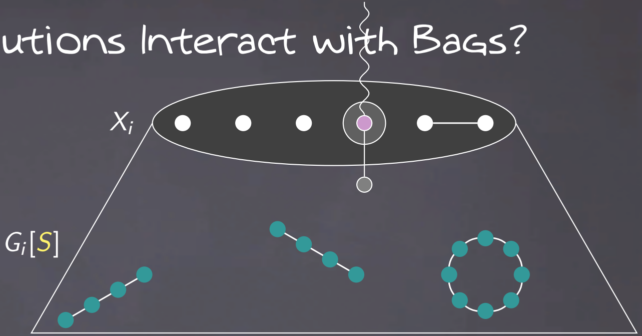
(each edge introduced exactly once)

Join: i has 2 children j and ℓ and $X_i = X_j = X_\ell$

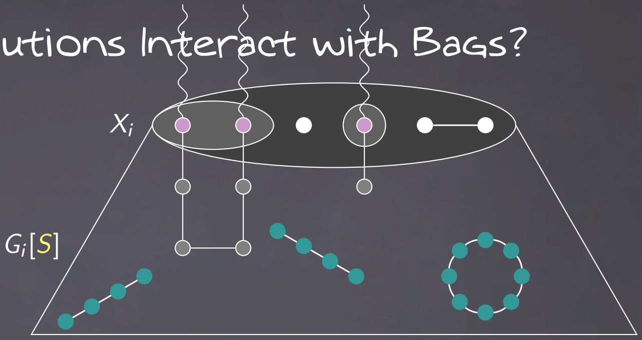
How do Solutions Interact with Bags?



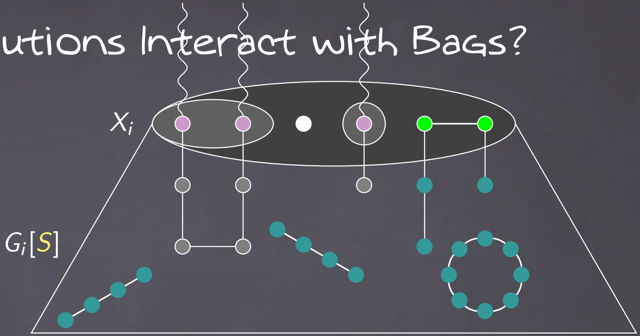
How do Solutions Interact with Bags?



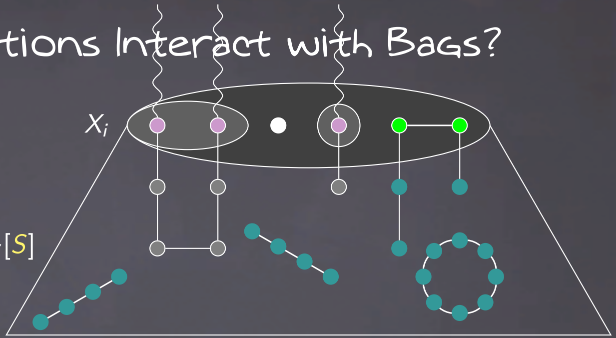
How do Solutions Interact with Bags?



How do Solutions Interact with Bags?



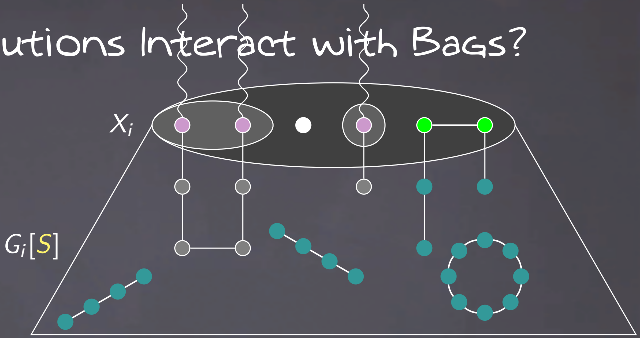
How do Solutions Interact with Bags?



Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X$
- #paths and #cycles completed "below the bag"

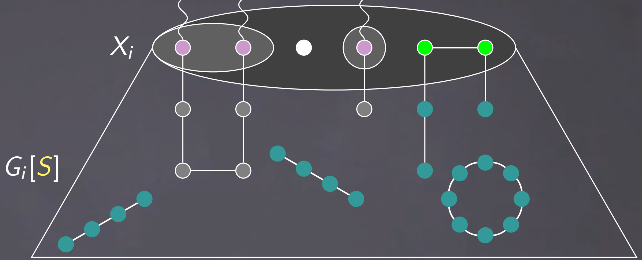
How do Solutions Interact with Bags?



Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

How do Solutions Interact with Bags?



Ingredients

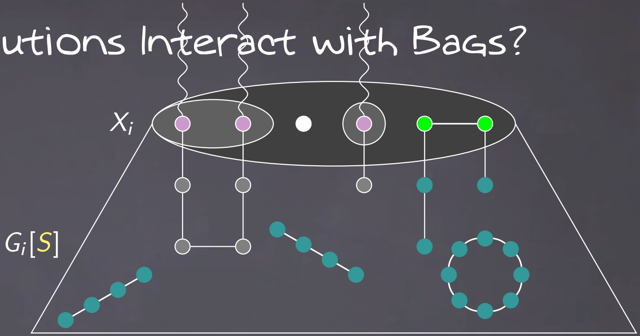
- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Semantics

$[d, P, p, c]_i =$ max. weight of any S with $\mathcal{M} \cap E(G_i) \subseteq S \subseteq E(G_i)$ and

1. each vertex $v \in X_i$ has degree $d(v)$ in $G_i[S]$,
2. for each $uv \in P$, $G_i[S]$ contains an alternating path…
   $u = v$: …from u avoiding $d^{-1}(1)$
   $u \neq v$: …from u to v
3. $G_i[S]$ contains p alt. paths & c alt. cycles avoiding $d^{-1}(1)$

How do Solutions Interact with Bags?



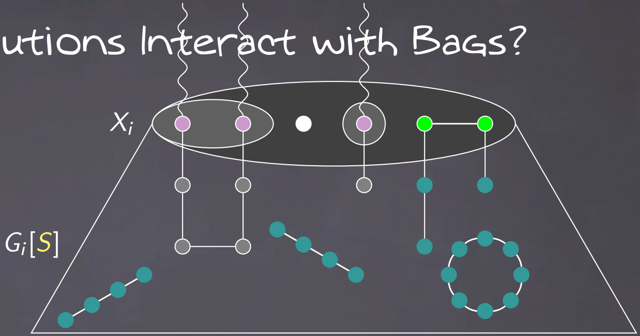
Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Leaf Bag

$[\varnothing, \varnothing, 0, 0]_i = 0$

How do Solutions Interact with Bags?



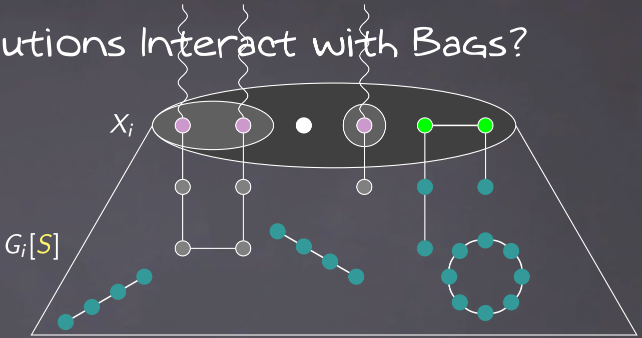
Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Introduce v (single child j)

$[d, P, p, c]_i = [d|_{v \to \bot}, P, p, c]_j$

How do Solutions Interact with Bags?

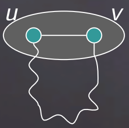


Ingredients

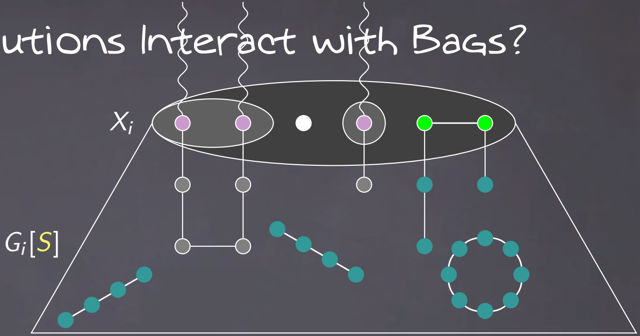
- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Introduce uv (single child j)

Case 1: $d(u) = d(v) = 2$



How do Solutions Interact with Bags?

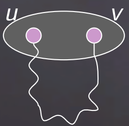


Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

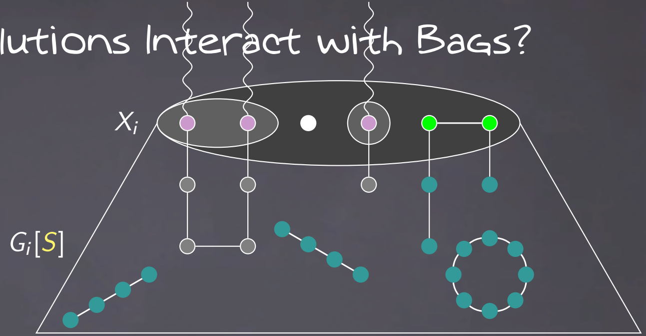
Introduce uv (single child j)

Case 1: $d(u) = d(v) = 2$



$$[d, P, p, c]_i = [d|_{u \to 1, v \to 1}, P + uv, p, c - 1]_j$$

How do Solutions Interact with Bags?



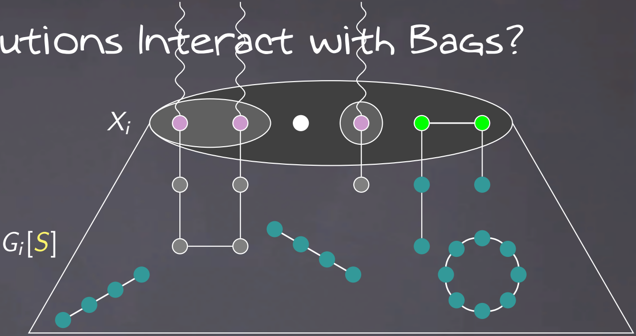
Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Introduce uv (single child j)

Case I: $d(u) = d(v) = 2$

How do Solutions Interact with Bags?



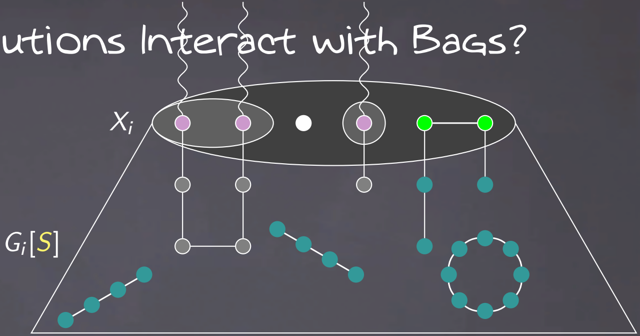
Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Forget v (single child j)

$$[d, P, p, c]_i = \max \left\{ \right.$$

How do Solutions Interact with Bags?



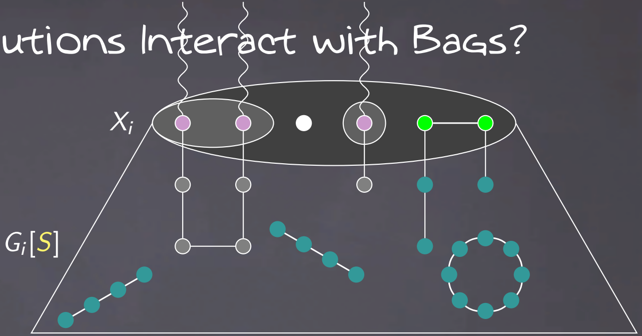
Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Forget v (single child j)

$$[d, P, p, c]_i = \max \begin{cases} [d|_{v\to 1}, P + vv, p - 1, c]_j \end{cases}$$

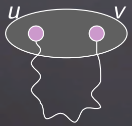
How do Solutions Interact with Bags?



Ingredients

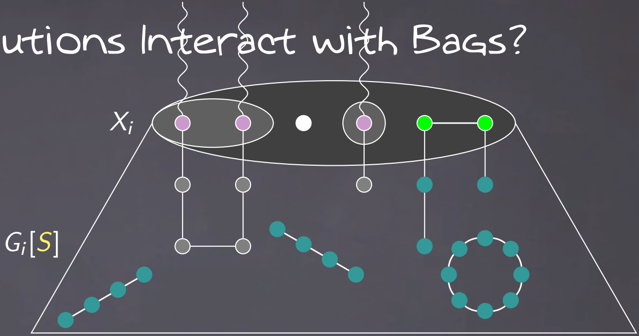
- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Forget v (single child j)



$$[d, P, p, c]_i = \max \begin{cases} [d|_{v \to 1}, P + vv, p - 1, c]_j \\ \max_{uu \in P} [d|_{v \to 1}, (P - uu) + uv, p, c]_j \end{cases}$$

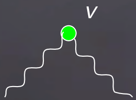
How do Solutions Interact with Bags?



Ingredients

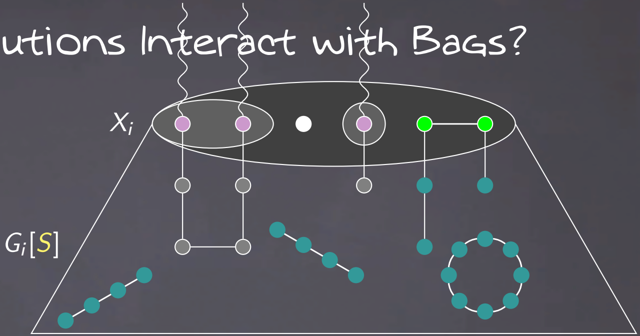
- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Forget v (single child j)



$$[d, P, p, c]_i = \max \begin{cases} [d|_{v\to 1}, P + vv, p - 1, c]_j \\ \max\limits_{uu \in P} [d|_{v\to 1}, (P - uu) + uv, p, c]_j \\ \max\limits_{x \in \{0,2\}} [d|_{v\to x}, P, p, c]_j \end{cases}$$

How do Solutions Interact with Bags?



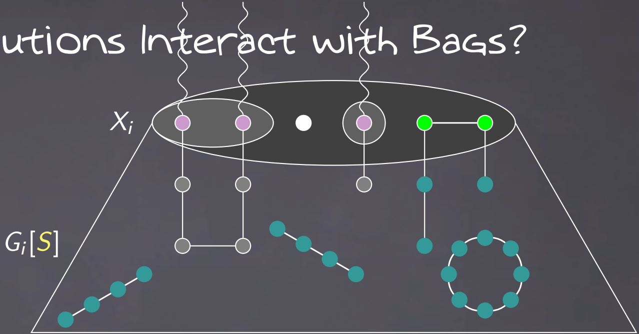
Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Join Bag (children j & ℓ)

$$[d, P, p, c]_i = \max_{d_j, P_j, p_j, c_j} \max_{\substack{P_\ell \\ P_j \sqcup P_\ell = P}} [d_j, P_j, p_j, c_j]_j + [d - d_j, P_\ell, p - p_j, c - c_j]_\ell$$

How do Solutions Interact with Bags?



Ingredients

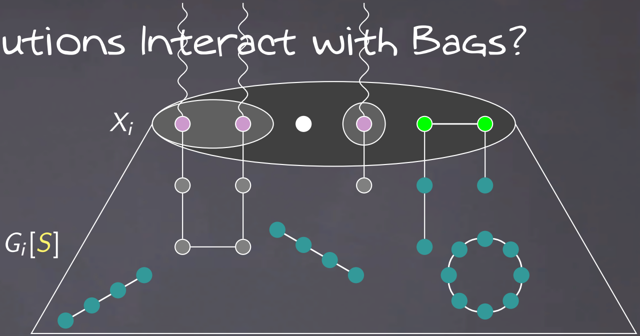
- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Join Bag (children j & ℓ)

$$[d, P, p, c]_i = \max_{d_j, P_j, p_j, c_j} \max_{\substack{P_\ell \\ P_j \sqcup P_\ell = P}} [d_j, P_j, p_j, c_j]_j + [d - d_j, P_\ell, p - p_j, c - c_j]_\ell$$

$\rightsquigarrow O(3^{\text{tw}} \cdot \text{tw}^{\text{tw}/2} \cdot \sigma_p \cdot \sigma_c)$ table entries

How do Solutions Interact with Bags?



Ingredients

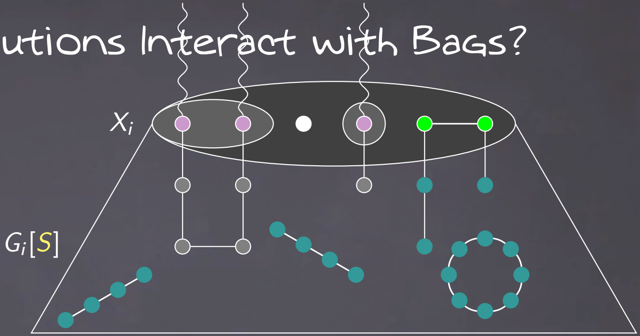
- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

Join Bag (children j & ℓ)

$$[d, P, p, c]_i = \max_{d_j, P_j, p_j, c_j} \max_{\substack{P_\ell \\ P_j \sqcup P_\ell = P}} [d_j, P_j, p_j, c_j]_j + [d - d_j, P_\ell, p - p_j, c - c_j]_\ell$$

$\rightsquigarrow O(2^{\text{tw}} \cdot \text{tw}^{\text{tw}/2} \cdot \sigma_p \cdot \sigma_c)$ table entries

How do Solutions Interact with Bags?



Ingredients

- degree-function $d : X \to \{0, 1, 2\}$
- "pairing" $\subseteq \binom{X}{2} \cup X \rightsquigarrow$ #matchings possibilities $\rightsquigarrow O(|X|^{|X|/2})$
- #paths and #cycles completed "below the bag"

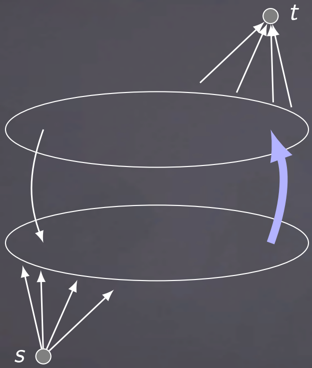
Join Bag (children j & ℓ)

$$[d, P, p, c]_i = \max_{d_j, P_j, p_j, c_j} \max_{\substack{P_\ell \\ P_j \sqcup P_\ell = P}} [d_j, P_j, p_j, c_j]_j + [d - d_j, P_\ell, p - p_j, c - c_j]_\ell$$

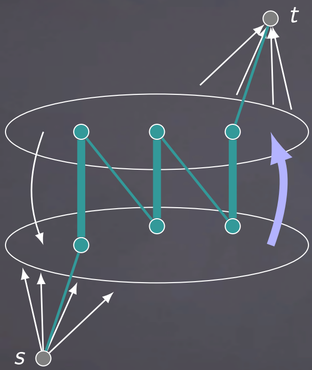
$\rightsquigarrow O(2^{\text{tw}} \cdot \text{tw}^{\text{tw}/2} \cdot \sigma_p \cdot \sigma_c)$ table entries and $O((\text{tw} + 2)^{\text{tw}} \cdot \sigma_p \cdot \sigma_c \cdot n)$ time


Too slow
in prac-
tice!

Integer Linear Program Formulation

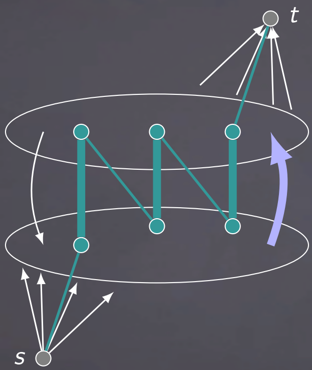


Integer Linear Program Formulation



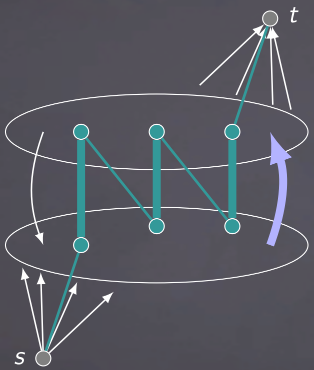
- chromosomes = disjoint s-t-paths

Integer Linear Program Formulation



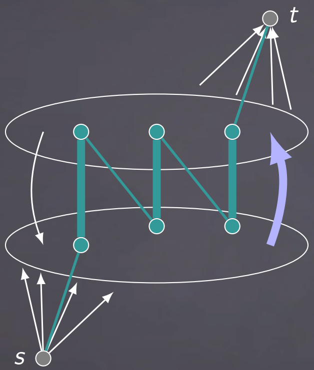
- chromosomes = disjoint s-t-paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \rightarrow v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s,t} \sum_v y_{vu} = \sum_v y_{uv}$
- path bounds: $\sum_v y_{vt} \leq \sigma$

Integer Linear Program Formulation



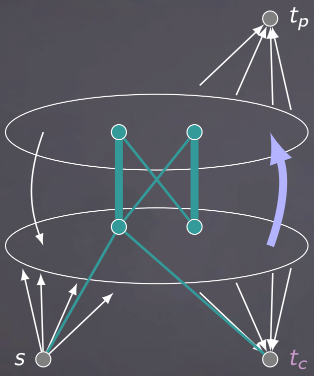
- chromosomes = disjoint s-t-paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \rightarrow v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s,t} \sum_v y_{vu} = \sum_v y_{uv}$
- path bounds: $\sum_v y_{vt} \leq \sigma$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} y_{uv} < |C|$

Integer Linear Program Formulation



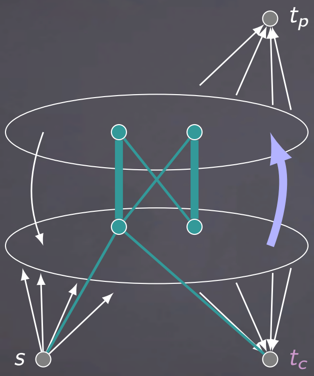
- chromosomes = disjoint s-t-paths
- bin. variables $\quad y_{uv} = 1 \Leftrightarrow u \to v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\quad \forall_{uv \in M} x_{uv} = 1$
- path preservation: $\quad \forall_{u \neq s,t} \sum_v y_{vu} = \sum_v y_{uv}$
- path bounds: $\quad \sum_v y_{vt} \leq \sigma$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\quad \sum_{uv \in C} y_{uv} < |C|$
- objective: $\quad \max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$

Integer Linear Program Formulation



- chromosomes = disjoint s-t-paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \to v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s,t} \sum_v y_{vu} = \sum_v y_{uv}$
- path bounds: $\sum_v y_{vt} \leq \sigma$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} y_{uv} < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$

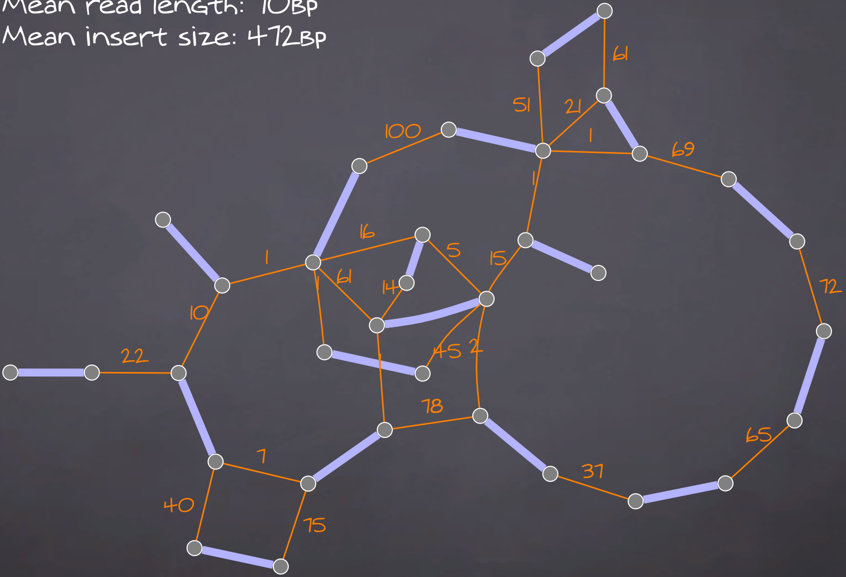
Integer Linear Program Formulation



- chromosomes = disjoint s–$\{t_p, t_c\}$–paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \to v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$

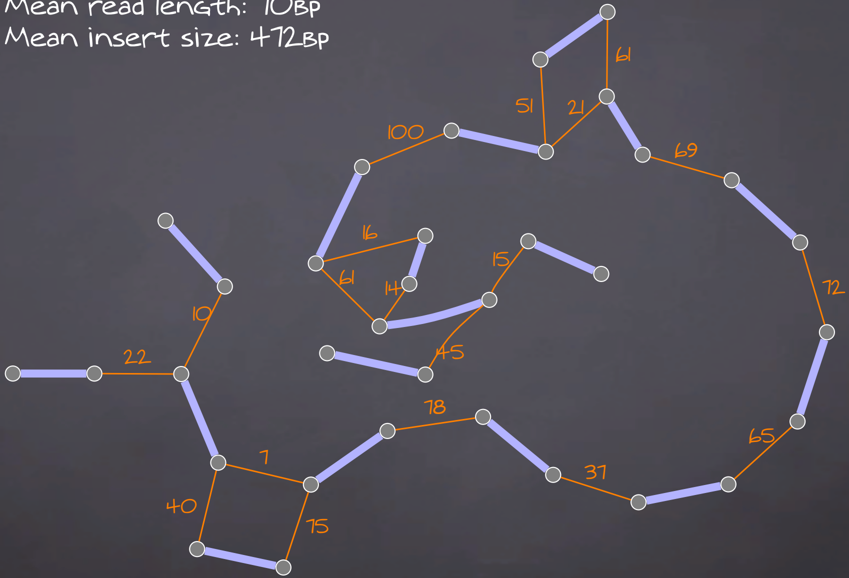
Extension: Contig Jumps

Mean read length: 70bp
Mean insert size: 472bp



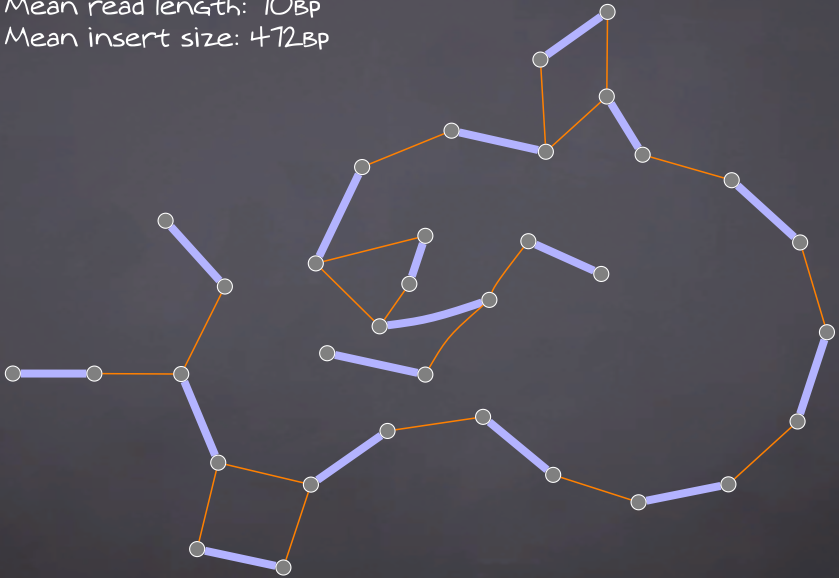
Extension: Contig Jumps

Mean read length: 70bp
Mean insert size: 472bp



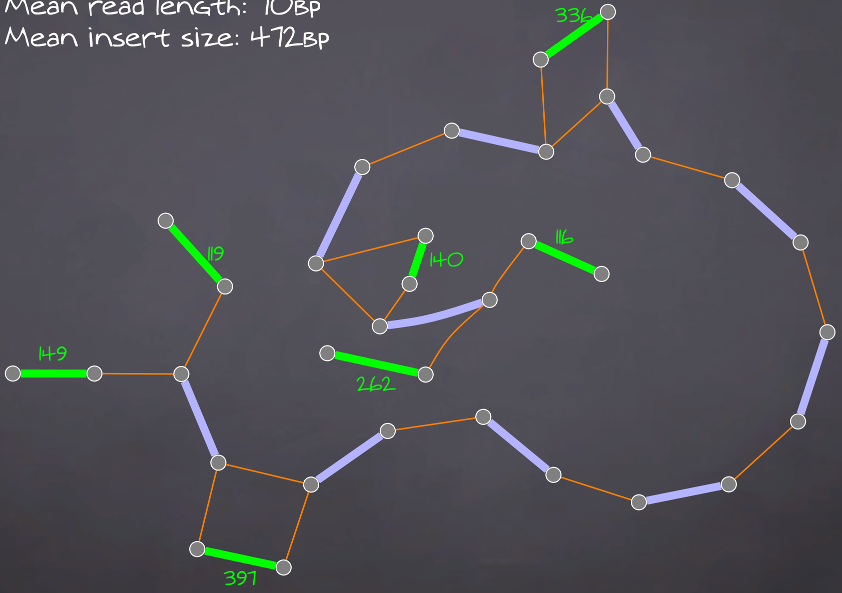
Extension: Contig Jumps

Mean read length: 70bp
Mean insert size: 472bp

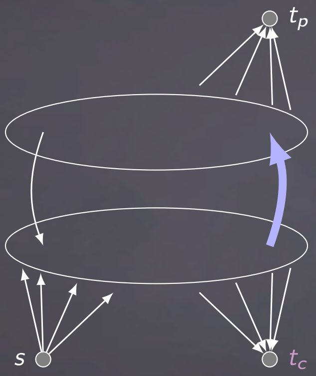


Extension: Contig Jumps

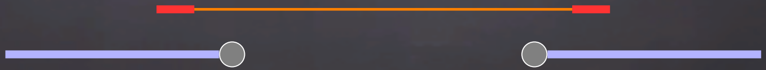
Mean read length: 70bp
Mean insert size: 472bp



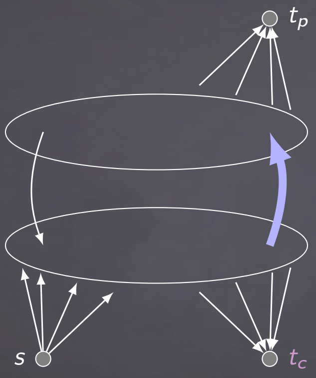
Integer Linear Program Formulation



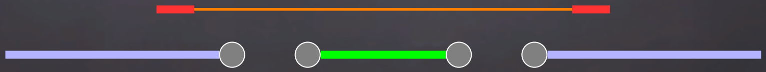
- chromosomes = disjoint s–$\{t_p, t_c\}$–paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \rightarrow v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$



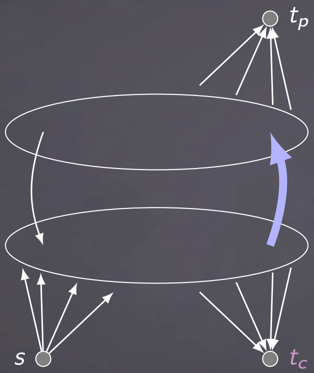
Integer Linear Program Formulation



- chromosomes = disjoint s–$\{t_p, t_c\}$–paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \rightarrow v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$

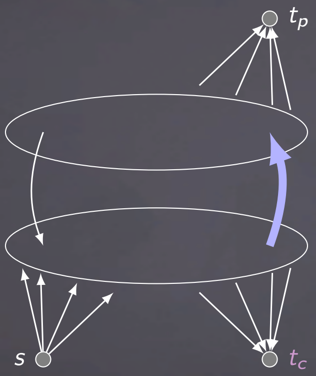


Integer Linear Program Formulation

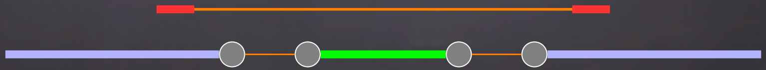


- chromosomes = disjoint s-$\{t_p, t_c\}$-paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \to v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$

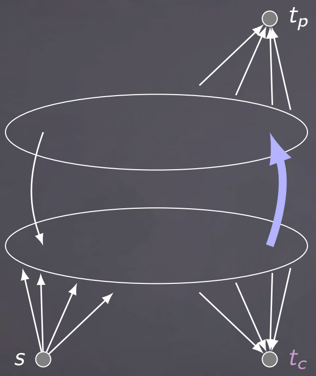
Integer Linear Program Formulation



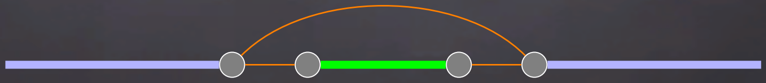
- chromosomes = disjoint s-$\{t_p, t_c\}$-paths
- bin. variables $\quad y_{uv} = 1 \Leftrightarrow u \rightarrow v$ used
  $\quad x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\quad \forall_{uv \in M} x_{uv} = 1$
- path preservation: $\quad \forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\quad \sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\quad \sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\quad \max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\quad \forall_u y_{ut_c} \leq y_{su}$



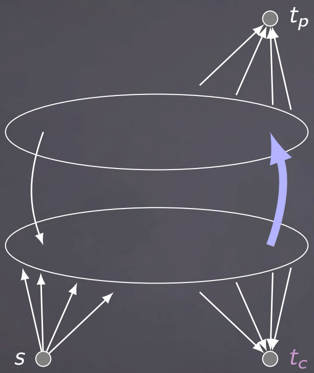
Integer Linear Program Formulation



- chromosomes = disjoint s–$\{t_p, t_c\}$–paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \rightarrow v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$



Integer Linear Program Formulation



- chromosomes = disjoint s-$\{t_p, t_c\}$-paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \rightarrow v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$

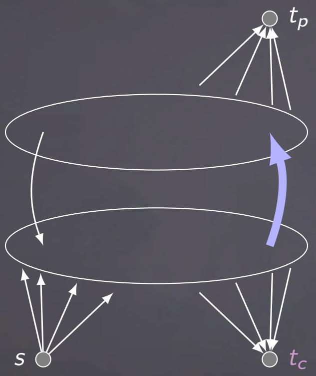
Jump Mechanics

for each non-contig uv,

1. introduce a variable z_{uv}
2. construct "jump network" between u and v that fits in the gap
3. add z_{uv} to $x_{\{u,v\}}$

extra: preprocess instance to finish incomplete jumps

Integer Linear Program Formulation



- chromosomes = disjoint s–$\{t_p, t_c\}$–paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \to v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu} + z_{uv} + z_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$
- jump mechanics

Jump Mechanics

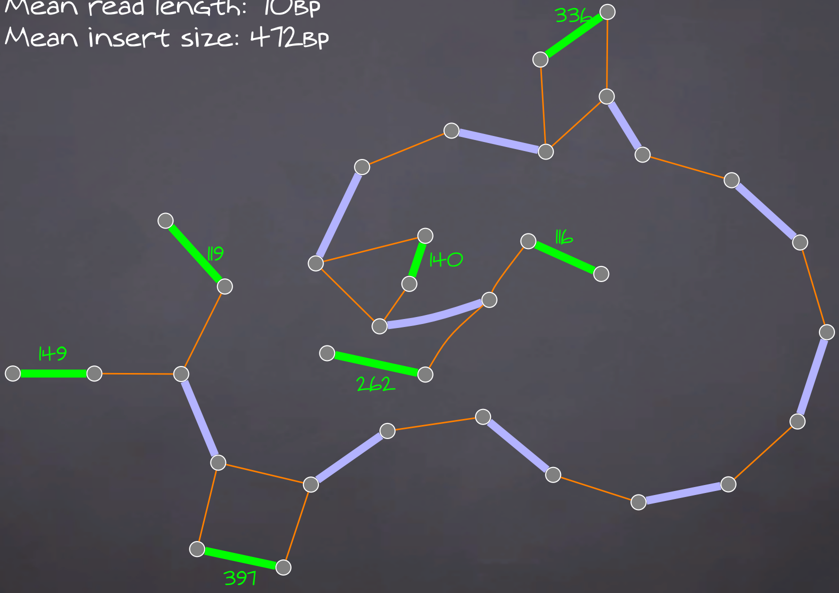
for each non-contig uv,

1. introduce a variable z_{uv}
2. construct "jump network" between u and v that fits in the gap
3. add z_{uv} to $x_{\{u,v\}}$

extra: preprocess instance to finish incomplete jumps

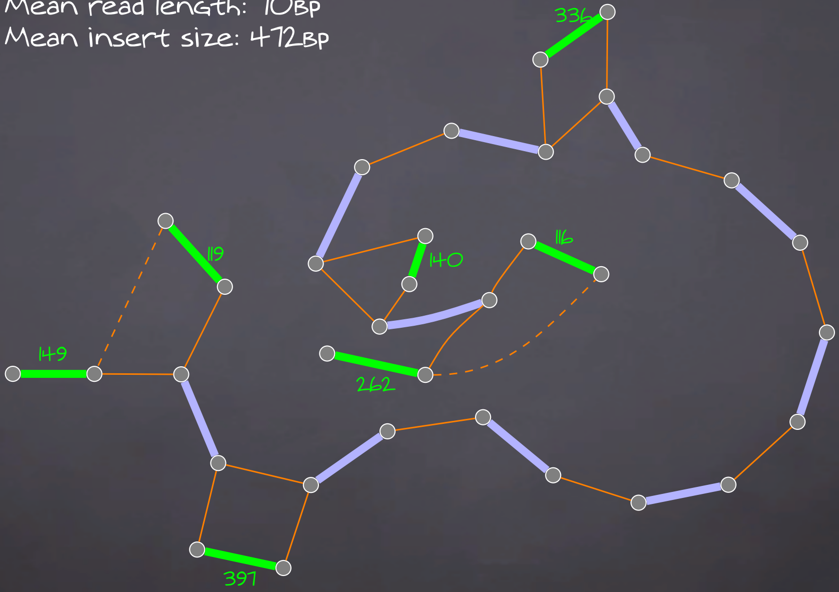
Extension: Contig Jumps

Mean read length: 70bp
Mean insert size: 472bp



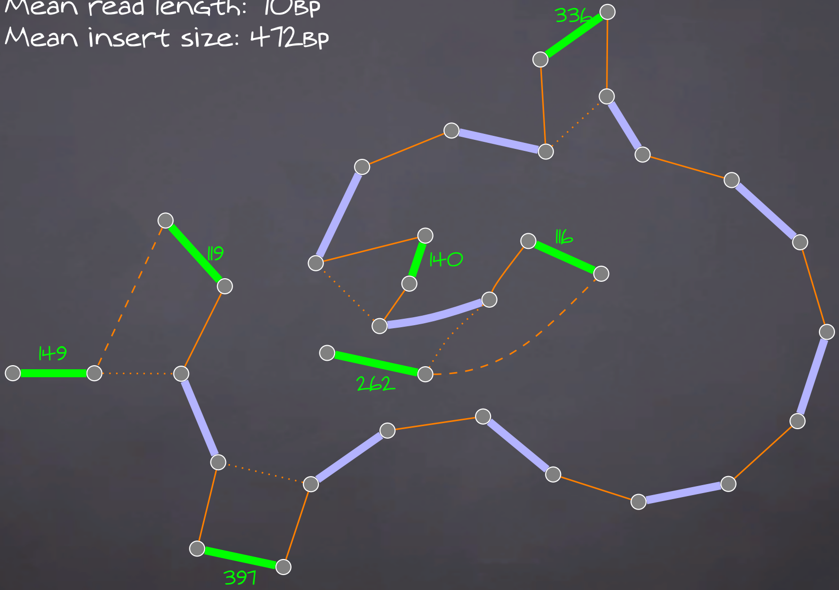
Extension: Contig Jumps

Mean read length: 70bp
Mean insert size: 472bp



Extension: Contig Jumps

Mean read length: 70bp
Mean insert size: 472bp

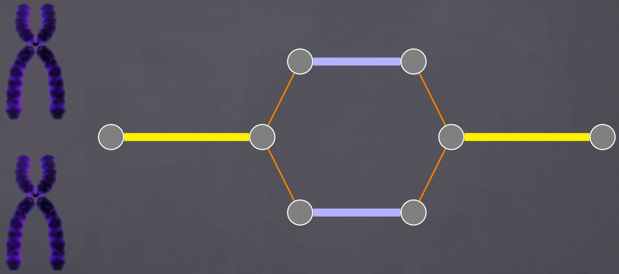


ILP Extension: Multiplicities

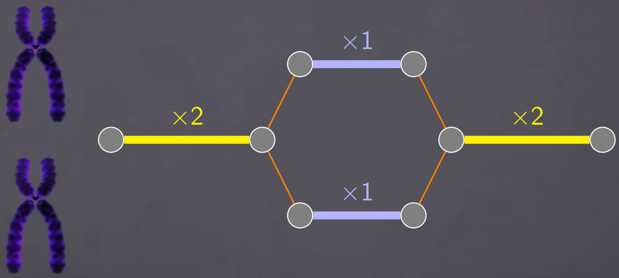
GGTGCGAGAGAGGTCATGGATTGCAACGA

GGTGCGAGAGGCCACTCCAATTGCAACGA

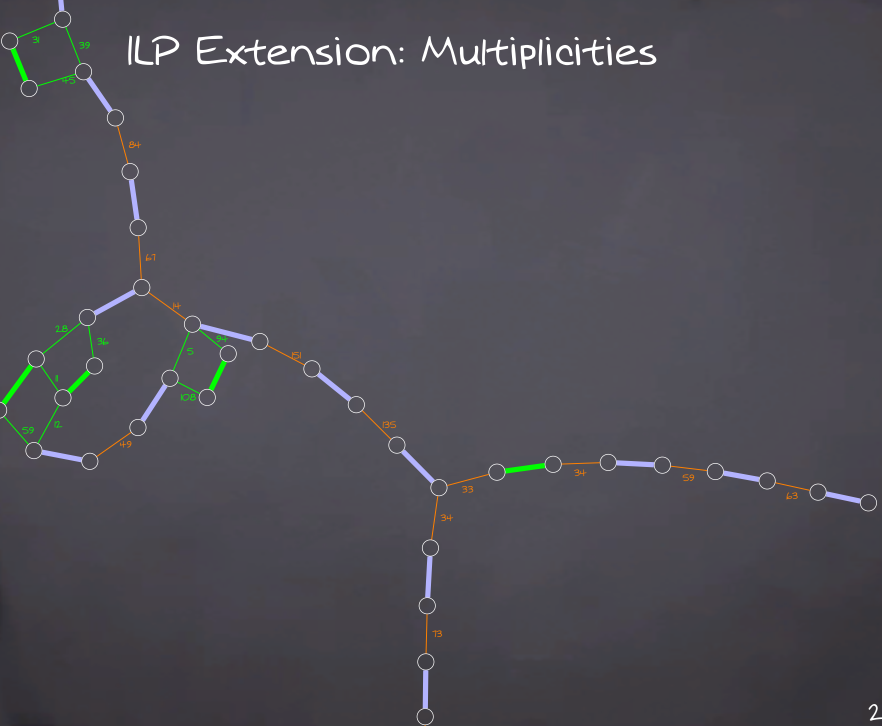
ILP Extension: Multiplicities



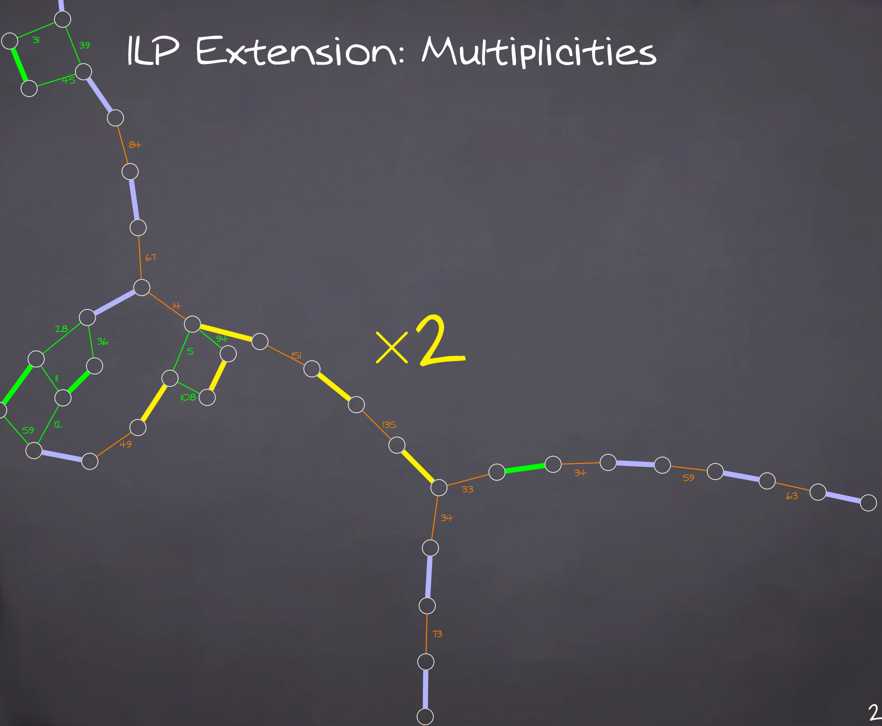
ILP Extension: Multiplicities



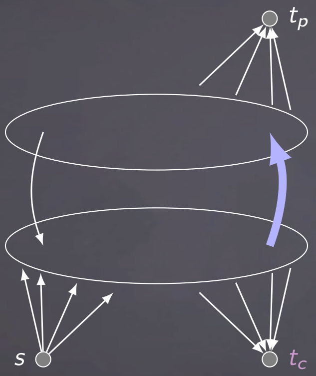
ILP Extension: Multiplicities



ILP Extension: Multiplicities



Integer Linear Program Formulation

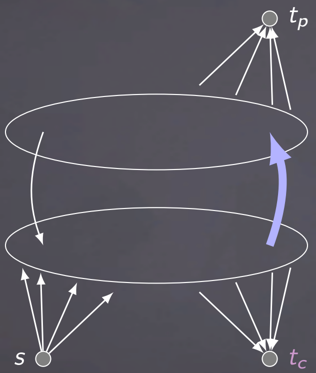


- chromosomes = disjoint s–$\{t_p, t_c\}$–paths
- bin. variables $y_{uv} = 1 \Leftrightarrow u \to v$ used
  $x_{\{u,v\}} = y_{uv} + y_{vu} + z_{uv} + z_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} = 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} (y_{uv} - y_{ut_c}) < |C|$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$
- jump mechanics

Multiplicities

1. make y_{uv}, $x_{\{u,v\}}$ integers in domain $[0, m(\{u,v\})]$
2. change callback

Integer Linear Program Formulation



- chromosomes = disjoint s–$\{t_p, t_c\}$–paths
- int. variables $y_{uv} = \ell \Leftrightarrow u \to v$ used ℓ times
  $x_{\{u,v\}} = y_{uv} + y_{vu} + z_{uv} + z_{vu}$
- force contigs: $\forall_{uv \in M} x_{uv} \geq 1$
- path preservation: $\forall_{u \neq s, t_p, t_c} \sum_v y_{vu} = \sum_v y_{uv}$
- path & cycle bounds: $\sum_v y_{vt_{\{p,c\}}} \leq \sigma_{\{p,c\}}$
- forbid cycles (row generation via callback):
  $\forall$ cycle C: $\sum_{uv \in C} y_{uv} \leq |C| \cdot m_{\max} \cdot \sum_{u \in C, v \notin C} y_{uv}$
- objective: $\max \sum_{e \in E} x_{\{u,v\}} \cdot \omega(e)$
- cycle consistency: $\forall_u y_{ut_c} \leq y_{su}$
- jump mechanics !!!

Multiplicities

1. make y_{uv}, $x_{\{u,v\}}$ integers in domain $[0, m(\{u,v\})]$
2. change callback

Linearization of Solutions

Problem

no unique chromosome-configuration explaining solution

Linearization of Solutions

Problem

no unique chromosome-configuration explaining solution

Linearization of Solutions

Problem

no unique chromosome-configuration explaining solution

Linearization of Solutions

Theorem

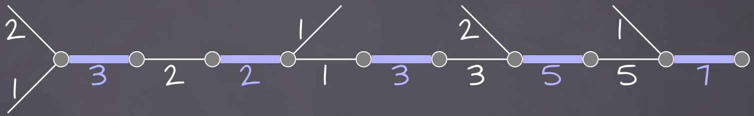
$(G, \mathcal{M}, m)$ uniquely linearizable $\Leftrightarrow$ no "ambigous paths"
(=alt. path of uniform multiplicity μ & each end incident to non-contig $< \mu$)

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 $< \mu$)



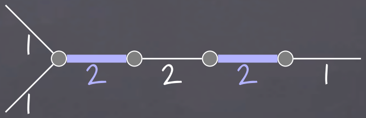
Linearization of Solutions

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



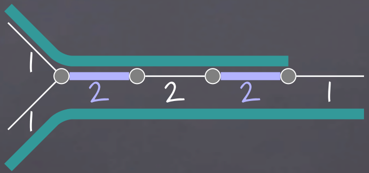
Linearization of Solutions

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



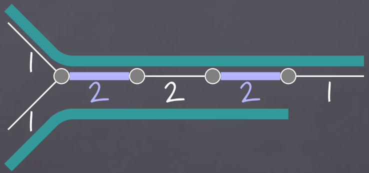
Linearization of Solutions

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



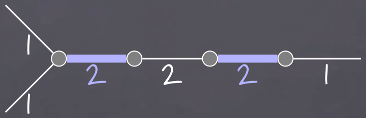
Linearization of Solutions

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):

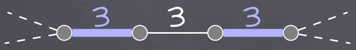
Linearization of Solutions

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

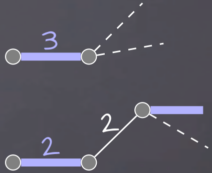
$(G, \mathcal{M}, m)$ **uniquely** linearizable $\Leftrightarrow$ no "ambigous paths"

(=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):



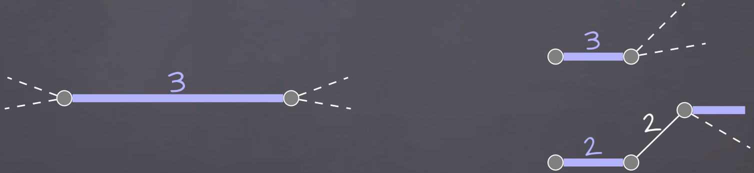
Linearization of Solutions

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 $< \mu$)

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 $< \mu$)

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 $< \mu$)

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 $< \mu$)

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity

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 $< \mu$)

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ 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 $< \mu$)

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ 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 $< \mu$)

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard

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 $< \mu$)

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard
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 $< \mu$)

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard
4. cut as few multiplicities as possible $\rightsquigarrow$ computationally hard

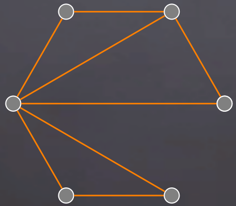
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard
4. cut as few multiplicities as possible $\rightsquigarrow$ computationally hard



Multiplicities

one &

#non-matching adj. to contig

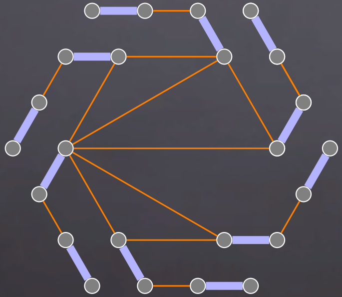
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguity ⇝ information loss
3. cut as few ends as possible ⇝ computationally hard
4. cut as few multiplicities as possible ⇝ computationally hard



Multiplicities

one &

#non-matching adj. to contig

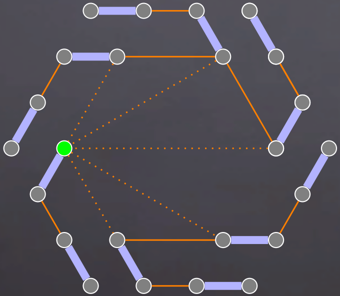
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard
4. cut as few multiplicities as possible $\rightsquigarrow$ computationally hard



Multiplicities

one &

#non-matching adj. to contig

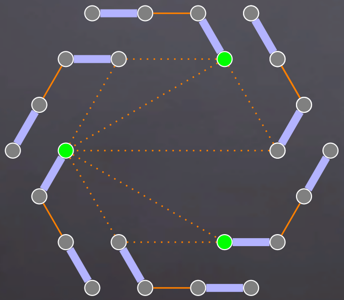
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguity ⇝ information loss
3. cut as few ends as possible ⇝ computationally hard
4. cut as few multiplicities as possible ⇝ computationally hard



Multiplicities
one &
#non-matching adj. to contig

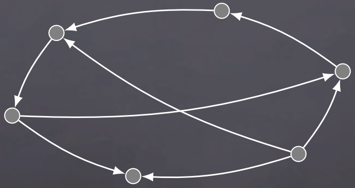
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard
4. cut as few multiplicities as possible $\rightsquigarrow$ computationally hard



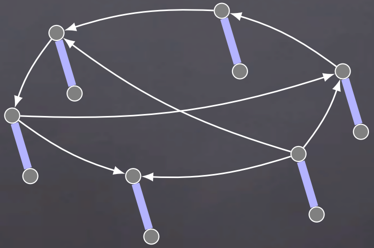
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguity ⇝ information loss
3. cut as few ends as possible ⇝ computationally hard
4. cut as few multiplicities as possible ⇝ computationally hard



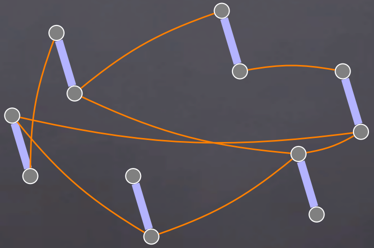
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard
4. cut as few multiplicities as possible $\rightsquigarrow$ computationally hard



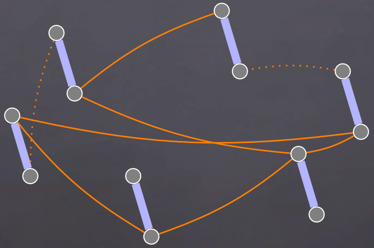
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily ⇝ missassembly
2. isolate each ambiguity ⇝ information loss
3. cut as few ends as possible ⇝ computationally hard
4. cut as few multiplicities as possible ⇝ computationally hard



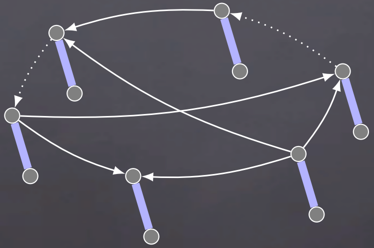
Linearization of Solutions

Theorem

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

Proposals

1. decide arbitrarily $\rightsquigarrow$ missassembly
2. isolate each ambiguity $\rightsquigarrow$ information loss
3. cut as few ends as possible $\rightsquigarrow$ computationally hard
4. cut as few multiplicities as possible $\rightsquigarrow$ computationally hard



Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings

Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings
- computationally hard problem for dense graphs with weights 0/1
- no constant-factor approx or subexponential-time algorithm for linear quasi trees with weights 0/1

Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings
- computationally hard problem for dense graphs with weights 0/1
- no constant-factor approx or subexponential-time algorithm for linear quasi trees with weights 0/1
- $O(n^2)$ time on unweighted cliques/co-bipartite/split

Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings
- computationally hard problem for dense graphs with weights 0/1
- no constant-factor approx or subexponential-time algorithm for linear quasi trees with weights 0/1
- $O(n^2)$ time on unweighted cliques/co-bipartite/split
- $O(n \cdot \sigma_p \cdot \sigma_c)$ time for constant treewidth

Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings
- computationally hard problem for dense graphs with weights 0/1
- no constant-factor approx or subexponential-time algorithm for linear quasi trees with weights 0/1
- $O(n^2)$ time on unweighted cliques/co-bipartite/split
- $O(n \cdot \sigma_p \cdot \sigma_c)$ time for constant treewidth
- 2-approximable in cliques/complete bipartite in $O(n^3)$ time

Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings
- computationally hard problem for dense graphs with weights 0/1
- no constant-factor approx or subexponential-time algorithm for linear quasi trees with weights 0/1
- $O(n^2)$ time on unweighted cliques/co-bipartite/split
- $O(n \cdot \sigma_p \cdot \sigma_c)$ time for constant treewidth
- 2-approximable in cliques/complete bipartite in $O(n^3)$ time
- $O(\sqrt{2}^n \operatorname{poly}(n))$ time exact algorithm

Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings
- computationally hard problem for dense graphs with weights 0/1
- no constant-factor approx or subexponential-time algorithm for linear quasi trees with weights 0/1
- $O(n^2)$ time on unweighted cliques/co-bipartite/split
- $O(n \cdot \sigma_p \cdot \sigma_c)$ time for constant treewidth
- 2-approximable in cliques/complete bipartite in $O(n^3)$ time
- $O(\sqrt{2}^n \text{poly}(n))$ time exact algorithm
- ILP formulation with contig jumps & multiplicities

Conclusion

What we saw

- 3-step sequencing technique:
  1. produce paired-end reads
  2. assemble reads to contigs
  3. scaffold contigs to chromosomes using read-pairings
- computationally hard problem for dense graphs with weights 0/1
- no constant-factor approx or subexponential-time algorithm for linear quasi trees with weights 0/1
- $O(n^2)$ time on unweighted cliques/co-bipartite/split
- $O(n \cdot \sigma_p \cdot \sigma_c)$ time for constant treewidth
- 2-approximable in cliques/complete bipartite in $O(n^3)$ time
- $O(\sqrt{2}^n \operatorname{poly}(n))$ time exact algorithm
- ILP formulation with contig jumps & multiplicities
- Linearization problem raised by multiplicities in solution

Conclusion

Outlook

- 3^{rd} generation sequencing: PacBio, Oxford Nanopore produces long reads (10–15kbp), but error-prone
  ⇝ correction using small reads?

Conclusion

Outlook

- 3rd generation sequencing: PacBio, Oxford Nanopore produces long reads (10-15kbp), but error-prone
  $\rightsquigarrow$ correction using small reads?
- generally: multi-library scaffolding

Conclusion

Outlook

- 3rd generation sequencing: PacBio, Oxford Nanopore produces long reads (10–15kbp), but error-prone
  $\rightsquigarrow$ correction using small reads?
- generally: multi-library scaffolding
- other sources for contig-connections (phylogenetic information?)

Conclusion

Outlook

- 3^{rd} generation sequencing: PacBio, Oxford Nanopore produces long reads (10-15kbp), but error-prone
  $\rightsquigarrow$ correction using small reads?
- generally: multi-library scaffolding
- other sources for contig-connections (phylogenetic information?)
- better parameters for Scaffolding and Scaffold Linearization
  $\rightsquigarrow$ analyze practical instances

Conclusion

Outlook

- 3rd generation sequencing: PacBio, Oxford Nanopore produces long reads (10-15kbp), but error-prone
  ⇝ correction using small reads?
- generally: multi-library scaffolding
- other sources for contig-connections (phylogenetic information?)
- better parameters for Scaffolding and Scaffold Linearization
  ⇝ analyze practical instances
- approximation/heuristics for Scaffold Linearization


Fin!