Tree Width
(PACE 2016 & 2017)

1805006 - Tanjeem Azwad Zaman
1805008 - Abdur Rafi
1805010 - Anwarul Bashir Shuaib
1805019 - MD Rownok Zahan Ratul
1805030 - Md Toki Tahmid

Basic Concepts and Problem Definition

1805006 - Tanjeem Azwad Zaman

Topics To Cover

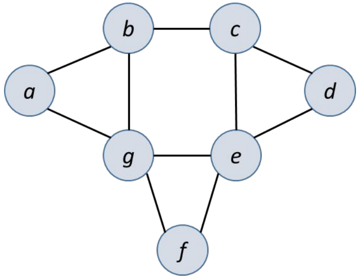
- General terminology
- Tree Decompositions
- Nice Tree Decompositions
- Tree Width
- Our Problem Definition
    - Optimization version
    - Decision Version

Topics To Cover

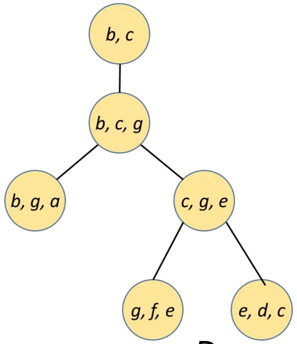
- General terminology
- Tree Decompositions
- Nice Tree Decompositions
- Tree Width
- Our Problem Definition
  - Optimization version
  - Decision Version

General Terminology / Definitions

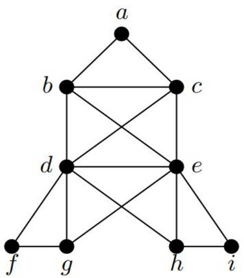
- Given a graph ***G***, with vertex set ***V(G)*** and edge set ***E(G)***
- In our context, Decomposition ***T*** is
  - Another graph-like ***<u>Structural Representation of G</u>***, where
  - Each node in T corresponds to ***<u>a subset of V(G)</u>*** **-> known as "Bag"s**
- A valid decomposition must have some other properties



G_1



D_1



G_2

D_2

Topics To Cover

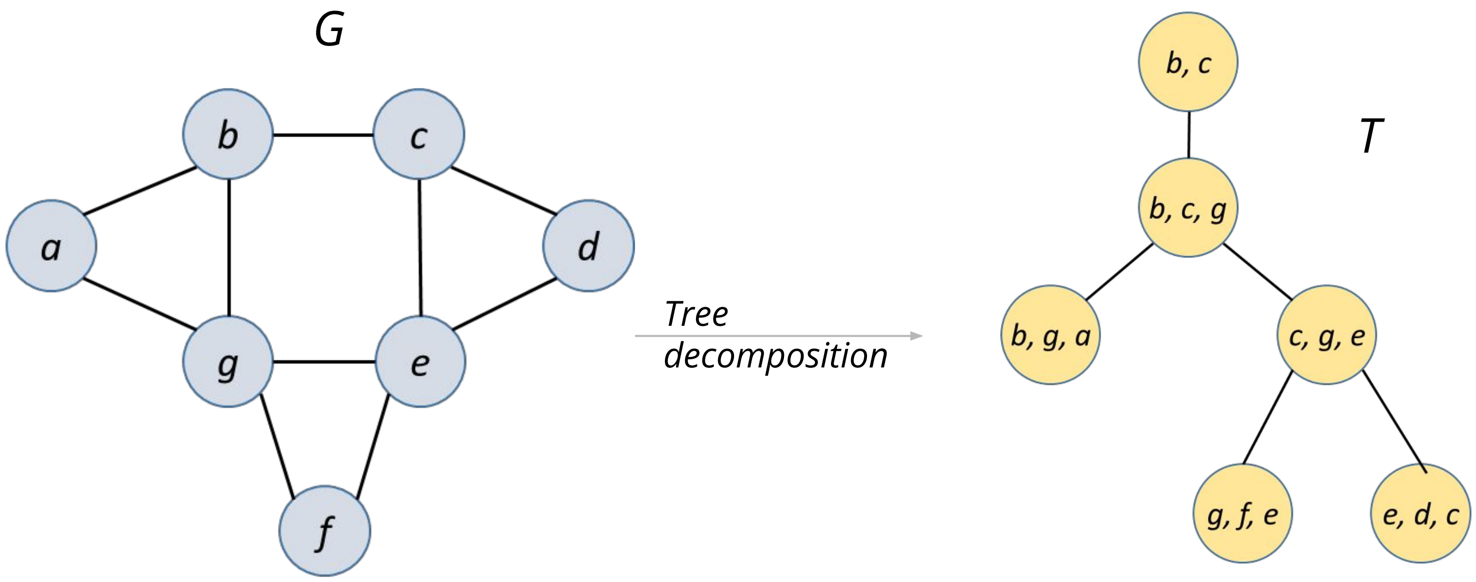
- General terminology
- Tree Decompositions
- Nice Tree Decompositions
- Tree Width
- Our Problem Definition
  - Optimization version
  - Decision Version

Tree Decomposition

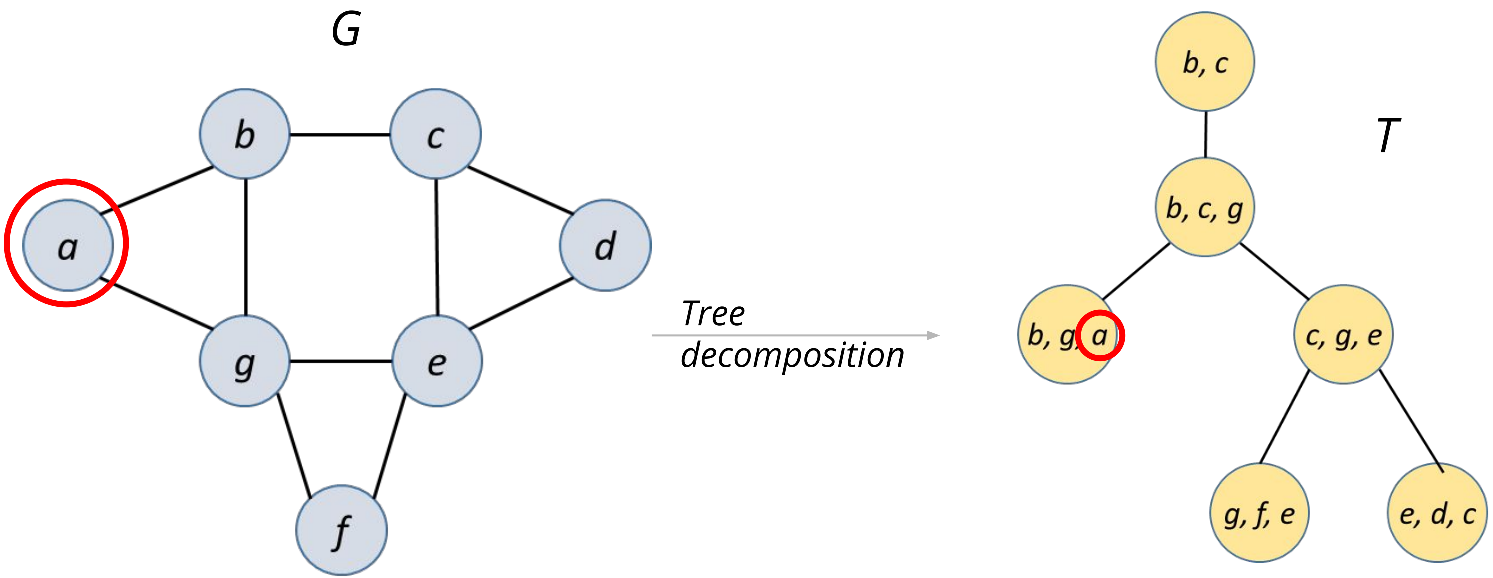
A tree decomposition is represented as: $\mathscr{T} = (\ T, \{\boldsymbol{X_t}\}_{t \epsilon V(t)}),$ where

Formal	*Simplified*
T is a tree	*T is a tree*
$\forall$ *t in V(T),* $\boldsymbol{X_t} \in \boldsymbol{V(G)}$	*each bag (corresponding to a tree node) is a subset of V(G)*
And the following 3 properties hold:	
1. $\bigcup_{t \in V(\boldsymbol{T})} \boldsymbol{X_t} = \boldsymbol{V(G)}$	*Every vertex of G is in at least 1 bag of T*
2. $\forall$ (u,v) $\in$ ***E(G),*** *there exists a node t in* ***T,*** *s.t both u and v belong to* $\boldsymbol{X_t}$	*For all edges in E(G), there is at least 1 bag in T that has both endpoints of the edge*
3. $\forall$ u $\in$ ***V(G)***, the set $\boldsymbol{T_u} = \{t \in \boldsymbol{V(T)} : u \in \boldsymbol{X_t}\}$	*All bags that contain any specific vertex of G, make a connected subtree in T*

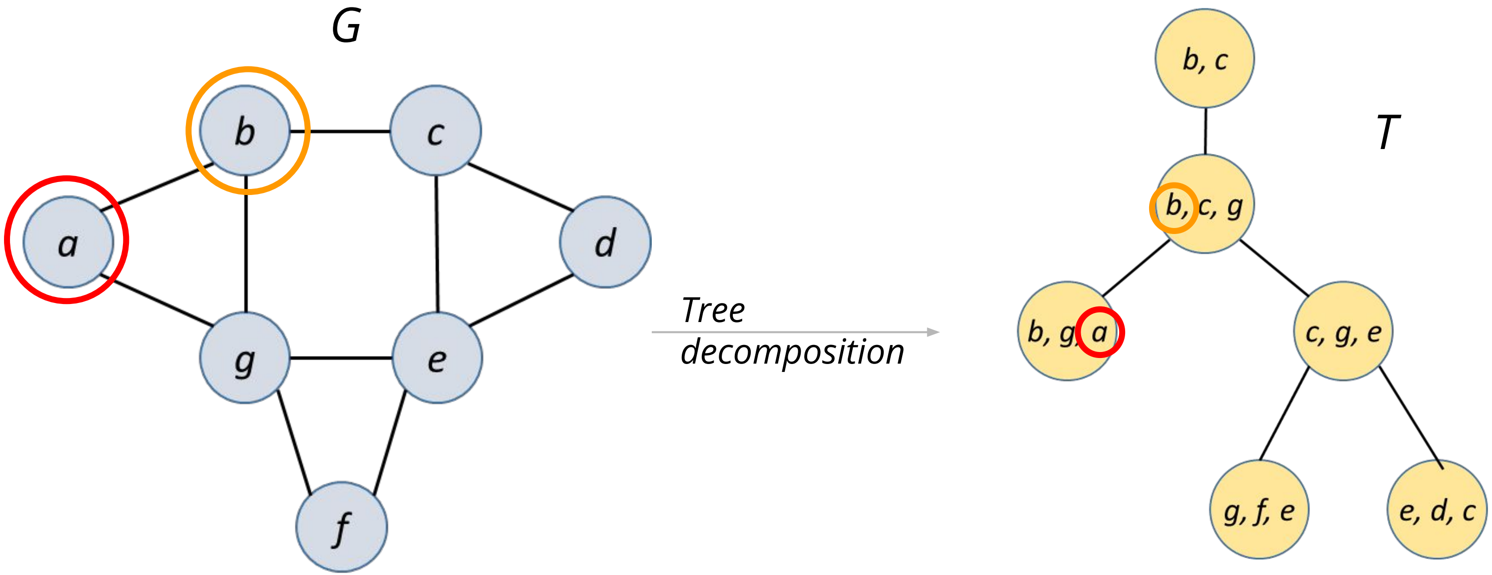
Example



Example
1. All nodes belong to at least 1 bag

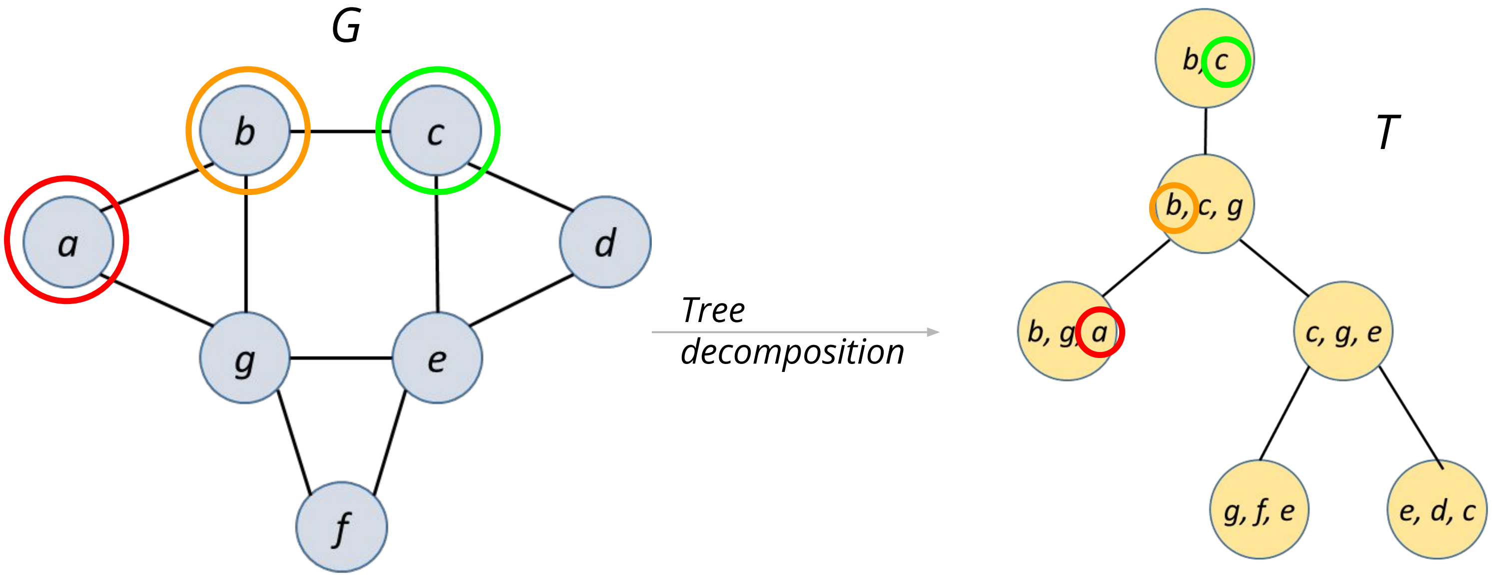


Example 1. All nodes belong to at least 1 bag

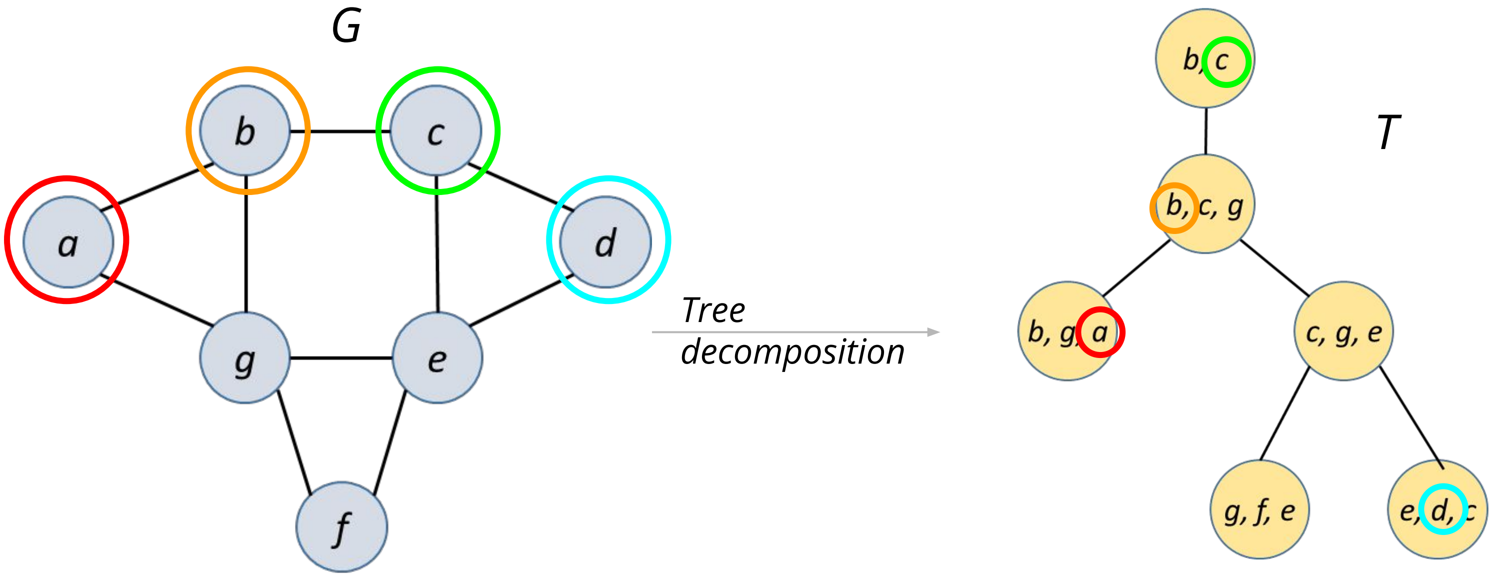


Example

1. All nodes belong to at least 1 bag

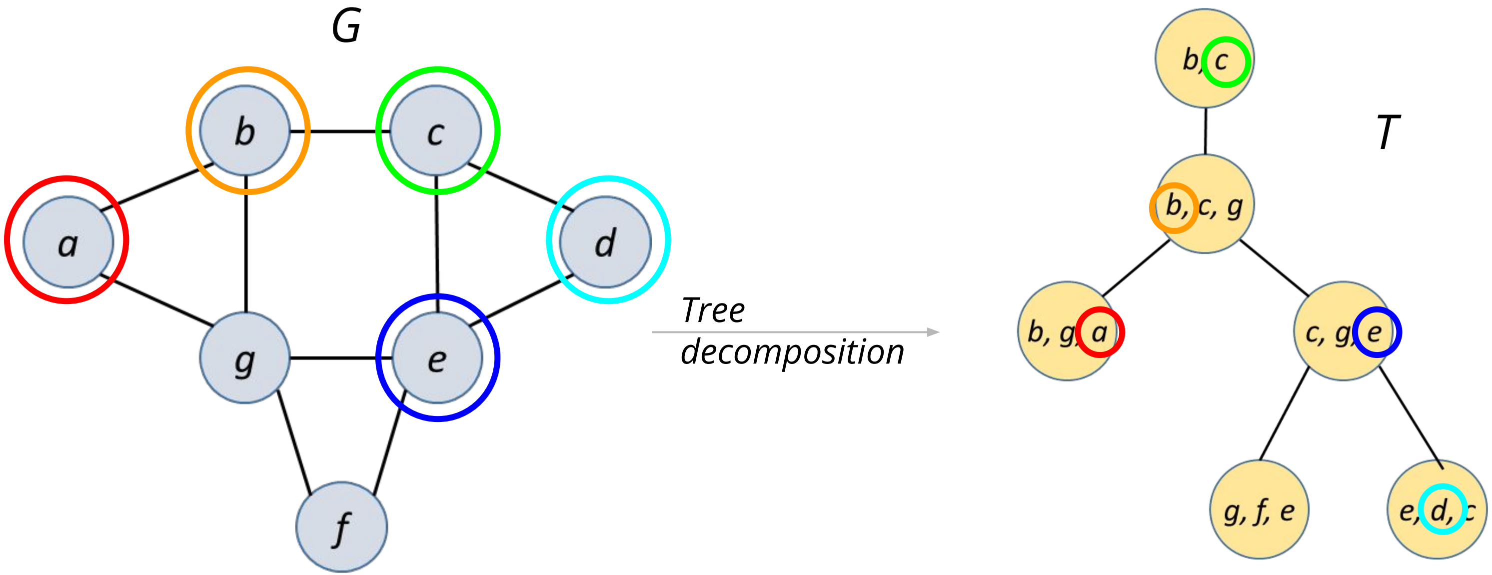


Example 1. All nodes belong to at least 1 bag

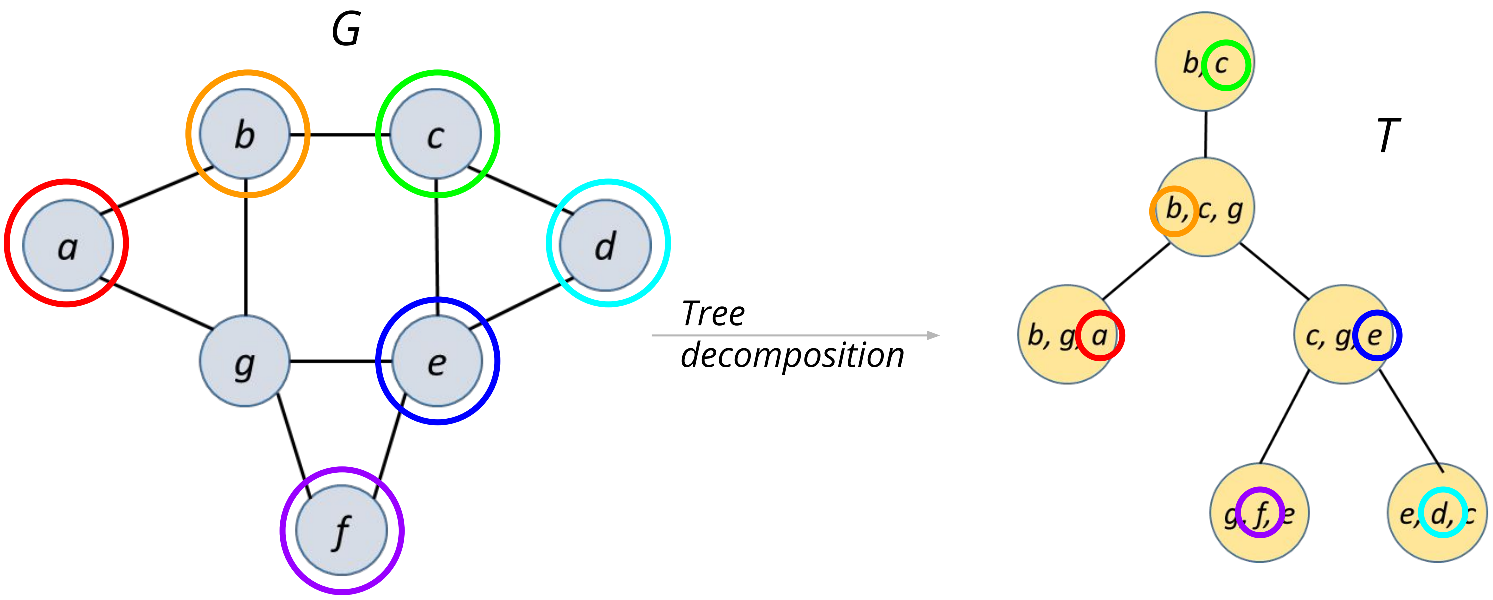


Example

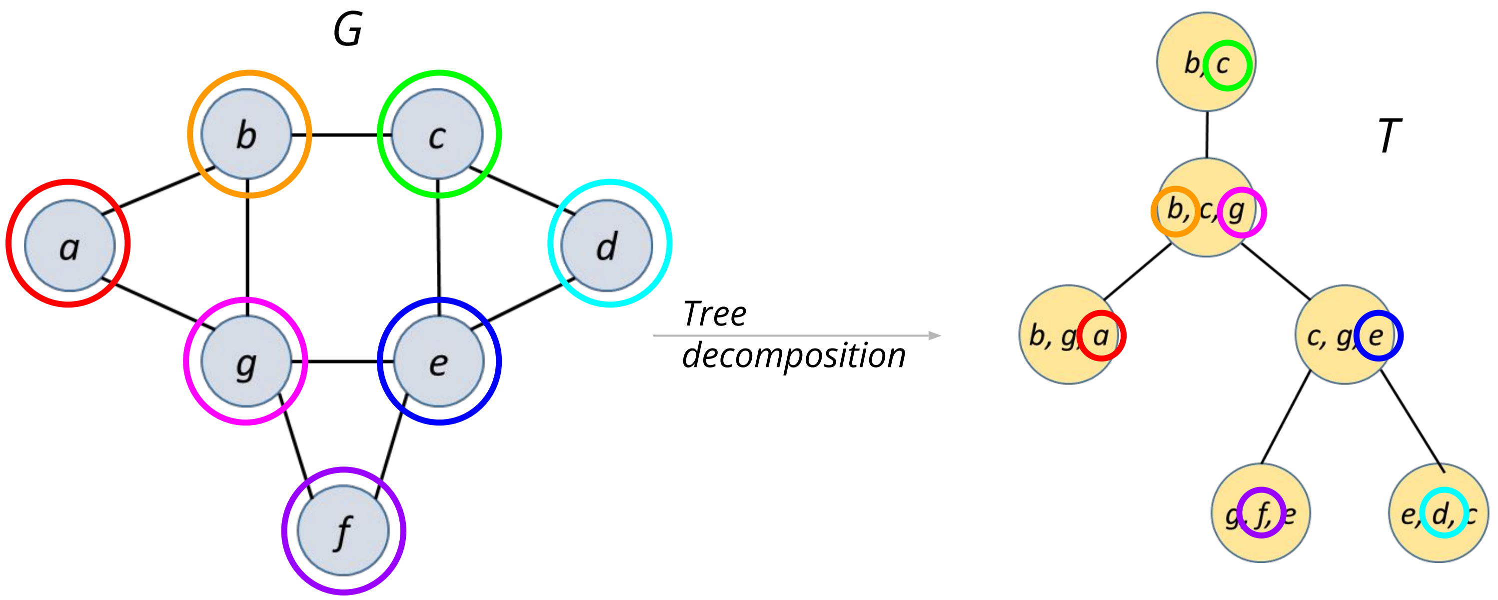
1. All nodes belong to at least 1 bag



Example 1. All nodes belong to at least 1 bag

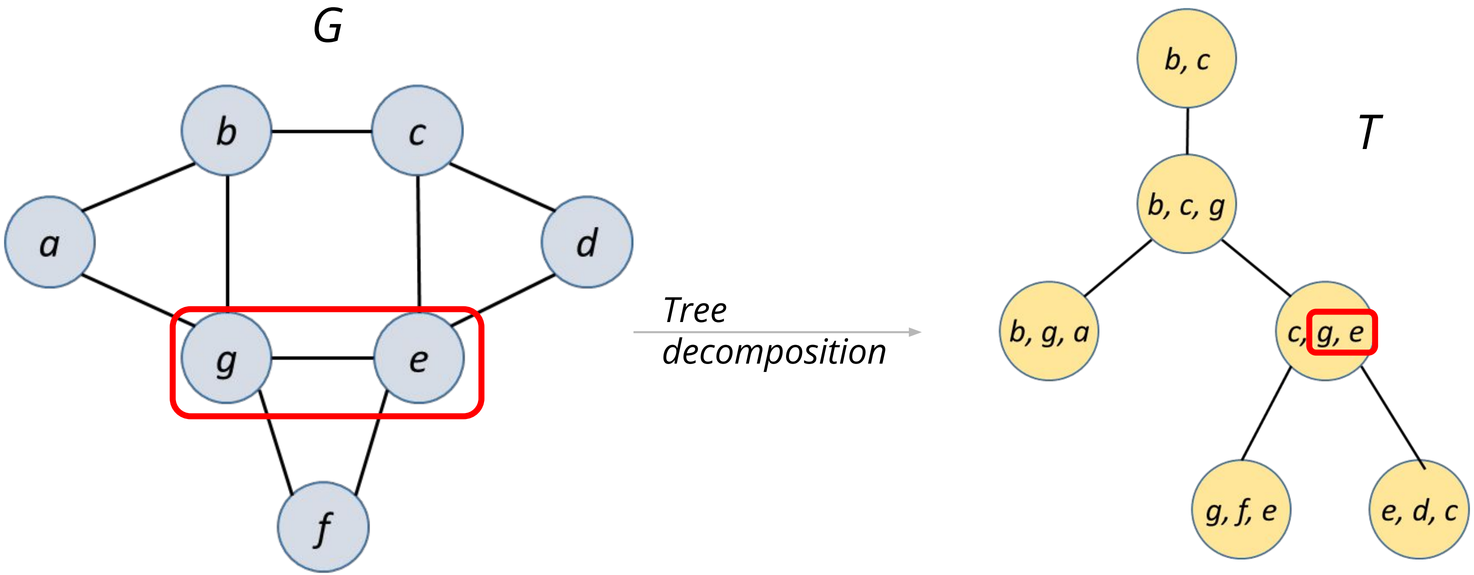


Example 1. All nodes belong to at least 1 bag

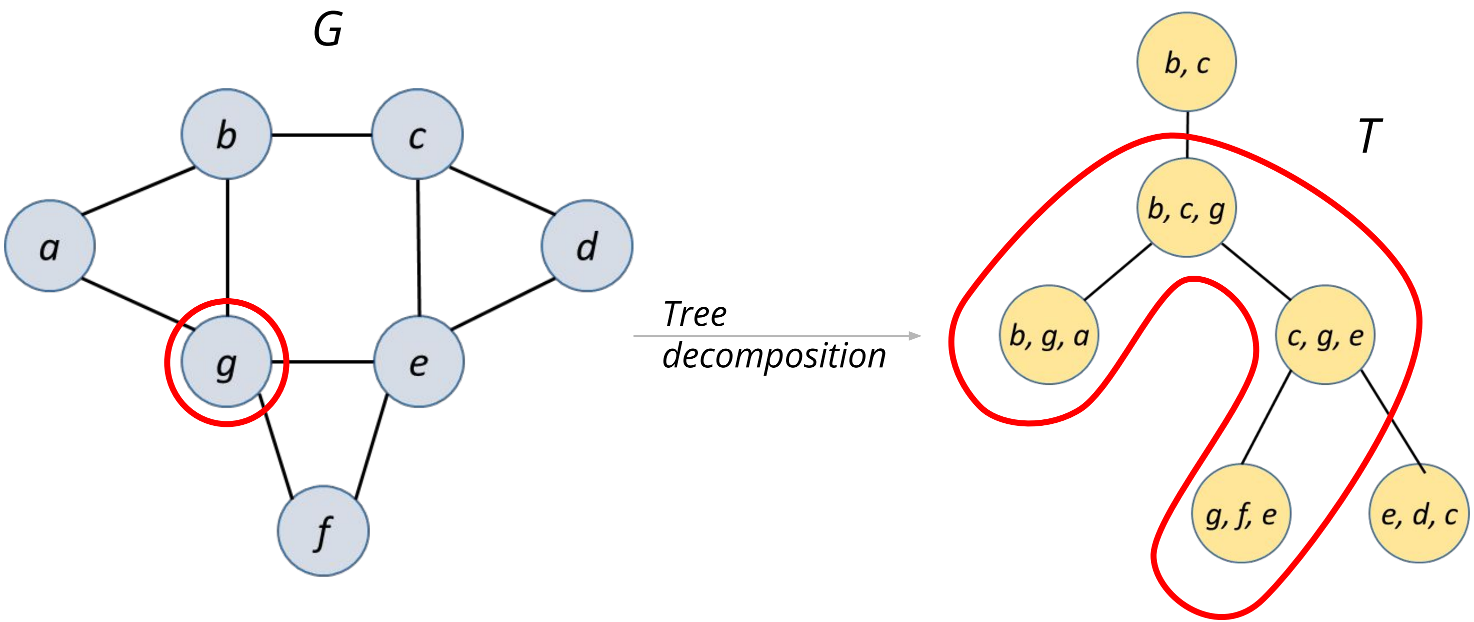


Example

2. For all edges, at least 1 bag has both endpoints. Eg: (g,e)



Example 3. All bags with a specific vertex will form a connected subtree



Topics To Cover

- General terminology
- Tree Decompositions
- Nice Tree Decompositions
- Tree Width
- Our Problem Definition
  - Optimization version
  - Decision Version

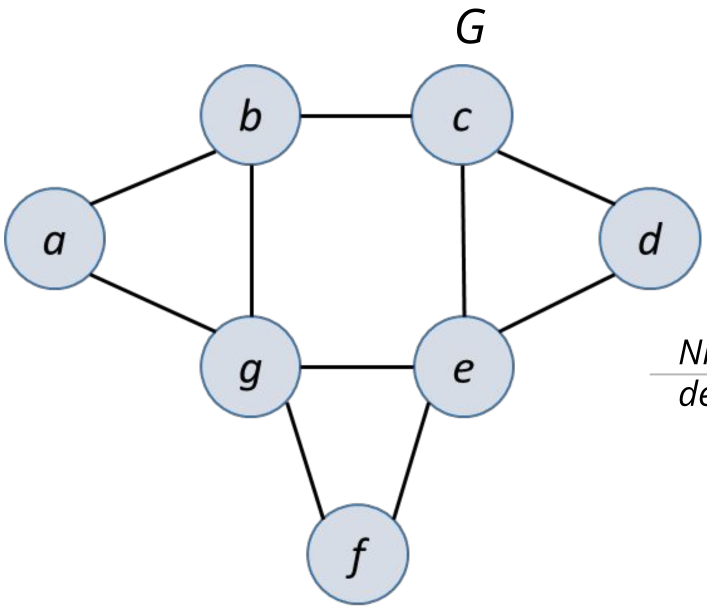
"Nice Tree" Decomposition

A tree decomposition where

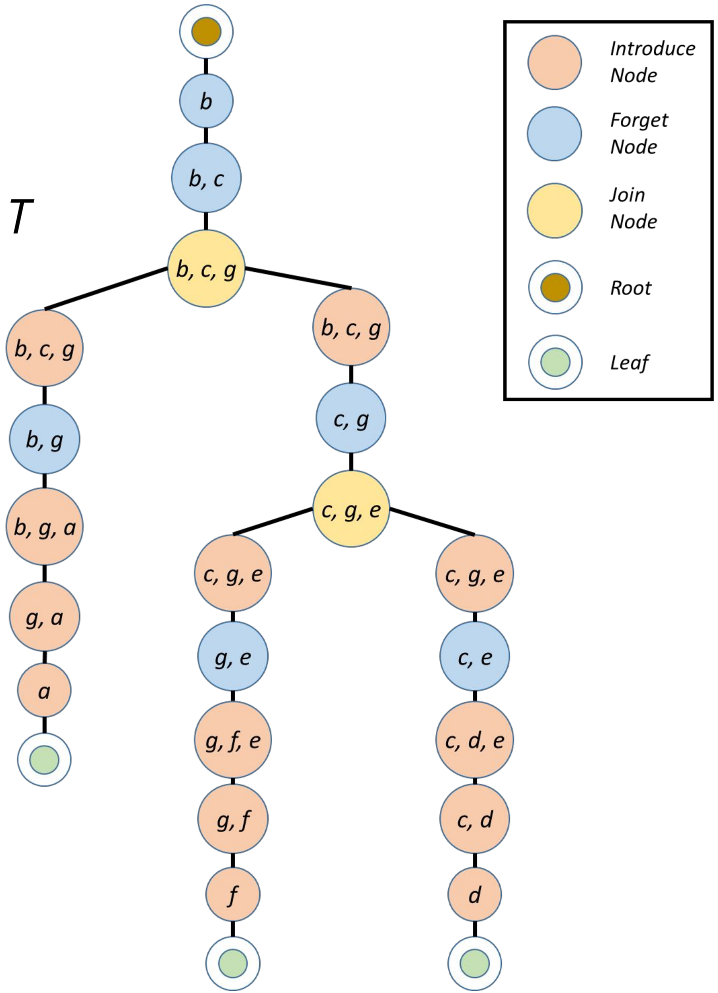
- The root and leaf bags are empty. $\boldsymbol{X_{root}} = \emptyset$, $\boldsymbol{X_{leaf}} = \emptyset$
- Each ***non-leaf node***, t is one of three types:

1. Introduce node:	
has 1 child t′, where $\boldsymbol{X_t} = \boldsymbol{X_{t'}} \cup \{v\}$ for $\boldsymbol{v}$ not in $\boldsymbol{X_{t'}}$	*A node with one child, and has an extra vertex not included in its child*
2. Forget Node:	
1 child t′, where $\boldsymbol{X_t} = \boldsymbol{X_{t'}} \setminus \{v\}$ for a $\boldsymbol{v}$ in $\boldsymbol{X_{t'}}$	A node with one child and a vertex less than its child
3. Join Node:	
2 children t_1, t_2 such that $\boldsymbol{X_t} = \boldsymbol{X_{t1}} = \boldsymbol{X_{t2}}$	A node with two childs, both identical to itself

Example:



Nice Tree decomposition



Why "Nice Tree" Decomposition?

Lemma: Given a graph G and its tree decomposition $\mathscr{T} = (\ T, \{\boldsymbol{X_t}\}_{t \epsilon V(t)})$, one can compute a nice tree decomposition in

- ***Time*** : O(k^2 ▪max(|V(T)|, V|(G)|)
- ***Width***: at most k
- ***# of nodes***: at most O(k|V(G)|)

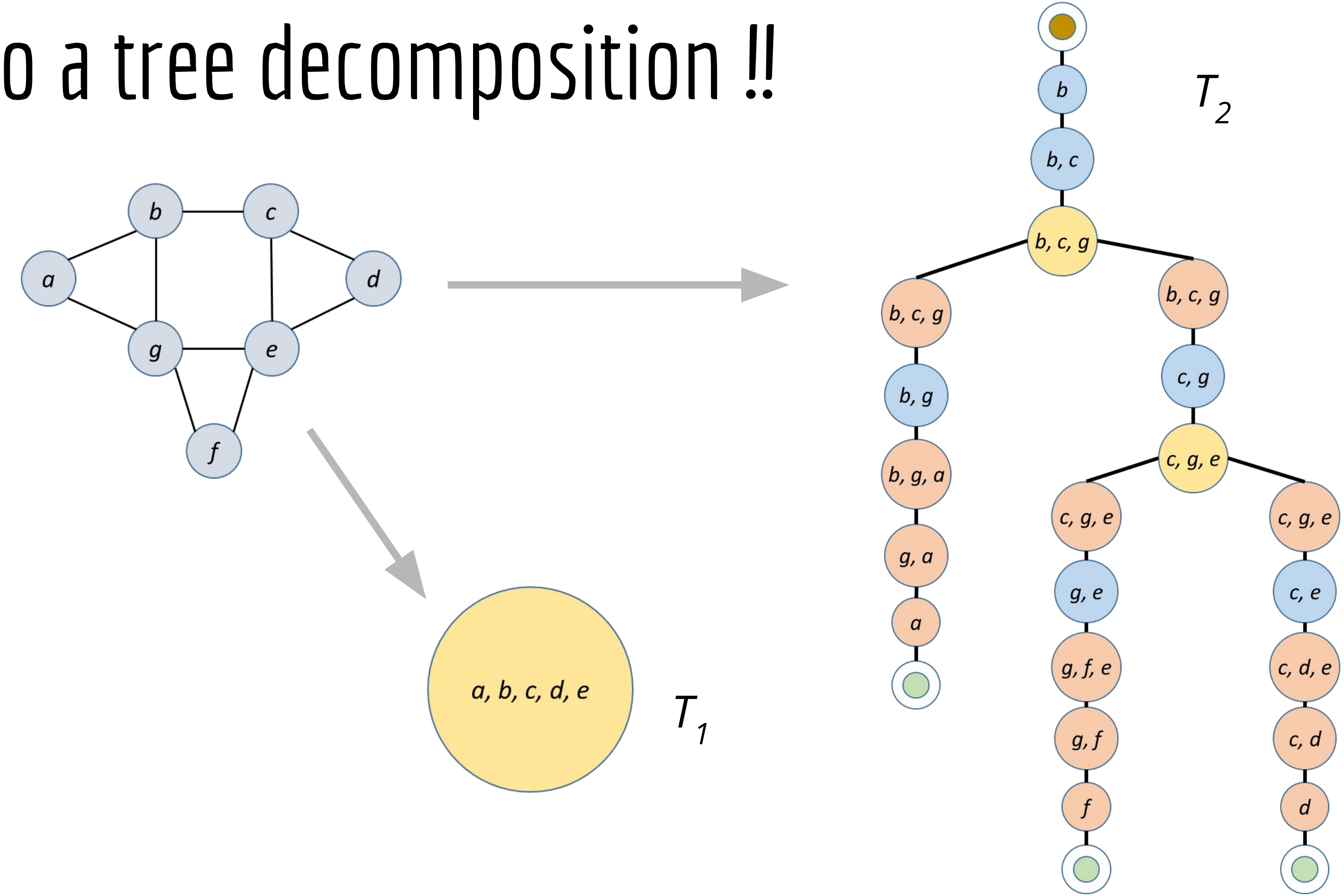
Thus, nice tree decompositions have the following pros, among many more:

1. ***Conducive to DP:*** Problems on graphs can be broken down into smaller subproblems corresponding to nodes of the decomposition.

2. ***Real-World Applications:*** Discussed later

3. ***Standard Form:*** Easier to work with when designing algorithms

Also a tree decomposition !!



Topics To Cover

- General terminology
- Tree Decompositions
- Nice Tree Decompositions
- Tree Width
- Our Problem Definition
  - Optimization version
  - Decision Version

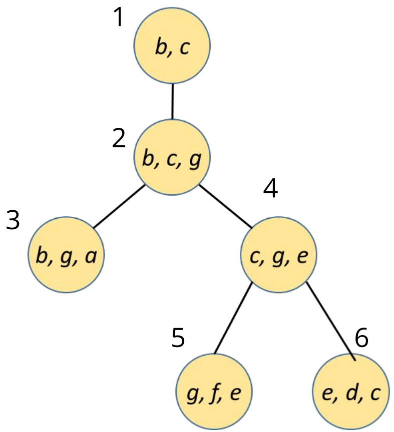
Tree Width

Width of a <u>bag</u>: (Size of the bag) - 1

Width of a <u>tree</u>: Maximum of the widths of its bags

Tree-Width of a <u>graph</u>: <u>*Minimum width*</u> among all tree decompositions of the graph

<u>*Example:*</u>



Bags:

$\boldsymbol{X_1}$ = {b,c}, $\boldsymbol{X_2}$ = {b,c, g}, $\boldsymbol{X_3}$ = {b, g, a}, $\boldsymbol{X_4}$ = {c, g, e}, $\boldsymbol{X_5}$ = {g, f, e}, $\boldsymbol{X_6}$ = {e,d,c},

Bag widths:

size of $|\boldsymbol{X_1}| = 2$, so width of $X_1 = 1$.

Similarly widths of $\boldsymbol{X_2}, \boldsymbol{X_3}, \boldsymbol{X_4}, \boldsymbol{X_5}, \boldsymbol{X_6}$ are all 2

Width of the tree = max (1,2,2,2,2,2) = 2

Topics To Cover

- General terminology
- Tree Decompositions
- Nice Tree Decompositions
- Tree Width
- Our Problem Definition
  - Optimization version
  - Decision Version

Problem Definition

- ***Optimization version:***

  Given an arbitrary graph, find its *tree width*

  *(i.e. minimum width among all possible tree decompositions)

  ** in most practical cases, the decomposition itself that gives the tree width is needed.

- ***Decision Version:***

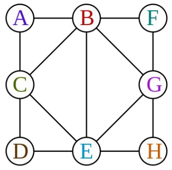
  Given an arbitrary graph and a positive integer k, is the tree-width of the graph ***at most k***?

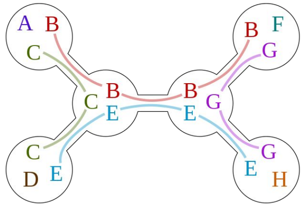
The Tree-Width Problem is NP-Complete

Reductions to/from Hard Problems

1805019

NP-Completeness of Computing Treewidth





- **<u>Decision Problem:</u>**

  Given, G(V,E), does G has a treewidth at most k?

- NP-Completeness proved in 1987
  - “Complexity of Finding Embeddings in a k-Tree”
  - Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski

Required Definitions

K-Chordal Graphs

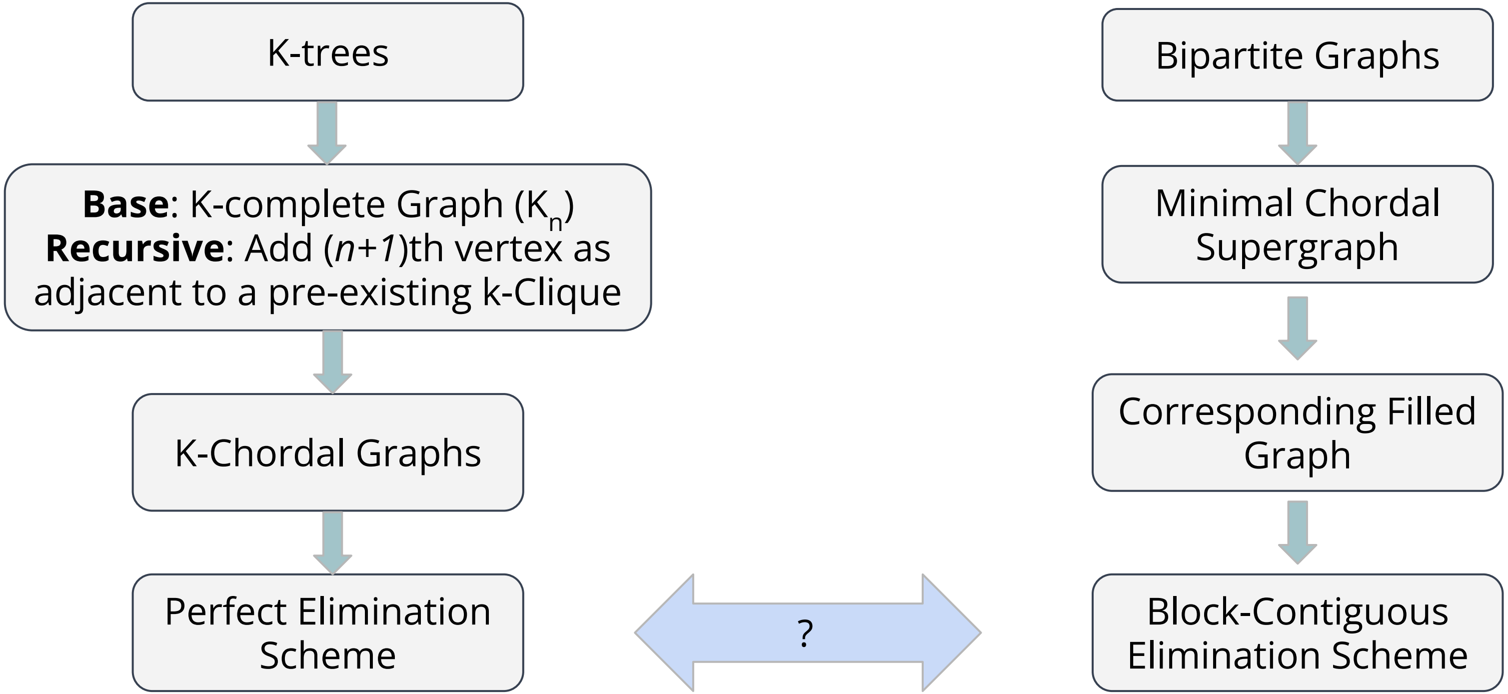
Block-Contiguous Elimination Scheme

Minimum Cut Linear Arrangement (MCLA)

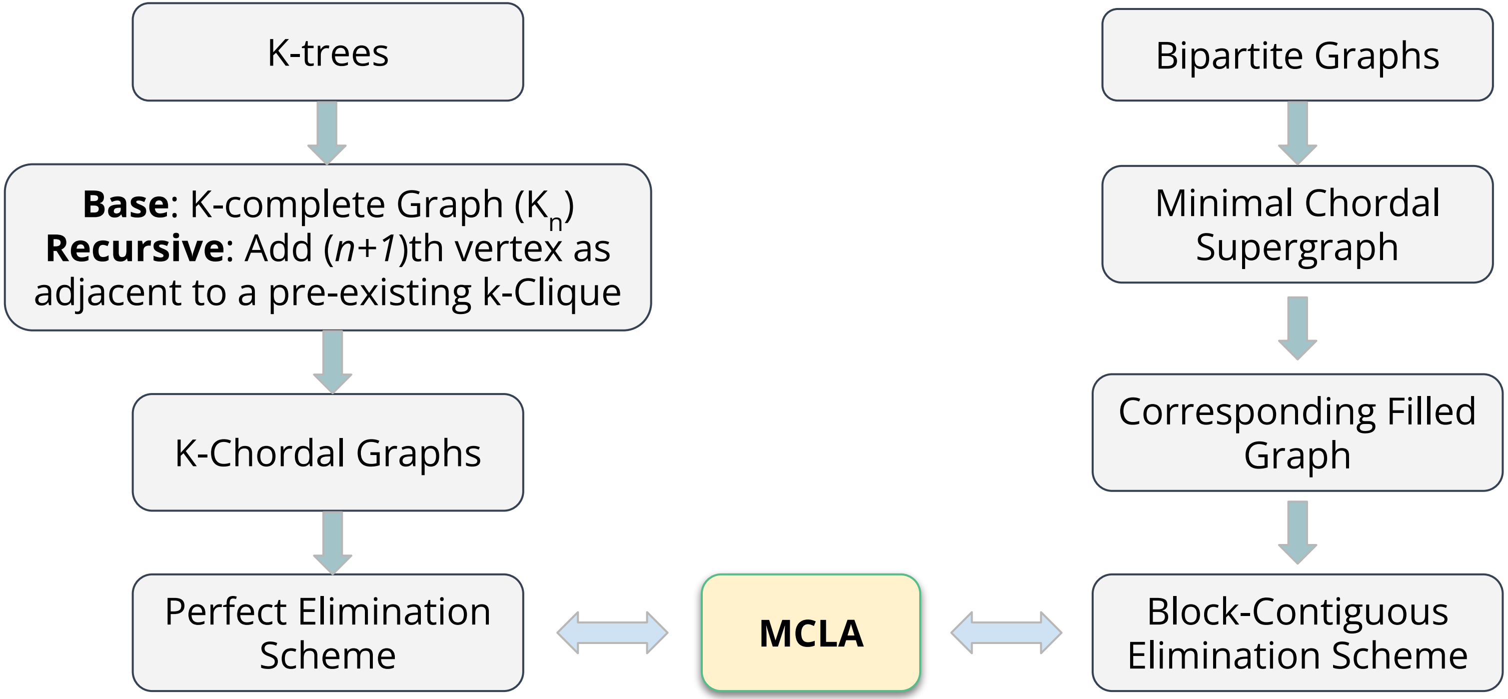
Reduction

$\text{MCLA} \leq_p \text{Treewidth}$

Intuition Behind the Proof



Intuition Behind the Proof



Formal Proof

Construction of Bipartite Graph

Input: G (V,E)

Output: G’ (A U B, E’)

Construction Rule:

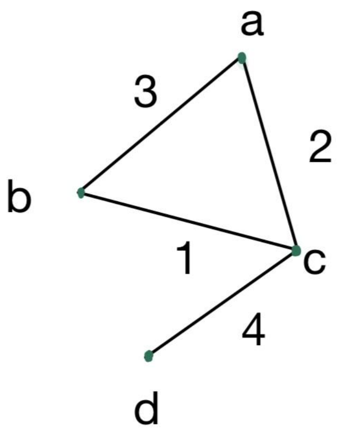
Defining Nodes:

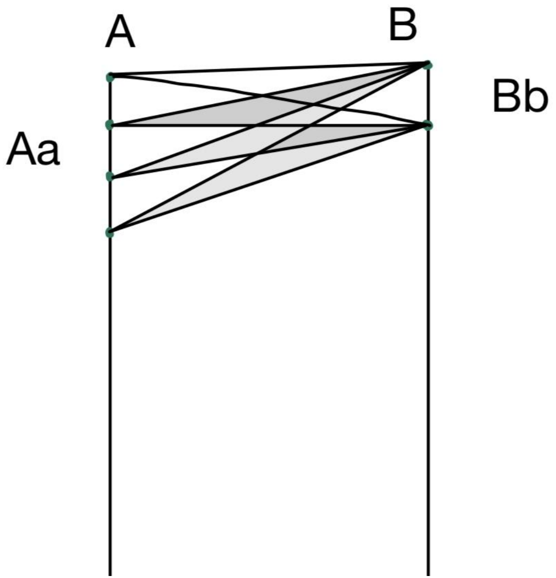
a. $\forall(x)$ ε V, add *Δ(G)+1* vertices in A as A_x and *Δ(G)+1-deg(x)* vertices in B as B_x
b. $\forall(e)$ ε E, add 2 vertices to B denoted by B_e

Defining Edges:

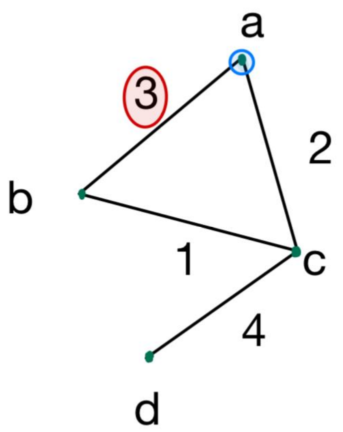
a. All vertices of A_x are adjacent to all vertices of B_x
b. All vertices of A_x are adjacent to all of B_e if *x* is incident to e

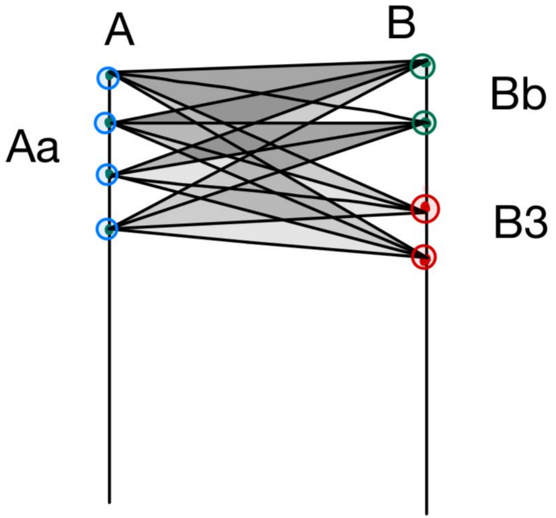
Example Bipartite Construction



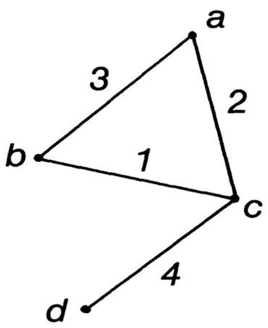


Example Bipartite Construction

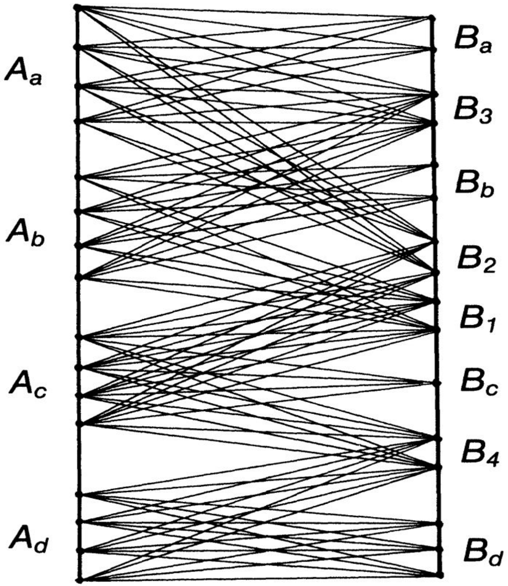




Example Bipartite Construction



G



C(G)

The Relation Between G and G’

Existing Algorithms & Experimental Results

Exponential Exact, Approximation & Randomized

118030

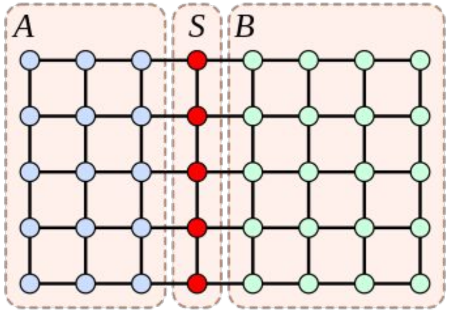
Exact Algorithms		
Positive-instance driven dynamic programming for treewidth ***Hisao Tamaki***	• Based on minimal separators and potential maximal cliques	• 2nd in PACE 2017: Exact Track
Jdrasil: A Modular Library for Computing Tree Decompositions ***Max Bannach, Sebastian Berndt, and Thorsten Ehlers***	• Supports parallel processing • Incorporate both heuristics and approximation algorithms too.	• 3rd in PACE 2017: Exact Track

Approximate Algorithms		
Finding all leftmost separators of size <= k ***Belbasi & Fürer (2021b)***	• Runs in $2^{6.755k} \cdot O(n \log n)$ • Approximation Ratio: 5K+1	• Focuses on improving the exponential value related to “K” by finding
An Improved Parameterized Algorithm for Treewidth (2022) ***Tuukka Korhonen, Daniel Lokshtanov***	• Runs in $2^{O(k^2)} n^{O(1)}$ • Approximation ratio: $(1 + \varepsilon)k$ [$\varepsilon \in (0, 1)$]	• First improvement on the dependency on k in algorithms for treewidth since the $2^{O(k^3)} n^{O(1)}$ time algorithm given by Bodlaender and Kloks [ICALP 1991]

Algorithm Selected For Implementation

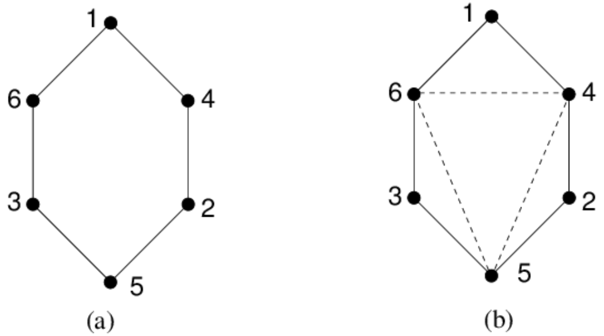
Positive-instance driven dynamic programming for treewidth (***Hisao Tamaki***)

Separator of Graph



A vertex set S ⊆V(G) is a separator of G if its removal increases the number of connected components of G

Potential Maximal Clique

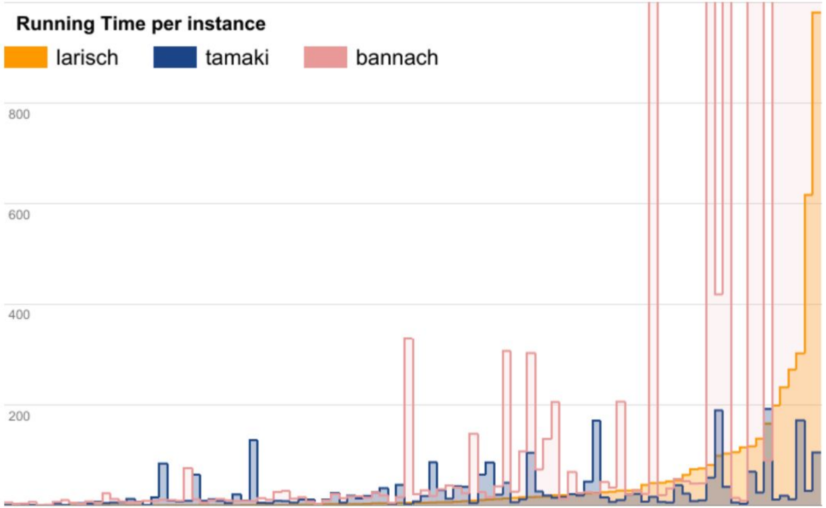


1. Find minimal triangulation G’ of graph G
2. Find a vertex set which induces a maximal clique in G’
3. This will be a potential maximal clique in G

If these objects can be listed in polynomial time for a class of graphs, the treewidth and the minimum fill-in are polynomially tractable for these graphs.

Positive-Instance Driven Algorithm: Driven by positive instances of dynamic programming, leading to efficient performance on benchmark instances.

Handling of Subproblems: Deals with subproblems through the novel use of auxiliary structures called O-blocks, leading to a binary recurrence that offers practical running time bounds



Existing Algorithms & Experimental Results

Heuristic & Meta-Heuristic

1805008

Flow-Cutter-2017

- 2nd place in pace 2017
- ??

Chordal Supergraph

- A chordal supergraph of G is a chordal graph G’ defined on the same set of vertex, where G is a subgraph of G’

Example image

Perfect Elimination Ordering

- An ordering of the vertex set of an undirected graph
- Neighbor of vertex v_i forms a clique in the graph induced by itself and the vertex appearing later
- Chordal graphs always have a perfect elimination ordering and can be determined in polynomial time??
- A tree decomposition of a graph can be constructed in polynomial time given a chordal supergraph and its perfect elimination ordering
- Example??

Relevance?

- Given an undirected graph and elimination order, we can construct the chordal supergraph and so, get the tree decomposition
- Commonly used algorithms try to guess the elimination order
- An approach to do so is called nested dissection
-

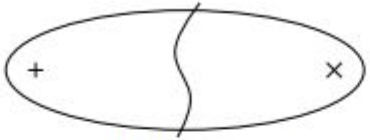
Guessing Elimination Order

- One approach is called nested dissection
- It consists of
  - Finding a small balanced separator
  - Placing these nodes at the end of elimination order
  - Removing the separator from the graph to get 2 sides
  - Run recursively on both sides
-

Core FlowCutter Algorithm

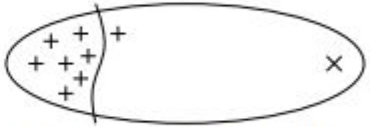
- A novel method to compute balanced graph cuts with minimum cut size
- Utilizes max flow min cut
- Considers unit flow
  - All edges have unit capacity
  - Flow through an edge can be either 0 or 1
- If the min cut is balanced then stop
- Otherwise suppose the source side have more nodes
- In this case we add new sources
- Among them, one is outside the source side, called the piercing node
- The piercing node is added to ensure that we get a new cut

Core FlowCutter Algorithm

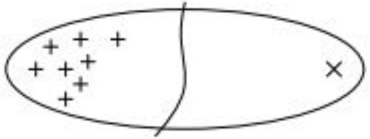


(a) Balanced cut C

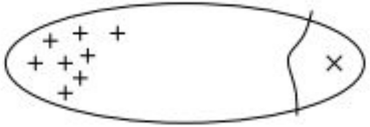
(b) Unbalanced cut C



(c) Extra sources to avoid C



(d) Source side cut C'



(e) Target side cut C'

Choosing Piercing Node

- 2 heuristics
  - Primary heuristic to select candidates of piercing nodes
  - Secondary heuristic to select among the candidates

Primary Heuristic

- Avoid augmenting path
  - If there is a non saturated path from the piercing node to any of the sink node, then making it a source will result in increase of net flow, thus increase in cut size
  - So we avoid such nodes to prevent increase in cut size
- If it is not possible, then choose any

Secondary Heuristic

- If there are multiple candidates available from primary heuristics, then consider 2 distance
- From piercing node to original source, ds
- From piercing node to original sink, dt
- Maximize dt - ds
- Why ??

Use in Calculating Treewidth

- We try to determine an elimination order of an undirected graph
- From an elimination order we can determine a chordal supergraph, from which we can determine tree decomposition in polynomial time
- The width of the decomposition depends on how minimum the chordal graph, found from the elimination order is.

Finding an elimination order

- One commonly used method is called nested dissection
- We first find a small balanced separator. This is where core FlowCutter algorithm is utilized
- We remove the separator from the graph
- The algorithm recursively continues on both sides.

Theoretical and Real-world Applications

1805010 - Anwarul Bashir Shuaib

Applications of Bounded Treewidth Graphs

- Many NP-hard problems can be solved in polynomial time for the class of bounded treewidth graphs. Some examples include:
    - Hamiltonian path
    - Network reliability
    - **Graph coloring**
    - **Independent Set problem**
- Some of the real-world applications include:
    - Identifying clusters in network analysis
    - Query optimization in database systems
    - Constraint Satisfaction Problems (CSP)
    - Dependencies and resource allocation in project planning
    - **Game theory**

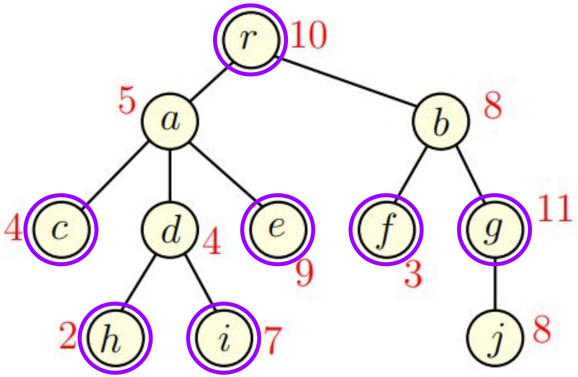
Maximum-Weighted Independent Set

- NP-hard for general graphs
- For trees, this can be done in O(n) time
- Dynamic programming -
  - Take MIS including v

$$W^+[v] = w(v) + \sum_{u \in C_v} W^-[u]$$

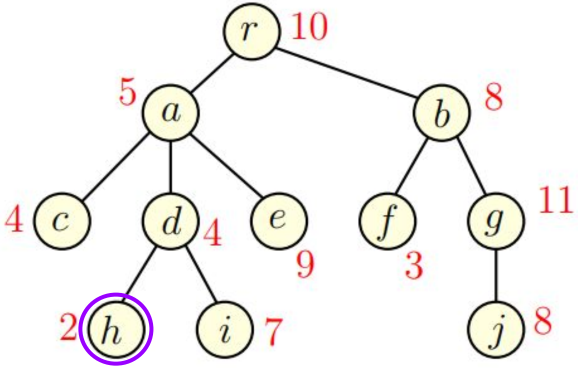
  - Take MIS excluding v

$$W^-[v] = \sum_{u \in C_v} max\{W^-[u], W^+[u]\}$$



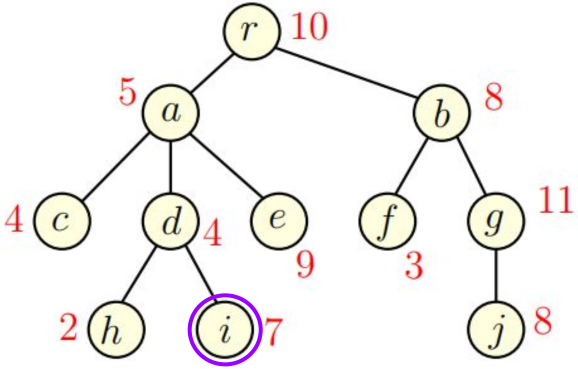
h	i	d	c	e	a	j	g	f	b	r
2	7	4 9	4	9	14 22	8	11 8	3	16 14	46 38

Maximum-Weighted Independent Set



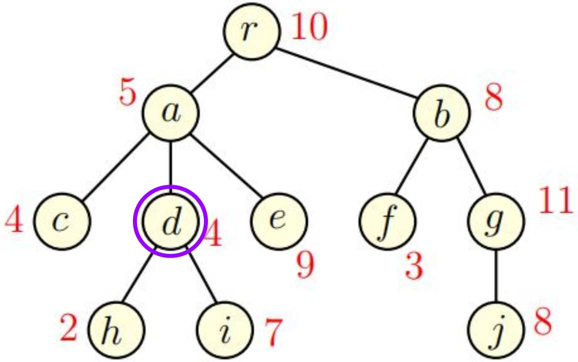
h	i	d	c	e	a	j	g	f	b	r
2										

Maximum-Weighted Independent Set



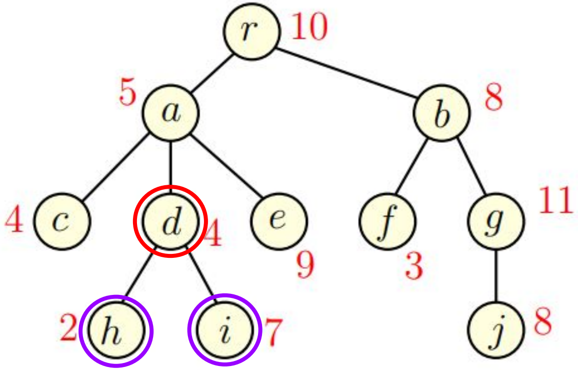
h	i	d	c	e	a	j	g	f	b	r
2	7									

Maximum-Weighted Independent Set



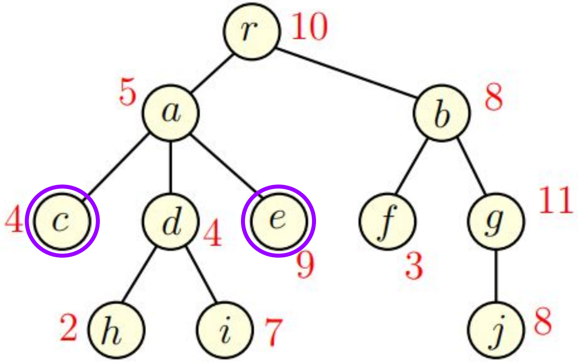
h	i	d	c	e	a	j	g	f	b	r
2	7	4								

Maximum-Weighted Independent Set



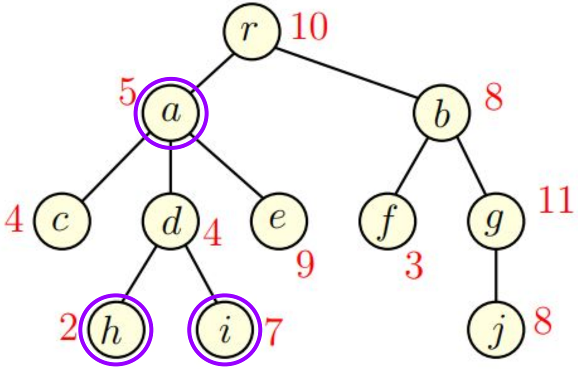
	h	i	d	c	e	a	j	g	f	b	r
Taken →	2	7	4								
Not taken →			9								

Maximum-Weighted Independent Set



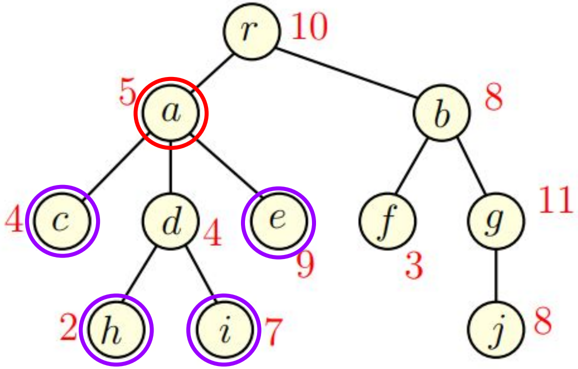
h	i	d	c	e	a	j	g	f	b	r
2	7	4 9	4	9						

Maximum-Weighted Independent Set



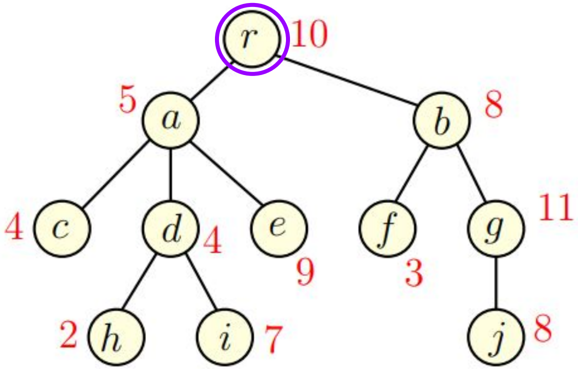
h	i	d	c	e	a	j	g	f	b	r
2	7	4 9	4	9	14					

Maximum-Weighted Independent Set



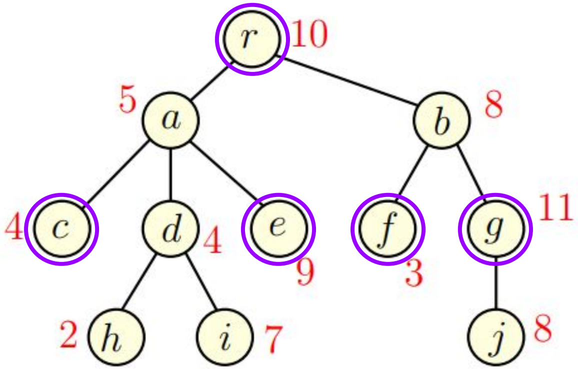
h	i	d	c	e	a	j	g	f	b	r
2	7	4 **9**	4	**9**	14 22					

Maximum-Weighted Independent Set



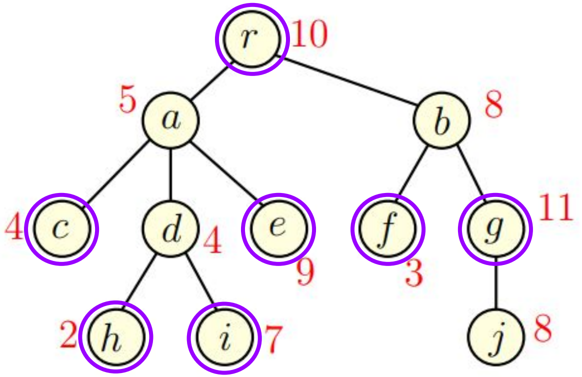
h	i	d	c	e	a	j	g	f	b	r
2	7	4 9	4	9	14 22	8	11 8	3	16 14	**46** 38

Maximum-Weighted Independent Set



h	i	d	c	e	a	j	g	f	b	r
2	7	4 **9**	**4**	**9**	14 22	8	**11** 8	**3**	16 14	**46** 38

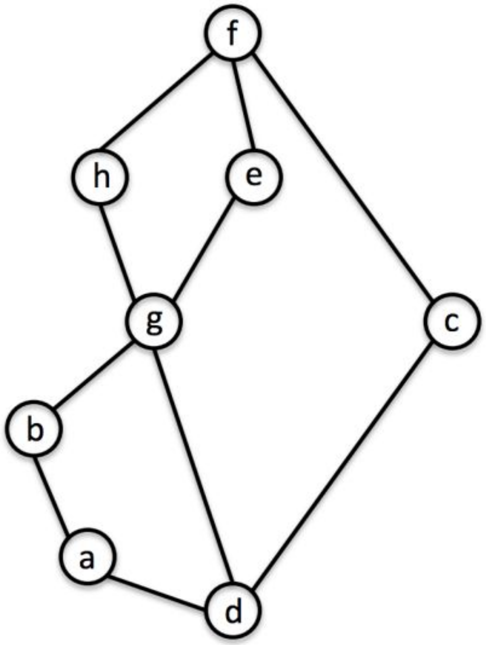
Maximum-Weighted Independent Set



h	i	d	c	e	a	j	g	f	b	r
2	**7**	4 **9**	**4**	**9**	14 22	8	**11** 8	**3**	16 14	**46** 38

Cops and Robber

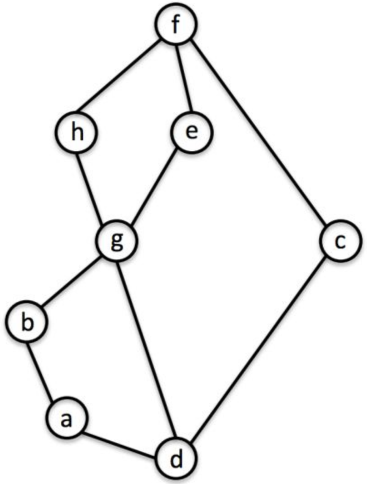
- Nodes = cities, Edges = roads
- Two participants -
  - A robber - can using edges
  - Some number of cops - can fly to nodes
- Everytime a cop is allowed to move, the robber can move to other vertices
- Cops win by trapping the robber; the robber wins by evading capture.

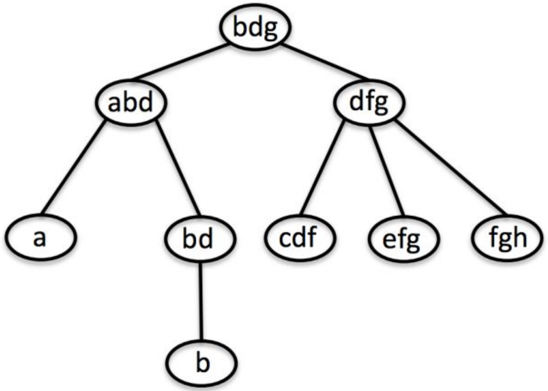


Cops and Robber

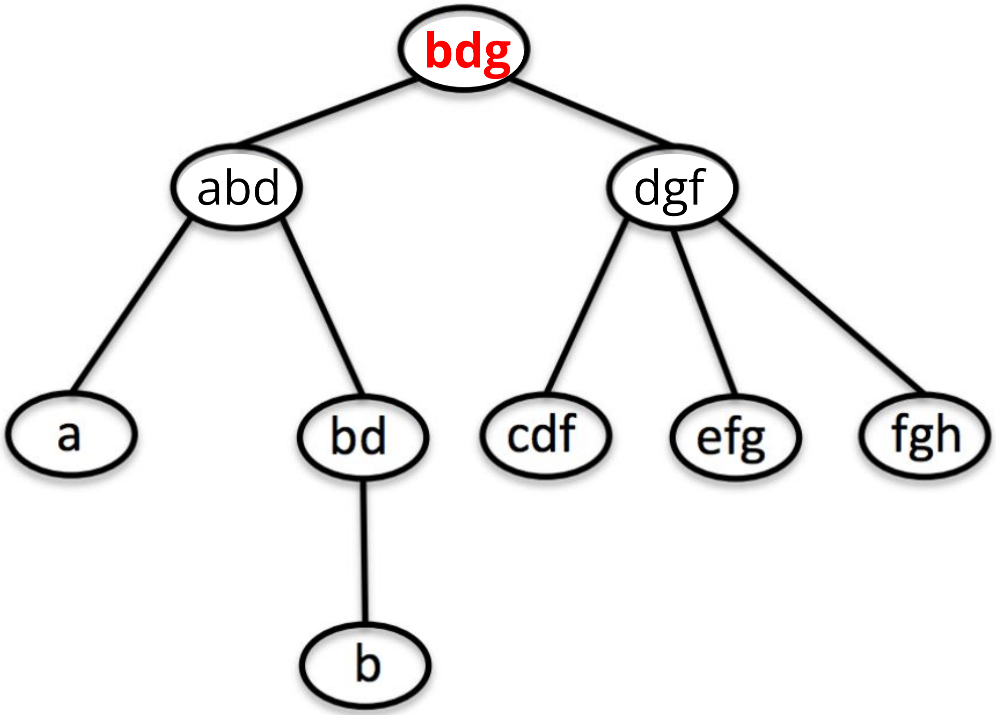
Concept:

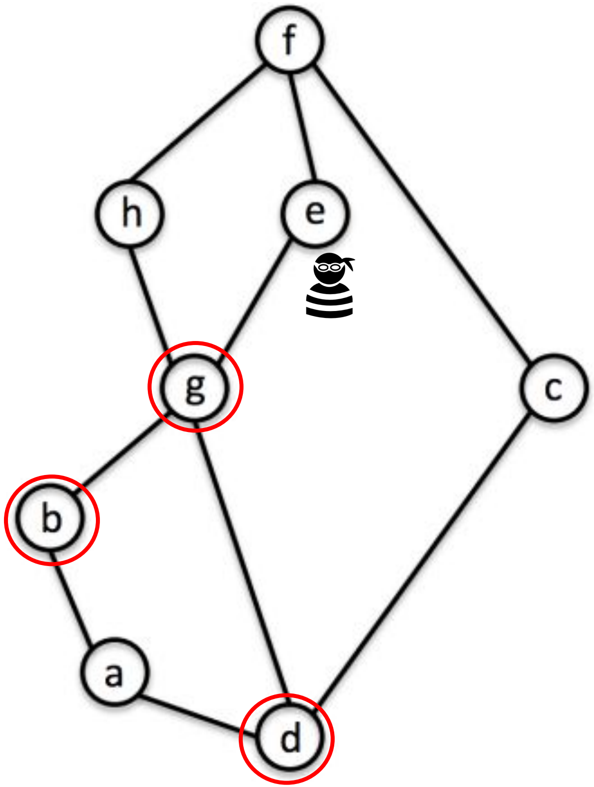
The treewidth of a graph G is at most *k* iff *k+1* cops can win the game.



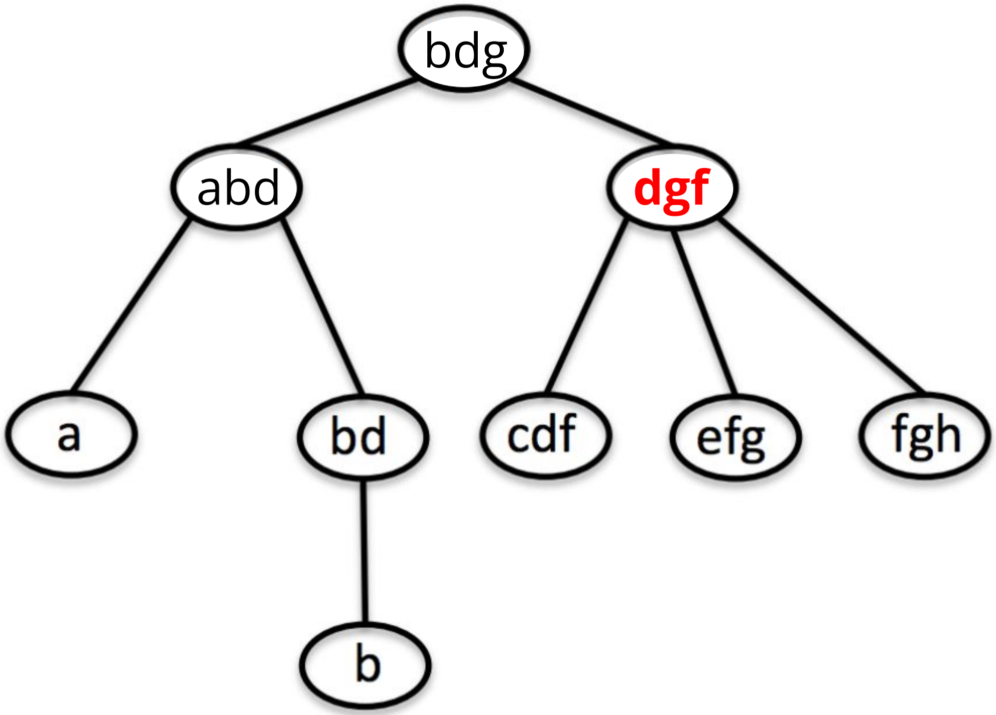


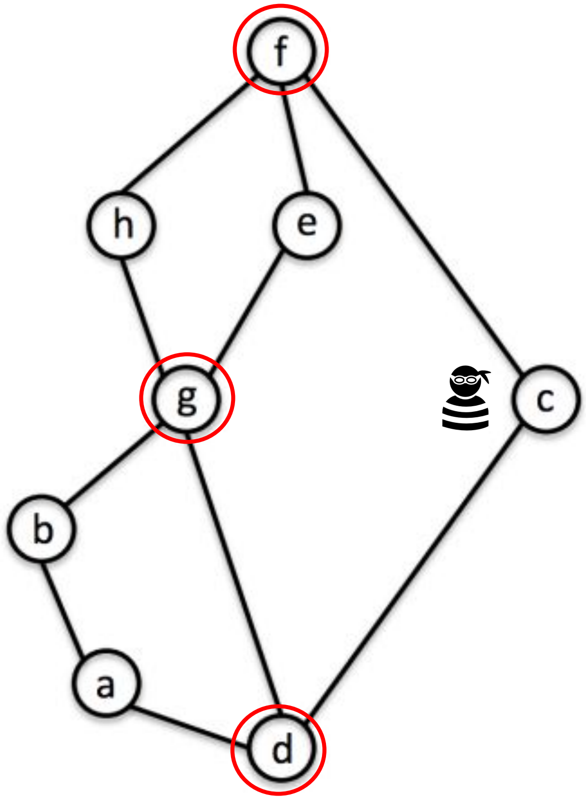
Cops and Robber



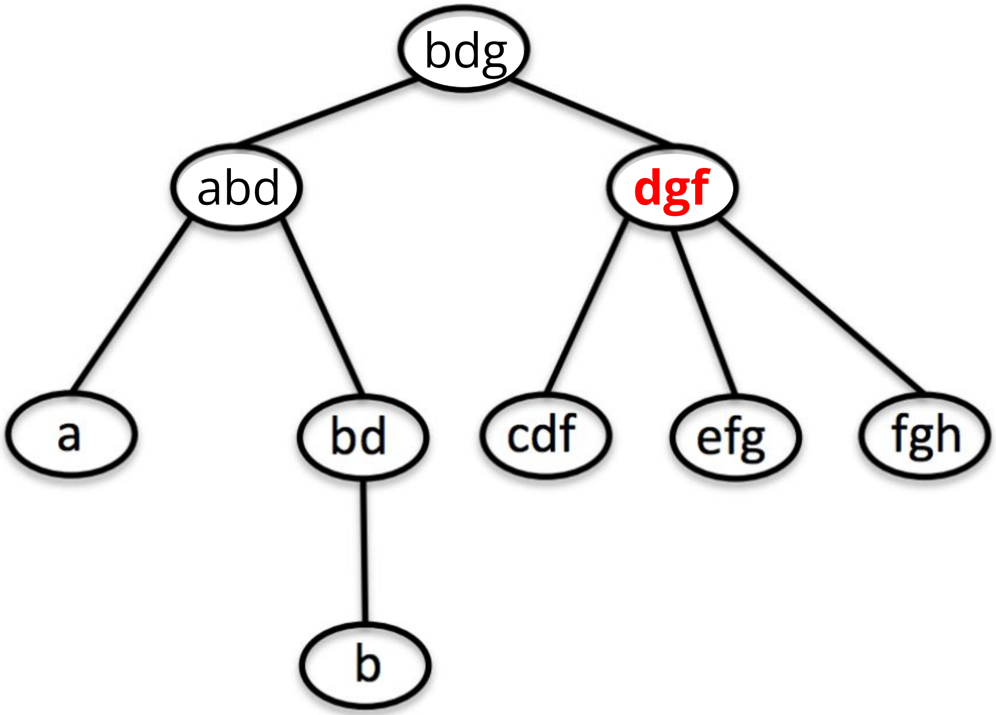


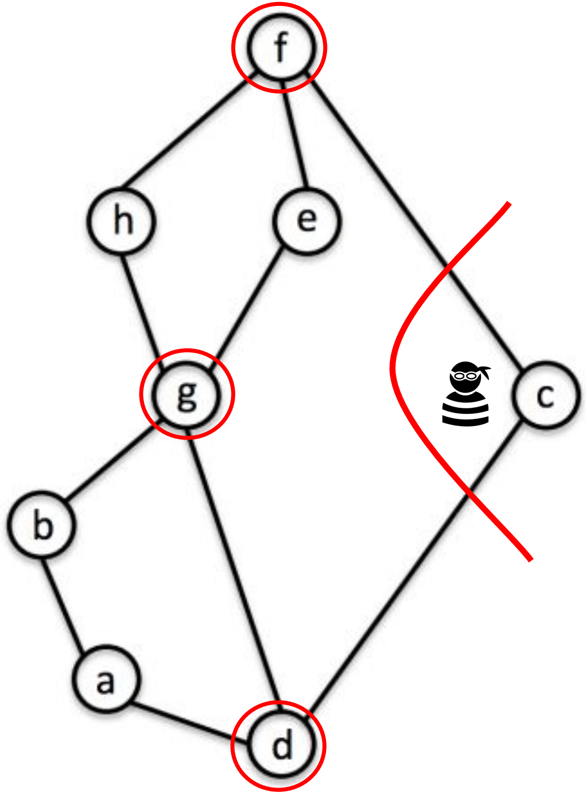
Cops and Robber



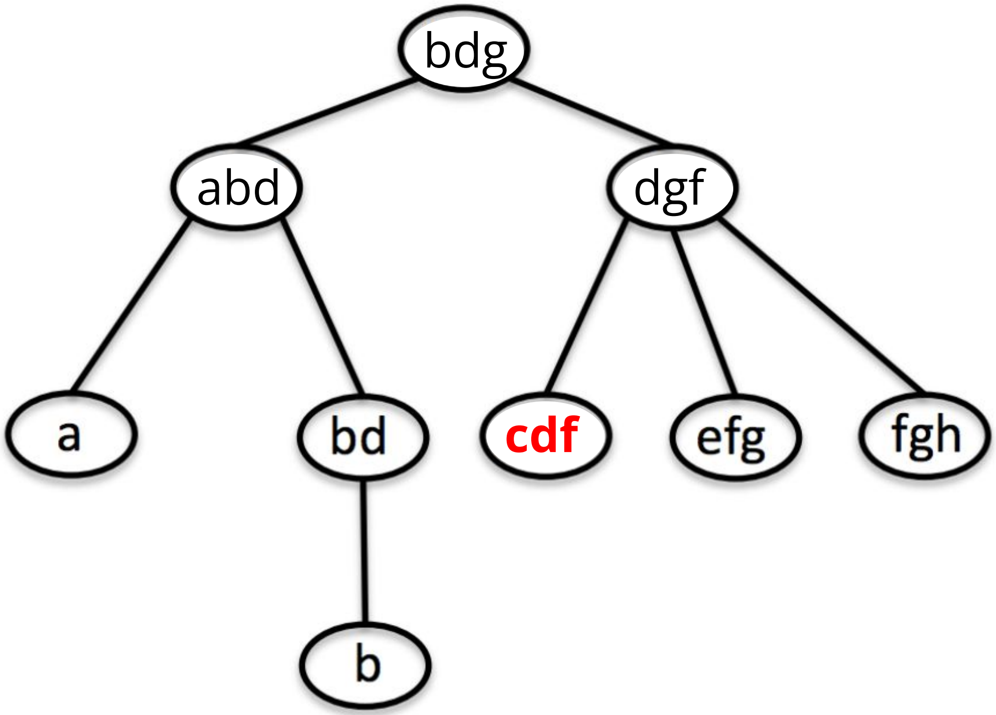


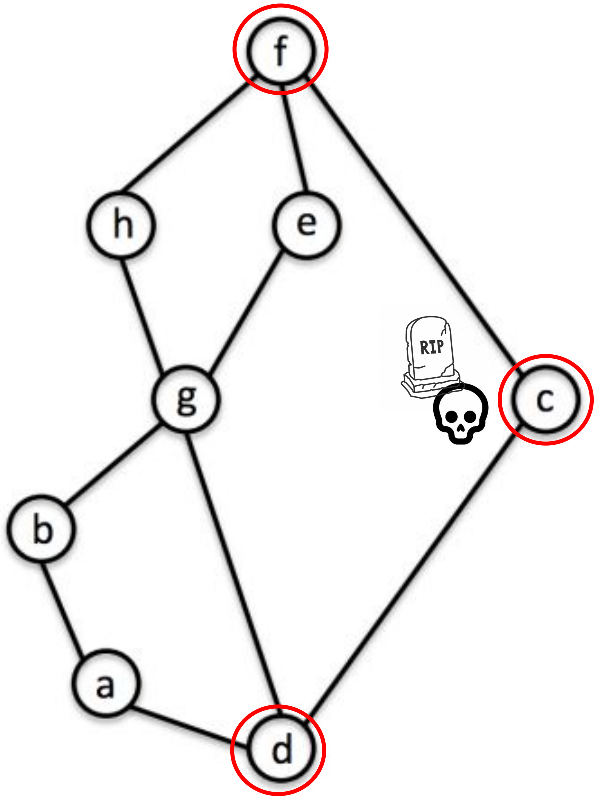
Cops and Robber





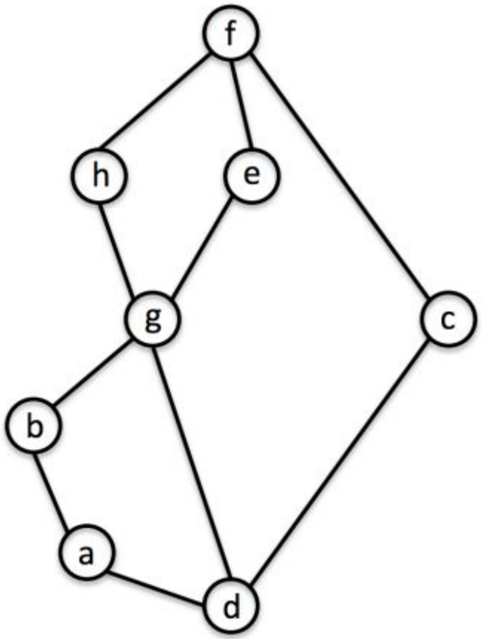
Cops and Robber





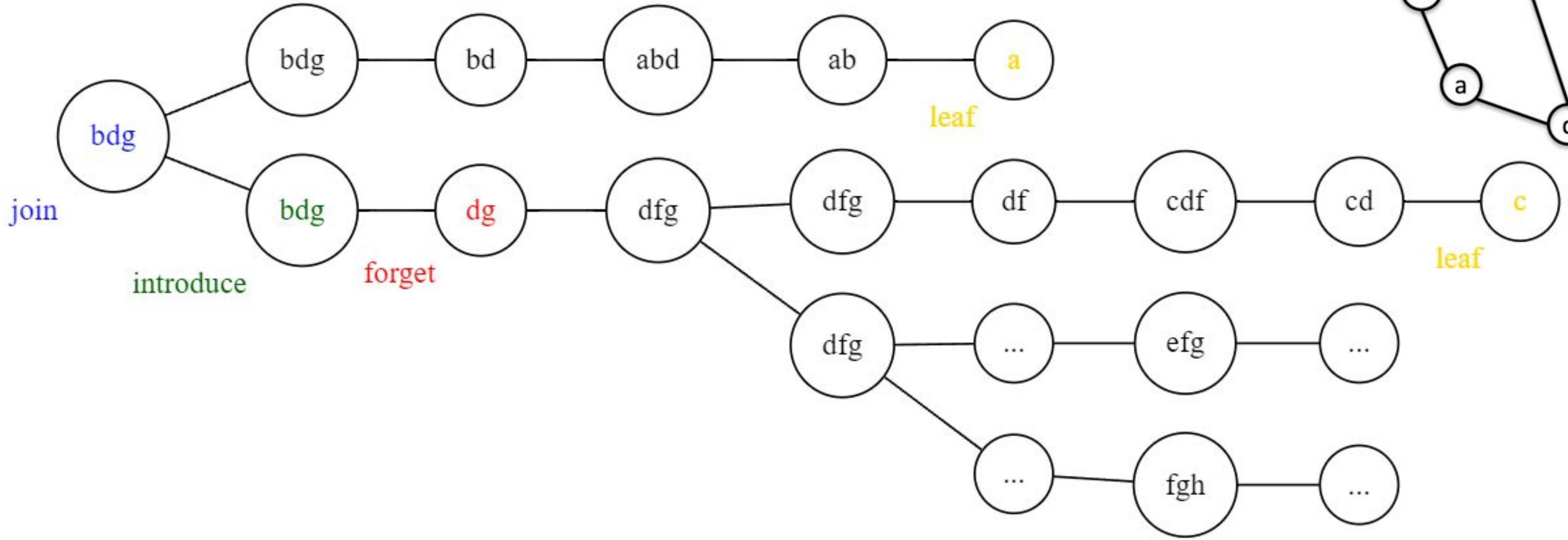
3-Coloring on Bounded Treewidth Graphs

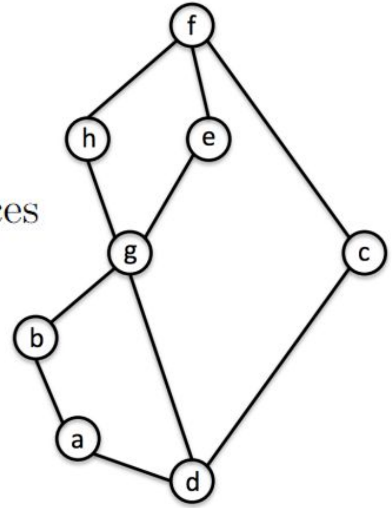
- 3-Coloring problem is Fixed Parameter Tractable (FPT) parameterized by treewidth
- Create a "*nice tree decomposition*" of the graph G
- Can be solved by keeping track of a table for each bag



3-Coloring on Bounded Treewidth Graphs

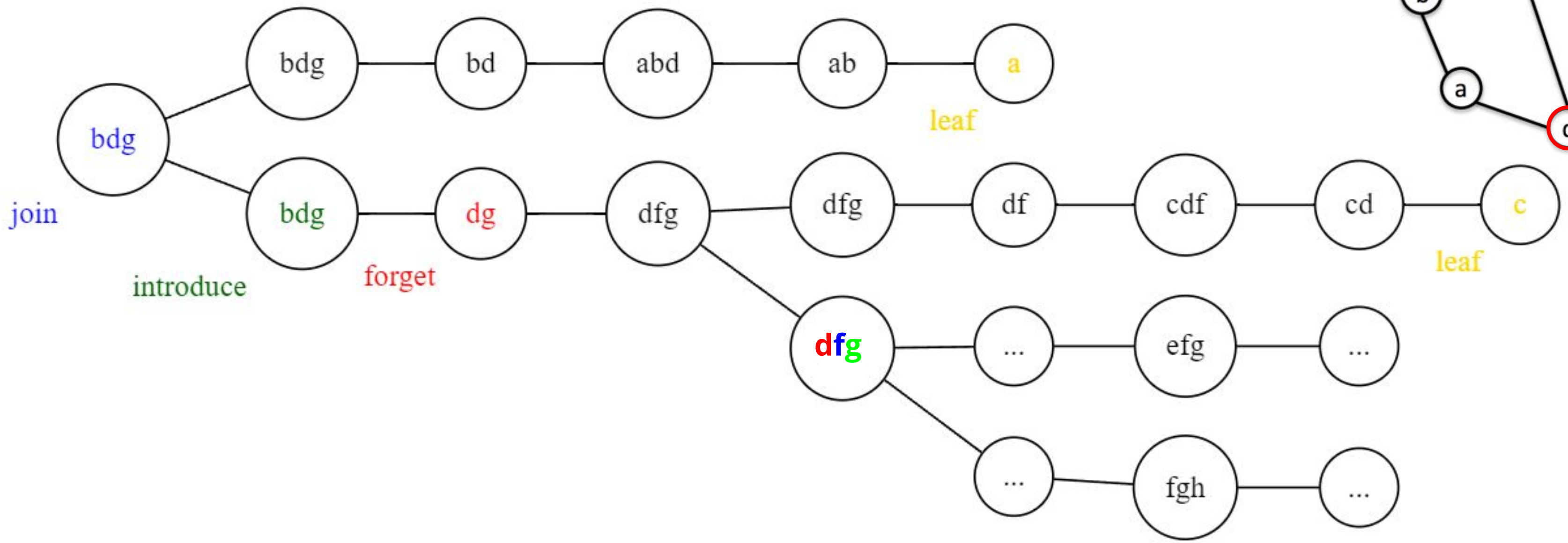
$$S[X, c] = \begin{cases} \text{true, if } c \text{ can be extended to proper coloring of descendent vertices} \\ \text{false, otherwise} \end{cases}$$

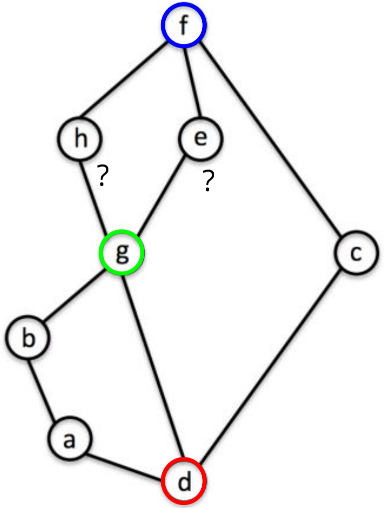




3-Coloring on Bounded Treewidth Graphs

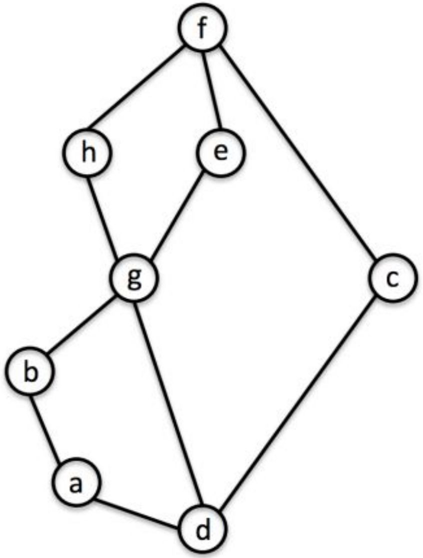
$S[dfg, \{rbg\}] = $ true, since both e and h can be colored red without conflict





3-Coloring on Bounded Treewidth Graphs

- For each bag and all possible coloring (*3^tw*, where *tw=3*), check if the vertices in the descendent nodes can be assigned any colors without conflict
- Start from the leaves, assign colors in bottom-up manner
- Check if the *S[root, c]* is true for any assignment of *c*
- If yes, the graph is 3-colorable, else not.
- Overall complexity: *O(3^tw)*n*



References

1. https://www.cs.cmu.edu/~odonnell/ - Algorithms for bounded treewidth
2. https://math.mit.edu/~apost/courses/18.204-2016/18.204_Gerrod_Voigt_final_paper.pdf - Survey Paper on Recent Findings in Treewidth
3. https://courses.engr.illinois.edu/cs374/fa2020/ - Maximum weighted independent set in a tree

Thank You

3-Coloring on Bounded Treewidth Graphs