Computing Width Parameters of Graphs

Tuukka Korhonen

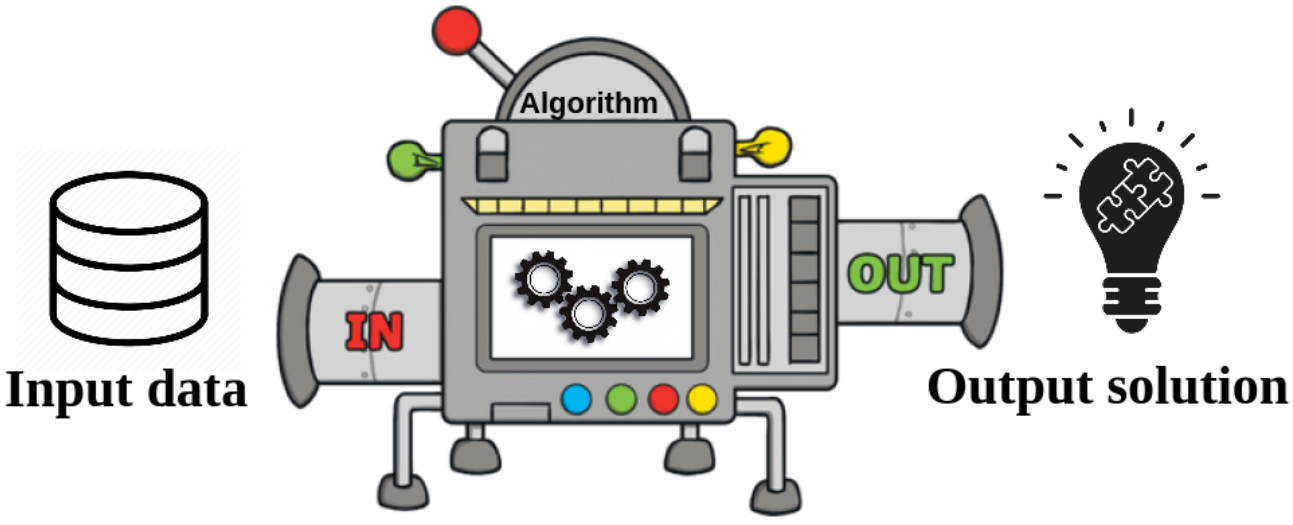
UNIVERSITY OF BERGEN

15 May 2024

Opponent: Hans L. Bodlaender
Opponent: Archontia C. Giannopoulou
Leader of the committee: Torstein J. F. Strømme
Leader of the defense: Tom Michoel
Main supervisor: Fedor V. Fomin
Co-supervisor: Petr A. Golovach

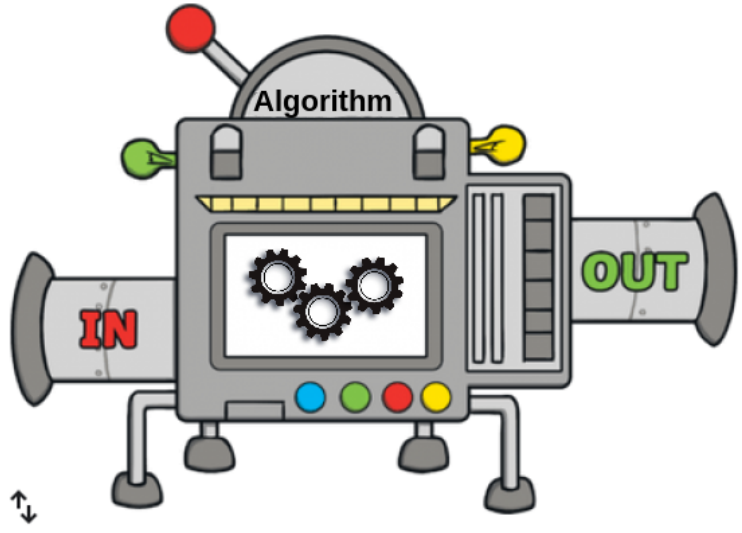
Algorithms

Algorithms



Algorithms

Map of norway



Bergen

Oslo

Names of two cities

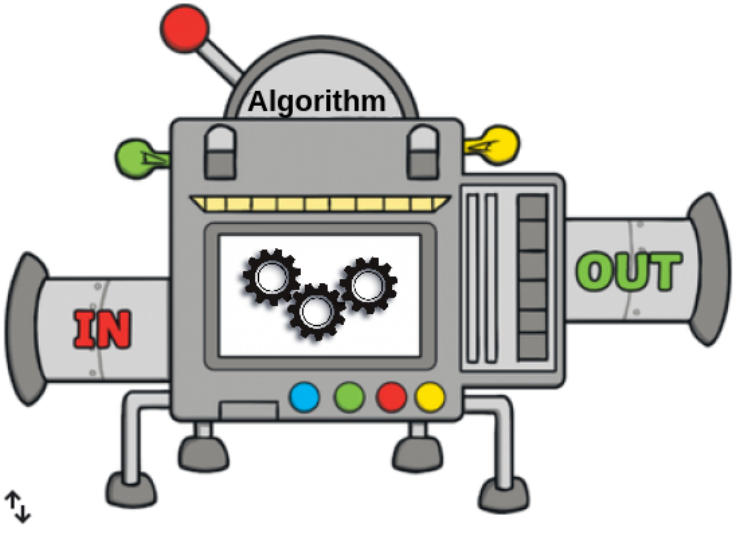
Algorithms

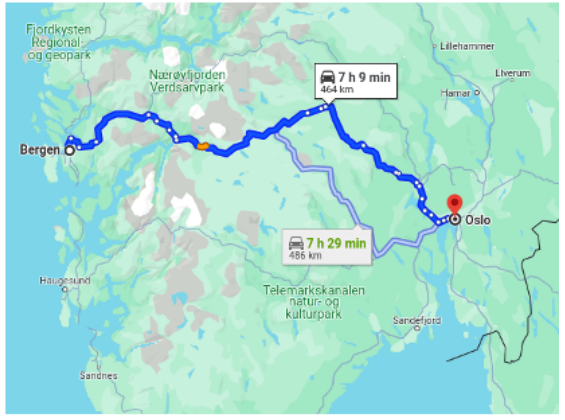
Map of norway

Bergen

Oslo

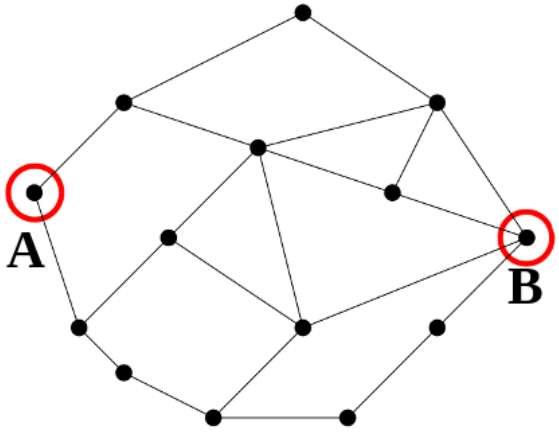
Names of two cities



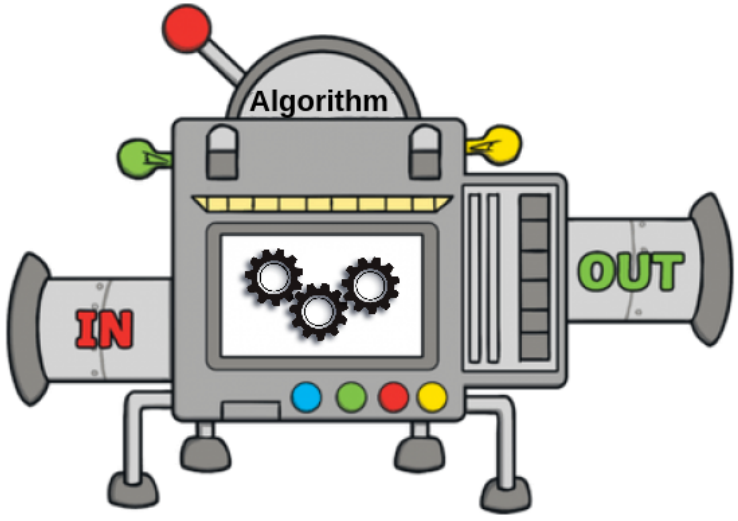


Fastest route between the cities

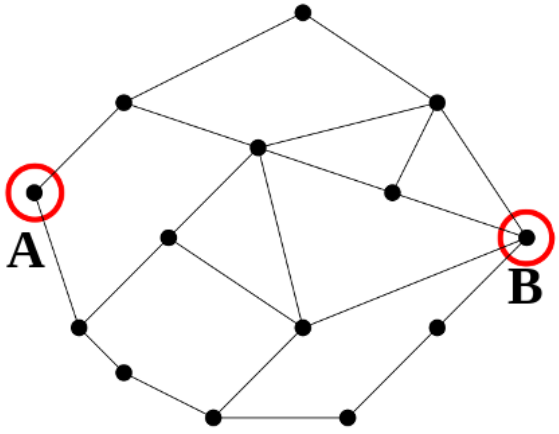
Algorithms



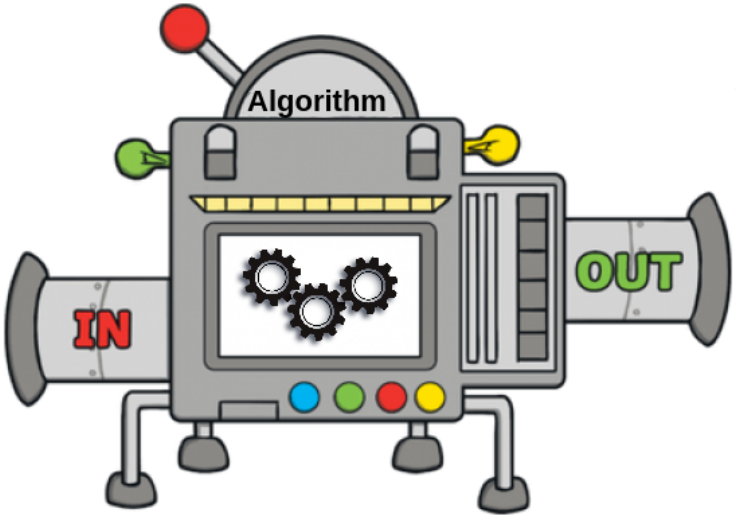
Graph with two marked vertices A and B

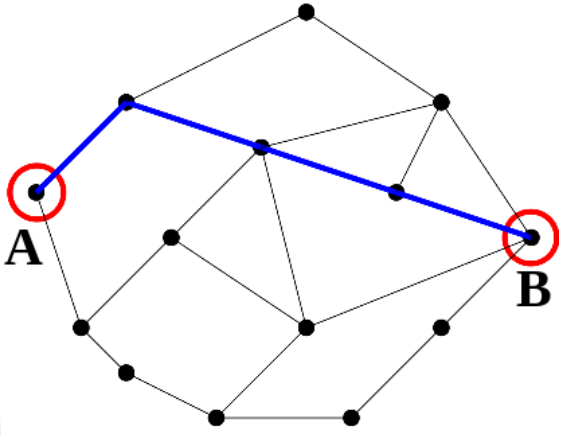


Algorithms



Graph with two marked vertices A and B



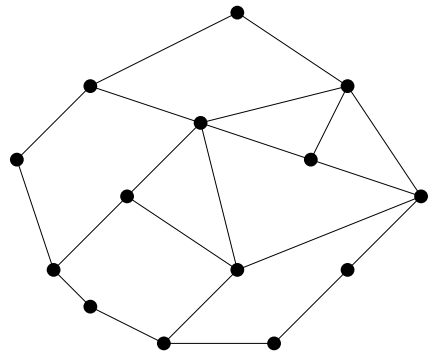


Shortest path between A and B

Graphs

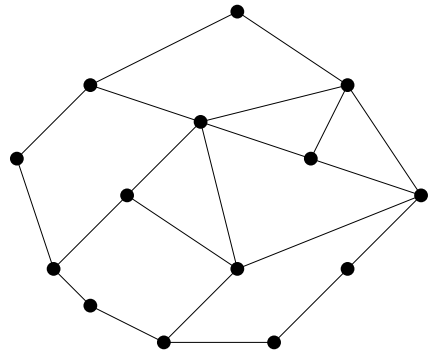
Graphs

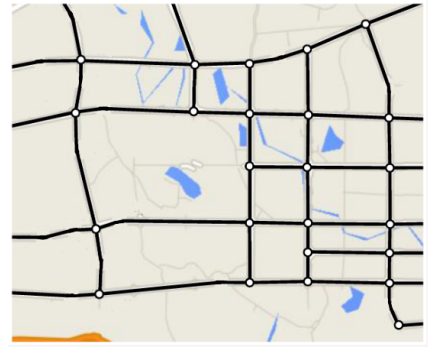
- Vertices (points) connected by edges (lines)



Graphs

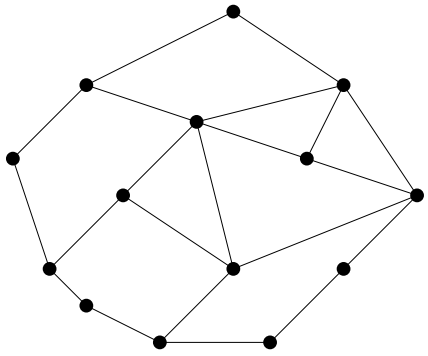
- Vertices (points) connected by edges (lines)
- Road networks

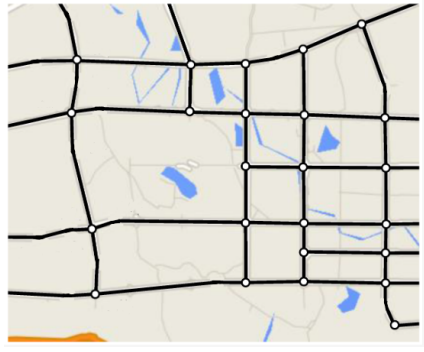




Graphs

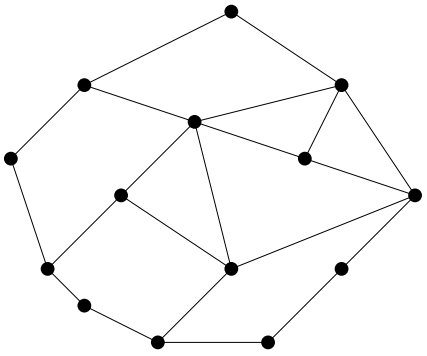
- Vertices (points) connected by edges (lines)
- Road networks
- Connections in social media

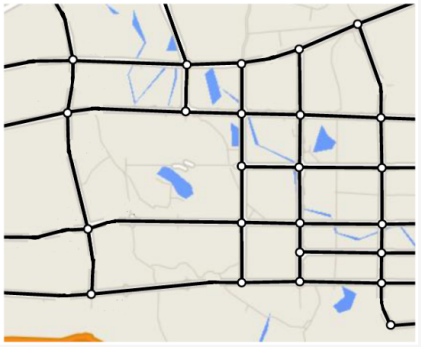


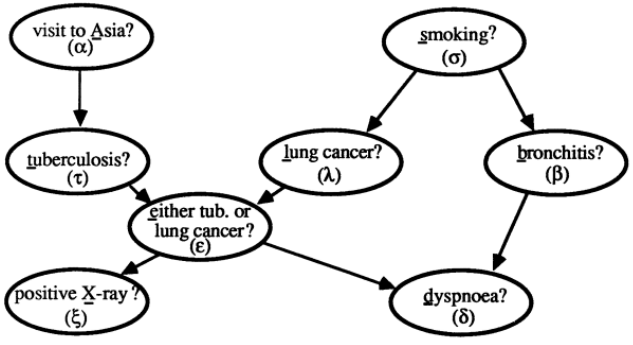


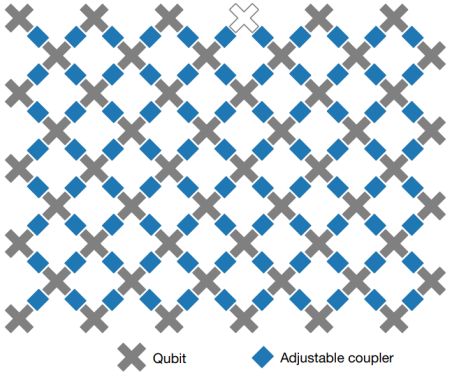
Graphs

- Vertices (points) connected by edges (lines)
- Road networks
- Connections in social media
- Interactions between variables









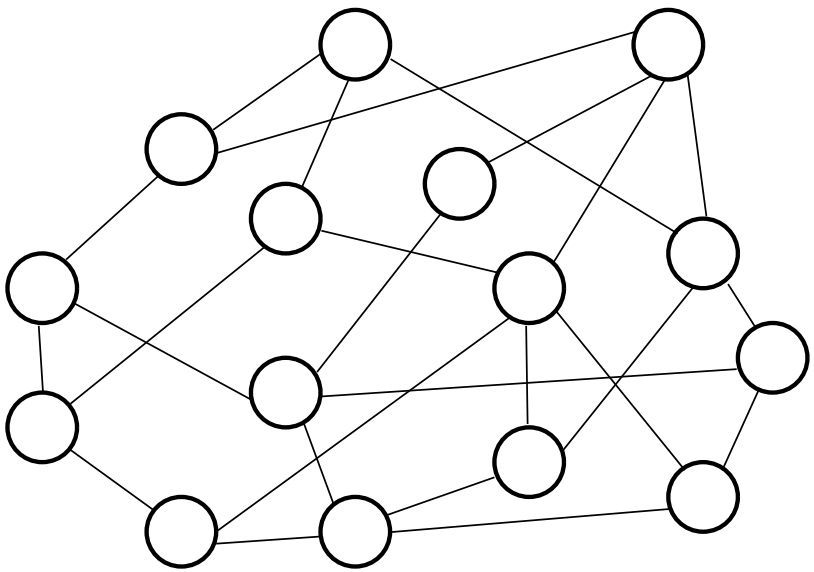
Simple but tough graph problem

The maximum independent set problem:

Simple but tough graph problem

The maximum independent set problem:

Input: A graph

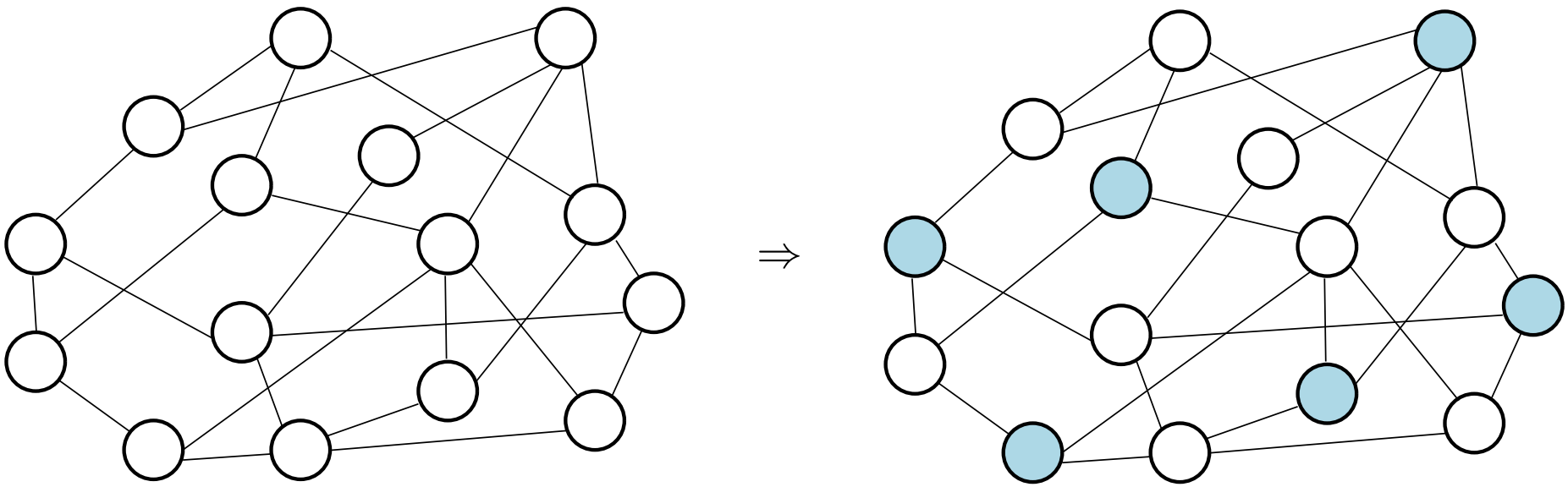


Simple but tough graph problem

The maximum independent set problem:

Input: A graph

Output: Largest set of vertices with no edges between them

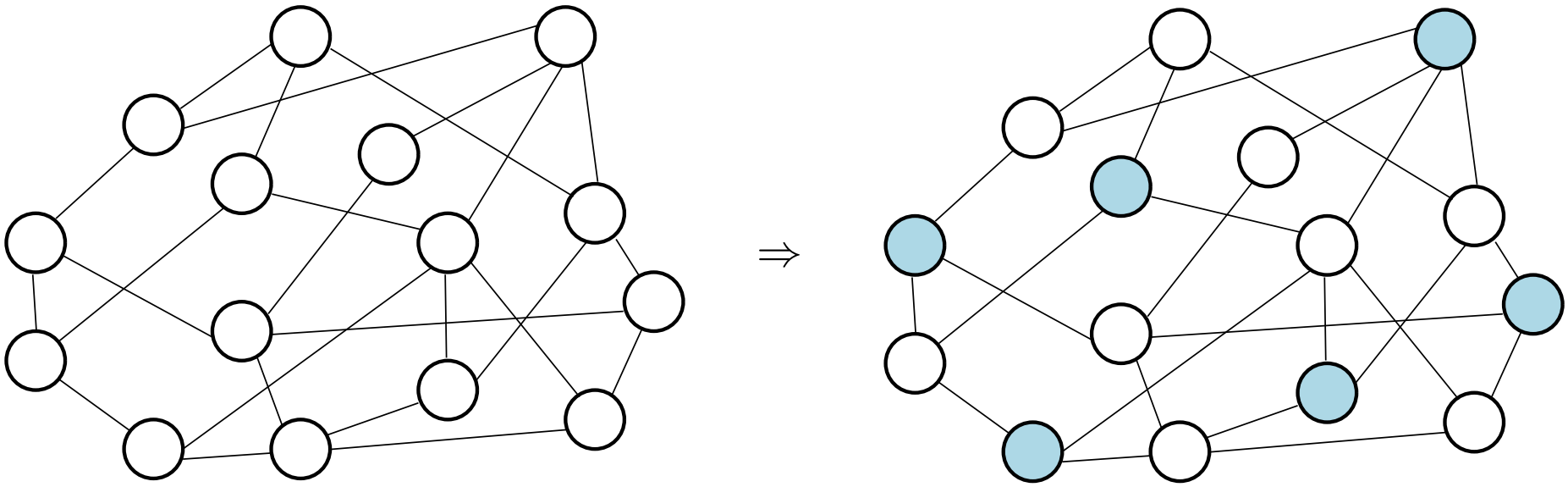


Simple but tough graph problem

The maximum independent set problem:

Input: A graph

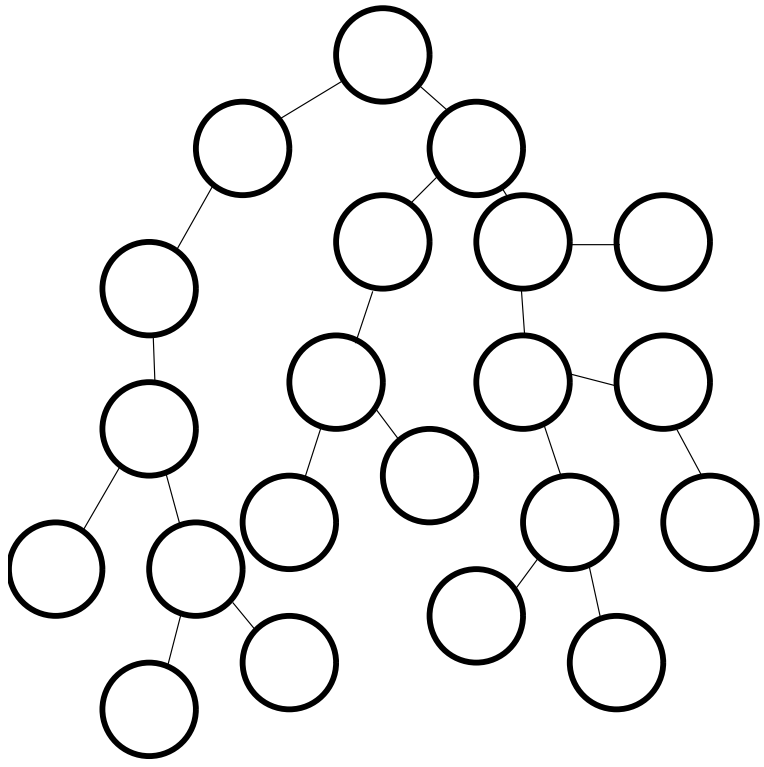
Output: Largest set of vertices with no edges between them



NP-hard $\rightarrow$ no efficient algorithm for finding an optimal solution

Structure matters

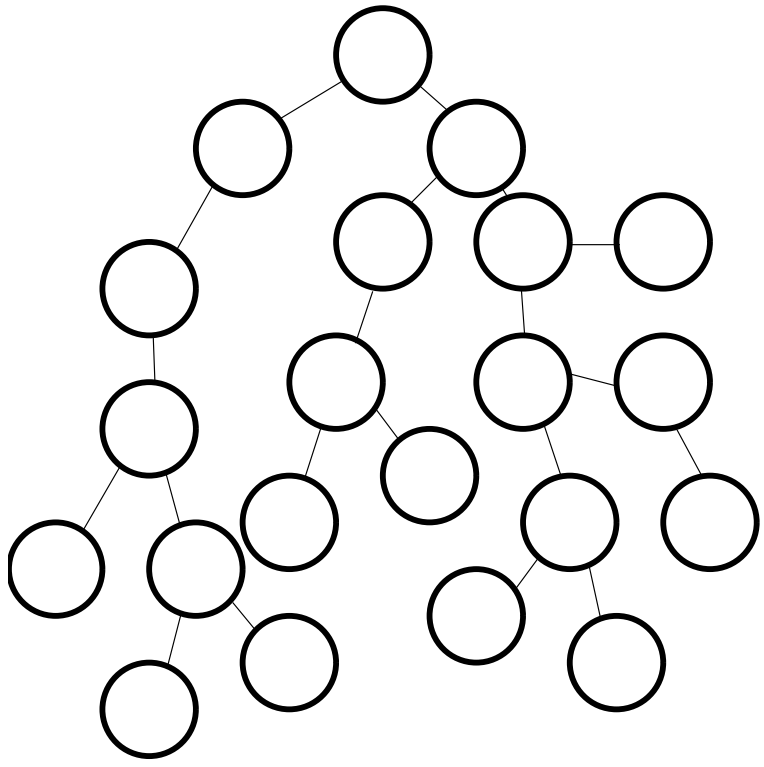
What if the input graph is a tree



Structure matters

What if the input graph is a tree

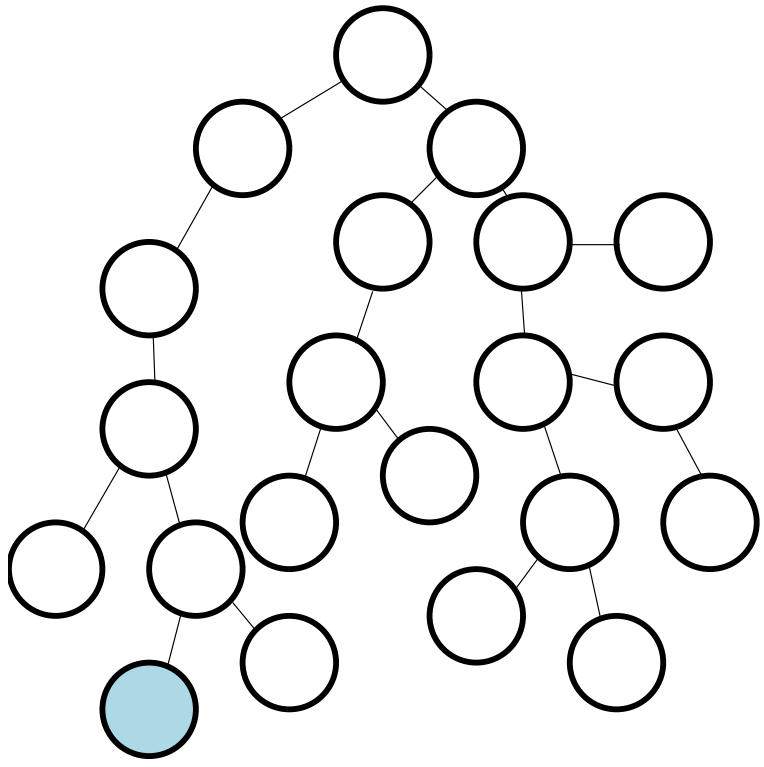
- No cycles



Structure matters

What if the input graph is a tree

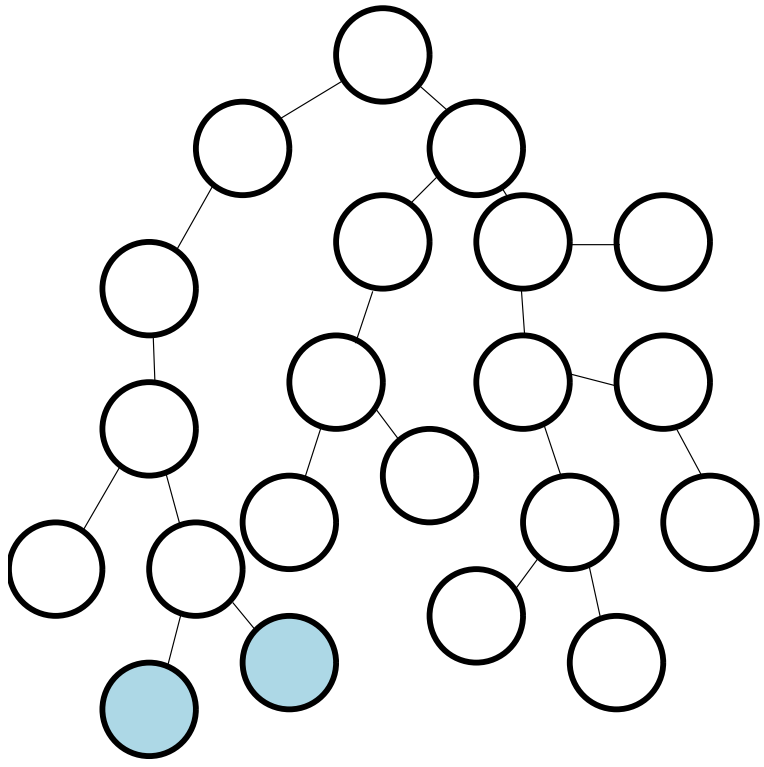
- No cycles



Structure matters

What if the input graph is a tree

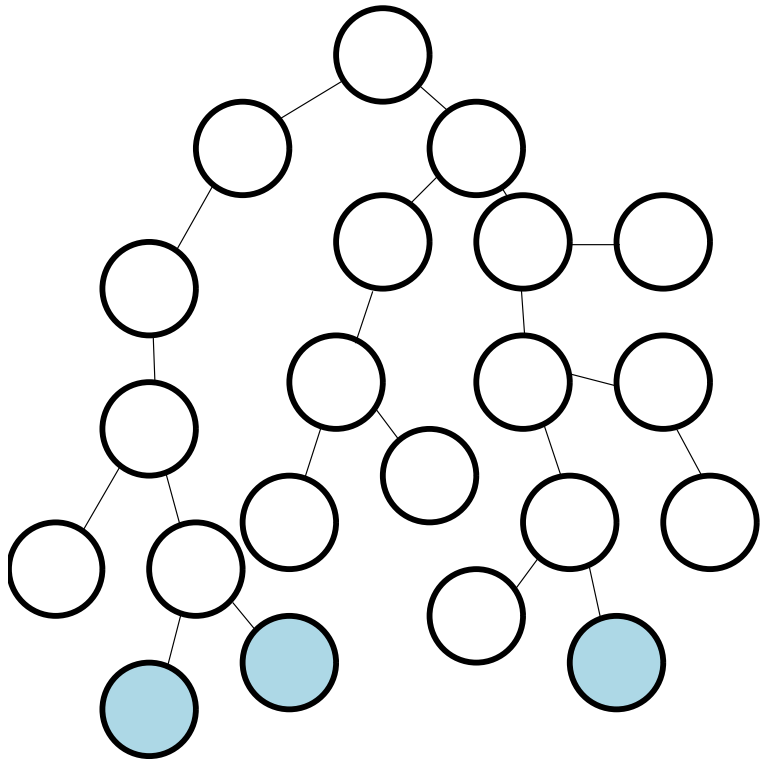
- No cycles



Structure matters

What if the input graph is a tree

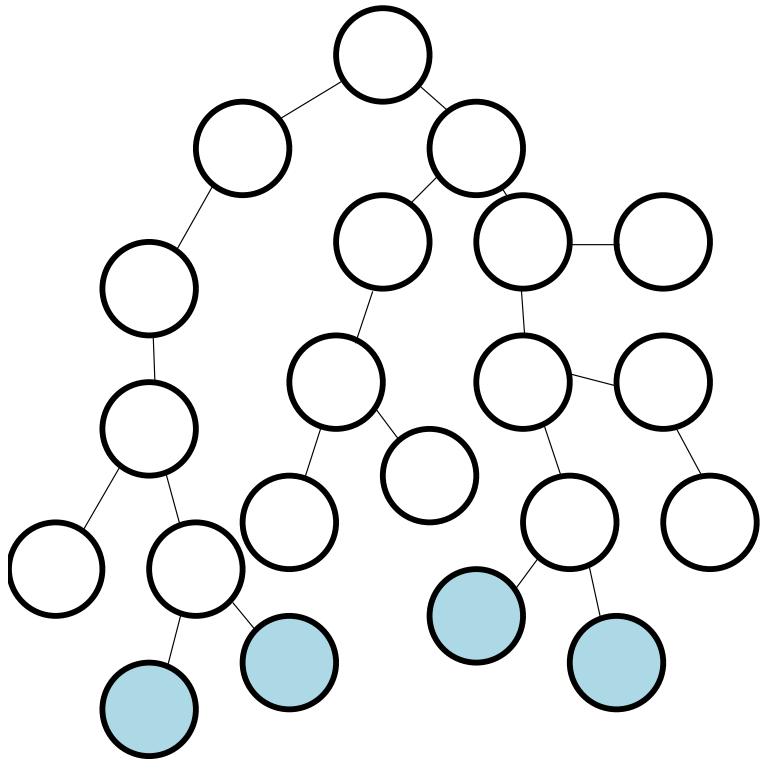
- No cycles



Structure matters

What if the input graph is a tree

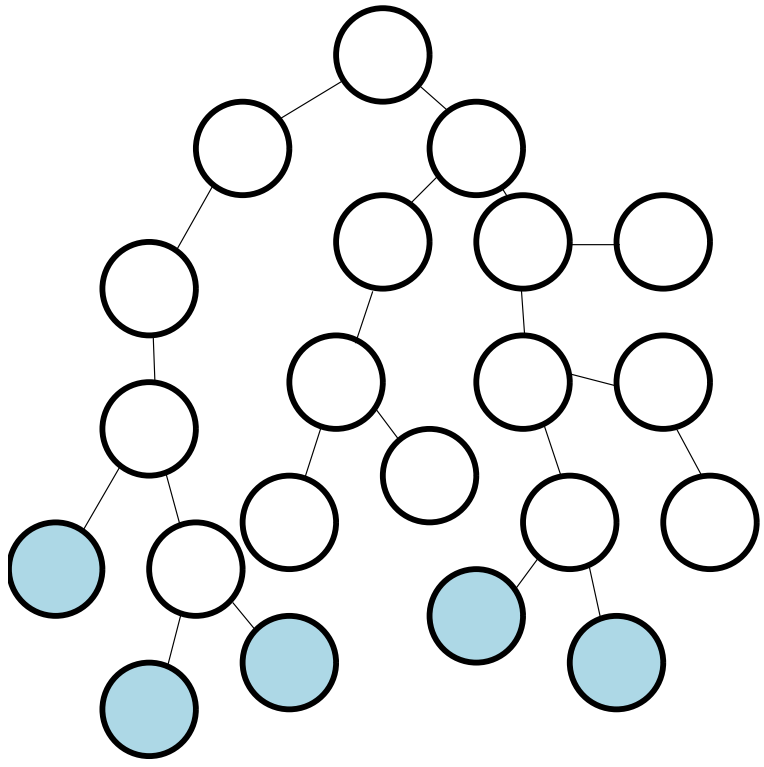
- No cycles



Structure matters

What if the input graph is a tree

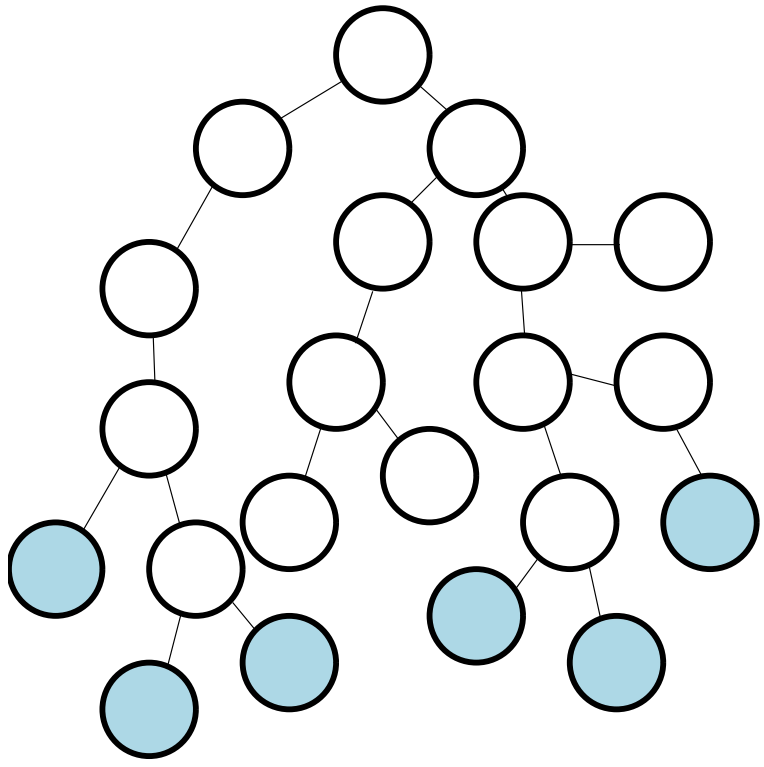
- No cycles



Structure matters

What if the input graph is a tree

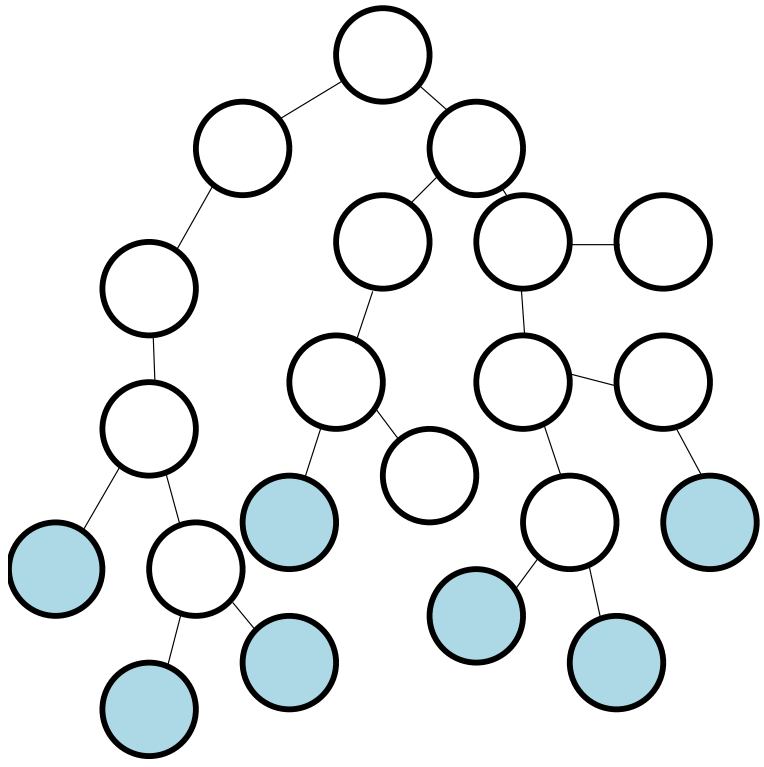
- No cycles



Structure matters

What if the input graph is a tree

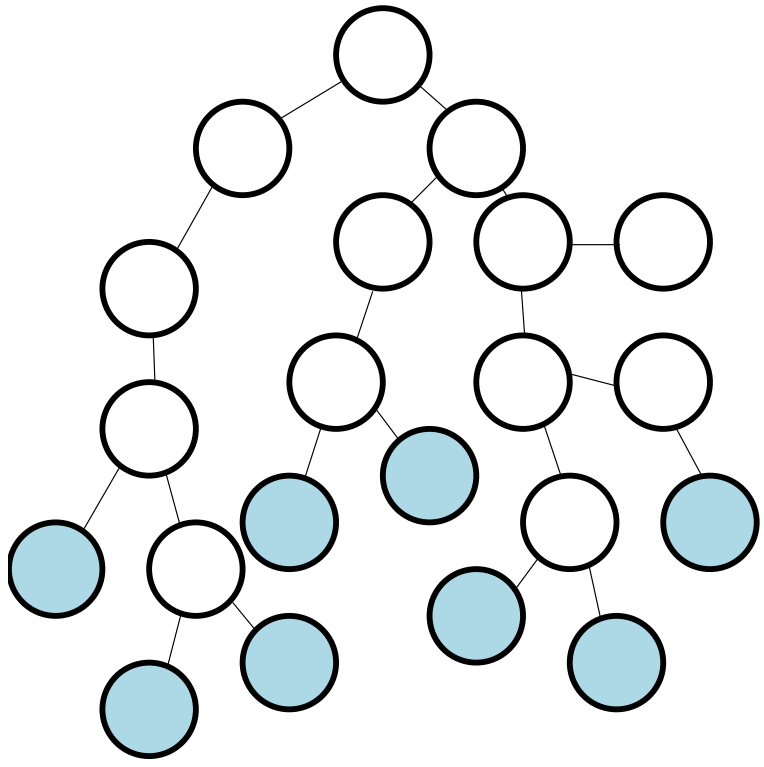
- No cycles



Structure matters

What if the input graph is a tree

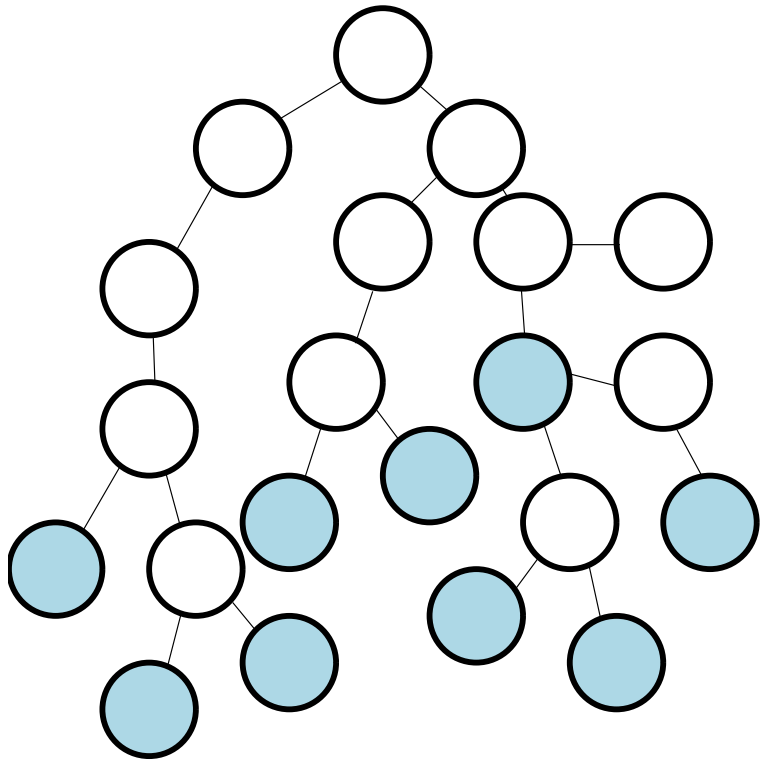
- No cycles



Structure matters

What if the input graph is a tree

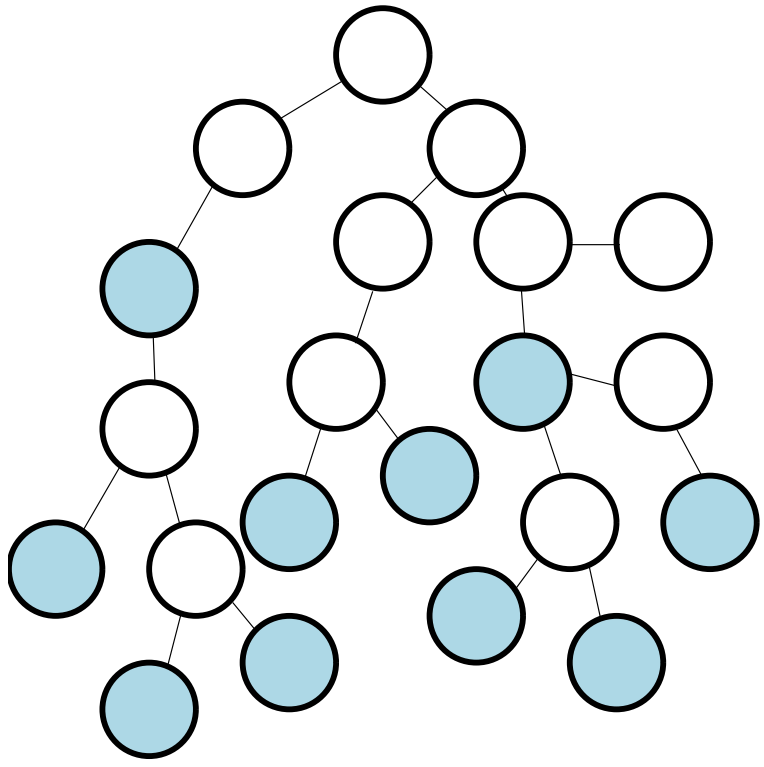
- No cycles



Structure matters

What if the input graph is a tree

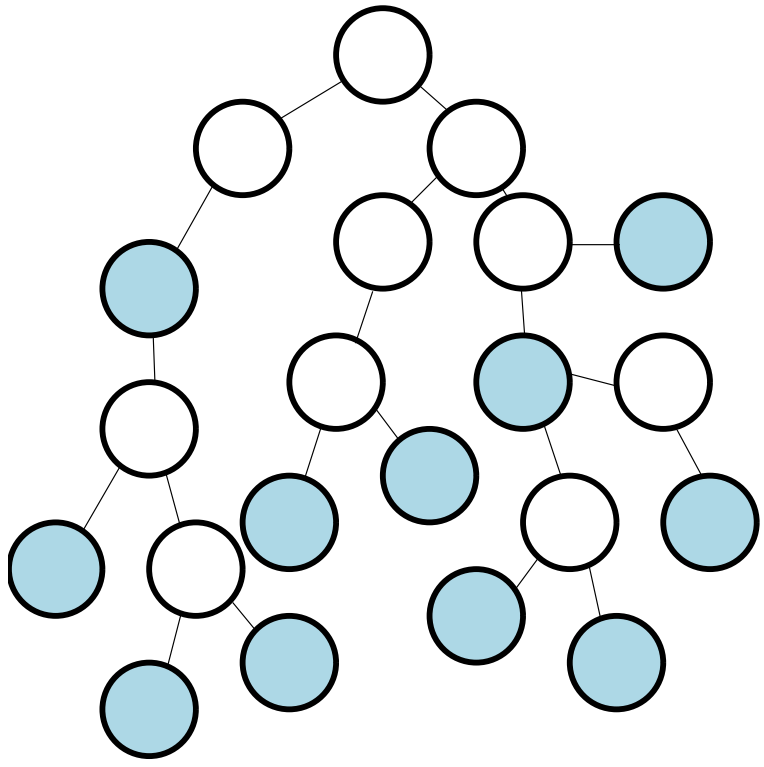
- No cycles



Structure matters

What if the input graph is a tree

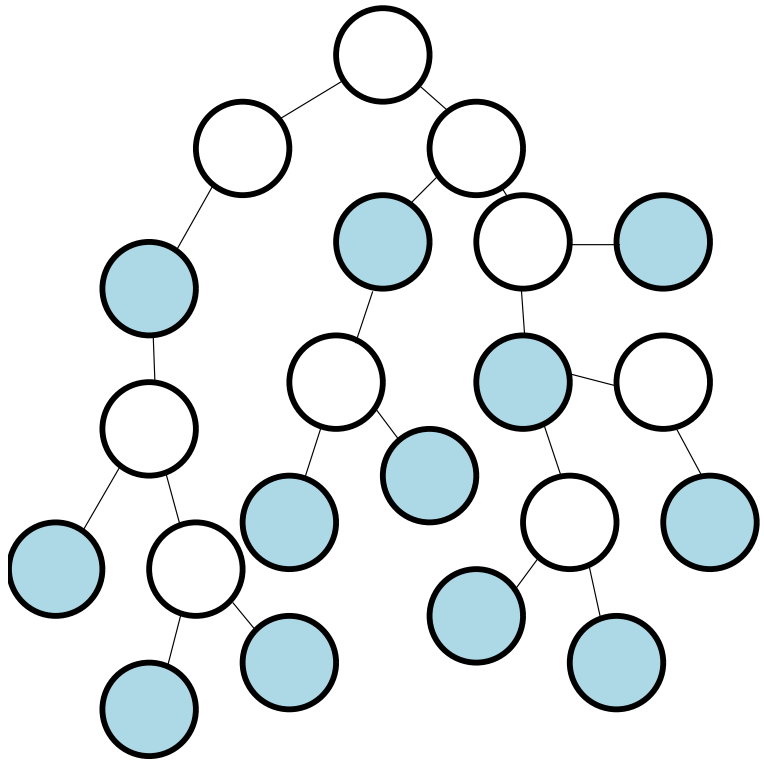
- No cycles



Structure matters

What if the input graph is a tree

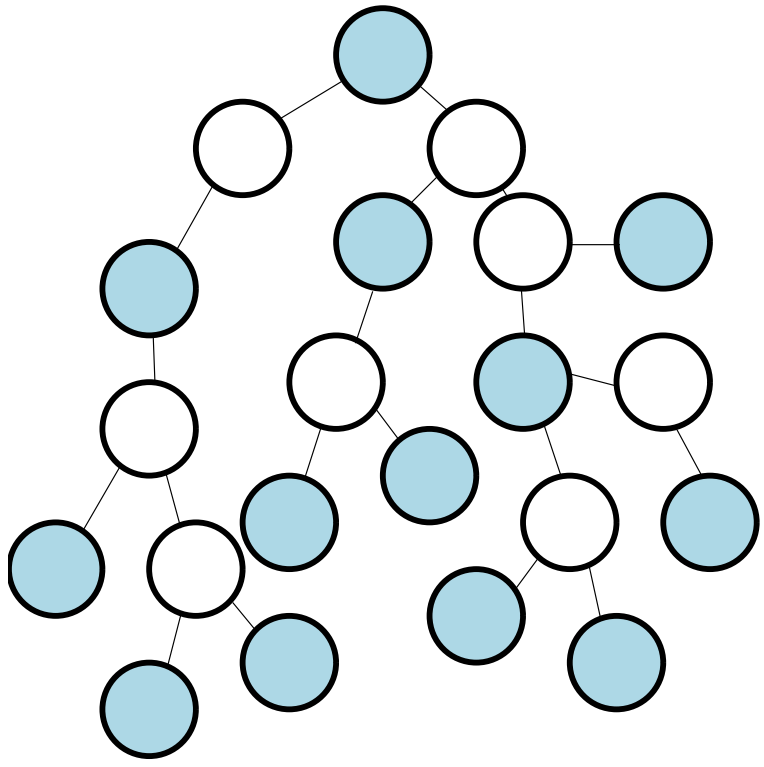
- No cycles



Structure matters

What if the input graph is a tree

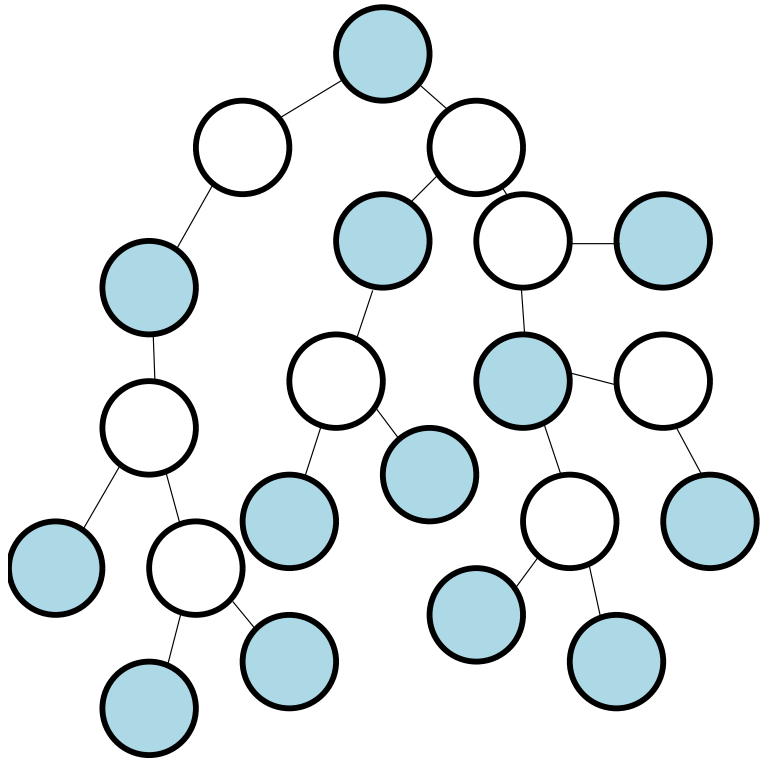
- No cycles



Structure matters

What if the input graph is a tree

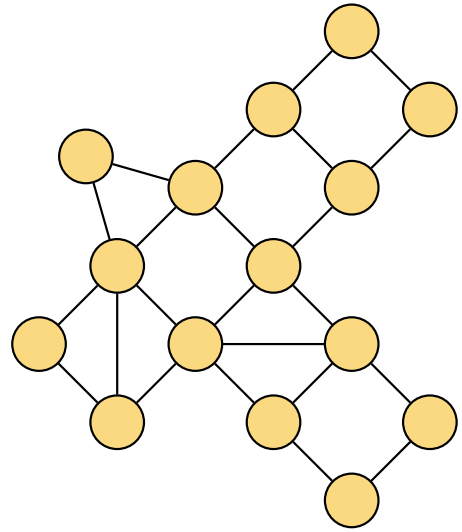
- No cycles



Algorithm to find an optimal solution in linear time

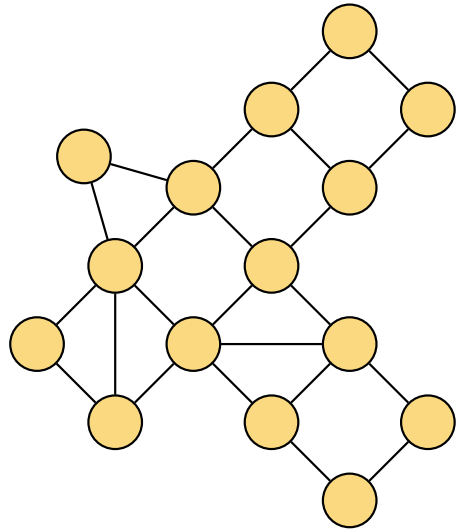
Almost trees

- What if a graph is not a tree, but almost?



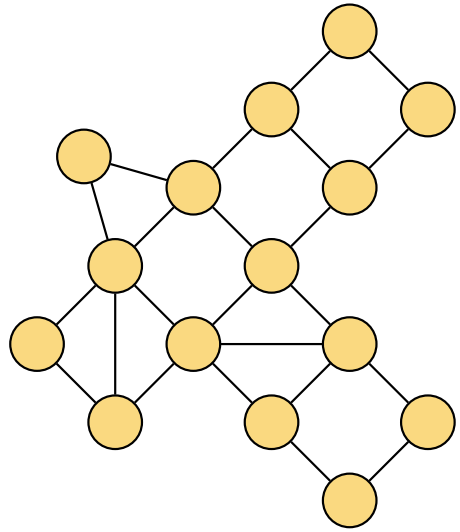
Almost trees

- What if a graph is not a tree, but almost?
- The treewidth of a graph [Robertson & Seymour'84, Arnborg & Proskurowski'89, Bertele & Brioschi'72, Halin'76]



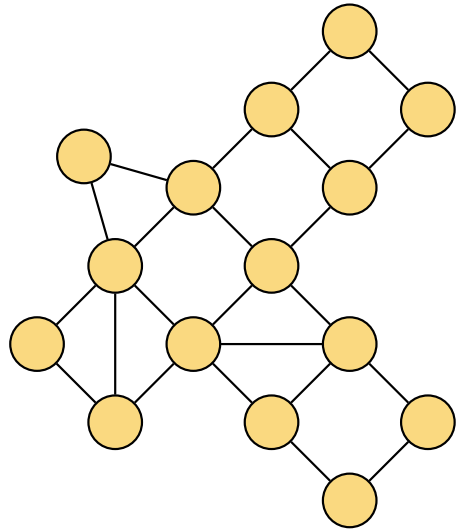
Almost trees

- What if a graph is not a tree, but almost?
- The treewidth of a graph [Robertson & Seymour'84, Arnborg & Proskurowski'89, Bertele & Brioschi'72, Halin'76]
- Trees have treewidth 1



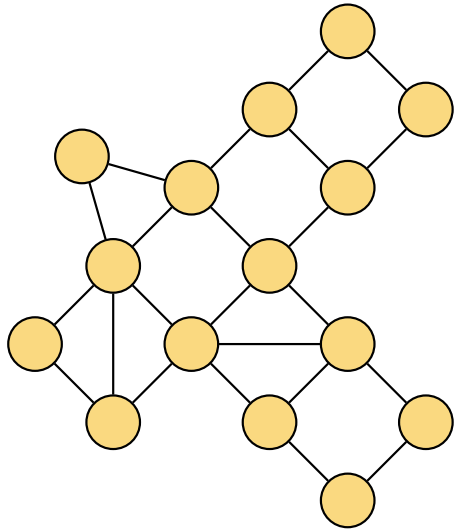
Almost trees

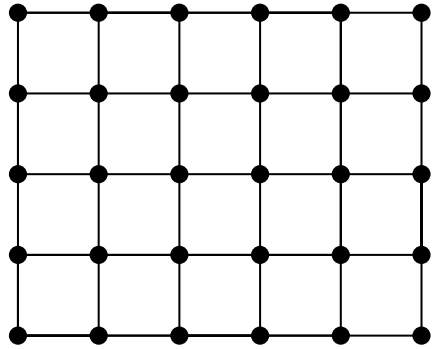
- What if a graph is not a tree, but almost?
- The treewidth of a graph [Robertson & Seymour'84, Arnborg & Proskurowski'89, Bertele & Brioschi'72, Halin'76]
- Trees have treewidth 1
- The example graph has treewidth 2



Almost trees

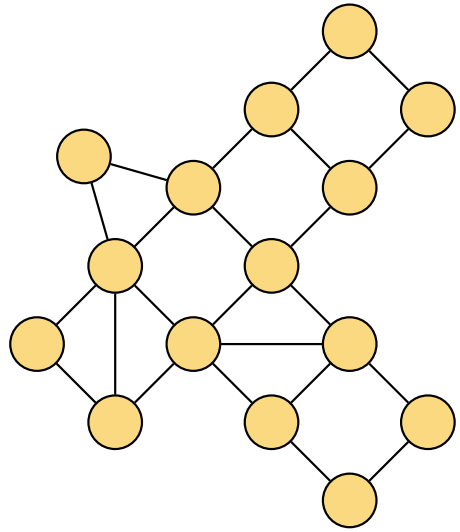
- What if a graph is not a tree, but almost?
- The treewidth of a graph [Robertson & Seymour'84, Arnborg & Proskurowski'89, Bertele & Brioschi'72, Halin'76]
- Trees have treewidth 1
- The example graph has treewidth 2
- The $n \times m$ grid has treewidth $\min(n, m)$

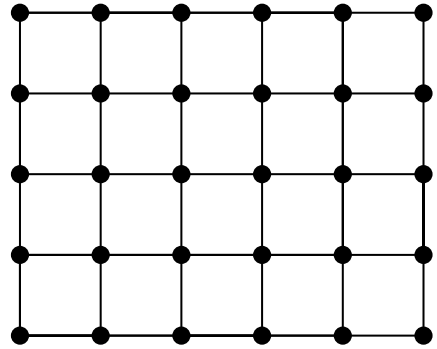




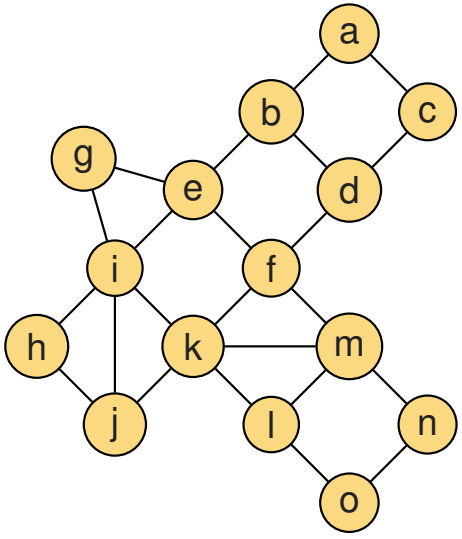
Almost trees

- What if a graph is not a tree, but almost?
- The treewidth of a graph [Robertson & Seymour'84, Arnborg & Proskurowski'89, Bertele & Brioschi'72, Halin'76]
- Trees have treewidth 1
- The example graph has treewidth 2
- The $n \times m$ grid has treewidth $\min(n, m)$
- Maximum independent set can be solved in linear time on graphs with constant treewidth [Arnborg & Proskurowski'89, Bodlaender'96]

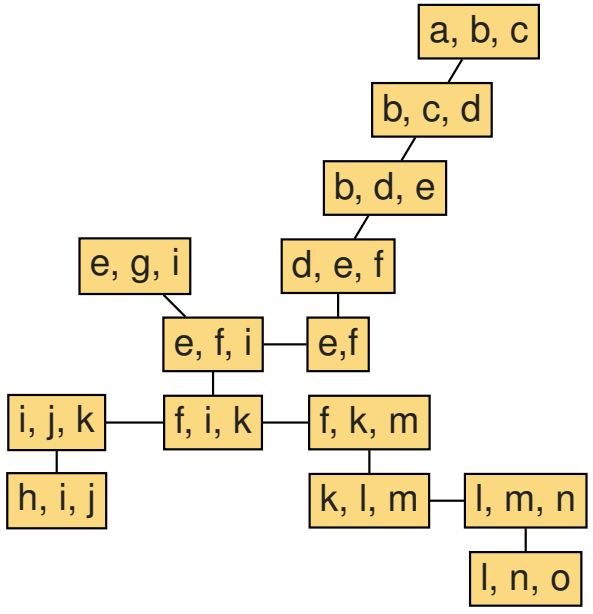




Definition of treewidth

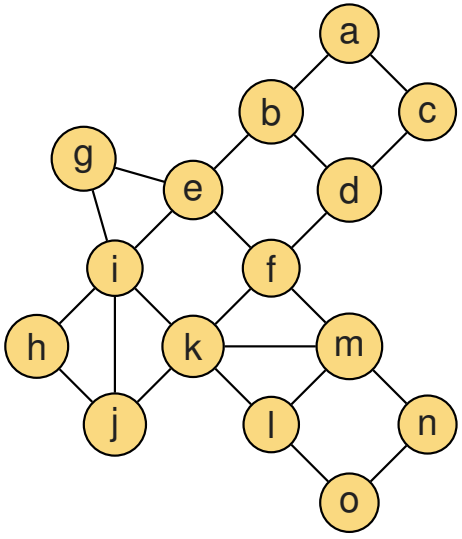


Graph G

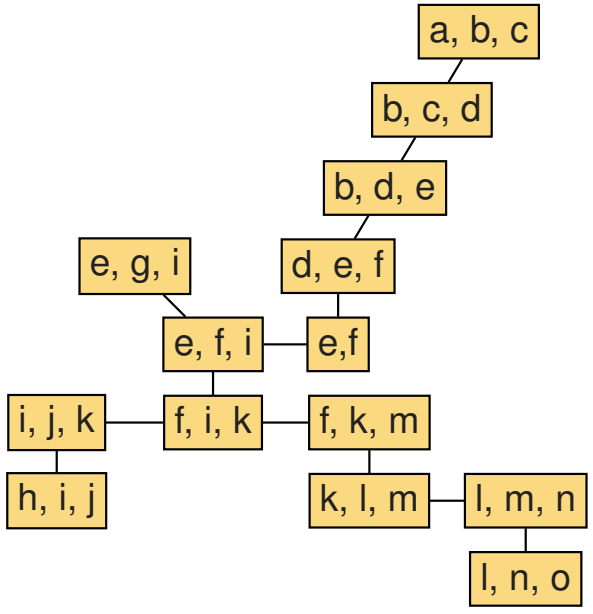


A tree decomposition of G

Definition of treewidth



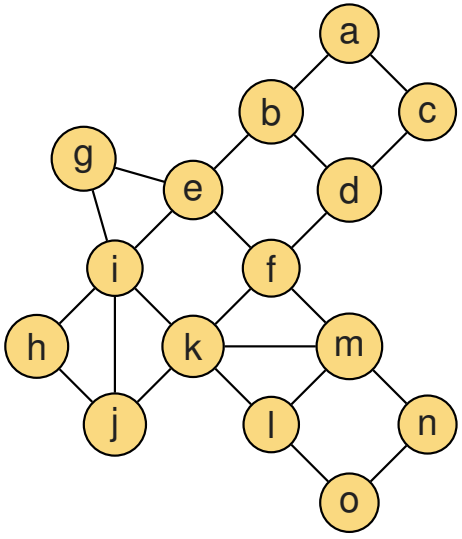
Graph G



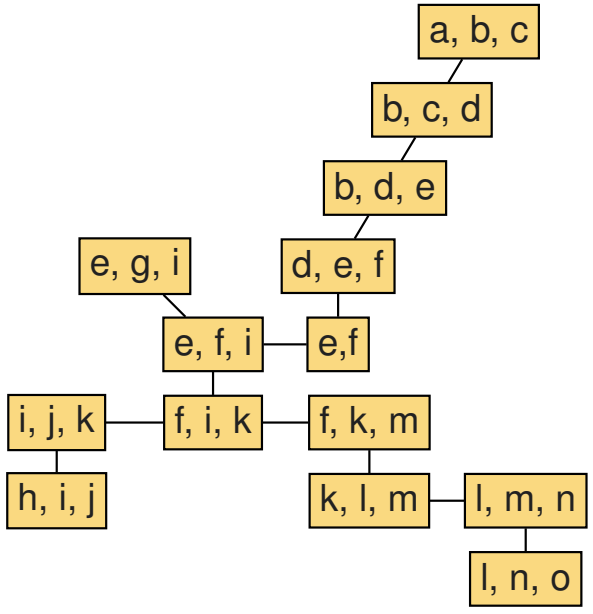
A tree decomposition of G

1. Every vertex should be in a bag

Definition of treewidth



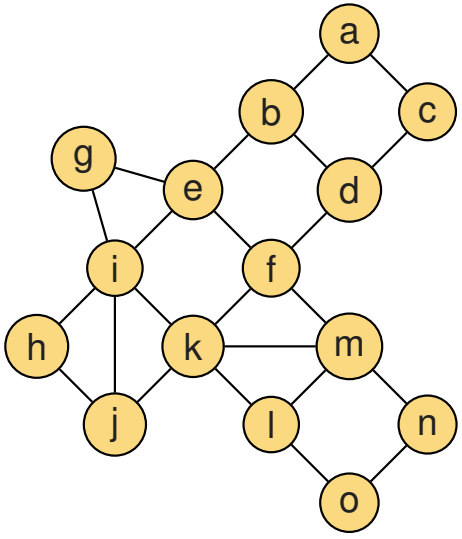
Graph G



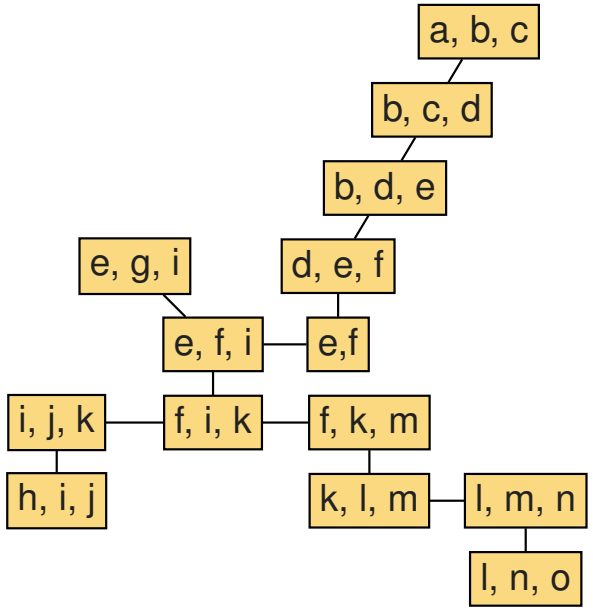
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag

Definition of treewidth



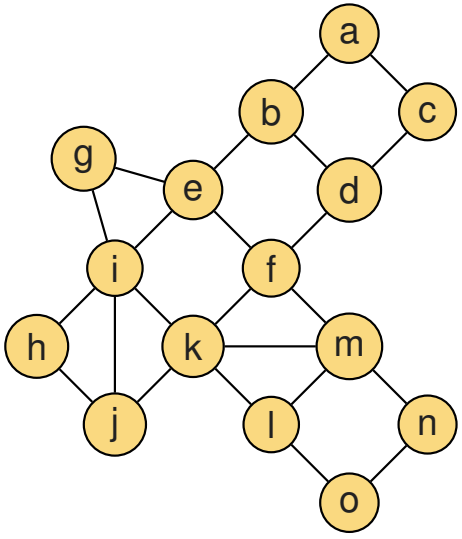
Graph G



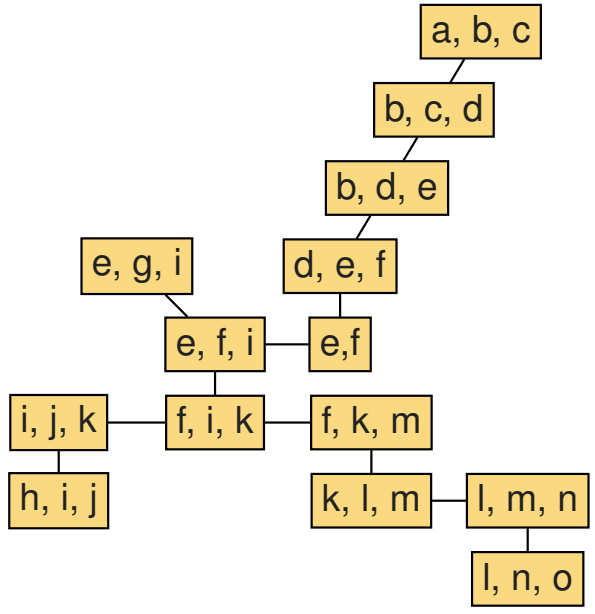
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree

Definition of treewidth



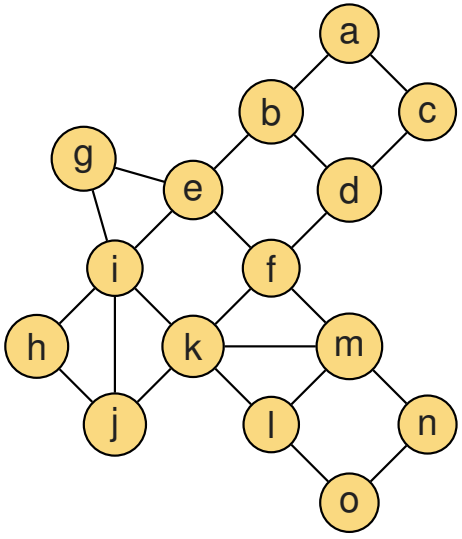
Graph G



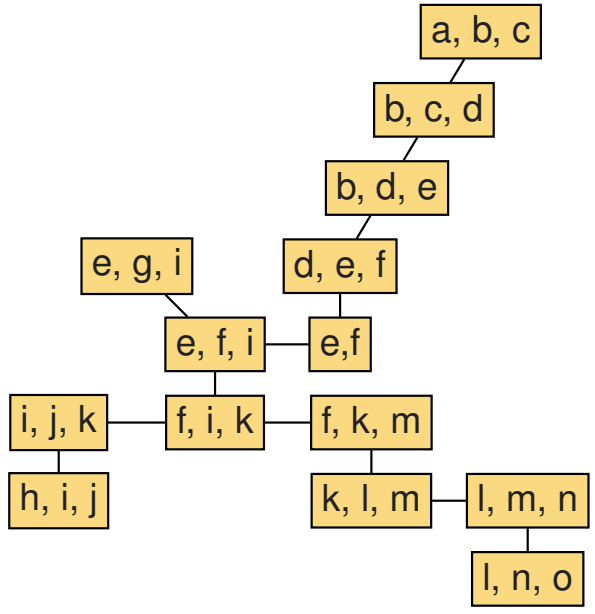
A tree decomposition of G

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree
4. Width = maximum bag size -1

Definition of treewidth



Graph G

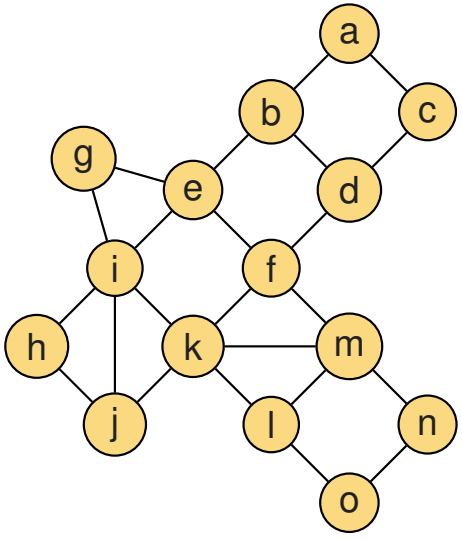


A tree decomposition of G

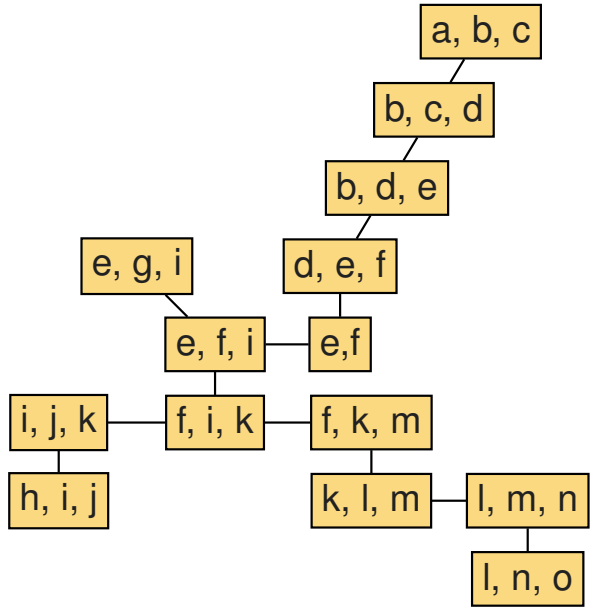
Width = 2

1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree
4. Width = maximum bag size -1

Definition of treewidth



Graph G

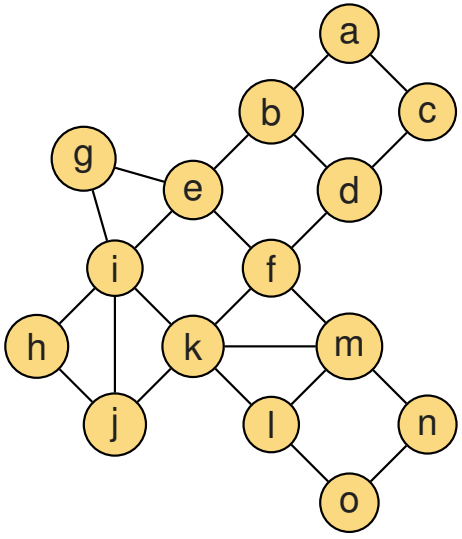


A tree decomposition of G

Width = 2

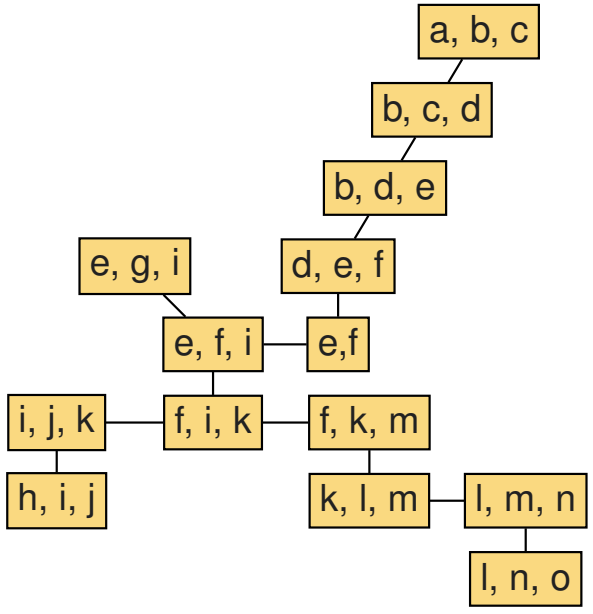
1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree
4. Width = maximum bag size -1
5. Treewidth of G = minimum width of tree decomposition of G

Definition of treewidth



Graph G

Treewidth 2

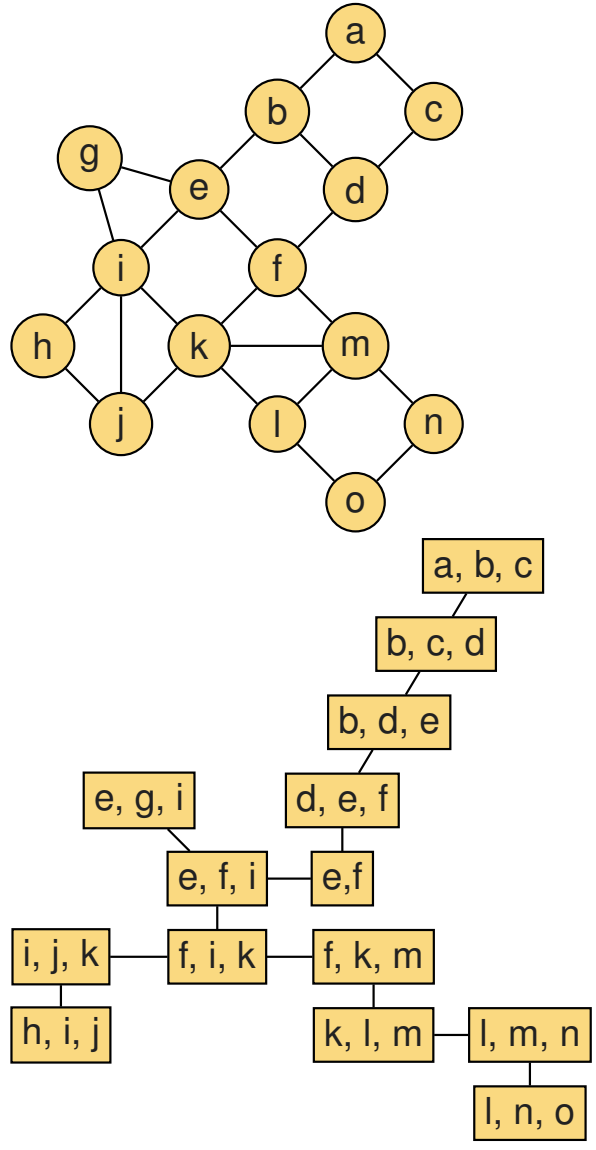


A tree decomposition of G

Width = 2

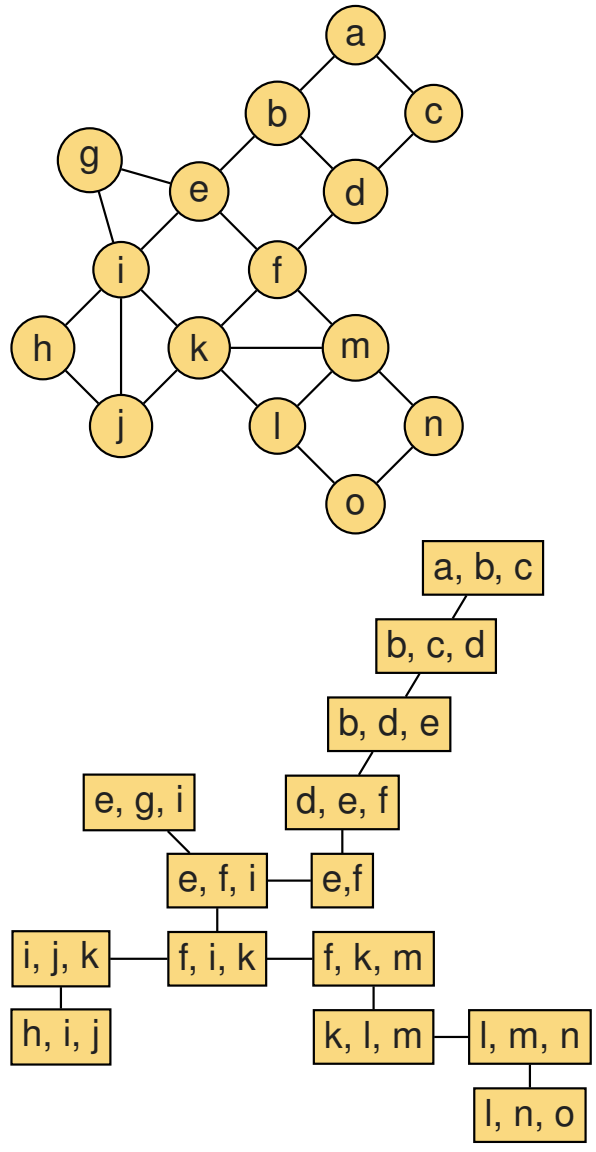
1. Every vertex should be in a bag
2. Every edge should be in a bag
3. For every vertex v, the bags containing v should form a connected subtree
4. Width = maximum bag size -1
5. Treewidth of G = minimum width of tree decomposition of G

Why treewidth



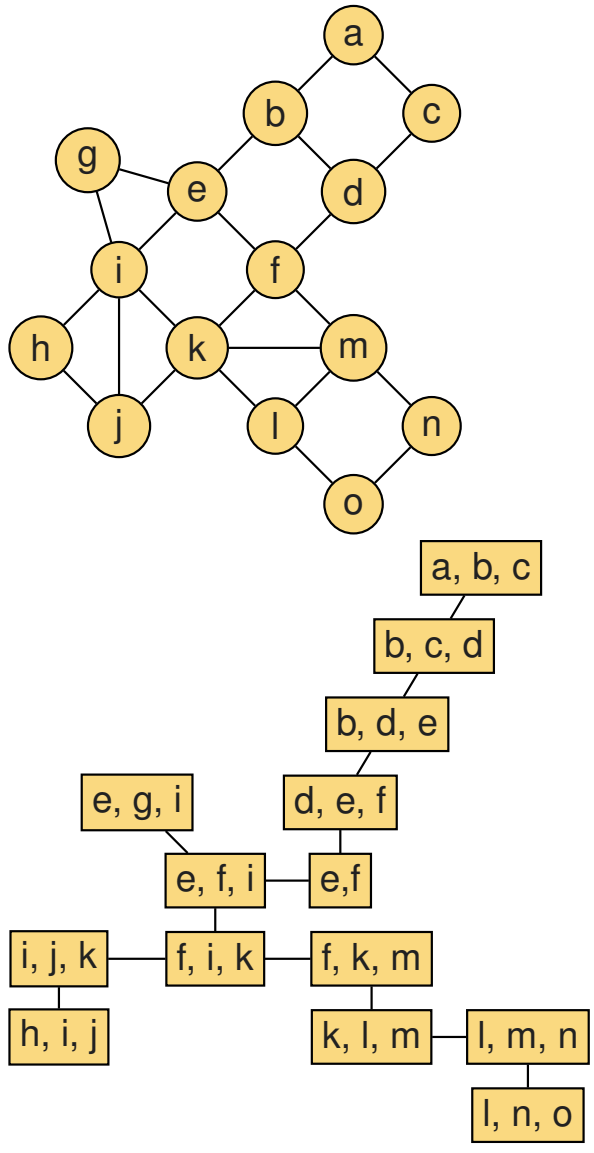
Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices



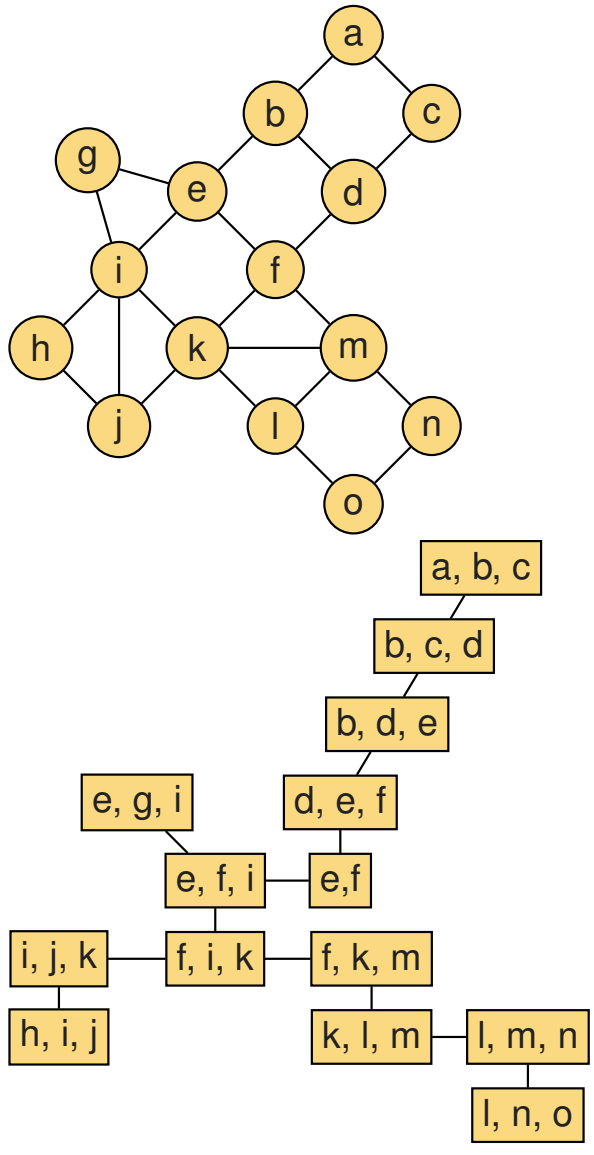
Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices
- Often $2^{\mathcal{O}(k)} n$ time algorithms



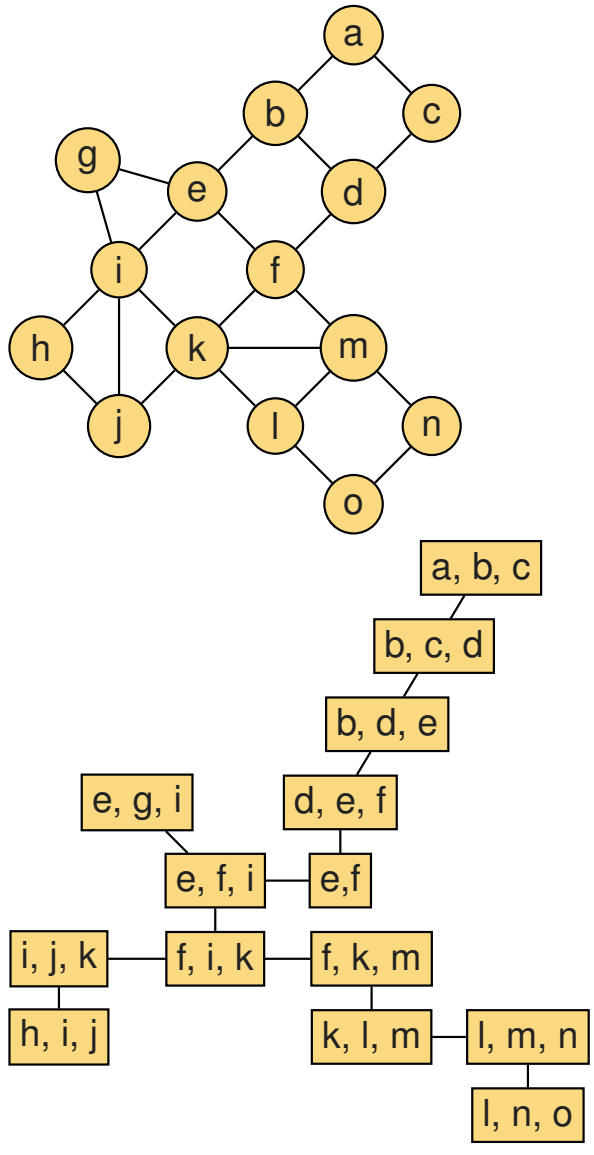
Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices
- Often $2^{\mathcal{O}(k)} n$ time algorithms
- Not only about graph problems:



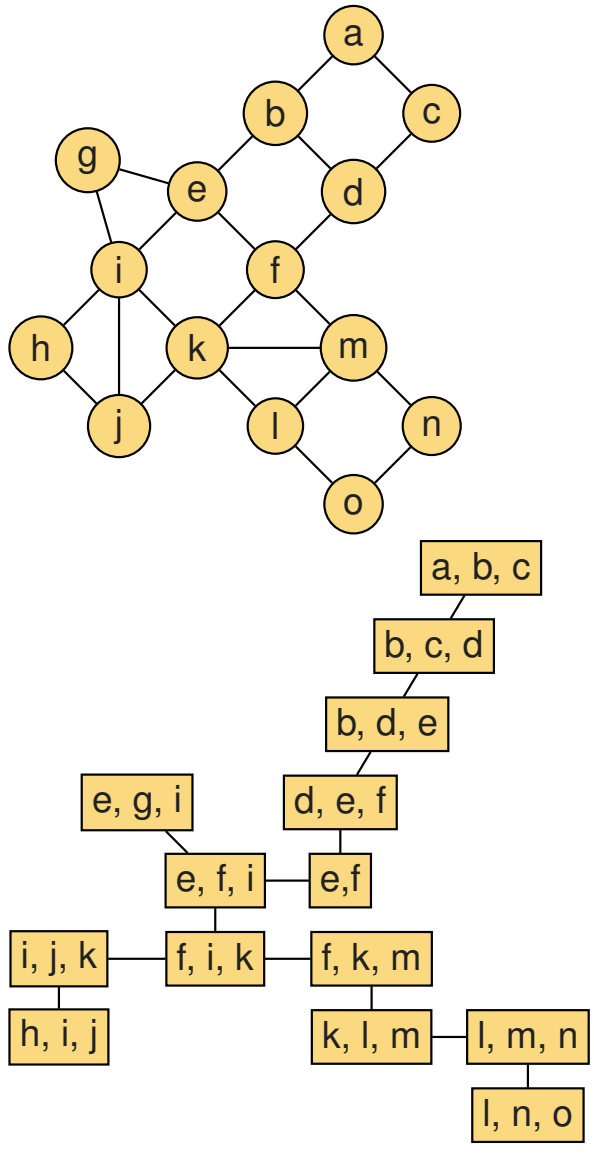
Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices
- Often $2^{\mathcal{O}(k)} n$ time algorithms
- Not only about graph problems:
  - Constraint satisfaction [Freuder '90, Dechter & Pearl '89]



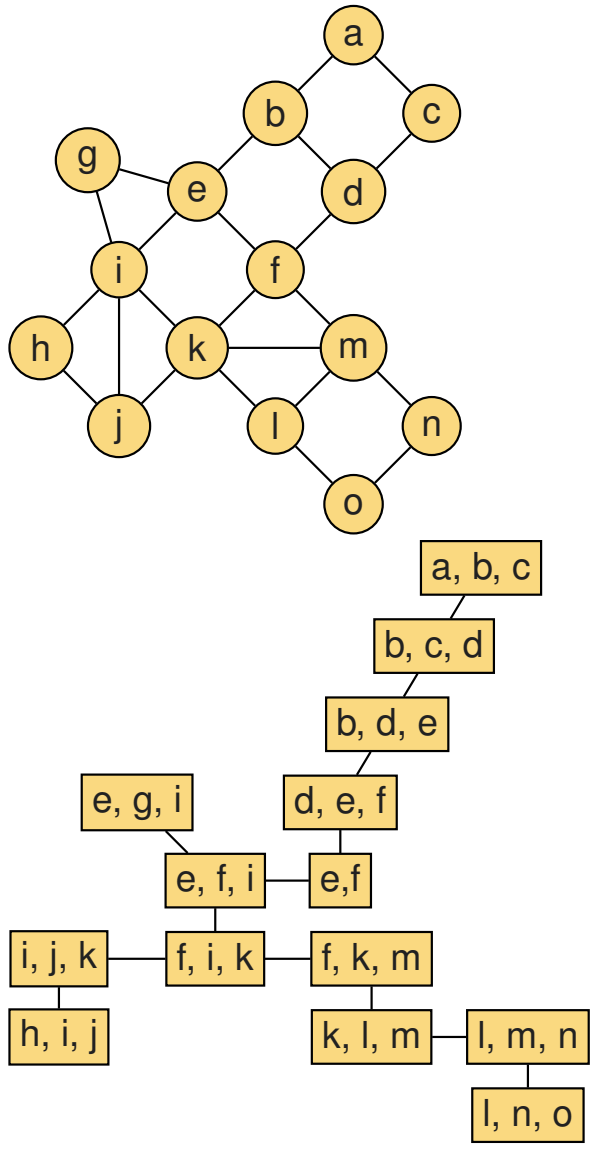
Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices
- Often $2^{\mathcal{O}(k)} n$ time algorithms
- Not only about graph problems:
  - Constraint satisfaction [Freuder '90, Dechter & Pearl '89]
  - Probabilistic inference [Lauritzen & Spiegelhalter '88]



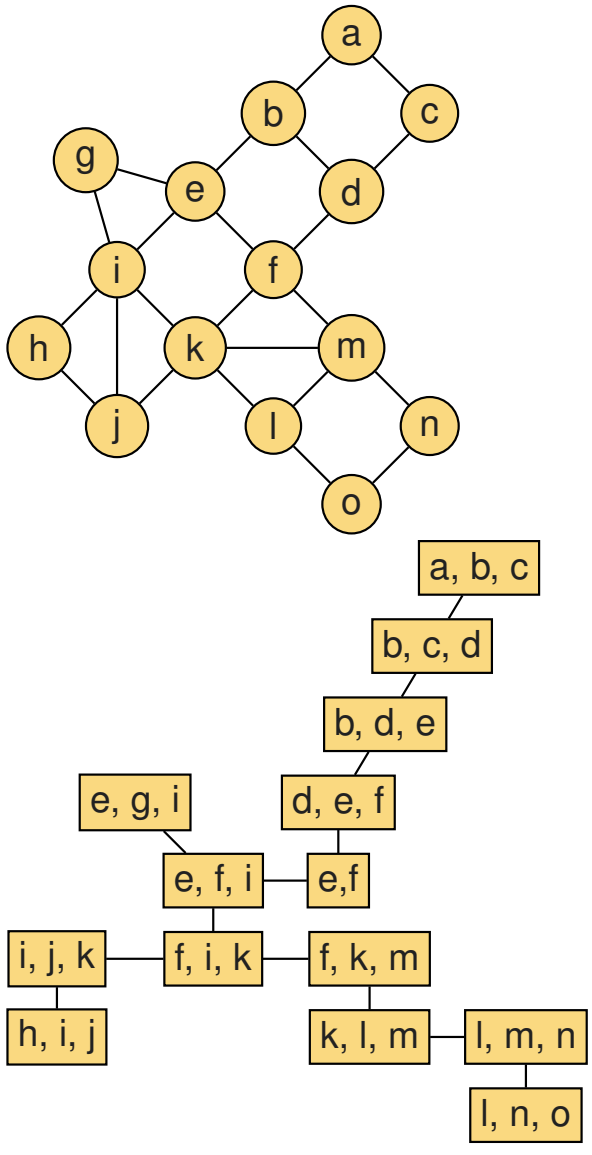
Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices
- Often $2^{\mathcal{O}(k)} n$ time algorithms
- Not only about graph problems:
  - Constraint satisfaction [Freuder '90, Dechter & Pearl '89]
  - Probabilistic inference [Lauritzen & Spiegelhalter '88]
  - Compiler optimization [Thorup '98, Bodlaender, Gusted & Telle '98]



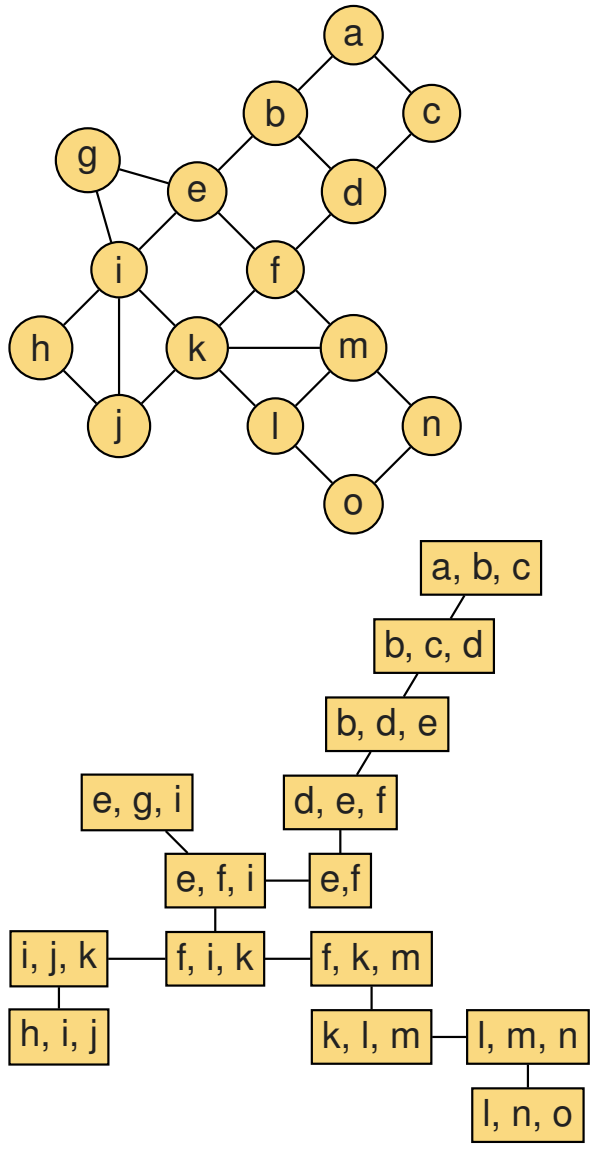
Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices
- Often $2^{\mathcal{O}(k)} n$ time algorithms
- Not only about graph problems:
  - Constraint satisfaction [Freuder '90, Dechter & Pearl '89]
  - Probabilistic inference [Lauritzen & Spiegelhalter '88]
  - Compiler optimization [Thorup '98, Bodlaender, Gusted & Telle '98]
  - Quantum computer simulation [Markov & Shi '08]



Why treewidth

- Many NP-hard graph problems can be solved in time $f(k) \cdot n$
  - k is the width of a given tree decomposition
  - n is the number of vertices
- Often $2^{\mathcal{O}(k)} n$ time algorithms
- Not only about graph problems:
  - Constraint satisfaction [Freuder '90, Dechter & Pearl '89]
  - Probabilistic inference [Lauritzen & Spiegelhalter '88]
  - Compiler optimization [Thorup '98, Bodlaender, Gusted & Telle '98]
  - Quantum computer simulation [Markov & Shi '08]



Need the tree decomposition!

Topic of this thesis:

Topic of this thesis:

Algorithms for computing small-width tree decompositions

Topic of this thesis:

Algorithms for computing small-width tree decompositions

Paper 1: Tuukka Korhonen. *A single-exponential time 2-approximation algorithm for treewidth.* (FOCS 2021, to appear in SICOMP)

Paper 2: Tuukka Korhonen and Daniel Lokshtanov. *An improved parameterized algorithm for treewidth.* (STOC 2023)

} Computing tree decompositions

Topic of this thesis:

Algorithms for computing small-width tree decompositions

Paper 1: Tuukka Korhonen. *A single-exponential time 2-approximation algorithm for treewidth.* (FOCS 2021, to appear in SICOMP)

Paper 2: Tuukka Korhonen and Daniel Lokshtanov. *An improved parameterized algorithm for treewidth.* (STOC 2023)

} Computing tree decompositions

Paper 3: Tuukka Korhonen, Konrad Majewski, Wojciech Nadara, Michał Pilipczuk, and Marek Sokołowski. *Dynamic treewidth.* (FOCS 2023)

Maintaining tree decompositions of dynamic graphs

Topic of this thesis:

Algorithms for computing small-width tree decompositions

Paper 1: Tuukka Korhonen. *A single-exponential time 2-approximation algorithm for treewidth.* (FOCS 2021, to appear in SICOMP)

Paper 2: Tuukka Korhonen and Daniel Lokshtanov. *An improved parameterized algorithm for treewidth.* (STOC 2023)

} Computing tree decompositions

Paper 3: Tuukka Korhonen, Konrad Majewski, Wojciech Nadara, Michał Pilipczuk, and Marek Sokołowski. *Dynamic treewidth.* (FOCS 2023)

Maintaining tree decompositions of dynamic graphs

Paper 4: Fedor V. Fomin and Tuukka Korhonen. *Fast FPT-approximation of branchwidth.* (STOC 2022, to appear in SICOMP)

Computing branch and rank decompositions

Previous work on treewidth computing

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]
- 4-approximation in $2^{\mathcal{O}(k)} \cdot n^2$ time, exact in $f(k) \cdot n^2$ time [Robertson & Seymour '86]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]
- 4-approximation in $2^{\mathcal{O}(k)} \cdot n^2$ time, exact in $f(k) \cdot n^2$ time [Robertson & Seymour '86]
- Constant-approximation in $k^{\mathcal{O}(k)} \cdot n \operatorname{polylog} n$ time [Matoušek&Thomas'91,Lagergren'91, Reed '92]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]
- 4-approximation in $2^{\mathcal{O}(k)} \cdot n^2$ time, exact in $f(k) \cdot n^2$ time [Robertson & Seymour '86]
- Constant-approximation in $k^{\mathcal{O}(k)} \cdot n \operatorname{polylog} n$ time [Matoušek&Thomas'91,Lagergren'91, Reed '92]
- Exact in $2^{\mathcal{O}(k^3)} \cdot n$ time [Bodlaender '96]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]
- 4-approximation in $2^{\mathcal{O}(k)} \cdot n^2$ time, exact in $f(k) \cdot n^2$ time [Robertson & Seymour '86]
- Constant-approximation in $k^{\mathcal{O}(k)} \cdot n \operatorname{polylog} n$ time [Matoušek&Thomas'91,Lagergren'91, Reed '92]
- Exact in $2^{\mathcal{O}(k^3)} \cdot n$ time [Bodlaender '96]
  - Using the dynamic programming of [Bodlaender & Kloks '96]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]
- 4-approximation in $2^{\mathcal{O}(k)} \cdot n^2$ time, exact in $f(k) \cdot n^2$ time [Robertson & Seymour '86]
- Constant-approximation in $k^{\mathcal{O}(k)} \cdot n \operatorname{polylog} n$ time [Matoušek&Thomas'91,Lagergren'91, Reed '92]
- Exact in $2^{\mathcal{O}(k^3)} \cdot n$ time [Bodlaender '96]
  - Using the dynamic programming of [Bodlaender & Kloks '96]
- $\mathcal{O}(\sqrt{\log k})$-approximation in polynomial time [Feige, Hajiaghayi & Lee '08]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]
- 4-approximation in $2^{\mathcal{O}(k)} \cdot n^2$ time, exact in $f(k) \cdot n^2$ time [Robertson & Seymour '86]
- Constant-approximation in $k^{\mathcal{O}(k)} \cdot n \operatorname{polylog} n$ time [Matoušek&Thomas'91,Lagergren'91, Reed '92]
- Exact in $2^{\mathcal{O}(k^3)} \cdot n$ time [Bodlaender '96]
  - Using the dynamic programming of [Bodlaender & Kloks '96]
- $\mathcal{O}(\sqrt{\log k})$-approximation in polynomial time [Feige, Hajiaghayi & Lee '08]
  - No polynomial-time constant-factor approximation, assuming the SSE-hypothesis [Wu, Austrin, Pitassi & Liu'14]

Previous work on treewidth computing

- NP-complete [Arnborg, Corneil, Proskurowski '87]
- $\mathcal{O}(n^{k+2})$ time [Arnborg, Corneil, Proskurowski '87]
- 4-approximation in $2^{\mathcal{O}(k)} \cdot n^2$ time, exact in $f(k) \cdot n^2$ time [Robertson & Seymour '86]
- Constant-approximation in $k^{\mathcal{O}(k)} \cdot n \operatorname{polylog} n$ time [Matoušek&Thomas'91,Lagergren'91, Reed '92]
- Exact in $2^{\mathcal{O}(k^3)} \cdot n$ time [Bodlaender '96]
  - Using the dynamic programming of [Bodlaender & Kloks '96]
- $\mathcal{O}(\sqrt{\log k})$-approximation in polynomial time [Feige, Hajiaghayi & Lee '08]
  - No polynomial-time constant-factor approximation, assuming the SSE-hypothesis [Wu, Austrin, Pitassi & Liu'14]
- 5-approximation in $2^{\mathcal{O}(k)} \cdot n$ time [Bodlaender,Drange,Dregi,Fomin,Lokshtanov&Pilipczuk'16]

New contributions to treewidth computing

New contributions to treewidth computing

Theorem (This thesis, paper 1)

There is a $2^{\mathcal{O}(k)} \cdot n$ time 2-approximation algorithm for treewidth.

New contributions to treewidth computing

Theorem (This thesis, paper 1)

There is a $2^{\mathcal{O}(k)} \cdot n$ time 2-approximation algorithm for treewidth.

- Can be compared to the 5-approximation in $2^{\mathcal{O}(k)} \cdot n$ time by [Bodlaender, Drange, Dregi, Fomin, Lokshtanov & Pilipczuk '16]

New contributions to treewidth computing

Theorem (This thesis, paper 1)

There is a $2^{\mathcal{O}(k)} \cdot n$ time 2-approximation algorithm for treewidth.

- Can be compared to the 5-approximation in $2^{\mathcal{O}(k)} \cdot n$ time by [Bodlaender, Drange, Dregi, Fomin, Lokshtanov & Pilipczuk '16]

Theorem (This thesis, paper 2)

There is a $2^{\mathcal{O}(k^2)} \cdot n^4$ time exact algorithm for treewidth.

New contributions to treewidth computing

Theorem (This thesis, paper 1)

There is a $2^{\mathcal{O}(k)} \cdot n$ time 2-approximation algorithm for treewidth.

- Can be compared to the 5-approximation in $2^{\mathcal{O}(k)} \cdot n$ time by [Bodlaender, Drange, Dregi, Fomin, Lokshtanov & Pilipczuk '16]

Theorem (This thesis, paper 2)

There is a $2^{\mathcal{O}(k^2)} \cdot n^4$ time exact algorithm for treewidth.

- Can be compared to the $2^{\mathcal{O}(k^3)} \cdot n$ time algorithm by [Bodlaender '96]

New contributions to treewidth computing

Theorem (This thesis, paper 1)

There is a $2^{\mathcal{O}(k)} \cdot n$ time 2-approximation algorithm for treewidth.

- Can be compared to the 5-approximation in $2^{\mathcal{O}(k)} \cdot n$ time by [Bodlaender, Drange, Dregi, Fomin, Lokshtanov & Pilipczuk '16]

Theorem (This thesis, paper 2)

There is a $2^{\mathcal{O}(k^2)} \cdot n^4$ time exact algorithm for treewidth.

- Can be compared to the $2^{\mathcal{O}(k^3)} \cdot n$ time algorithm by [Bodlaender '96]
- Solves the open problem of whether there is a $2^{o(k^3)} \cdot n^{\mathcal{O}(1)}$ time exact algorithm for treewidth

Dynamic treewidth

Dynamic treewidth

Goal: Maintain a good tree decomposition of a treewidth-k graph that is updated by

- edge additions
- edge deletions

Dynamic treewidth

Goal: Maintain a good tree decomposition of a treewidth-k graph that is updated by

- edge additions
- edge deletions

Also, maintain any dynamic programming scheme on the decomposition

Dynamic treewidth

Goal: Maintain a good tree decomposition of a treewidth-k graph that is updated by

- edge additions
- edge deletions

Also, maintain any dynamic programming scheme on the decomposition

By [Bodlaender '96], $2^{\mathcal{O}(k^3)} \cdot n$ update time

Dynamic treewidth

Goal: Maintain a good tree decomposition of a treewidth-k graph that is updated by

- edge additions
- edge deletions

Also, maintain any dynamic programming scheme on the decomposition

By [Bodlaender '96], $2^{\mathcal{O}(k^3)} \cdot n$ update time

Can we do $o(n)$ update time for fixed k?

Dynamic treewidth

Goal: Maintain a good tree decomposition of a treewidth-k graph that is updated by

- edge additions
- edge deletions

Also, maintain any dynamic programming scheme on the decomposition

By [Bodlaender '96], $2^{\mathcal{O}(k^3)} \cdot n$ update time

Can we do $o(n)$ update time for fixed k?

- $\mathcal{O}(\log n)$ update time for $k = 2$ [Bodlaender '93]

Dynamic treewidth

Goal: Maintain a good tree decomposition of a treewidth-k graph that is updated by

- edge additions
- edge deletions

Also, maintain any dynamic programming scheme on the decomposition

By [Bodlaender '96], $2^{\mathcal{O}(k^3)} \cdot n$ update time

Can we do $o(n)$ update time for fixed k?

- $\mathcal{O}(\log n)$ update time for $k = 2$ [Bodlaender '93]
- $n^{o(1)}$ amortized update time, but only $n^{o(1)}$-approximate, and only for bounded-degree graphs [Goranci, Räcke, Saranurak & Tan '21]

Dynamic treewidth

Goal: Maintain a good tree decomposition of a treewidth-k graph that is updated by

- edge additions
- edge deletions

Also, maintain any dynamic programming scheme on the decomposition

By [Bodlaender '96], $2^{\mathcal{O}(k^3)} \cdot n$ update time

Can we do $o(n)$ update time for fixed k?

- $\mathcal{O}(\log n)$ update time for $k = 2$ [Bodlaender '93]
- $n^{o(1)}$ amortized update time, but only $n^{o(1)}$-approximate, and only for bounded-degree graphs [Goranci, Räcke, Saranurak & Tan '21]

Theorem (This thesis, paper 3)

Data structure for maintaining 6-approximate tree decomposition of a dynamic graph with treewidth at most k, with amortized update time $f(k) \cdot n^{o(1)}$. Supports also maintaining any dynamic programming scheme.

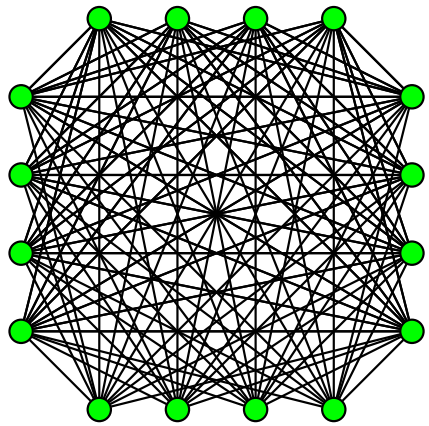
Rankwidth

Rank decompositions and rankwidth

Rankwidth

Rank decompositions and rankwidth

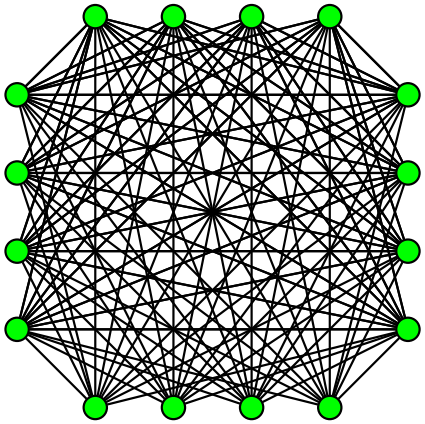
- Similar to tree decompositions and treewidth, but suitable also for dense graphs



Rankwidth

Rank decompositions and rankwidth

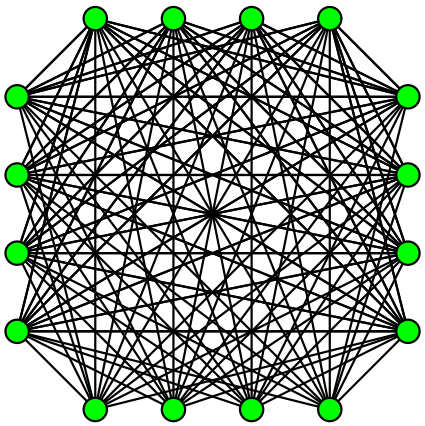
- Similar to tree decompositions and treewidth, but suitable also for dense graphs
- Introduced by [Oum & Seymour '06] to approximate cliquewidth, which was introduced by [Courcelle, Engelfriet & Rozenberg '93]



Rankwidth

Rank decompositions and rankwidth

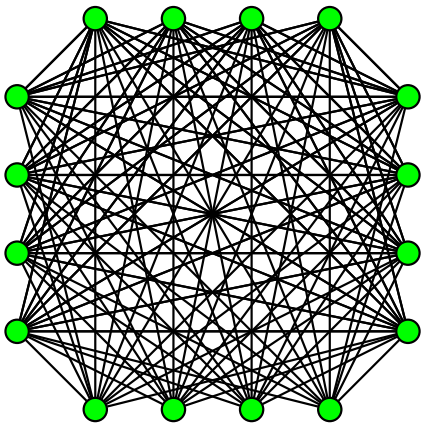
- Similar to tree decompositions and treewidth, but suitable also for dense graphs
- Introduced by [Oum & Seymour '06] to approximate cliquewidth, which was introduced by [Courcelle, Engelfriet & Rozenberg '93]
- $\mathcal{O}(8^k \cdot n^9 \log n)$ time 3-approximation [Oum&Seymour'06]



Rankwidth

Rank decompositions and rankwidth

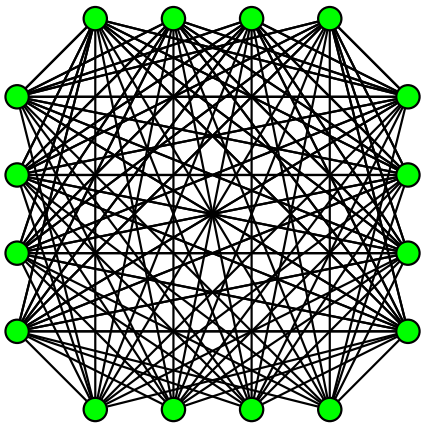
- Similar to tree decompositions and treewidth, but suitable also for dense graphs
- Introduced by [Oum & Seymour '06] to approximate cliquewidth, which was introduced by [Courcelle, Engelfriet & Rozenberg '93]
- $\mathcal{O}(8^k \cdot n^9 \log n)$ time 3-approximation [Oum&Seymour'06]
- $f(k) \cdot n^3$ time exact [Hlinený & Oum'08]



Rankwidth

Rank decompositions and rankwidth

- Similar to tree decompositions and treewidth, but suitable also for dense graphs
- Introduced by [Oum & Seymour '06] to approximate cliquewidth, which was introduced by [Courcelle, Engelfriet & Rozenberg '93]
- $\mathcal{O}(8^k \cdot n^9 \log n)$ time 3-approximation [Oum&Seymour'06]
- $f(k) \cdot n^3$ time exact [Hlinený & Oum'08]



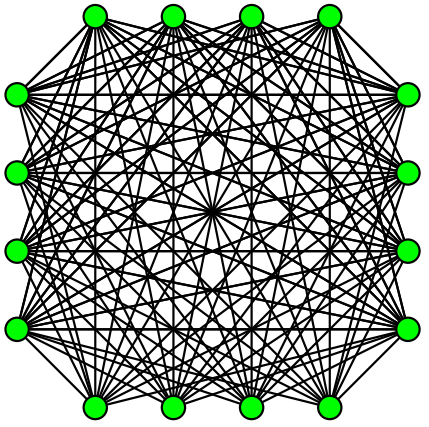
Theorem (This thesis, paper 4)

There is a $2^{2^{\mathcal{O}(k)}} \cdot n^2$ time 2-approximation algorithm for rankwidth.

Rankwidth

Rank decompositions and rankwidth

- Similar to tree decompositions and treewidth, but suitable also for dense graphs
- Introduced by [Oum & Seymour '06] to approximate cliquewidth, which was introduced by [Courcelle, Engelfriet & Rozenberg '93]
- $\mathcal{O}(8^k \cdot n^9 \log n)$ time 3-approximation [Oum&Seymour'06]
- $f(k) \cdot n^3$ time exact [Hlinený & Oum'08]



Theorem (This thesis, paper 4)

There is a $2^{2^{\mathcal{O}(k)}} \cdot n^2$ time 2-approximation algorithm for rankwidth.

Improves algorithms parameterized by rankwidth/cliquewidth from $f(k)n^3$ to $f(k)n^2$

Method

New method for computing decompositions of graphs: Local improvement

Method

New method for computing decompositions of graphs: Local improvement

- Repeatedly re-arrange the tree decomposition to make the largest bag smaller

Method

New method for computing decompositions of graphs: Local improvement

- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- Inspired by the proofs of [Thomas'90, Bellenbaum & Diestel '02] about lean tree decompositions

Method

New method for computing decompositions of graphs: Local improvement

- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- Inspired by the proofs of [Thomas'90, Bellenbaum & Diestel '02] about lean tree decompositions
- New ideas in both the graph-theoretic part of the re-arrangement, and in the efficient algorithmic implementation

Method

New method for computing decompositions of graphs: Local improvement

- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- Inspired by the proofs of [Thomas'90, Bellenbaum & Diestel '02] about lean tree decompositions
- New ideas in both the graph-theoretic part of the re-arrangement, and in the efficient algorithmic implementation
  - Introduced in Paper 1 for 2-approximating treewidth

Method

New method for computing decompositions of graphs: Local improvement

- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- Inspired by the proofs of [Thomas'90, Bellenbaum & Diestel '02] about lean tree decompositions
- New ideas in both the graph-theoretic part of the re-arrangement, and in the efficient algorithmic implementation
  - Introduced in Paper 1 for 2-approximating treewidth
  - Generalized in Paper 2 for exact treewidth

Method

New method for computing decompositions of graphs: Local improvement

- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- Inspired by the proofs of [Thomas'90, Bellenbaum & Diestel '02] about lean tree decompositions
- New ideas in both the graph-theoretic part of the re-arrangement, and in the efficient algorithmic implementation
  - Introduced in Paper 1 for 2-approximating treewidth
  - Generalized in Paper 2 for exact treewidth
  - Applied in Paper 3 for dynamic treewidth

Method

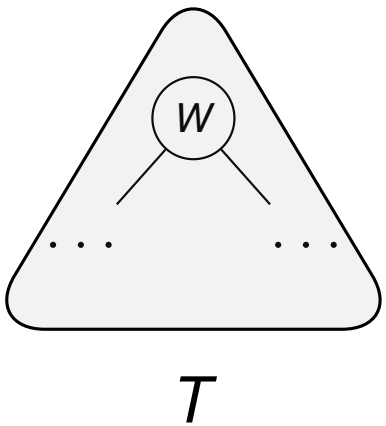
New method for computing decompositions of graphs: Local improvement

- Repeatedly re-arrange the tree decomposition to make the largest bag smaller
- Inspired by the proofs of [Thomas'90, Bellenbaum & Diestel '02] about lean tree decompositions
- New ideas in both the graph-theoretic part of the re-arrangement, and in the efficient algorithmic implementation
  - Introduced in Paper 1 for 2-approximating treewidth
  - Generalized in Paper 2 for exact treewidth
  - Applied in Paper 3 for dynamic treewidth
  - Extended in Paper 4 for 2-approximating rankwidth

Improvement operation for 2-approximating treewidth

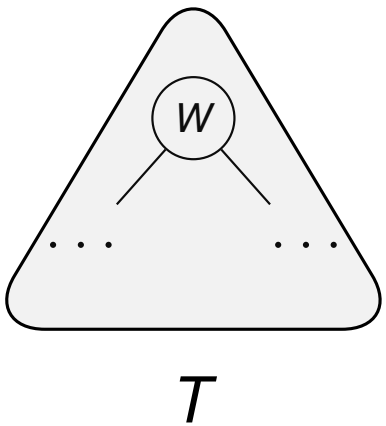
Improvement operation for 2-approximating treewidth

- Let W be the largest bag of a tree decomposition T of width $\geq 2k + 2$



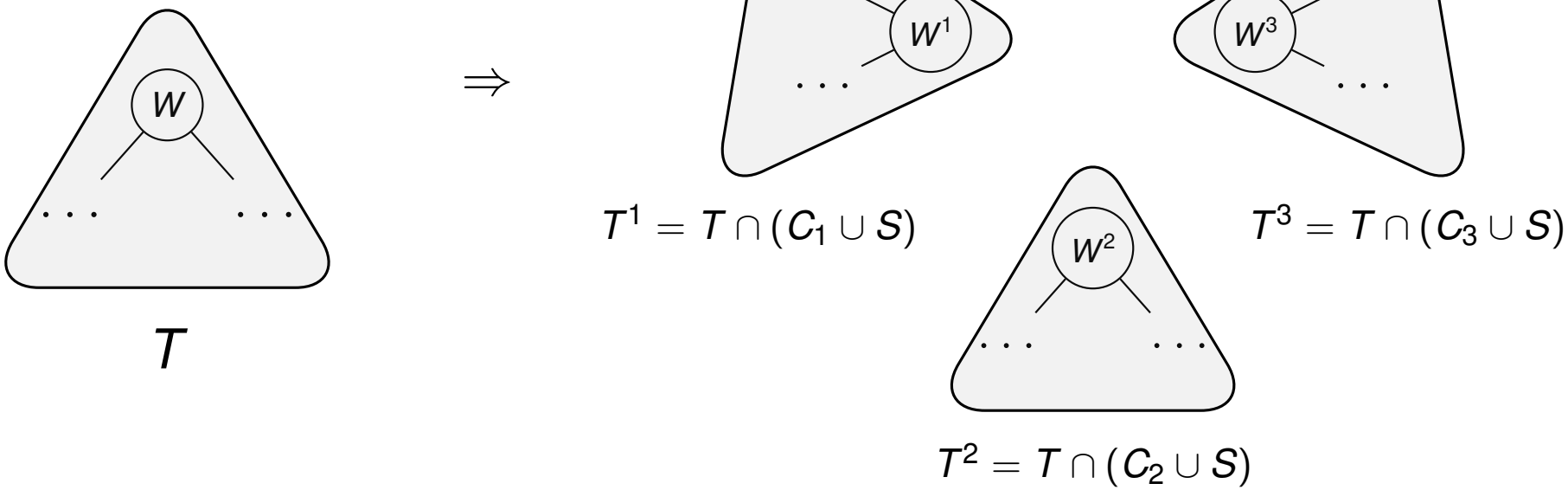
Improvement operation for 2-approximating treewidth

- Let W be the largest bag of a tree decomposition T of width $\geq 2k+2$
- Take a small balanced separator S of W with a partition (C_1, C_2, C_3, S) of V



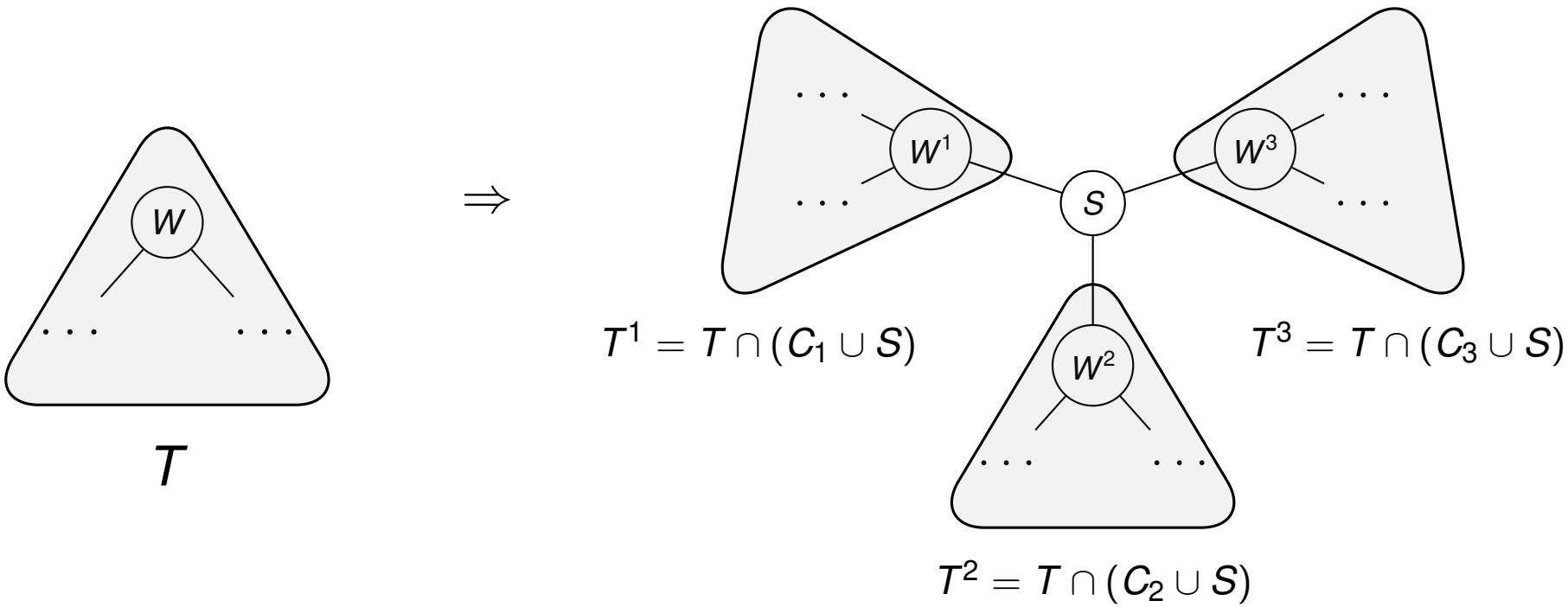
Improvement operation for 2-approximating treewidth

- Let W be the largest bag of a tree decomposition T of width $\geq 2k+2$
- Take a small balanced separator S of W with a partition (C_1, C_2, C_3, S) of V
- For each $i \in \{1,2,3\}$, obtain a tree decomposition $T^i = T \cap (C_i \cup S)$ by setting $B^i = B \cap (C_i \cup S)$ for each bag B of T.



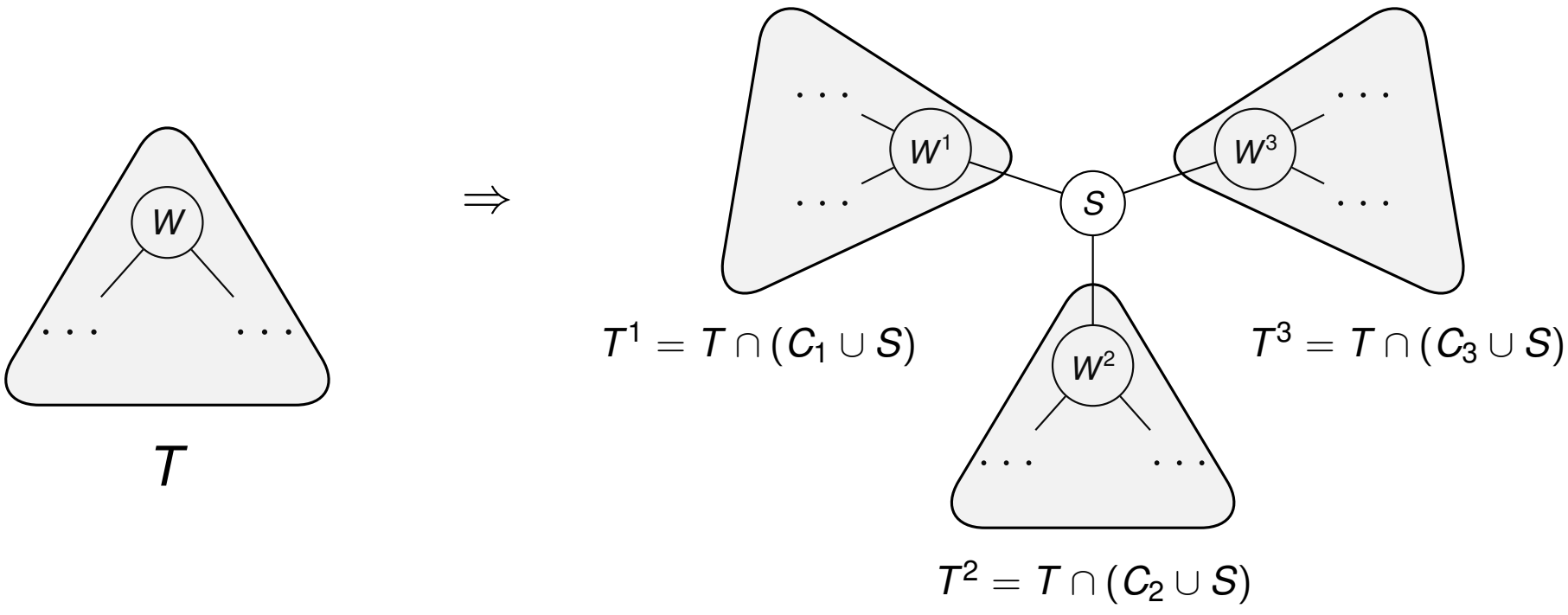
Improvement operation for 2-approximating treewidth

- Let W be the largest bag of a tree decomposition T of width $\geq 2k+2$
- Take a small balanced separator S of W with a partition (C_1, C_2, C_3, S) of V
- For each $i \in \{1,2,3\}$, obtain a tree decomposition $T^i = T \cap (C_i \cup S)$ by setting $B^i = B \cap (C_i \cup S)$ for each bag B of T.
- The following is almost a tree decomposition of G:



Improvement operation for 2-approximating treewidth

- Let W be the largest bag of a tree decomposition T of width $\geq 2k+2$
- Take a small balanced separator S of W with a partition (C_1, C_2, C_3, S) of V
- For each $i \in \{1,2,3\}$, obtain a tree decomposition $T^i = T \cap (C_i \cup S)$ by setting $B^i = B \cap (C_i \cup S)$ for each bag B of T.
- The following is almost a tree decomposition of G:



Except that vertices in S may violate the connectedness condition

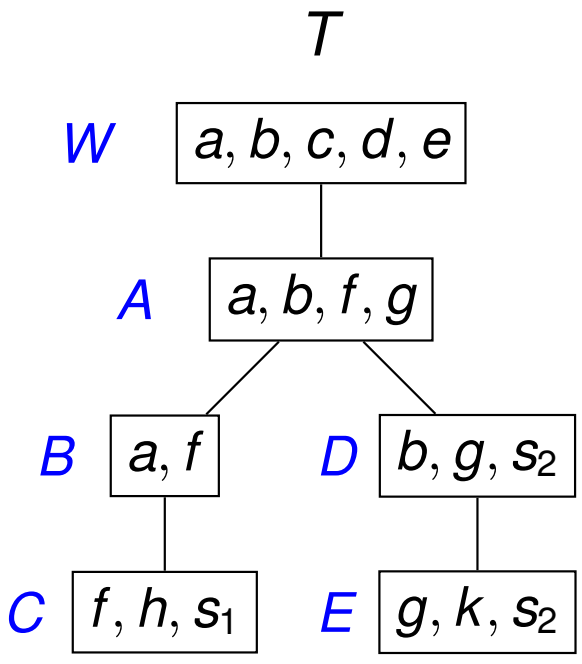
Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of S to bags

Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of S to bags

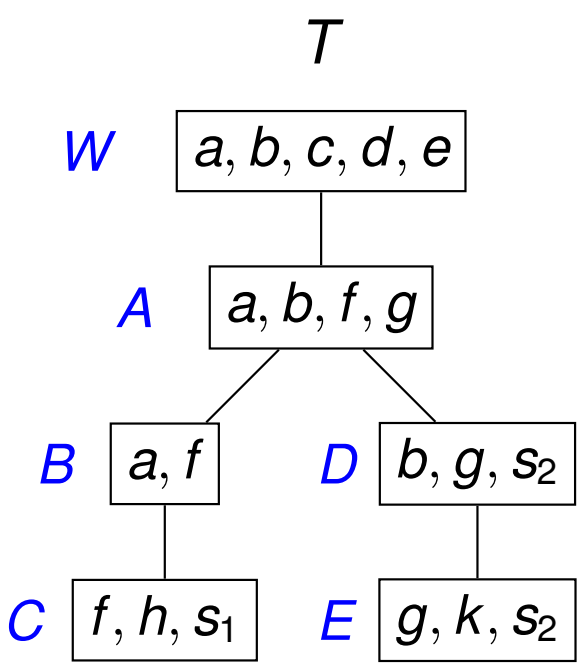
Example: Let $(C_1, C_2, C_3, S) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{s_1, s_2\})$ be the partition:

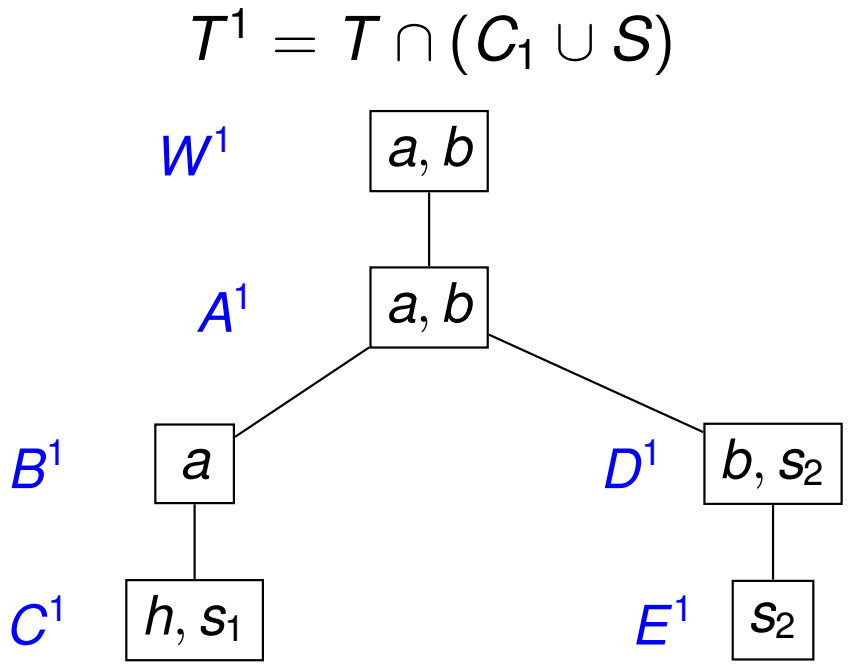


Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of S to bags

Example: Let $(C_1, C_2, C_3, S) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{s_1, s_2\})$ be the partition:

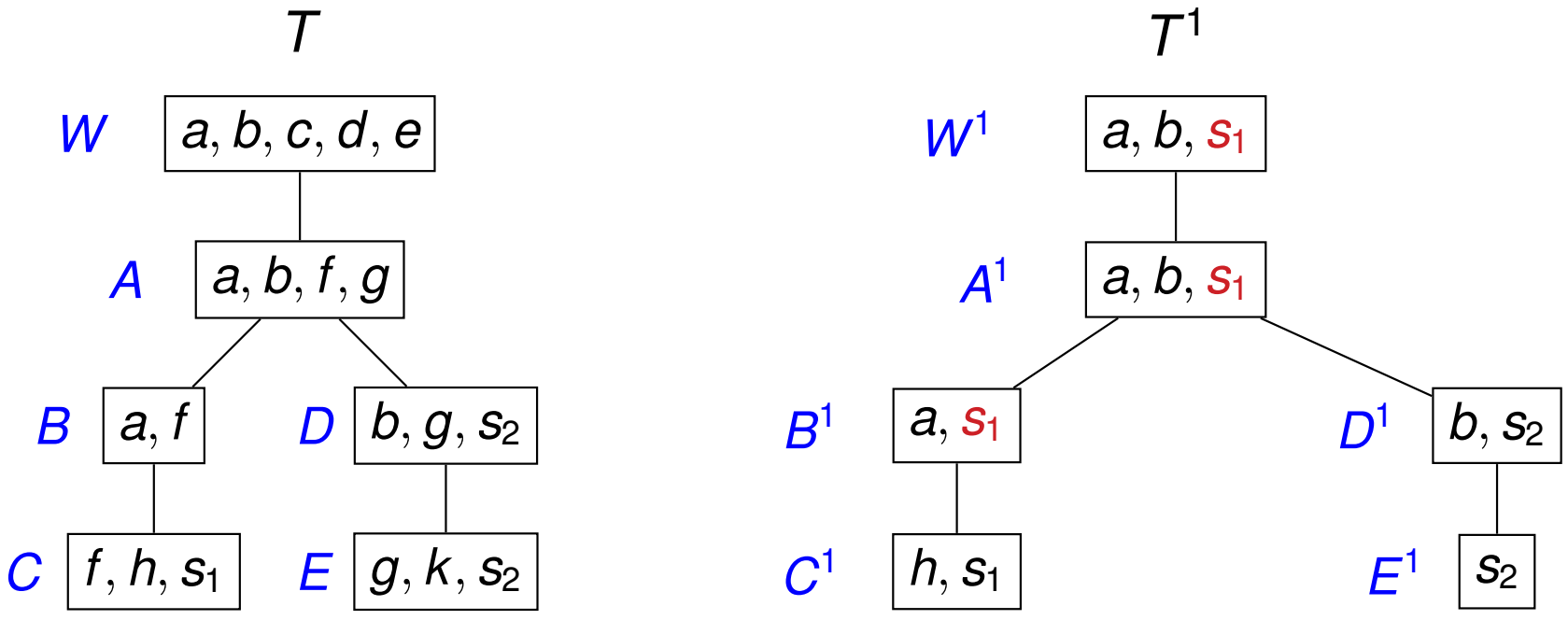




Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of S to bags

Example: Let $(C_1, C_2, C_3, S) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{s_1, s_2\})$ be the partition:

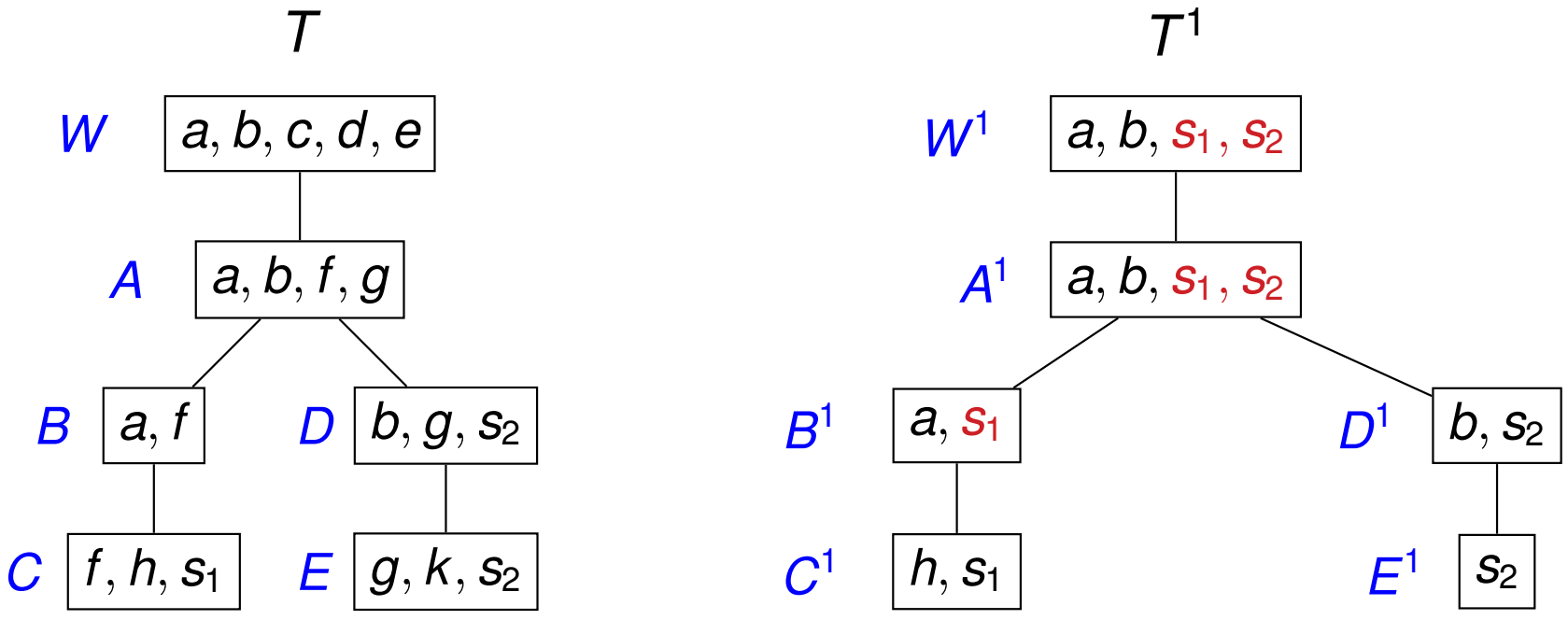


- Insert s_1 to B^1, A^1, and W^1

Fixing a tree decomposition

- Fix the connectedness condition by inserting vertices of S to bags

Example: Let $(C_1, C_2, C_3, S) = (\{a, b, h\}, \{c, d, f\}, \{e, g, k\}, \{s_1, s_2\})$ be the partition:



- Insert s_1 to B^1, A^1, and W^1
- Insert s_2 to A^1 and W^1

Analysis of the improvement

- Each bag B is replaced by bags B^1, B^2, B^3

Analysis of the improvement

- Each bag B is replaced by bags B^1, B^2, B^3

Lemma

If the balanced separator S is chosen according to specific criteria, then $|B^i| \leq |B|$ for all bags B and each $i \in \{1, 2, 3\}$.

Analysis of the improvement

- Each bag B is replaced by bags B^1, B^2, B^3

Lemma

If the balanced separator S is chosen according to specific criteria, then $|B^i| \leq |B|$ for all bags B and each $i \in \{1, 2, 3\}$.

- $|B^i| = |B|$ holds only in a degenerate case where we can throw B^j for $j \neq i$ away

Analysis of the improvement

- Each bag B is replaced by bags B^1, B^2, B^3

Lemma

If the balanced separator S is chosen according to specific criteria, then $|B^i| \leq |B|$ for all bags B and each $i \in \{1, 2, 3\}$.

- $|B^i| = |B|$ holds only in a degenerate case where we can throw B^j for $j \neq i$ away
- For the bag W, $|W^i| < |W|$ is ensured by the definition of the balanced separator

Analysis of the improvement

- Each bag B is replaced by bags B^1, B^2, B^3

Lemma

If the balanced separator S is chosen according to specific criteria, then $|B^i| \leq |B|$ for all bags B and each $i \in \{1, 2, 3\}$.

- $|B^i| = |B|$ holds only in a degenerate case where we can throw B^j for $j \neq i$ away
- For the bag W, $|W^i| < |W|$ is ensured by the definition of the balanced separator

$\Rightarrow$ The number of bags of size $|W|$ decreases

Conclusion

New method for computing width parameters of graphs: Local improvement

Conclusion

New method for computing width parameters of graphs: Local improvement

- Solutions to open problems about computing treewidth, dynamic treewidth, and computing rankwidth

Conclusion

New method for computing width parameters of graphs: Local improvement

- Solutions to open problems about computing treewidth, dynamic treewidth, and computing rankwidth
- Inspired by a proof about lean tree decompositions [Thomas'90, Bellenbaum & Diestel '02]

Conclusion

New method for computing width parameters of graphs: Local improvement

- Solutions to open problems about computing treewidth, dynamic treewidth, and computing rankwidth
- Inspired by a proof about lean tree decompositions [Thomas'90, Bellenbaum & Diestel '02]

Future directions:

Conclusion

New method for computing width parameters of graphs: Local improvement

- Solutions to open problems about computing treewidth, dynamic treewidth, and computing rankwidth
- Inspired by a proof about lean tree decompositions [Thomas'90, Bellenbaum & Diestel '02]

Future directions:

- Prove $2^{\Omega(k)}$ lower bound for treewidth under ETH ($2^{\Omega(\sqrt{k})}$ known)

Conclusion

New method for computing width parameters of graphs: Local improvement

- Solutions to open problems about computing treewidth, dynamic treewidth, and computing rankwidth
- Inspired by a proof about lean tree decompositions [Thomas'90, Bellenbaum & Diestel '02]

Future directions:

- Prove $2^{\Omega(k)}$ lower bound for treewidth under ETH ($2^{\Omega(\sqrt{k})}$ known)
- Dynamic treewidth in amortized $f(k) \cdot \operatorname{polylog} n$ time?

Conclusion

New method for computing width parameters of graphs: Local improvement

- Solutions to open problems about computing treewidth, dynamic treewidth, and computing rankwidth
- Inspired by a proof about lean tree decompositions [Thomas'90, Bellenbaum & Diestel '02]

Future directions:

- Prove $2^{\Omega(k)}$ lower bound for treewidth under ETH ($2^{\Omega(\sqrt{k})}$ known)
- Dynamic treewidth in amortized $f(k) \cdot \operatorname{polylog} n$ time?

Thank you!