CS 5003: Parameterized Algorithms

Lectures 32-33

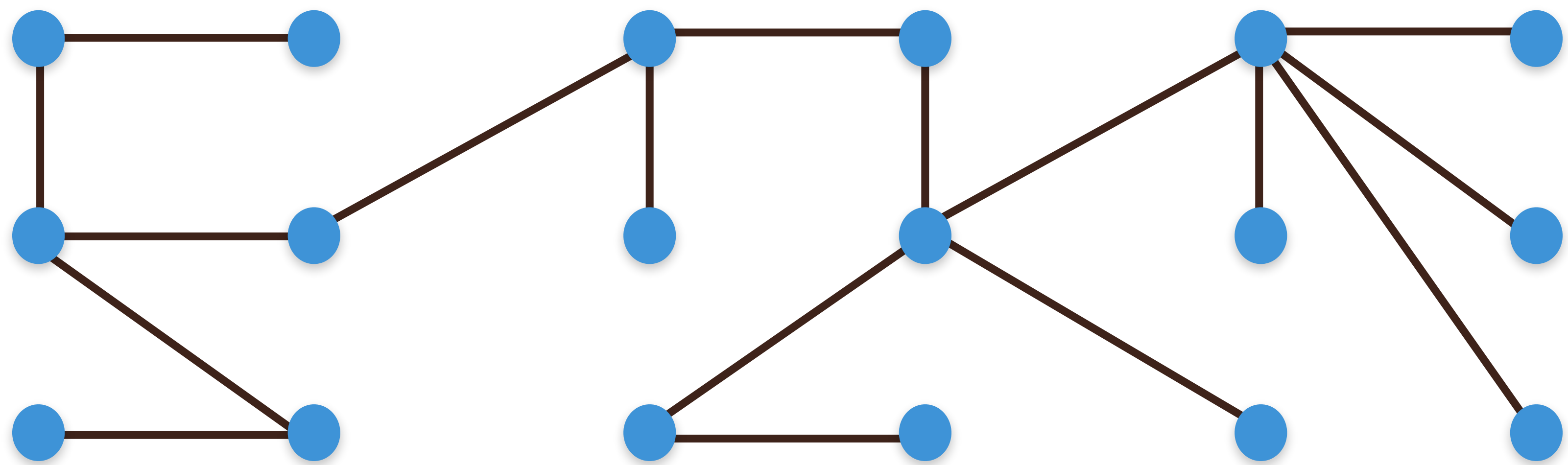
Krithika Ramaswamy

IIT Palakkad

Reference Books: Parameterized Algorithms by Cygan et al. and Kernelization by Fomin et al.

Trees

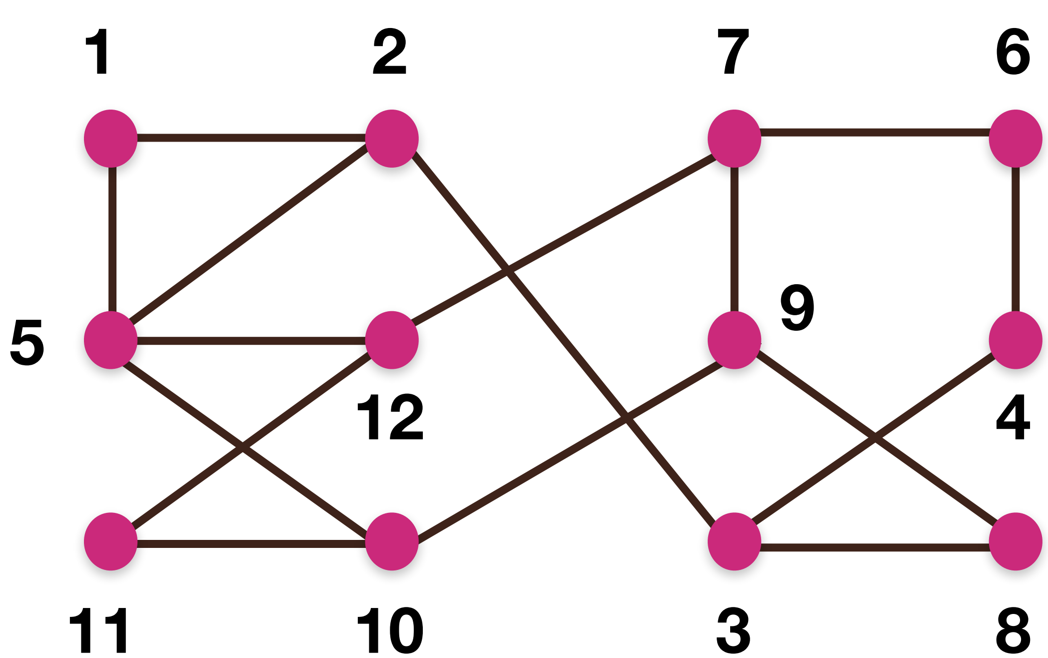
Undirected connected acyclic graphs

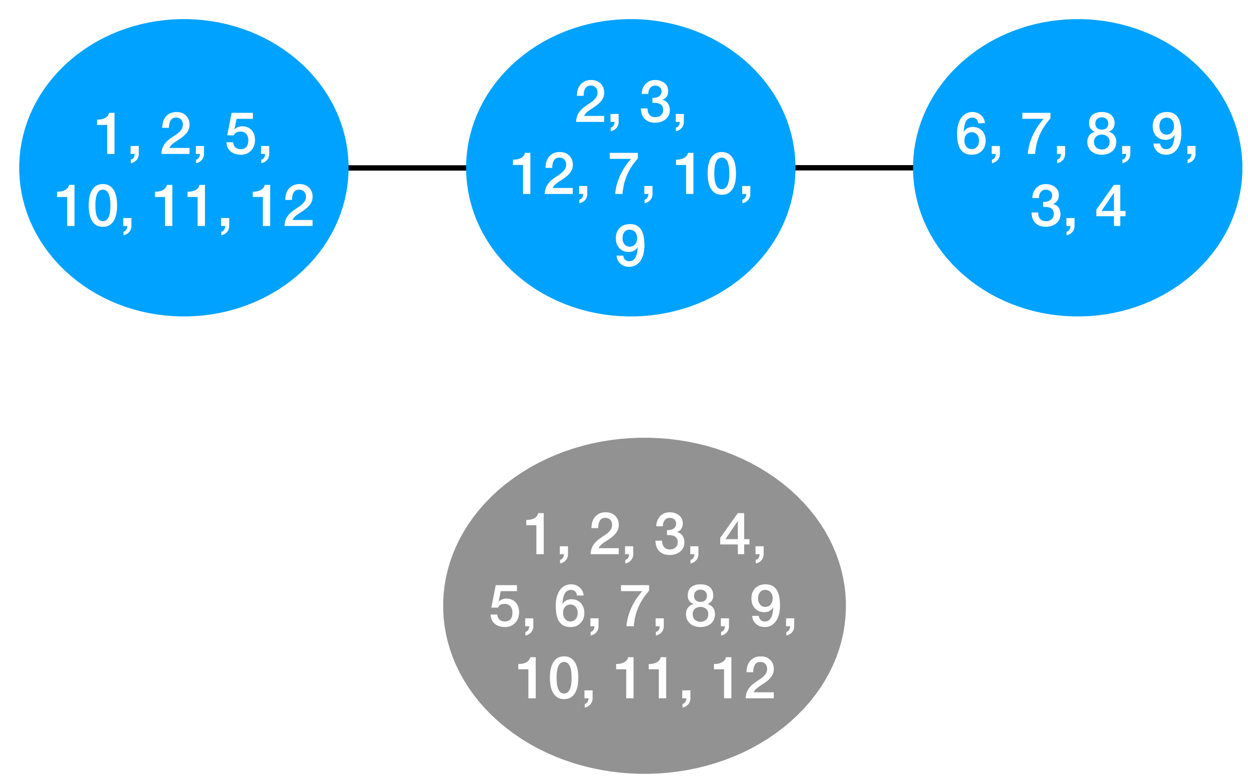


* Many problems that are NP-hard in general graphs are polynomial-time solvable in trees
  * Longest Path, Minimum Vertex Coloring
  * Minimum Feedback Vertex Set, Maximum Clique
  * Maximum Independent Set
  * Minimum Dominating Set

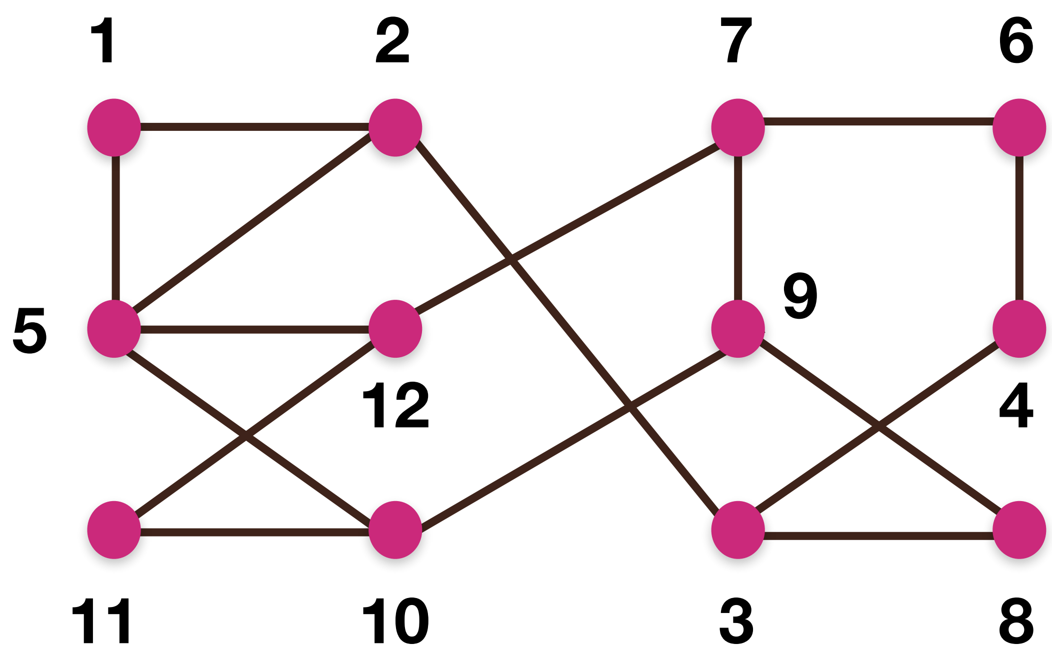
Treewidth

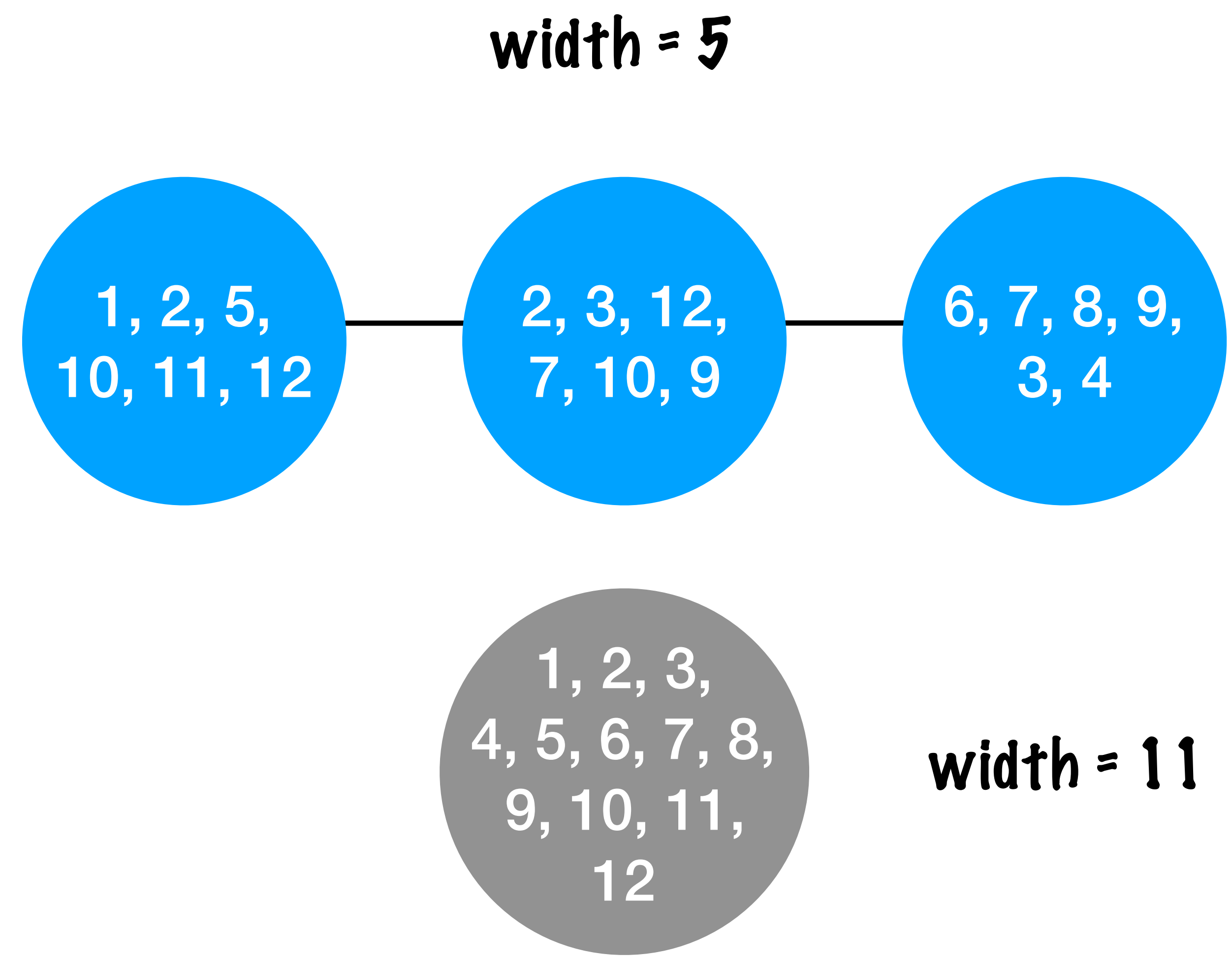
* A measure of how close a graph is to a tree
* A tree decomposition of a graph G is a pair (T,B) where T is a tree and B: V(T) -> $2^{V(G)}$ satisfies the following
  * For each vertex v in G, there is a node x in V(T) such that v is in B(x)
  * For each edge e={u, v} in G, there is a node x in V(T) such that u are v are in B(x)
  * For each vertex v in G, the set $\{x \in V(T) : v \in B(x)\}$ induces a connected graph
* The sets B(x) for node x in T are referred to as bags of the decomposition





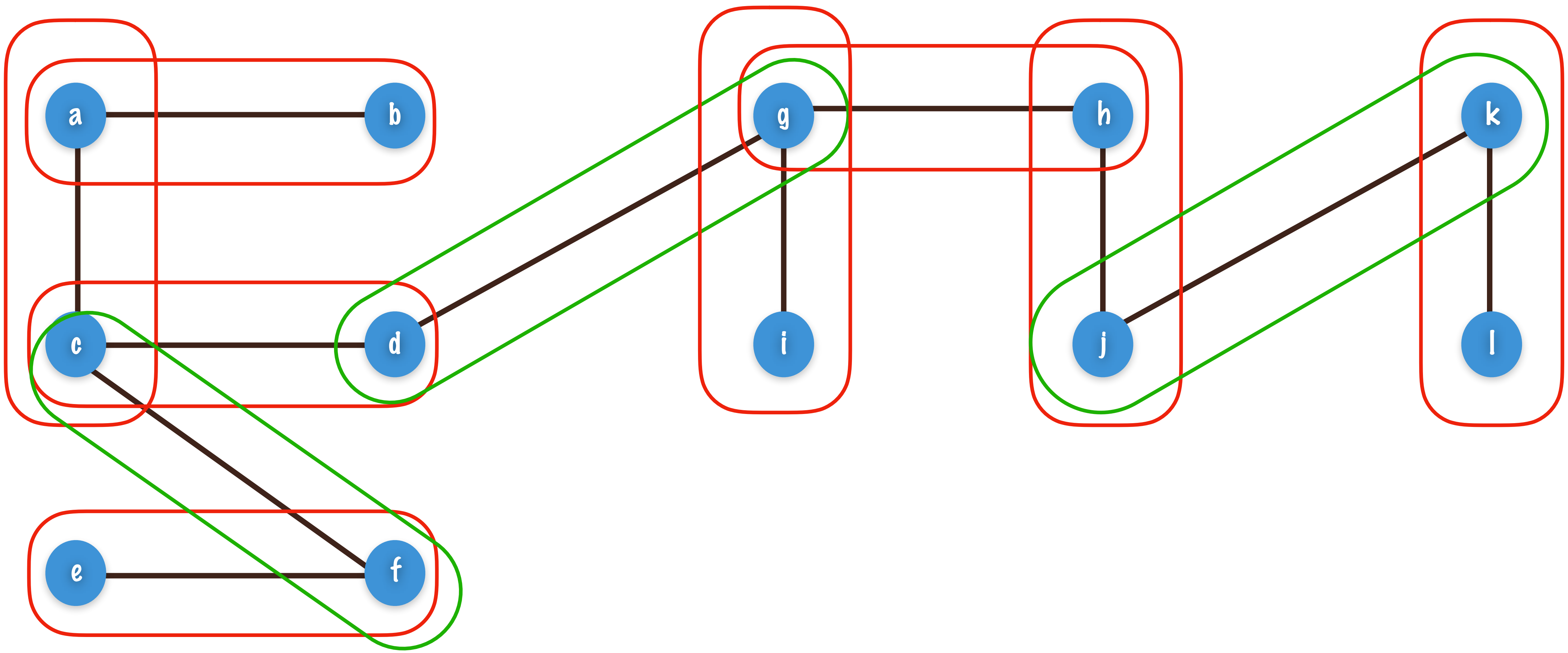
Treewidth



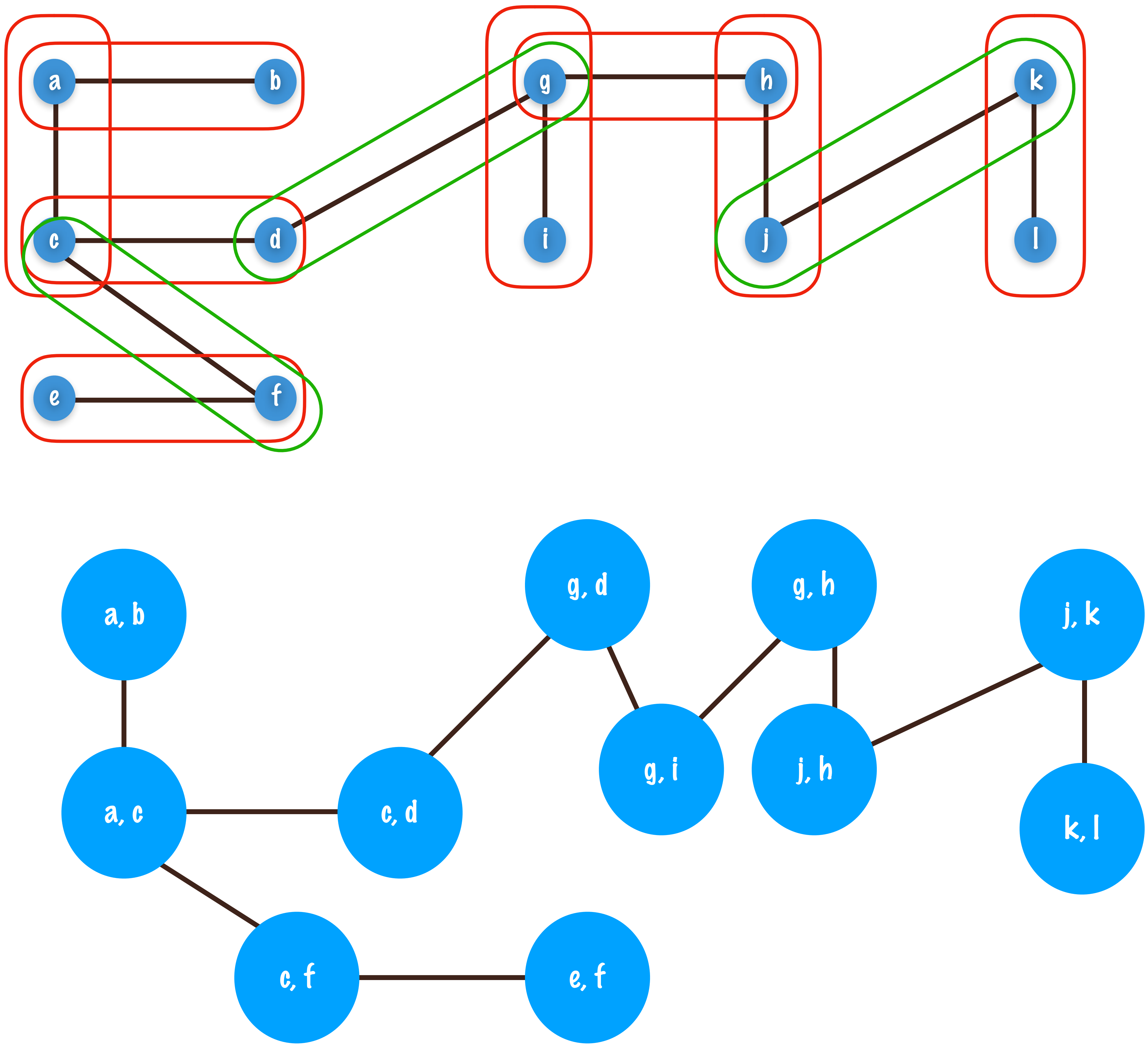


* Width of a tree decomposition T = w(T) = max $\{|B(x)| : x \in V(T)\} - 1$
* Treewidth of G, tw(G) = min {w(T) : T is a tree decomposition of G}
* An optimal tree decomposition of G is a tree decomposition of G of width tw(G).
* If G is a tree, then tw(G) <= 1

Treewidth of Trees - Intuition



Treewidth of Trees - Intuition



Computing Treewidth

* Width of a tree decomp T = $w(T) = \max\{|B(x)| : x \in V(T)\} - 1$
* Treewidth of G, $tw(G) = \min\{w(T) : T$ is a tree decomp of $G\}$
* An optimal tree decomp of G is a tree decomp of G of width tw(G)
* Computing tw(G) is NP-hard in general
* A brute-force algorithm
  * Given G, enumerate all pairs (T, B) s.t T is a tree and $B: V(T) \rightarrow 2^{V(G)}$
    * Check if (T, B) is a tree decomposition of G
  * Output tree decomposition (T, B) that has minimum width

How many nodes are there in T?

Computing Treewidth

Definition: A simple tree decomposition (T, B) is one where there is no pair of distinct nodes x and y in T such that $B(x) \subseteq B(y)$

Lemma: Any simple tree decomposition (T, B) of G satisfies $|V(T)| <= |V(G)|$.

Theorem: For any G, there is an opt tree decomposition that is simple.

Algorithm to compute tw and opt tree decomp

* Given G, enumerate all pairs (T, B)
    * T is a tree on at most $|V(G)|$ nodes ($<= n(n^2$ choose $n-1)$ choices)
    * B: $V(T) \to 2^{V(G)}$ ($<= (2^n)^n$ choices)
    * Check if (T, B) is a tree decomposition of G (polynomial time)
* Output tree decomposition (T, B) that has minimum width

$2^{O(n^2)}$ time algorithm

Simple Tree Decomposition

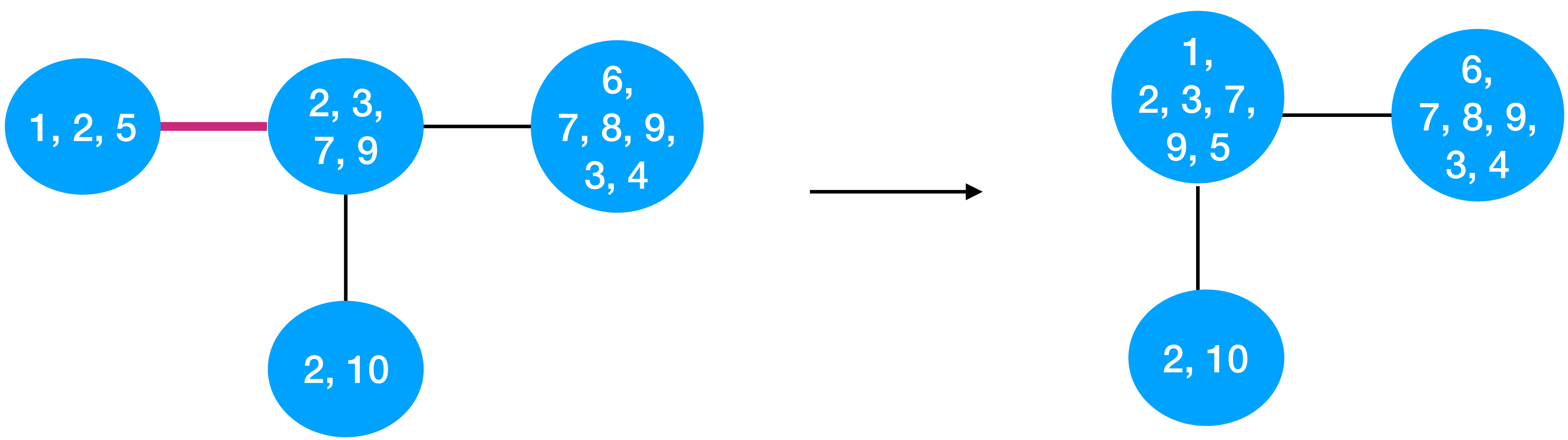
Lemma: Any simple tree decomposition (T, B) of G satisfies |V(T)| <= |V(G)|.

* Root T at r
* For each vertex v in G, let C(v) denote the node in T that is closest to r
* **Claim:** For each node x in T, there is a vertex v in G such that C(v) = x.
  * Suppose not.
  * Let x be a node in T for which there is not vertex v in G with C(v) = x
  * If x = r then B(r) = $\varnothing$ implying that (T, B) is not simple
  * If x ≠ r then let y be the parent of x
    * Consider v $\in$ B(x),
      * There is a node z(v) in T with C(v) = z(v)
      * z(v) is closer to r than x and v $\in$ B(z(v))
      * Then, v $\in$ B(y)
    * Thus, B(x) $\subseteq$ B(y) implying that (T, B) is not simple

Computing Simple Optimal Tree Decompositions

Lemma: There is a polynomial time algorithm that given a tree decomposition (T, B) of G, outputs a simple tree decomposition (T′, B′) such that for every node x′ in T′, there is a node x in T with B(x) = B′(x′).

- Contracting an edge of a tree decomposition



- Results in another tree decomposition

Computing Simple Optimal Tree Decompositions

Lemma: There is a polynomial time algorithm that given a tree decomposition (T, B) of G, outputs a simple tree decomposition (T′, B′) such that for every node x′ in T′, there is a node x in T with B(x) = B′(x′).

* Initialize (T′, B′) = (T, B)
* As long as there is an edge {x, y} in T′ with $B'(x) \subseteq B'(y)$
  * Contract {x, y}
* Output (T′, B′)

Claim: (T′, B′) is simple

* Suppose not. Let x and y be distinct nodes in T′ such that $B'(x) \subseteq B'(y)$
* Let P denote the path from x to y in T′ and let x′ be the vertex succeeding x in P
* As $B'(x) \subseteq B'(y)$, by the property of tree decompositions, for each vertex v in G with $v \in B'(x)$, we have $v \in B'(x')$
* {x, x′} is an edge in T′ with $B'(x) \subseteq B'(y)$ (Algorithm would have contracted {x, x′})

Computing Treewidth

Proposition 14.21 (Seymour and Thomas (1994); Bodlaender (1996); Feige et al. (2008); Fomin et al. (2015a); Bodlaender et al. (2016a)). *Let G be an n-vertex graph and k be a positive integer. Then, the following algorithms to compute treewidth exist.*

- *There exists an algorithm running in time $\mathcal{O}(1.7347^n)$ to compute* $\mathrm{tw}(G)$.
- *There exists an algorithm with running time $2^{\mathcal{O}(k^3)}n$ to decide whether an input graph G has treewidth at most k.*
- *There exists an algorithm with running time $2^{\mathcal{O}(k)}n$ that either decides that the input graph G does not have treewidth at most k, or concludes that it has treewidth at most $5k$.*
- *There exists a polynomial time approximation algorithm with ratio $\mathcal{O}(\sqrt{\log \mathrm{tw}(G)})$ for treewidth.*
- *If G is a planar graph then there is a polynomial time approximation algorithm with ratio $\frac{3}{2}$ for treewidth. Furthermore, if G belongs to a family of graphs that exclude a fixed graph H as a minor, then there is a constant factor approximation for treewidth.*

We remark that all the algorithms in this proposition also compute (in the same running time) a tree decomposition of the appropriate width. For example, the third algorithm either decides that the input graph G does not have treewidth at most k or computes a tree decomposition of width at most $5k$.

Image Source: Kernelization by Fomin et al., Cambridge University Press, 2019