Tree-width

Group: 4 - Omega

Group Members:
1805006 - Tanjeem Azwad Zaman
1805008 - Abdur Rafi
1805010 - Anwarul Bashir Shuaib
1805019 - MD Rownok Jahan Ratul
1805030 - MD Toki Tahmid

Problem Definition, List of Algorithms, and Towards an Exact Algorithm

1805006 - Tanjeem Azwad Zaman

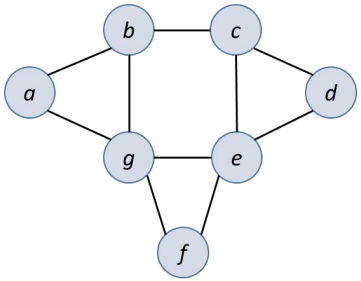
Topics To Cover

- Problem Definition
  - General Terminology
  - Tree Decomposition
  - Tree-Width
  - Problem Definition
- List of Algorithms
  - Exact
  - Approximate
  - Metaheuristic and Heuristic
- Towards an Exact Algorithm
  - Maximal potential clique
  - Minimal Separator

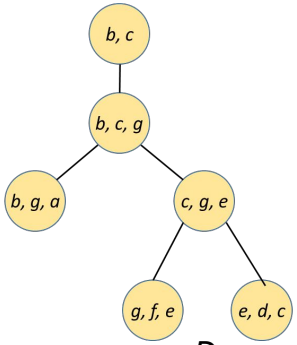
Problem Definition

General Terminology

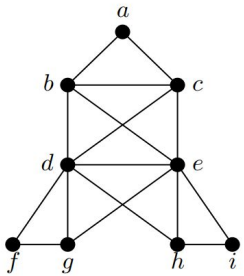
- Given a graph ***G***, with vertex set ***V(G)*** and edge set ***E(G)***
- In our context, Decomposition ***T*** is
  - Another graph-like ***<u>Structural Representation of G</u>***, where
  - Each node in T corresponds to ***<u>a subset of V(G)</u>*** **-> known as "Bag"s**
- A valid decomposition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



G_1



D_1



G_2

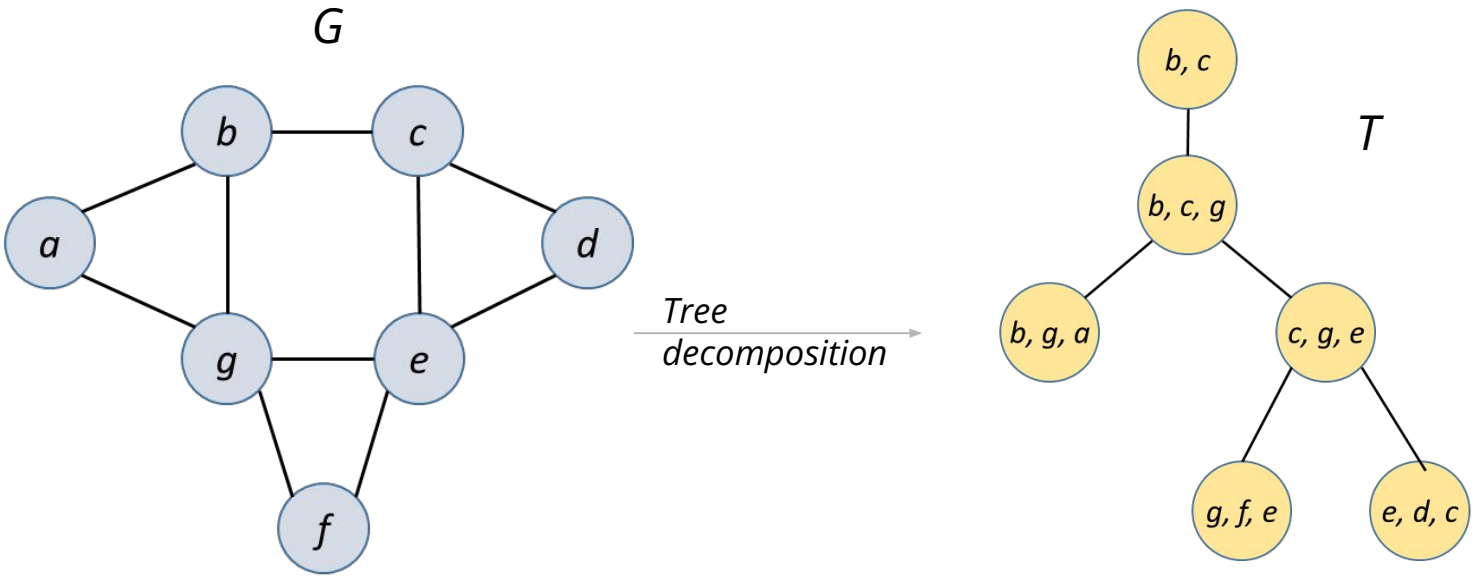
D_2

Tree Decomposition

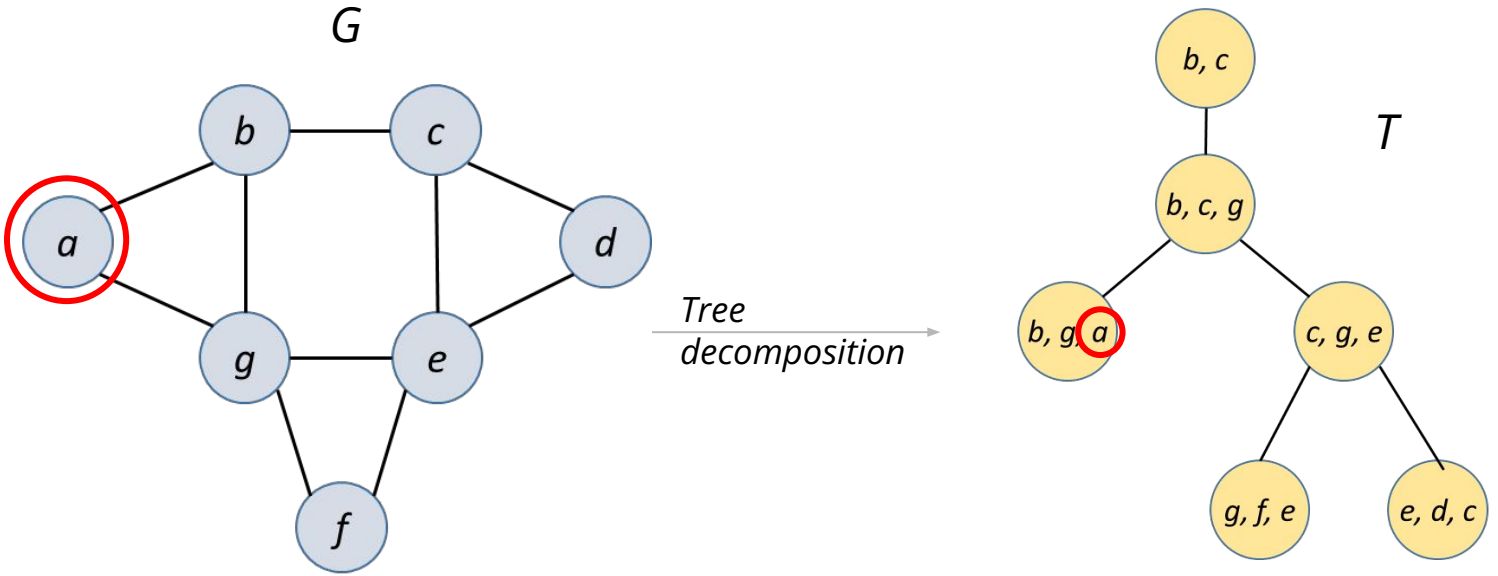
A tree decomposition is represented as: $\mathcal{T} = (\ T\ ,\ \{X_t\}_{t \in V(t)}\ )$, where

Formal	***Informal***
T is a tree	*T is* ***a tree***
$\{X_t\}\ \forall\ t$ *in V(T),* $X_t \in$ ***V(G)***	*Each bag (corresponding to a tree node) is a subset of V(G).* ***Set of all such bags***
And the following 3 properties hold:	
1. $\cup_{t \in V(T)} X_t = V(G)$	***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* X_t	***For all edges*** *in G, there is* ***at least 1 bag in T*** *that* ***has both endpoints*** *of the edge*
3. $\forall$ u $\in$ ***V(G)***, the set T_u = {t $\in$ ***V(T)*** : u $\in X_t$}	*All* ***bags that contain*** *any specific* ***vertex*** *of G,* ***make a connected subtree*** *in T*

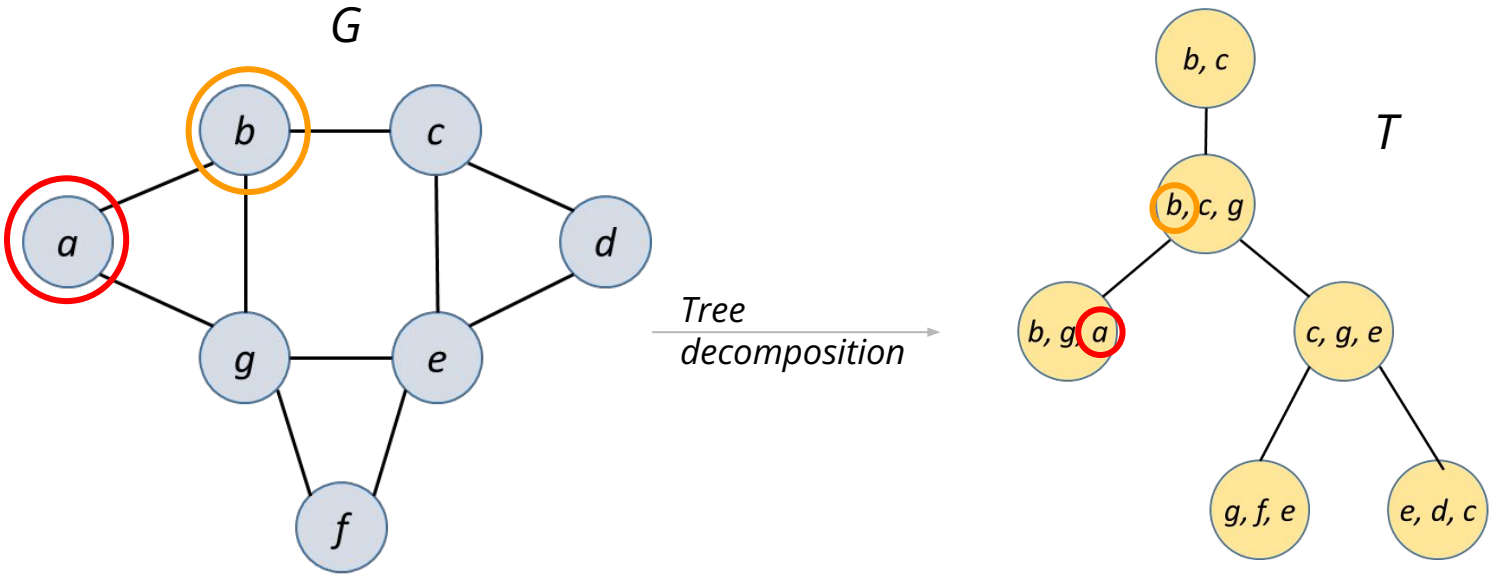
Example



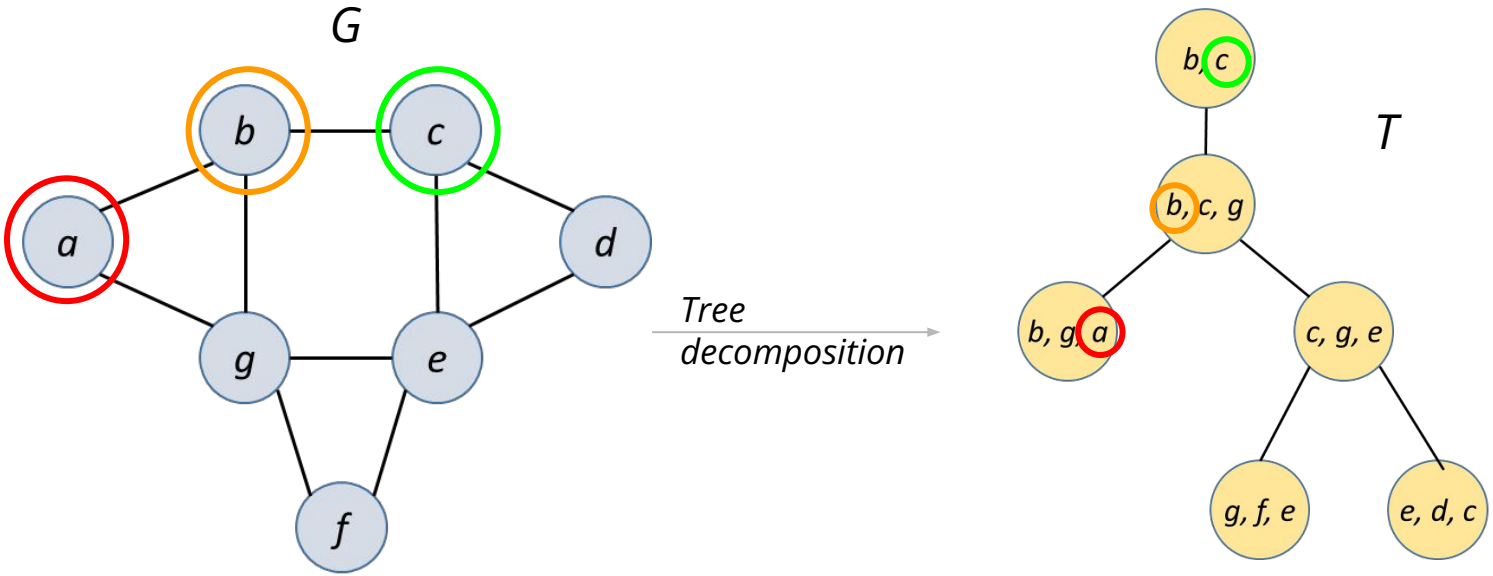
Example 1. All nodes belong to at least 1 bag



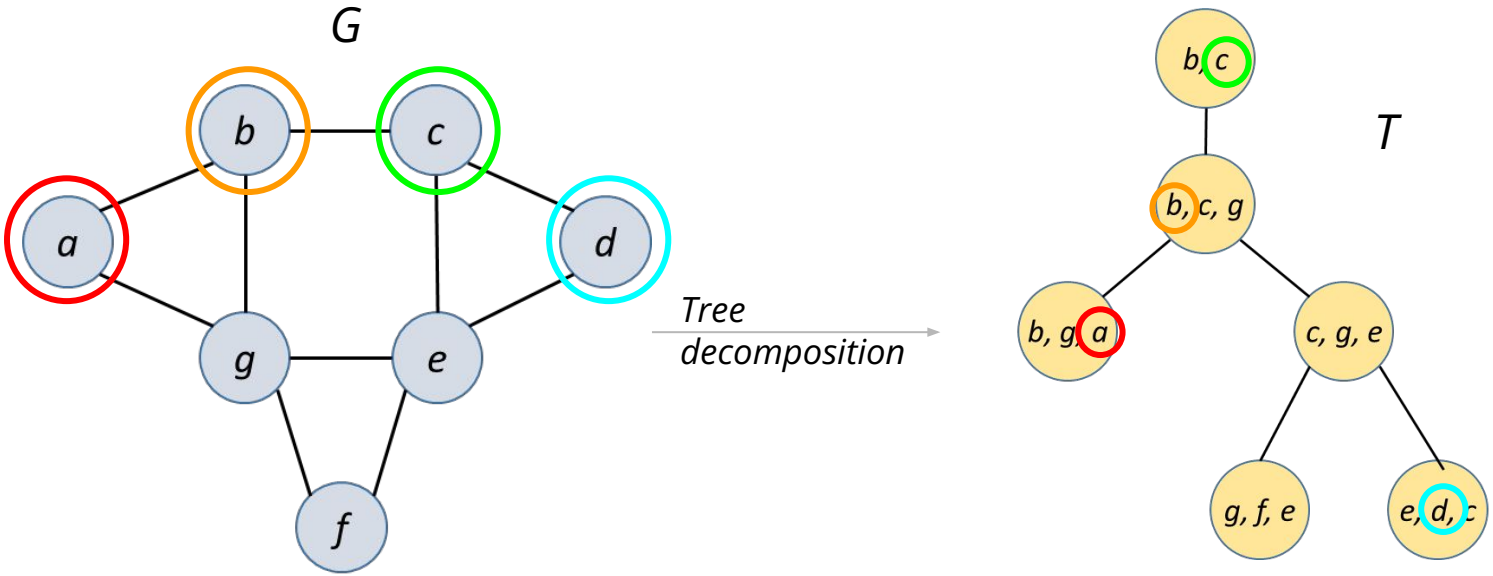
Example 1. All nodes belong to at least 1 bag



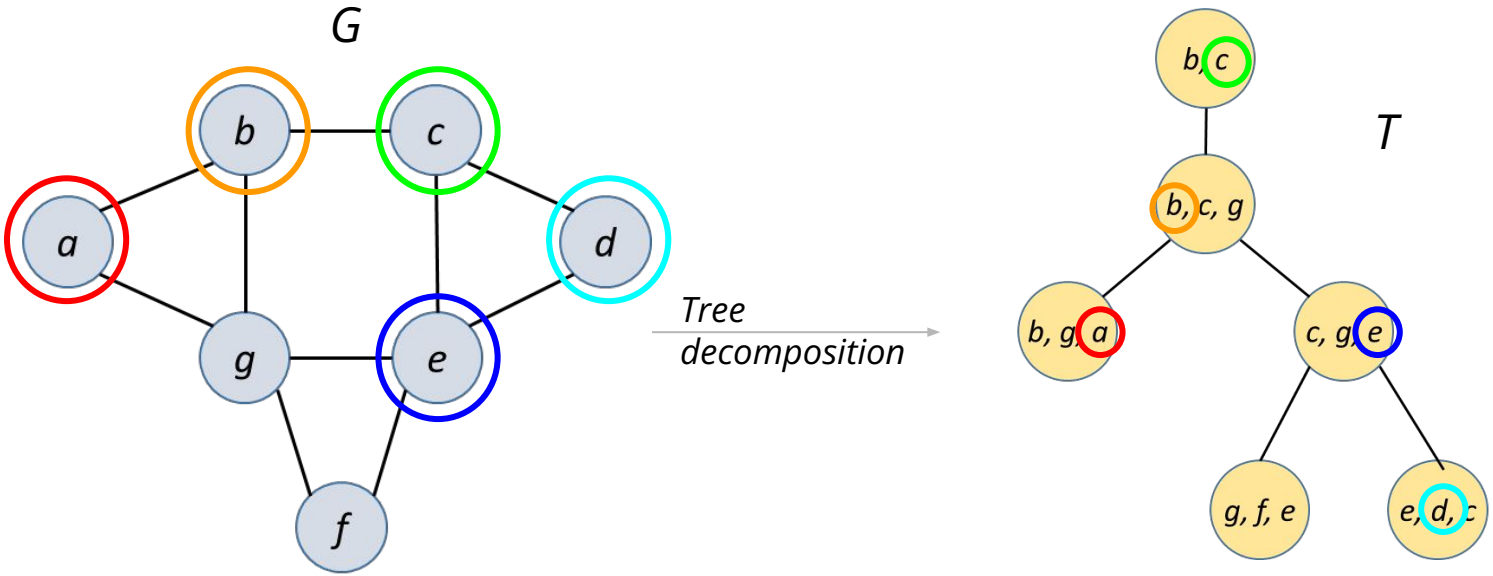
Example 1. All nodes belong to at least 1 bag



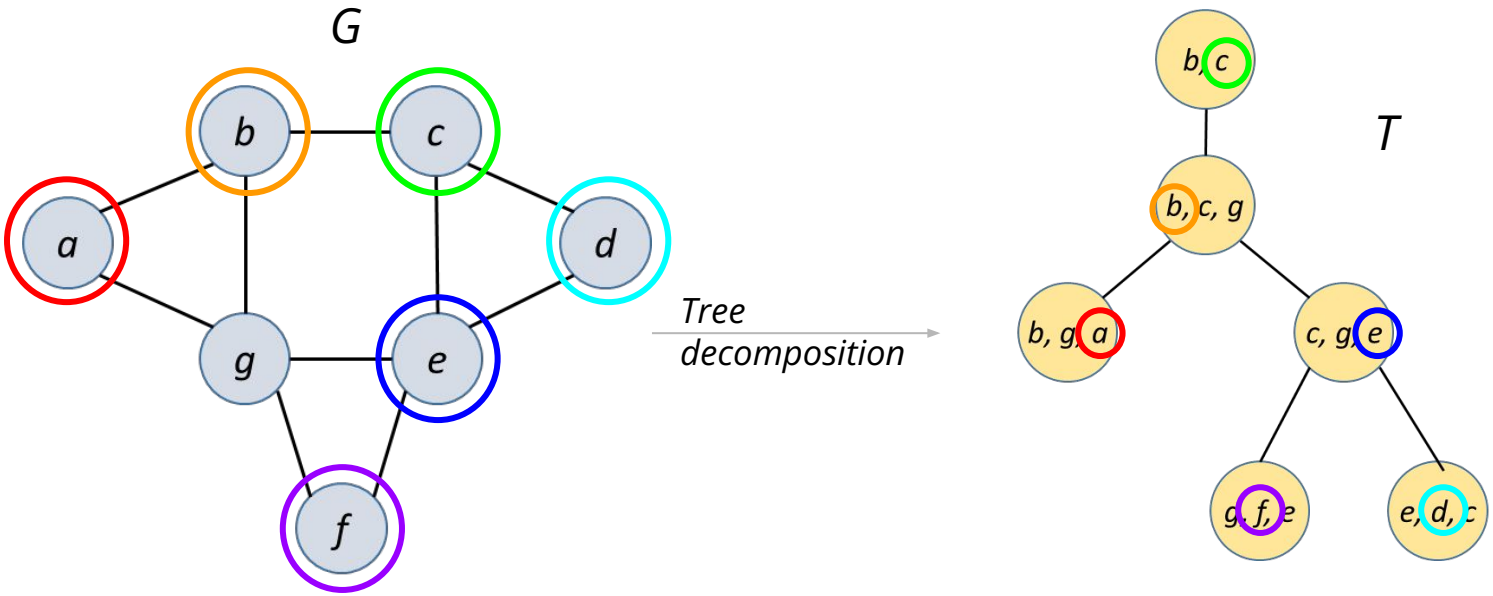
Example 1. All nodes belong to at least 1 bag



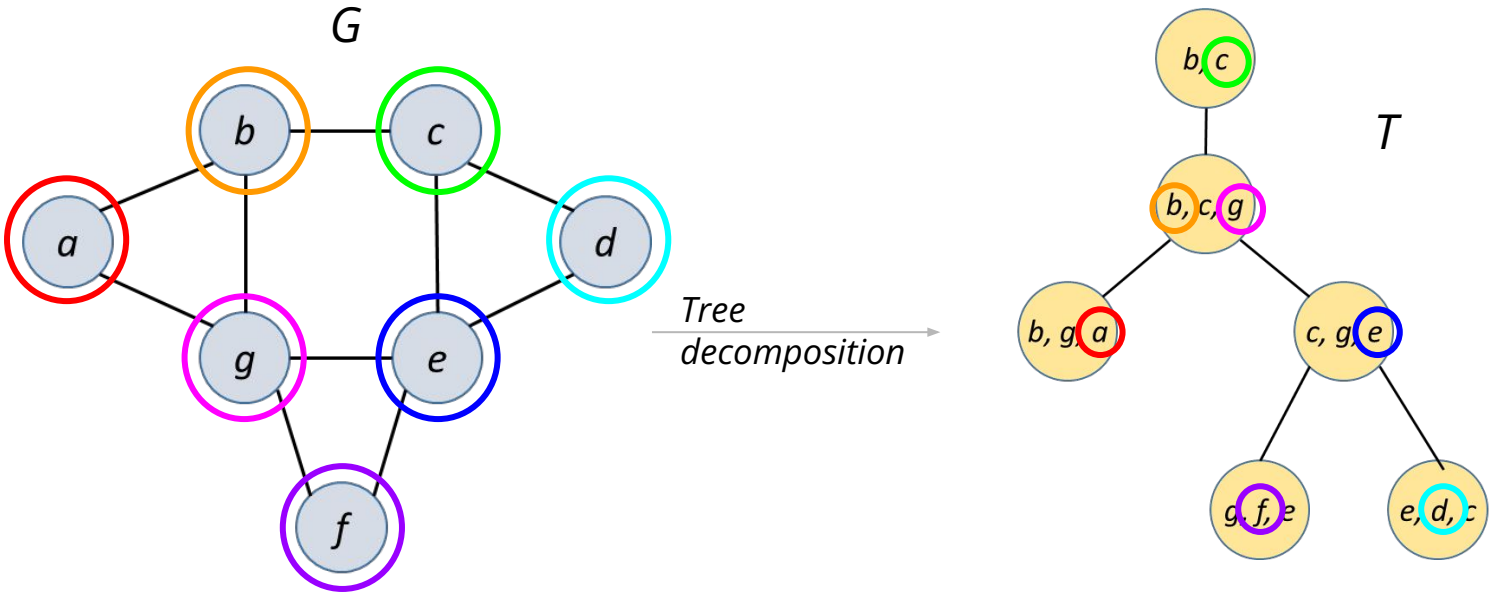
Example 1. All nodes belong to at least 1 bag



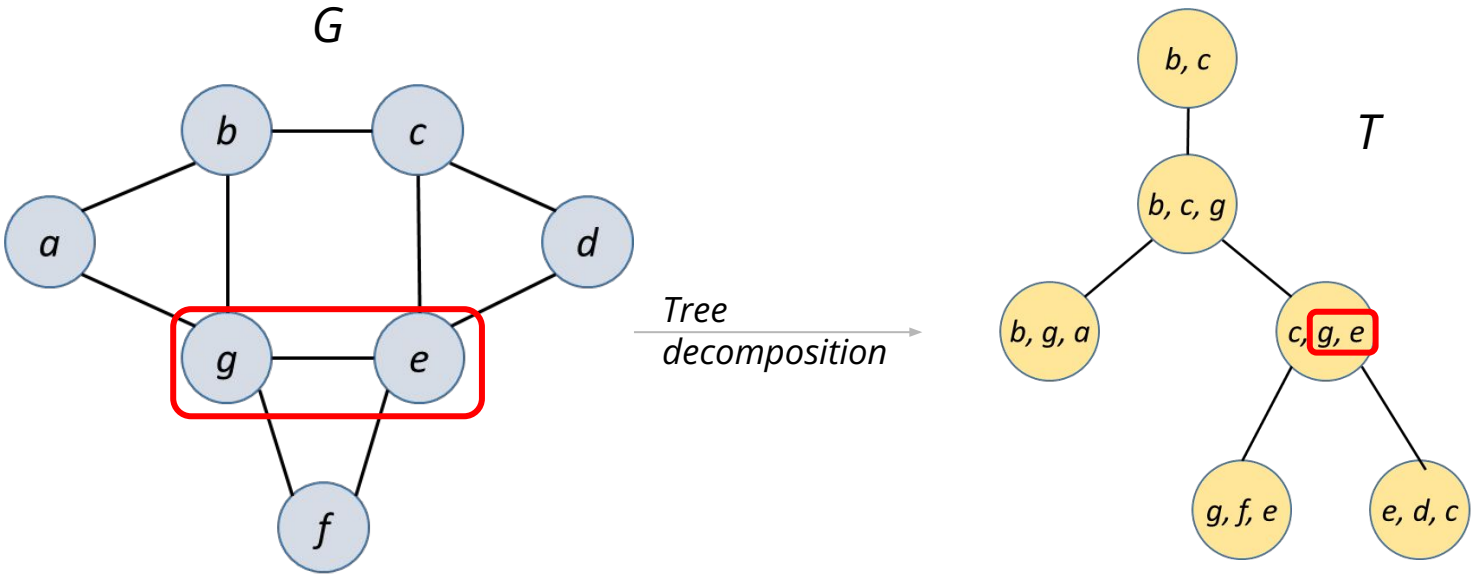
Example 1. All nodes belong to at least 1 bag



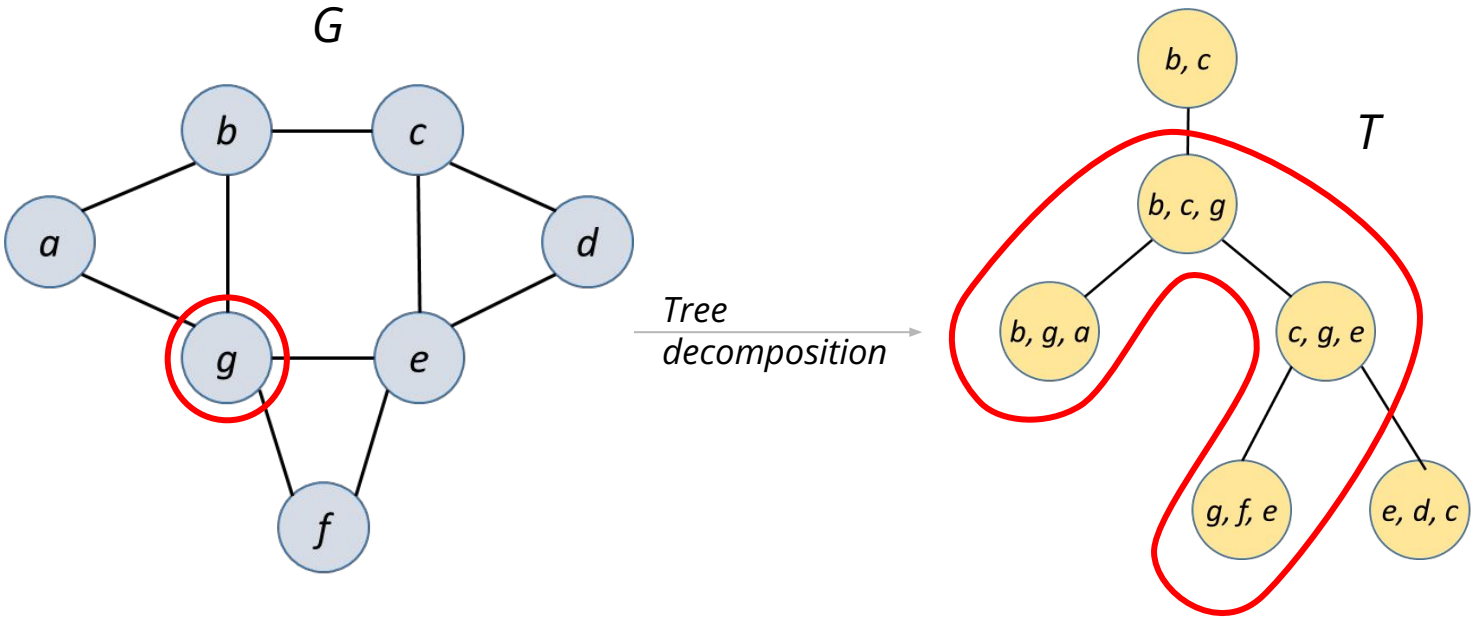
Example 1. All nodes belong to at least 1 bag



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



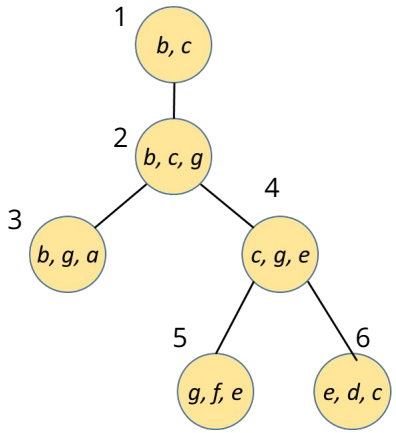
Example 3. All bags with a specific vertex will form a connected subtree



Treewidth

Width of a bag	(*Size of the bag*) - 1
Width of a tree	*Maximum* of the widths of *its bags*
Treewidth of a graph	*Minimum width* among *all tree decompositions* of the graph

Example:



Bags:

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

Bag widths:

size of $|X_1|$ = 2, so width of X_1 = 1.

Similarly widths of X_2, X_3, X_4, X_5, X_6 are all 2

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

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 Treewidth Problem is NP-Complete

List of Existing Algorithms

Exact Algorithms

Algorithm Name	Runtime/Approx Ratio	Authors	Year	Comments
PID - ACP (DP based Algorithm) (Positive Instance Driven)	$O(n^{k+2})$	Hisao Tamaki	2016	• 1st in PACE 2016: Exact Track
PID - Bouchitte Todinca	O(l-block ^{k} * O-block ^{k}) (emperical only)	Hisao Tamaki	2017	• 2nd in PACE 2017: Exact Track • Based on *minimal separators* and *potential maximal cliques*
Jdrasil: A Modular Library for Computing Tree Decompositions	Library of several algorithms	Max Bannach, Sebastian Berndt, and Thorsten Ehlers	2017	• 3rd in PACE 2017: Exact Track • Supports parallel processing
Large induced subgraphs via triangulations and CMSO	$O(1.7347^n)$	Fedor Fomin, Ioan Todinca, Yngve Villanger	2015	• Based on minimum triangulation

Approximate Algorithms

Algorithm Name	***Runtime/Approx Ratio***	***Authors***	***Year***	***Comments***
Finding all leftmost separators of size <= k	• Runs in $2^{6.755k} \cdot O(n \log n)$ • Approximation Ratio: 5K+1	Belbasi & Fürer	2021	• Focuses on improving the exponential value related to “K”
An Improved Parameterized Algorithm for Treewidth	• Runs in $2^{O(k^2)}n^{O(1)}$ • Approximation ratio: $(1 + \varepsilon)k$ [$\varepsilon \in (0, 1)$]	Tuukka Korhonen, Daniel Lokshtanov	2022	• 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]

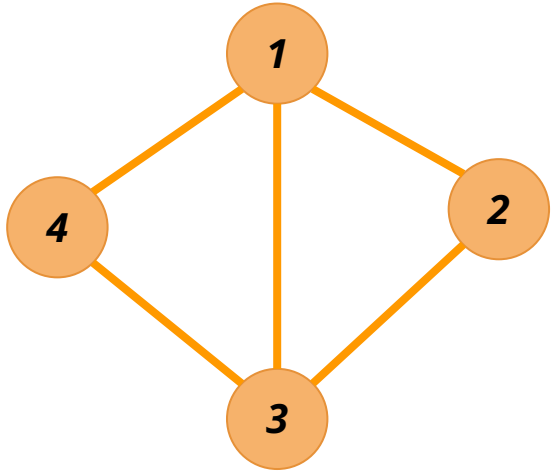
Metaheuristic and Heuristic Algorithms

Algorithm Name	Authors	Year	Comment
Genetic Algorithm for Treewidth	*Gogate, V., & Dechter, R.*	2002	Metaheuristic
FlowCutter PACE17	*Hamann, M., & Strasser, B.*	2017	Heuristic
Arnborg & Proskurowski's Heuristic	*Arnborg, S., & Proskurowski, A. (1989)*	1989	Heuristic
TreewidthDP (Dynamic Programming)	*Koster, R., Bodlaender, H.L., & Van Hoesel, S.P. (2005).*	2005	Exact / Heuristic

Towards an Exact Algorithm

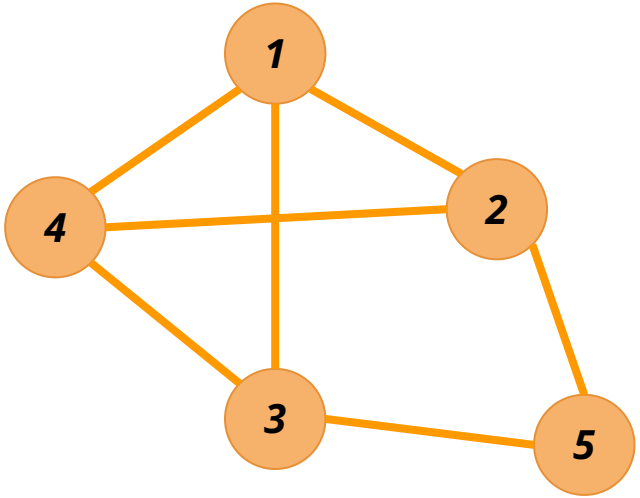
Potential maximal clique

- Chordal Graph:



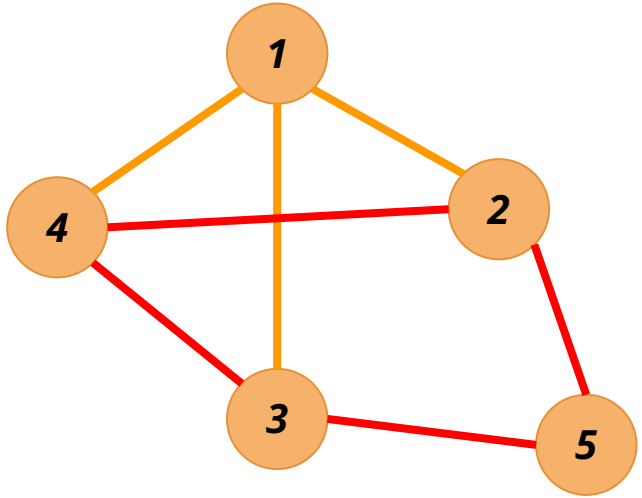
Potential maximal clique

- Chordal Graph:
- Minimal Chordal Completion:



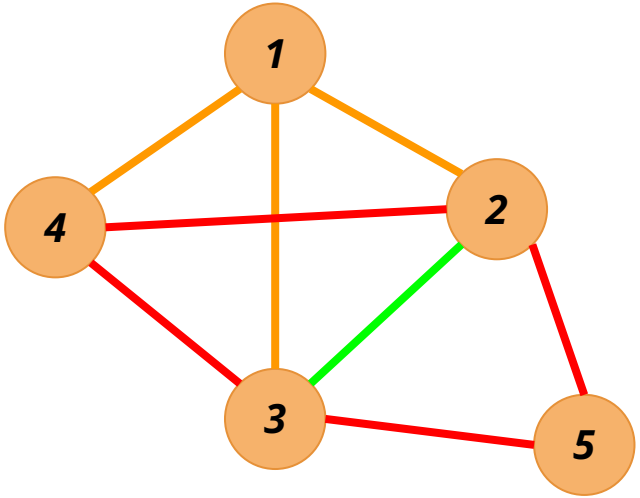
Potential maximal clique

- Chordal Graph:
- Minimal Chordal Completion:
  - A ***chordal version*** of a graph with ***minimum Extra Edges***



Potential maximal clique

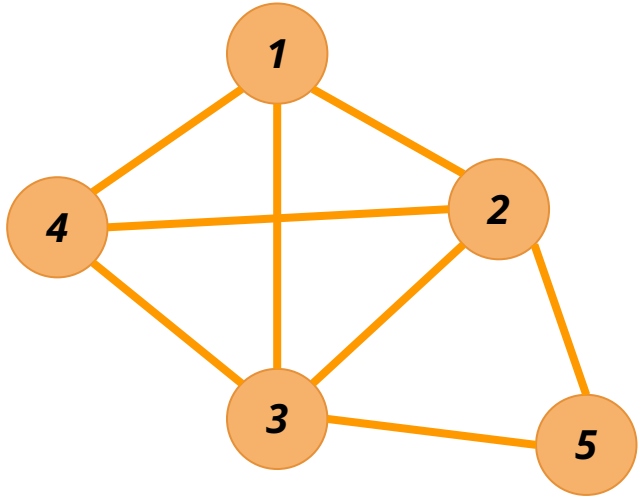
- Chordal Graph:
- Minimal Chordal Completion:
  - A ***chordal version*** of a graph with ***minimum Extra Edges***



Potential maximal clique

- Chordal Graph:

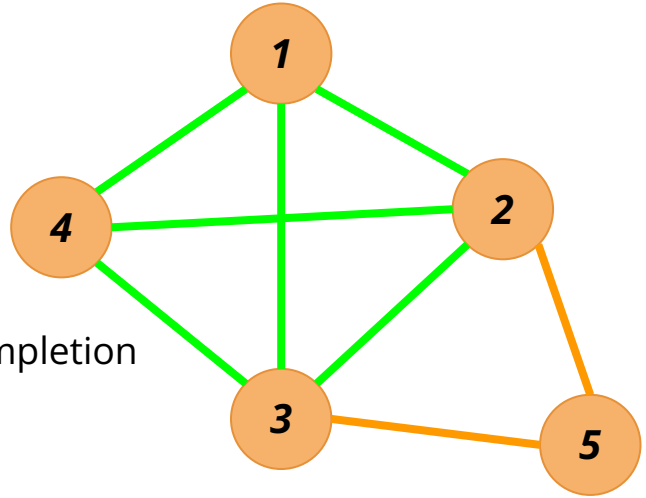
- Minimal Chordal Completion:
  - A ***chordal version*** of a graph with ***minimum Extra Edges***



Potential maximal clique

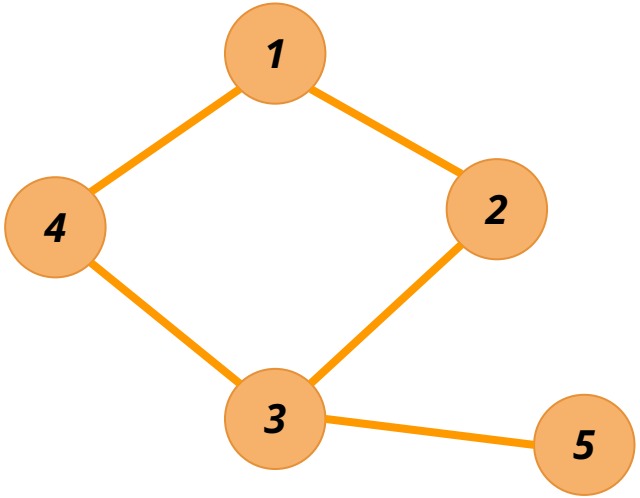
- Chordal Graph:

- Minimal Chordal Completion:
  - A ***chordal version*** of a graph with ***minimum Extra Edges***
- Potential Maximal Clique:
  - A maximal clique in any minimal chordal completion



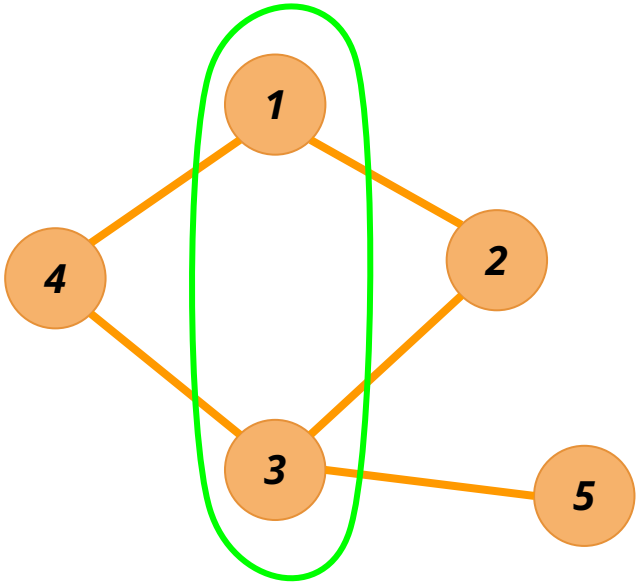
Minimal Separator

- Separator:
  - ***Set of Vertices*** which, if removed, divides Graph into 2 or more components



Minimal Separator

- Separator:
  - Set of Vertices which, if removed, divides Graph into 2 or more components
  - Eg: {1,3}

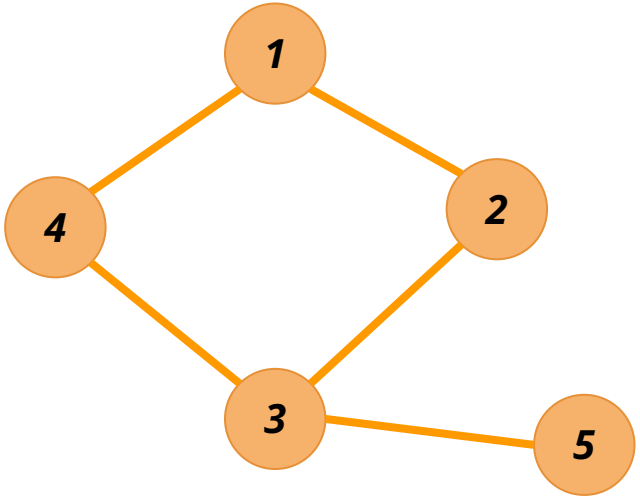


Minimal Separator

- Separator:
  - Set of Vertices which, if removed, divides Graph into 2 or more components
  - Eg: {1,3} -> {4}, {2}, {5}

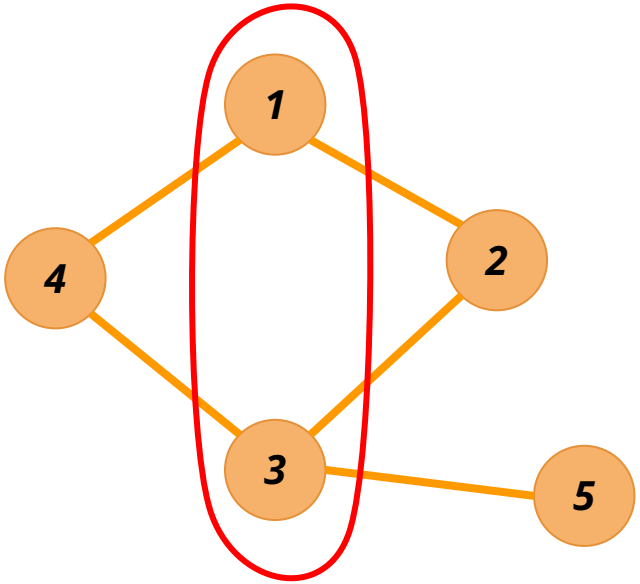
Minimal Separator

- Separator:
  - ***Set of Vertices*** which, if removed, divides Graph into 2 or more components
- Minimal Separator:
  - Separator s.t. no subset is also a separator



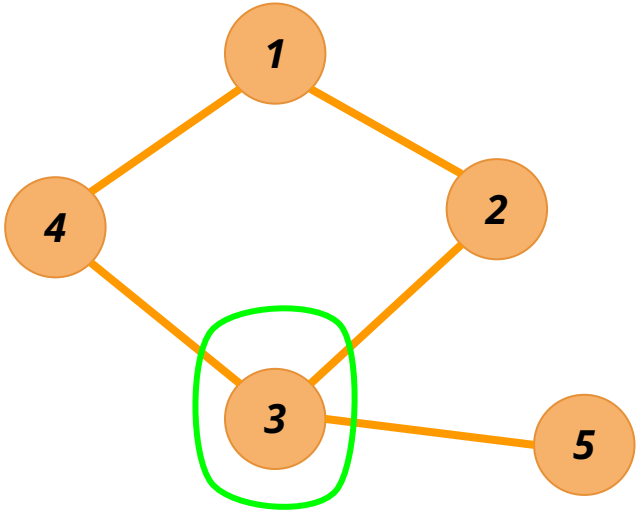
Minimal Separator

- Separator:
  - ***Set of Vertices*** which, if removed, divides Graph into 2 or more components
- Minimal Separator:
  - Separator s.t. no subset is also a separator
  - Eg {1,3} not minimal



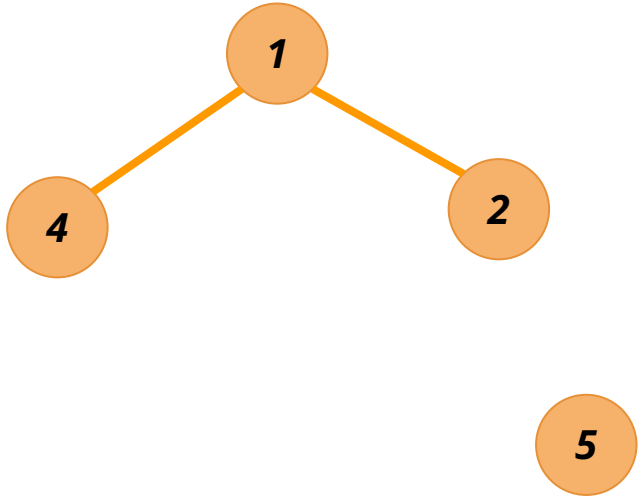
Minimal Separator

- Separator:
  - ***Set of Vertices*** which, if removed, divides Graph into 2 or more components
- Minimal Separator:
  - Separator s.t. no subset is also a separator
  - Eg {1,3} not minimal
  - Since {3} is also a seperator



Minimal Separator

- Separator:
  - ***Set of Vertices*** which, if removed, divides Graph into 2 or more components
- Minimal Separator:
  - Separator s.t. no subset is also a separator
  - Eg {1,3} not minimal
  - Since {3} is also a seperator



Positive-Instance-Driven Bouchitte and Todinca Algorithm

1805019 - MD Rownok Zahan Ratul

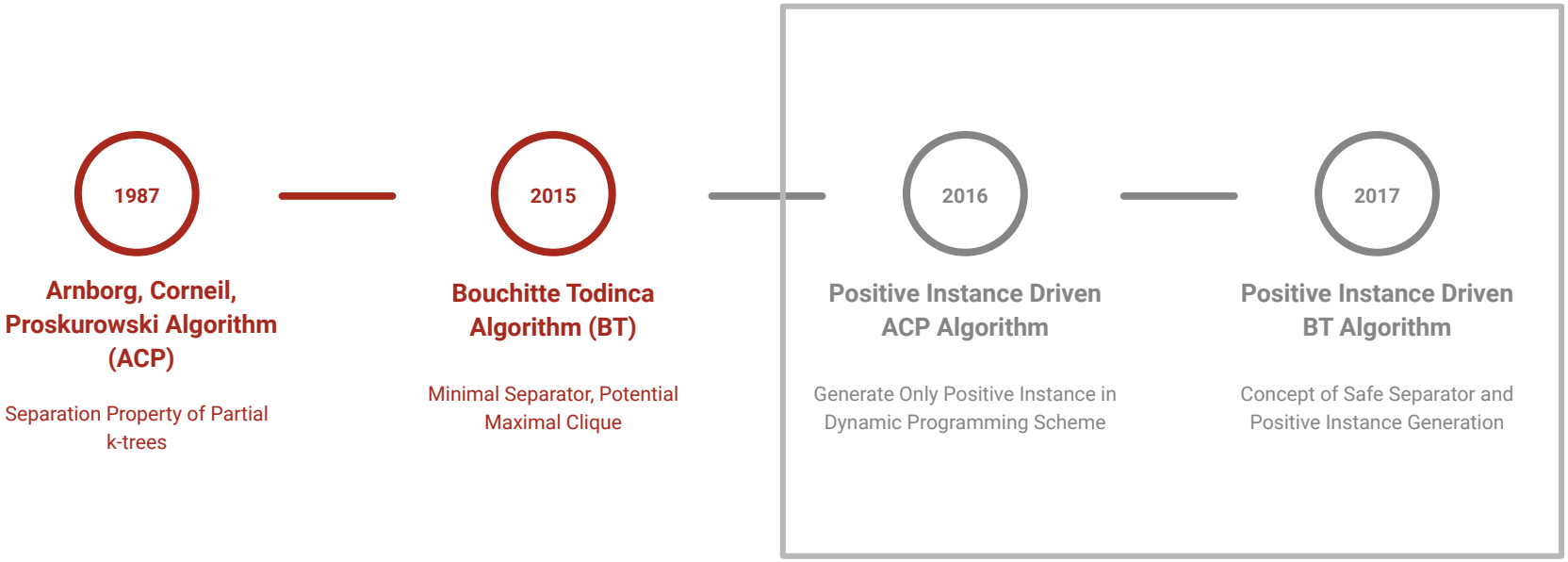
Outline

- Treewidth
  - Problem Definition & Formulation
- Approach towards exact algorithms
  - Minimal Separators
  - Potential Maximal Clique
- PID-BT algorithm
  - Definitions
  - Basic BT algorithm
  - Idea behind PID-BT
  - Analysis of PID-BT
- Implementation of PID-BT
  - Analytics

Outline

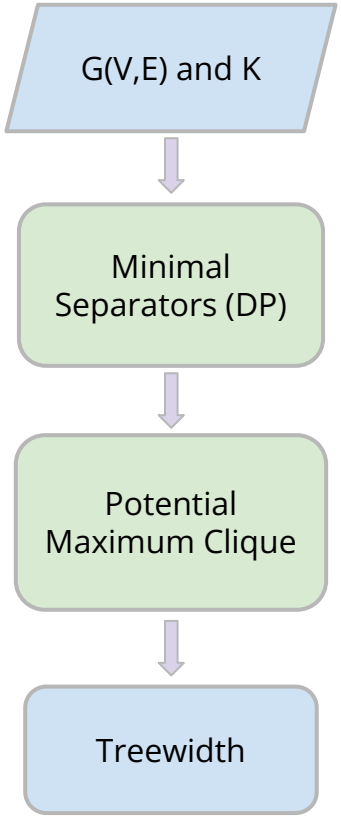
- Treewidth
  - Problem Definition & Formulation
- Approach towards exact algorithms
  - Minimal Separators
  - Potential Maximal Clique
- PID-BT algorithm
  - Definitions
  - Basic BT algorithm
  - Idea behind PID-BT
  - Analysis of PID-BT
- Implementation of PID-BT
  - Analytics

Approach Towards Exact Algorithms

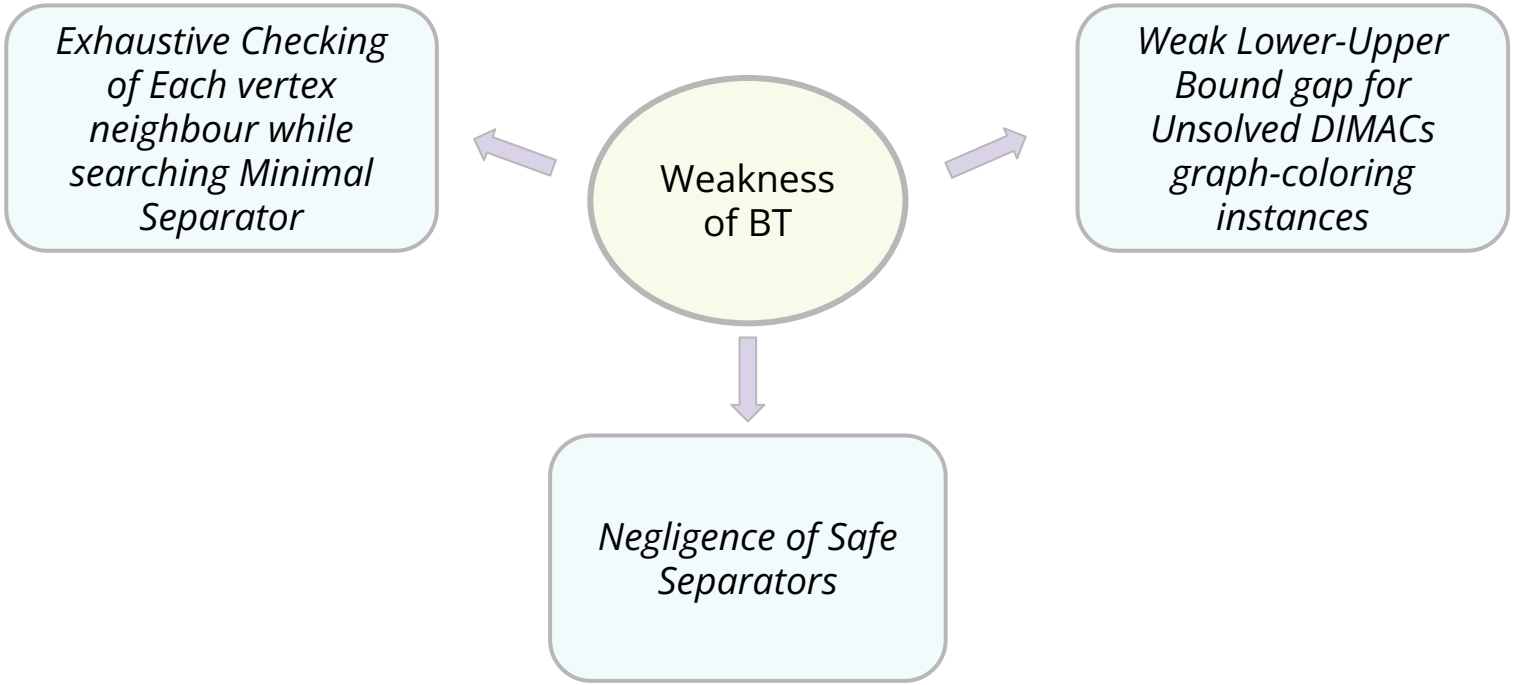


Bouchitte Todinca Algorithm

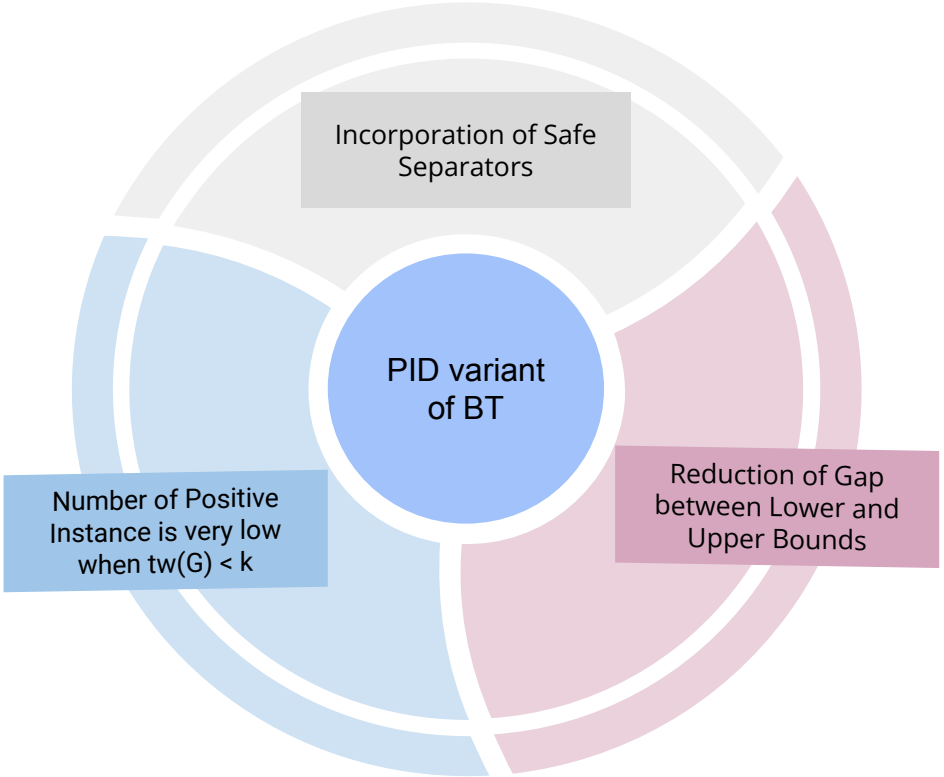
- Given, G(V, E) and K
  - Find Minimal Separator Set using Dynamic Programming
  - Derive the Potential Maximum Clique set based on the Associated Minimal Separator set
  - Infer the tree decomposition using Cuts using the Maximum Clique set
  - Return Treewidth



Scope of BT-algorithm



Incorporation of PID with BT-algorithm



Challenges ...

Challenge of PID incorporation with BT

Connected Subset of vertices for construction of Minimal Separator

Recurrence for Deciding Feasibility of C set

May involve an indefinite number of Connected vertex search space (exponential)

Challenges ...

Challenge of PID incorporation with BT

Connected Subset of vertices for construction of Minimal Separator

Solution
I-blocks and O-blocks

Recurrence for Deciding Feasibility of C set

May involve an indefinite number of Connected vertex search space (exponential)

PID-BT Algorithm

1. Let $\mathscr{I}_0 = \emptyset$ and $\mathscr{O}_0 = \emptyset$.
2. Initialize $\mathscr{P}_0$ and $\mathscr{S}_0$ to $\emptyset$.
3. Set $j = 0$.
4. For each $v \in V(G)$, if $N[v]$ is a potential maximal clique with $|N[v]| \leq k+1$ then add $N[v]$ to $\mathscr{P}_0$ and if, moreover, support$(N[v]) = \emptyset$ then do the following.
   (a) Add $N[v]$ to $\mathscr{S}_0$.
   (b) If outlet$(N[v]) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(N[v]), N[v])$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
5. Set $i = 0$.
6. Repeat the following and stop repetition when j is not incremented during the iteration step.
   (a) While $i < j$, do the following.
      i. Increment i and let $\mathscr{I}_i$ be $\mathscr{I}_{i-1} \cup \{C_i\}$.
      ii. Initialize $\mathscr{O}_i$ to $\mathscr{O}_{i-1}$, $\mathscr{P}_i$ to $\mathscr{P}_{i-1}$, and $\mathscr{S}_i$ to $\mathscr{S}_{i-1}$.
      iii. For each $B \in \mathscr{O}_{i-1}$ such that $C_i \subseteq B$ and $|N(C_i) \cup N(B)| \leq k+1$, let $K = N(C_i) \cup N(B)$ and do the following.
         A. If K is a potential maximal clique, then add K to $\mathscr{P}_i$.
         B. If $|K| \leq k$ and there is a full component A associated with K (which is unique), then add A to $\mathscr{O}_i$.
      iv. Let A be the full component associated with $N(C_i)$ and add A to $\mathscr{O}_i$.
      v. For each $A \in \mathscr{O}_i \setminus \mathscr{O}_{i-1}$ and $v \in N(A)$, let $K = N(A) \cup (n(v) \cap A)$ and if $|K| \leq k+1$ and K is a potential maximal clique then add K to $\mathscr{P}_i$.
      vi. For each $K \in \mathscr{P}_i \setminus \mathscr{S}_{i-1}$, if support$(K) \subseteq \mathscr{I}_i$ then add K to $\mathscr{S}_i$ and do the following: if outlet$(K) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(K), K)$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
7. If there is some $K \in \mathscr{S}_j$ such that outlet$(K) = \emptyset$, then answer "YES"; otherwise, answer "NO".

PID-BT Algorithm

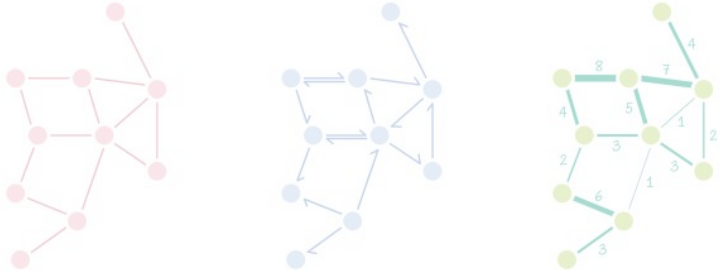
1. Let $\mathscr{I}_0 = \emptyset$ and $\mathscr{O}_0 = \emptyset$.
2. Initialize $\mathscr{P}_0$ and $\mathscr{S}_0$ to $\emptyset$.
3. Set $j = 0$.
4. For each $v \in V(G)$, if $N[v]$ is a potential maximal clique with $|N[v]| \leq k+1$ then add $N[v]$ to $\mathscr{P}_0$ and if, moreover, support$(N[v]) = \emptyset$ then do the following.
   (a) Add $N[v]$ to $\mathscr{S}_0$.
   (b) If outlet$(N[v]) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(N[v]), N[v])$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
5. Set $i = 0$.
6. Repeat the following and stop repetition when j is not incremented during the iteration step.
   (a) While $i < j$, do the following.
      i. Increment i and let $\mathscr{I}_i$ be $\mathscr{I}_{i-1} \cup \{C_i\}$.
      ii. Initialize $\mathscr{O}_i$ to $\mathscr{O}_{i-1}$, $\mathscr{P}_i$ to $\mathscr{P}_{i-1}$, and $\mathscr{S}_i$ to $\mathscr{S}_{i-1}$.
      iii. For each $B \in \mathscr{O}_{i-1}$ such that $C_i \subseteq B$ and $|N(C_i) \cup N(B)| \leq k+1$, let $K = N(C_i) \cup N(B)$ and do the following.
         A. If K is a potential maximal clique, then add K to $\mathscr{P}_i$.
         B. If $|K| \leq k$ and there is a full component A associated with K (which is unique), then add A to $\mathscr{O}_i$.
      iv. Let A be the full component associated with $N(C_i)$ and add A to $\mathscr{O}_i$.
      v. For each $A \in \mathscr{O}_i \setminus \mathscr{O}_{i-1}$ and $v \in N(A)$, let $K = N(A) \cup (n(v) \cap A)$ and if $|K| \leq k+1$ and K is a potential maximal clique then add K to $\mathscr{P}_i$.
      vi. For each $K \in \mathscr{P}_i \setminus \mathscr{S}_{i-1}$, if support$(K) \subseteq \mathscr{I}_i$ then add K to $\mathscr{S}_i$ and do the following: if outlet$(K) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(K), K)$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
7. If there is some $K \in \mathscr{S}_j$ such that outlet$(K) = \emptyset$, then answer "YES"; otherwise, answer "NO".

PID-BT Algorithm

1. Let $\mathscr{I}_0 = \emptyset$ and $\mathscr{O}_0 = \emptyset$.
2. Initialize $\mathscr{P}_0$ and $\mathscr{S}_0$ to $\emptyset$.
3. Set $j = 0$.
4. For each $v \in V(G)$, if $N[v]$ is a potential maximal clique with $|N[v]| \leq k+1$ then add $N[v]$ to $\mathscr{P}_0$ and if, moreover, support$(N[v]) = \emptyset$ then do the following.
   (a) Add $N[v]$ to $\mathscr{S}_0$.
   (b) If outlet$(N[v]) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(N[v]), N[v])$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
5. Set $i = 0$.
6. Repeat the following and stop repetition when j is not incremented during the iteration step.
   (a) While $i < j$, do the following.
      i. Increment i and let $\mathscr{I}_i$ be $\mathscr{I}_{i-1} \cup \{C_i\}$.
      ii. Initialize $\mathscr{O}_i$ to $\mathscr{O}_{i-1}$, $\mathscr{P}_i$ to $\mathscr{P}_{i-1}$, and $\mathscr{S}_i$ to $\mathscr{S}_{i-1}$.
      iii. For each $B \in \mathscr{O}_{i-1}$ such that $C_i \subseteq B$ and $|N(C_i) \cup N(B)| \leq k+1$, let $K = N(C_i) \cup N(B)$ and do the following.
         A. If K is a potential maximal clique, then add K to $\mathscr{P}_i$.
         B. If $|K| \leq k$ and there is a full component A associated with K (which is unique), then add A to $\mathscr{O}_i$.
      iv. Let A be the full component associated with $N(C_i)$ and add A to $\mathscr{O}_i$.
      v. For each $A \in \mathscr{O}_i \setminus \mathscr{O}_{i-1}$ and $v \in N(A)$, let $K = N(A) \cup (n(v) \cap A)$ and if $|K| \leq k+1$ and K is a potential maximal clique then add K to $\mathscr{P}_i$.
      vi. For each $K \in \mathscr{P}_i \setminus \mathscr{S}_{i-1}$, if support$(K) \subseteq \mathscr{I}_i$ then add K to $\mathscr{S}_i$ and do the following: if outlet$(K) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(K), K)$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
7. If there is some $K \in \mathscr{S}_j$ such that outlet$(K) = \emptyset$, then answer "YES"; otherwise, answer "NO".

You're right, the algorithm is messy!

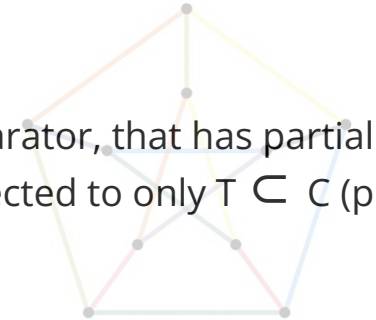
I-Blocks & O-Blocks



- Observations:
  - For each separator, total ordering of vertex separates the associated partition of V(G)
  - **Inbound** Partition:
    - If some connected set C has a neighbour set N(C) preceding C
  - Otherwise, **outbound** partition
- Facts:
  - For any inbound vertex set C, neighbour of C, N(C) is a **minimal separator**
- Formally,
  - A pair of (N(C), C) defined as a full-block is an **I-block** if C is inbound and |N(C)| <= k
  - A pair of (N(C), C) defined as a full-block is an **O-block** if C is outbound and |N(C)| <= k

Feasible Potential Maximal Clique

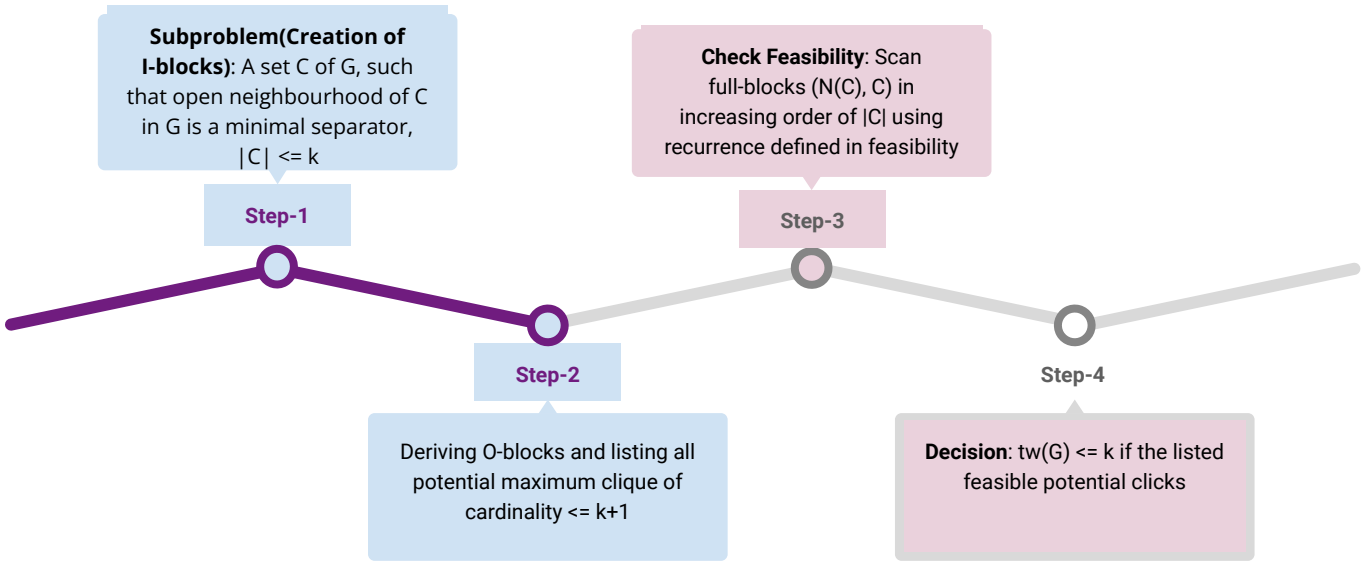
- Define G[C], C $\subset$ V(G), as a graph from taking the closed neighbourhood of C and completing N(C), neighbours of C a clique
- **Feasible**:
  - A vertex set C is feasible if treewidth(G[C]) <= k
- Support(C):
  - The set of partitions arising from maintaining C as a separator, that has partial coverage
  - That is, some subset of the separated vertex set is connected to only T $\subset$ C (proper subset of C)
- A Potential Maximal Clique Ω is feasible if,
  - $|\Omega|$ <= k+1
  - For every **C ϵ support(Ω)** is feasible



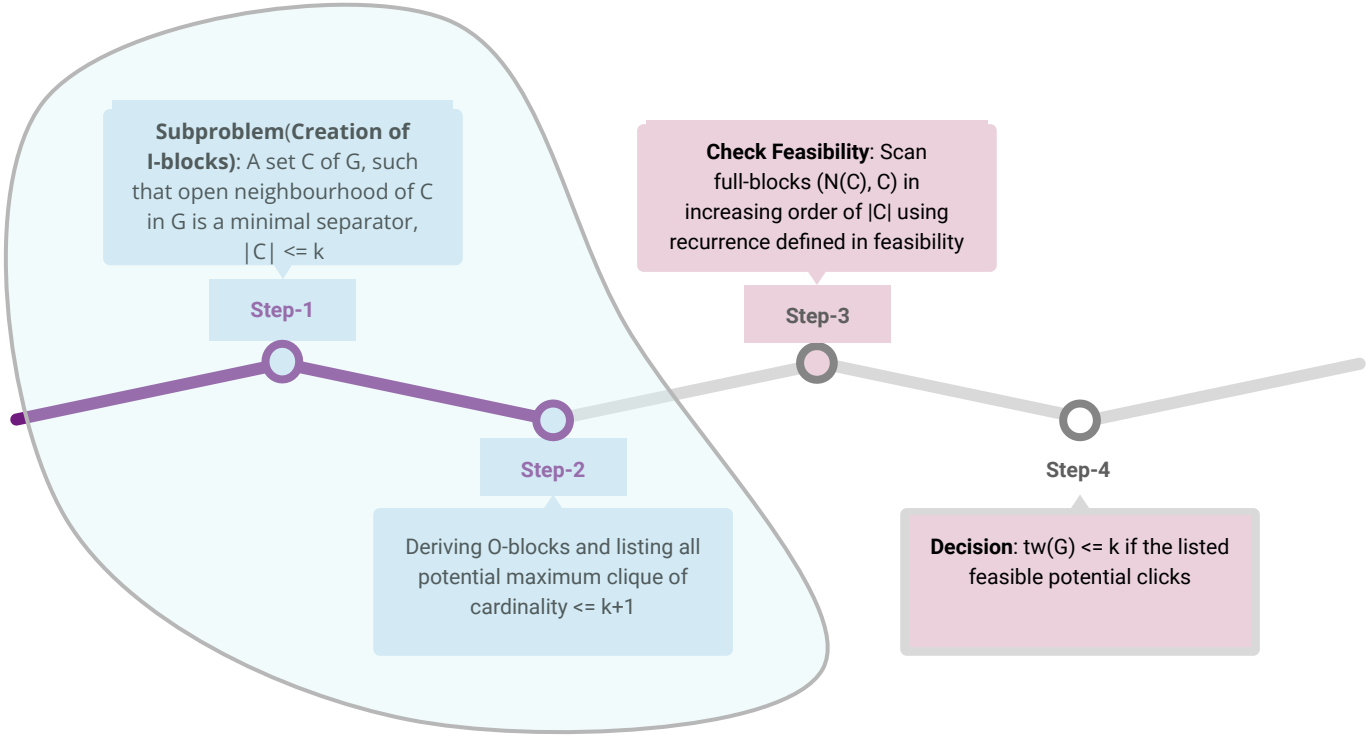
Theorem

The treewidth of a given graph G(V,E), $tw(G) \leq k$ **if and only if** G has a feasible potential maximal clique Ω with non-full associated outbound component (all components associated are feasible).

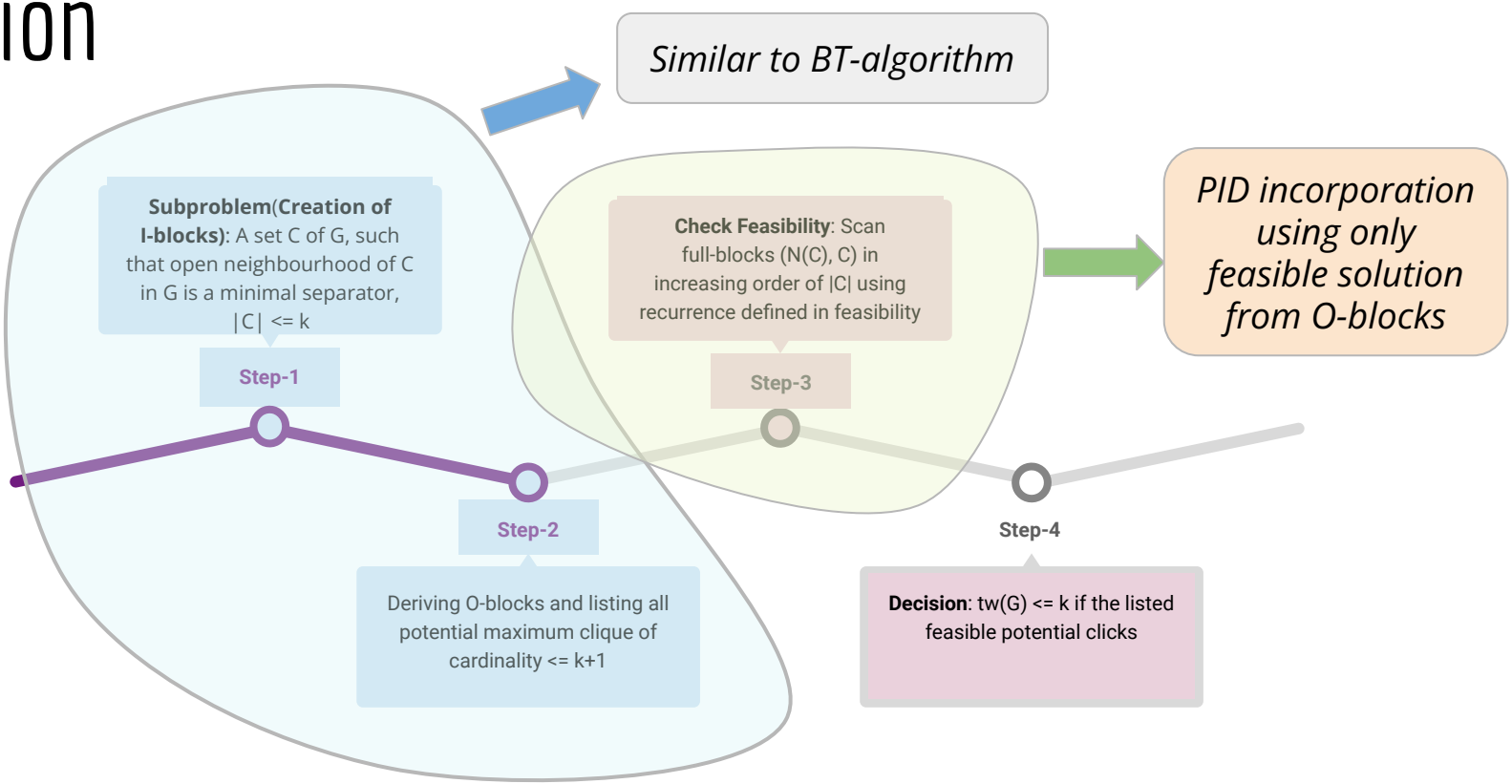
Breaking Down the Algorithm



Intuition


Subproblem(Creation of I-blocks): A set C of G, such that open neighbourhood of C in G is a minimal separator, |C| <= k
Step-1
Step-2
Deriving O-blocks and listing all potential maximum clique of cardinality <= k+1
Check Feasibility: Scan full-blocks (N(C), C) in increasing order of |C| using recurrence defined in feasibility
Step-3
Step-4
Decision: tw(G) <= k if the listed feasible potential clicks

Intuition


Similar to BT-algorithm
Subproblem(Creation of I-blocks): A set C of G, such that open neighbourhood of C in G is a minimal separator, |C| <= k
Step-1
Step-2
Deriving O-blocks and listing all potential maximum clique of cardinality <= k+1
Check Feasibility: Scan full-blocks (N(C), C) in increasing order of |C| using recurrence defined in feasibility
Step-3
Step-4
Decision: tw(G) <= k if the listed feasible potential clicks
PID incorporation using only feasible solution from O-blocks

PID-BT Algorithm

1. Let $\mathscr{I}_0 = \emptyset$ and $\mathscr{O}_0 = \emptyset$.
2. Initialize $\mathscr{P}_0$ and $\mathscr{S}_0$ to $\emptyset$.
3. Set $j = 0$.
4. For each $v \in V(G)$, if $N[v]$ is a potential maximal clique with $|N[v]| \leq k+1$ then add $N[v]$ to $\mathscr{P}_0$ and if, moreover, support$(N[v]) = \emptyset$ then do the following.
   (a) Add $N[v]$ to $\mathscr{S}_0$.
   (b) If outlet$(N[v]) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(N[v]), N[v])$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
5. Set $i = 0$.
6. Repeat the following and stop repetition when j is not incremented during the iteration step.
   (a) While $i < j$, do the following.
      i. Increment i and let $\mathscr{I}_i$ be $\mathscr{I}_{i-1} \cup \{C_i\}$.
      ii. Initialize $\mathscr{O}_i$ to $\mathscr{O}_{i-1}$, $\mathscr{P}_i$ to $\mathscr{P}_{i-1}$, and $\mathscr{S}_i$ to $\mathscr{S}_{i-1}$.
      iii. For each $B \in \mathscr{O}_{i-1}$ such that $C_i \subseteq B$ and $|N(C_i) \cup N(B)| \leq k+1$, let $K = N(C_i) \cup N(B)$ and do the following.
         A. If K is a potential maximal clique, then add K to $\mathscr{P}_i$.
         B. If $|K| \leq k$ and there is a full component A associated with K (which is unique), then add A to $\mathscr{O}_i$.
      iv. Let A be the full component associated with $N(C_i)$ and add A to $\mathscr{O}_i$.
      v. For each $A \in \mathscr{O}_i \setminus \mathscr{O}_{i-1}$ and $v \in N(A)$, let $K = N(A) \cup (n(v) \cap A)$ and if $|K| \leq k+1$ and K is a potential maximal clique then add K to $\mathscr{P}_i$.
      vi. For each $K \in \mathscr{P}_i \setminus \mathscr{S}_{i-1}$, if support$(K) \subseteq \mathscr{I}_i$ then add K to $\mathscr{S}_i$ and do the following: if outlet$(K) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(K), K)$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
7. If there is some $K \in \mathscr{S}_j$ such that outlet$(K) = \emptyset$, then answer "YES"; otherwise, answer "NO".

Algorithm Outline

Creation of I-blocks

⇩

Deriving O-blocks & listing potential maximal clicks

⇩

Deriving Feasible Potential Maximal clique using bottom to top approach

⇩

Applying the theorem for decision

PID-BT Algorithm

1. Let $\mathscr{I}_0 = \emptyset$ and $\mathscr{O}_0 = \emptyset$.
2. Initialize $\mathscr{P}_0$ and $\mathscr{S}_0$ to $\emptyset$.
3. Set $j = 0$.
4. For each $v \in V(G)$, if $N[v]$ is a potential maximal clique with $|N[v]| \leq k+1$ then add $N[v]$ to $\mathscr{P}_0$ and if, moreover, support$(N[v]) = \emptyset$ then do the following.
   (a) Add $N[v]$ to $\mathscr{S}_0$.
   (b) If outlet$(N[v]) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(N[v]), N[v])$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
5. Set $i = 0$.
6. Repeat the following and stop repetition when j is not incremented during the iteration step.
   (a) While $i < j$, do the following.
      i. Increment i and let $\mathscr{I}_i$ be $\mathscr{I}_{i-1} \cup \{C_i\}$.
      ii. Initialize $\mathscr{O}_i$ to $\mathscr{O}_{i-1}$, $\mathscr{P}_i$ to $\mathscr{P}_{i-1}$, and $\mathscr{S}_i$ to $\mathscr{S}_{i-1}$.
      iii. For each $B \in \mathscr{O}_{i-1}$ such that $C_i \subseteq B$ and $|N(C_i) \cup N(B)| \leq k+1$, let $K = N(C_i) \cup N(B)$ and do the following.
         A. If K is a potential maximal clique, then add K to $\mathscr{P}_i$.
         B. If $|K| \leq k$ and there is a full component A associated with K (which is unique), then add A to $\mathscr{O}_i$.
      iv. Let A be the full component associated with $N(C_i)$ and add A to $\mathscr{O}_i$.
      v. For each $A \in \mathscr{O}_i \setminus \mathscr{O}_{i-1}$ and $v \in N(A)$, let $K = N(A) \cup (n(v) \cap A)$ and if $|K| \leq k+1$ and K is a potential maximal clique then add K to $\mathscr{P}_i$.
      vi. For each $K \in \mathscr{P}_i \setminus \mathscr{S}_{i-1}$, if support$(K) \subseteq \mathscr{I}_i$ then add K to $\mathscr{S}_i$ and do the following: if outlet$(K) \neq \emptyset$ then let $C = \text{crib}(\text{outlet}(K), K)$ and, provided that $C \neq C_h$ for $1 \leq h \leq j$, increment j and let $C_j = C$.
7. If there is some $K \in \mathscr{S}_j$ such that outlet$(K) = \emptyset$, then answer "YES"; otherwise, answer "NO".

Algorithm Outline

step-1 → *Creation of I-blocks*

step-2 → *Deriving O-blocks & listing potential maximal clicks*

step-3 → *Deriving Feasible Potential Maximal clique using bottom to top approach*

step-4 → *Applying the theorem for decision*

That’s the Algorithm,

Easy Right (?!!)

Exact Implementation and Towards Heuristics

1805030 - Md Toki Tahmid

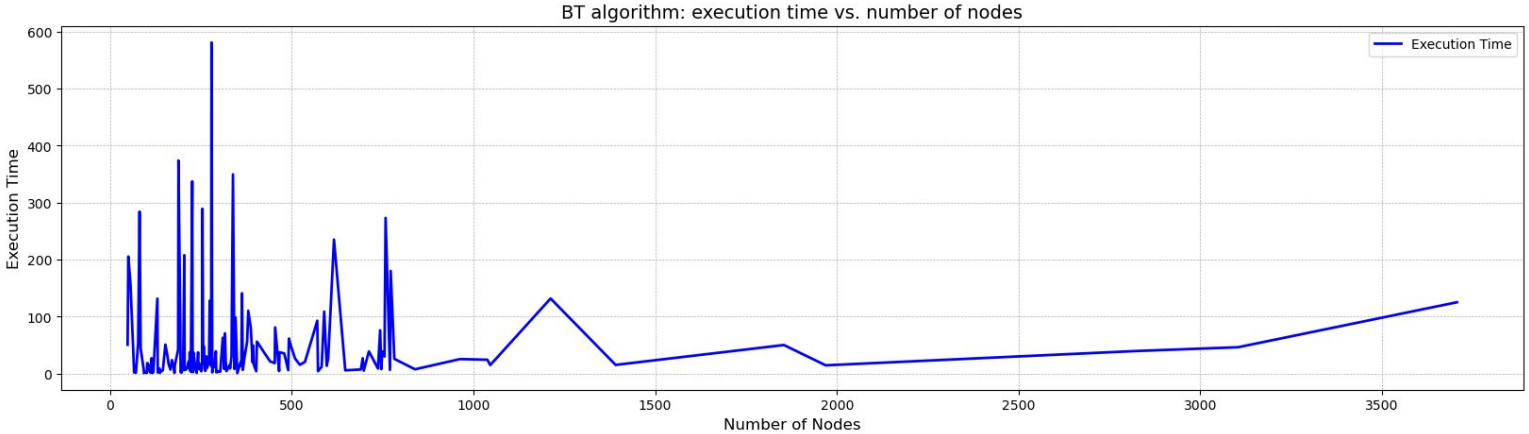
Analysis of BT-algorithm on Real Test Cases

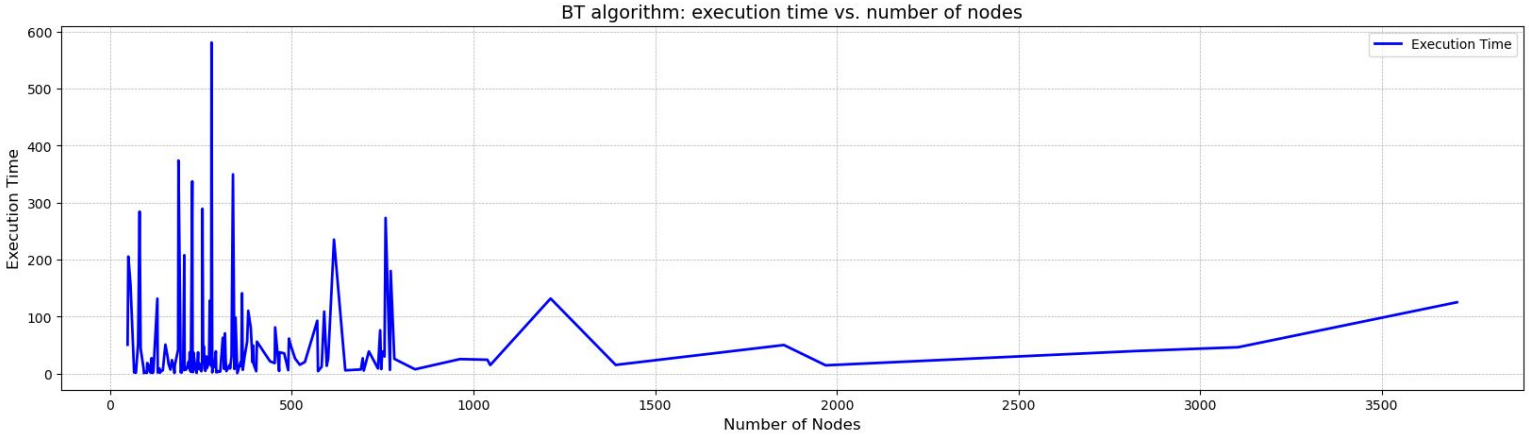
Dataset for the Exact Track

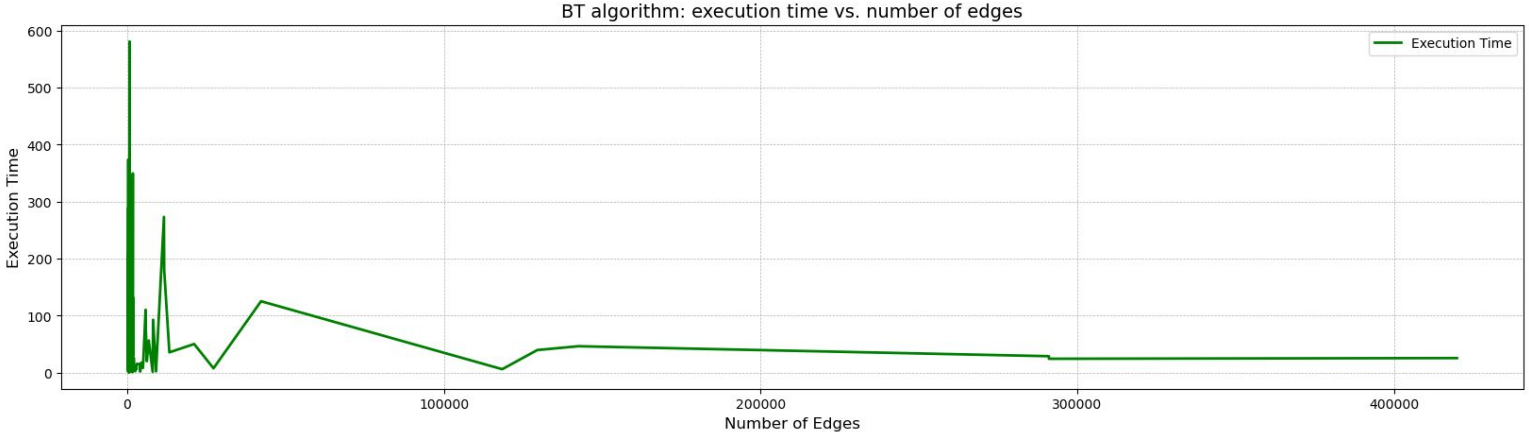
These instances were used in The Second Parameterized Algorithms and Computational Experiments Challenge (PACE 2017).

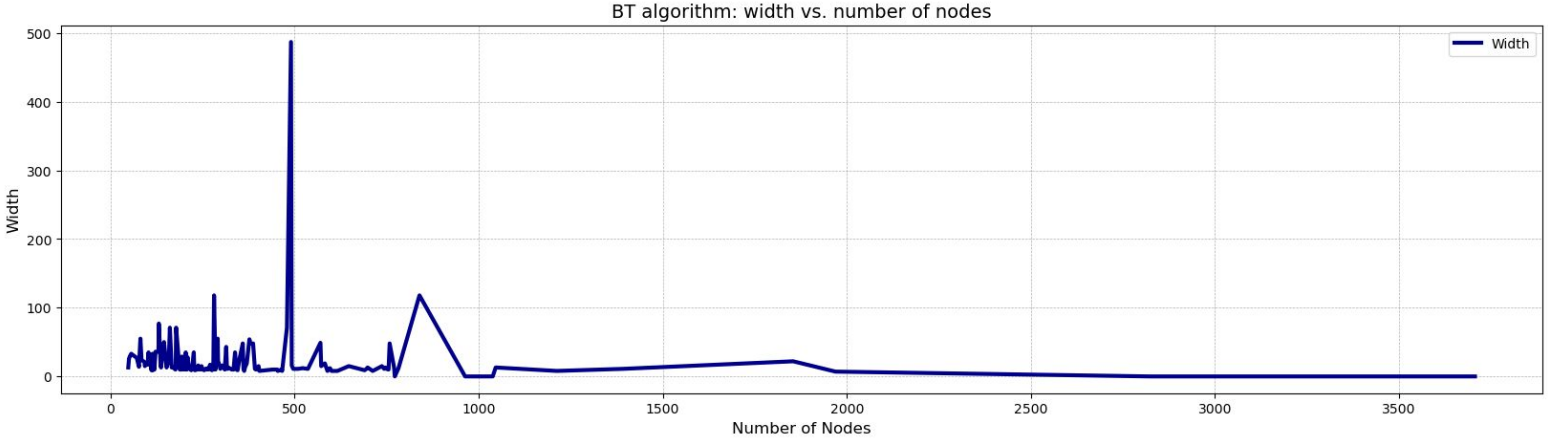
Half of the instances were made public before the challenge and the other half remained hidden until the conclusion of the challenge.

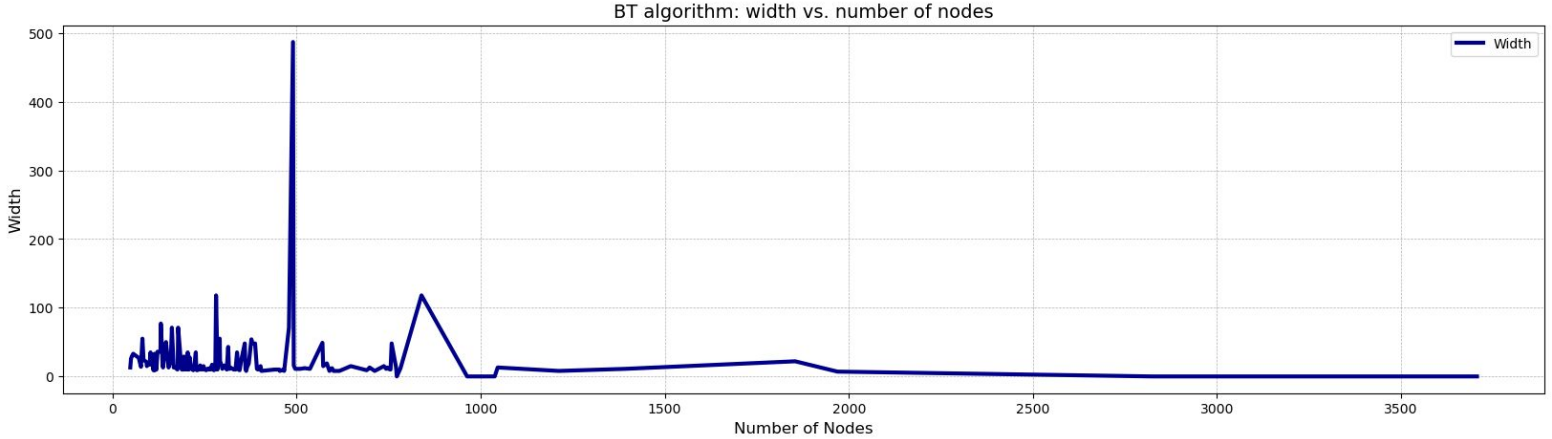
200 graphs for the exact track

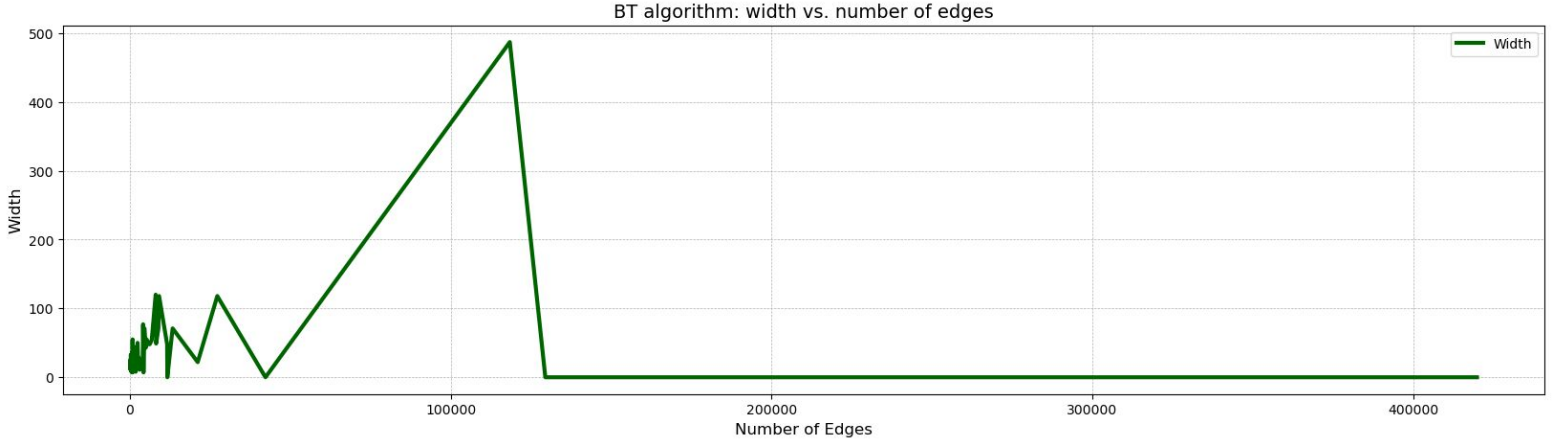

BT algorithm: execution time vs. number of nodes
Execution Time
Number of Nodes
Execution Time
0
100
200
300
400
500
600
0
500
1000
1500
2000
2500
3000
3500


BT algorithm: execution time vs. number of nodes
Execution Time
Number of Nodes
0
500
1000
1500
2000
2500
3000
3500
0
100
200
300
400
500
600


BT algorithm: execution time vs. number of edges
Execution Time
Number of Edges
0
100000
200000
300000
400000
0
100
200
300
400
500
600


BT algorithm: width vs. number of nodes
Width
Width
500
400
300
200
100
0
0
500
1000
1500
2000
2500
3000
3500
Number of Nodes


BT algorithm: width vs. number of nodes
Width
Width
Number of Nodes
0
100
200
300
400
500
0
500
1000
1500
2000
2500
3000
3500


BT algorithm: width vs. number of edges
Width
Width
Number of Edges
0
100
200
300
400
500
0
100000
200000
300000
400000

Min Degree Heuristics

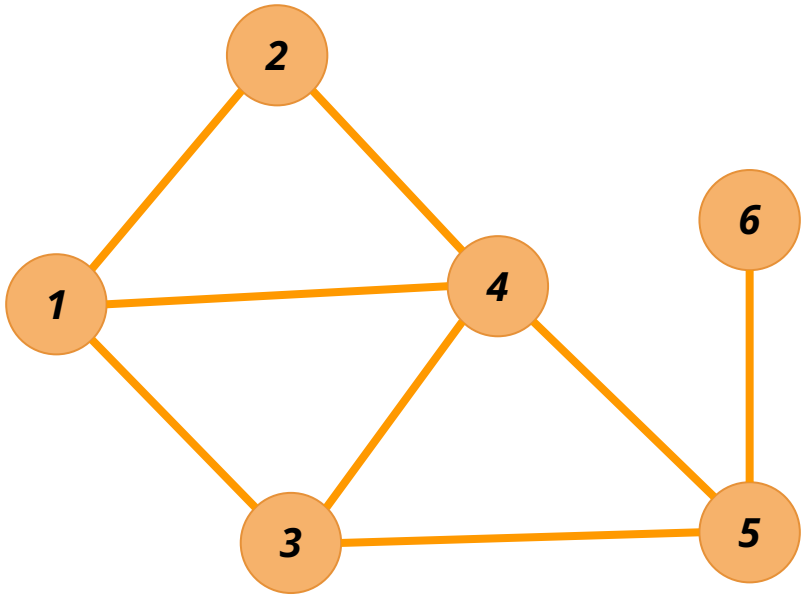
Algorithm

While the graph has more than one node do:

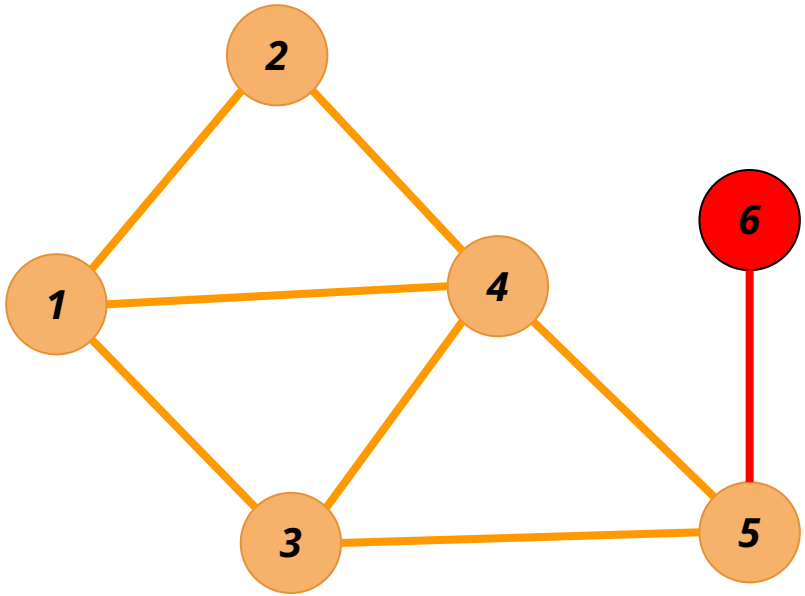
1. Select a node 'u' with the minimum degree in the graph.
2. Form a bag containing 'u' and its neighbors.
3. For each pair of neighbors 'v' and 'w' of 'u':
4. If 'v' and 'w' are not connected, add an edge between them.
5. Remove 'u' from the graph along with its incident edges.
6. Update the degrees of the remaining nodes.
7. If all remaining nodes form a clique, terminate the algorithm.

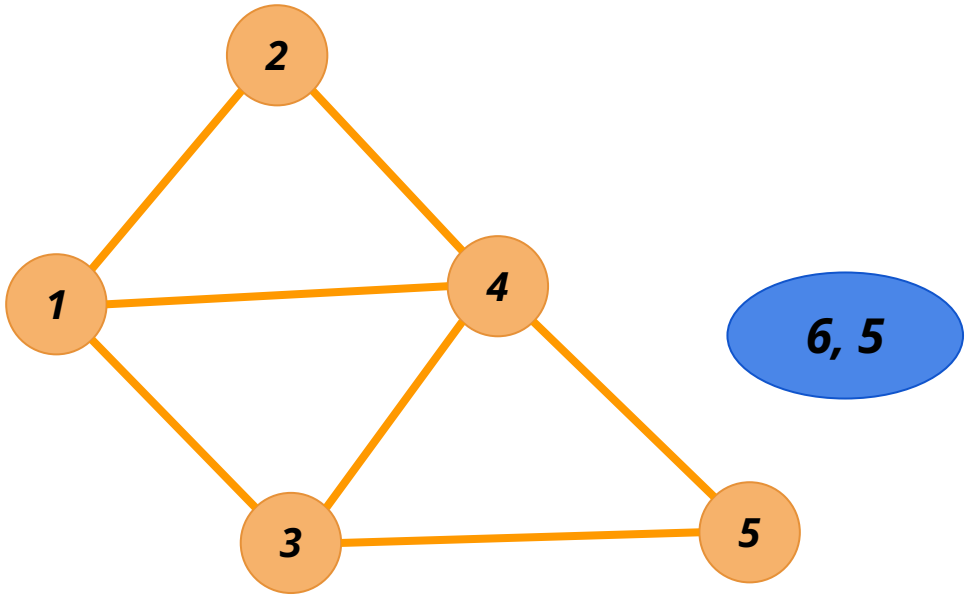
1. End While
2. Return the constructed tree decomposition.

Simulation

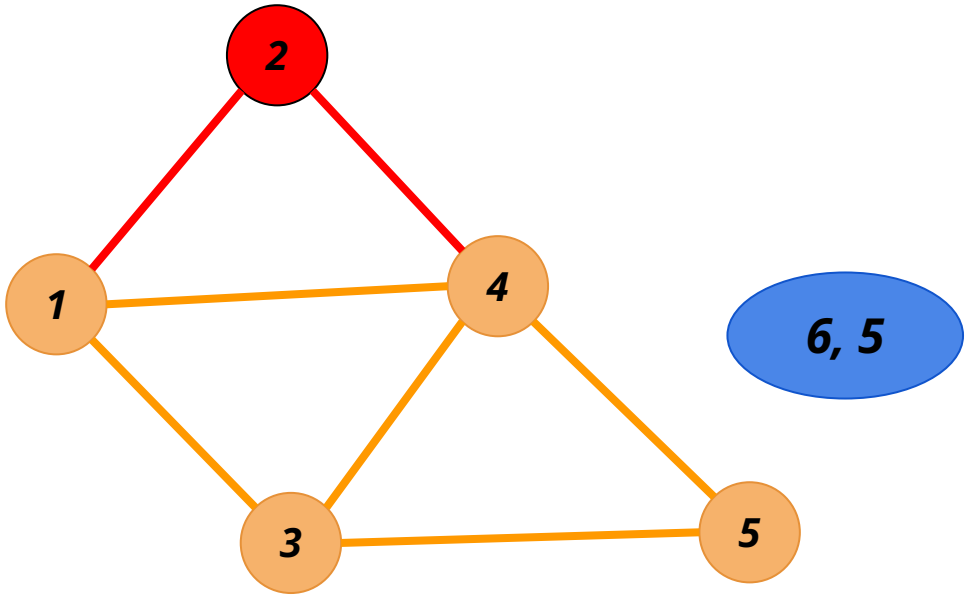


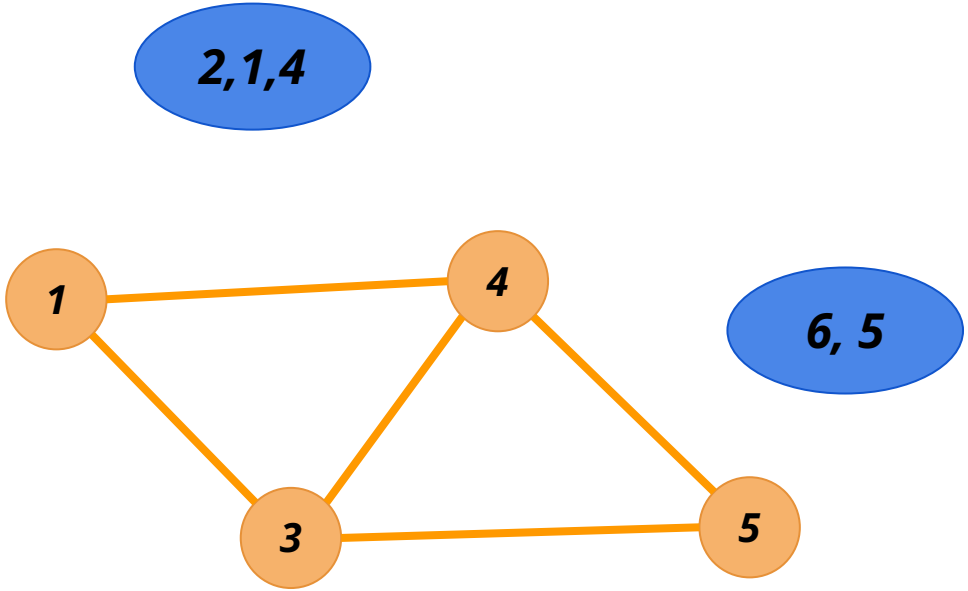
Min degree node: 6 —› (5)



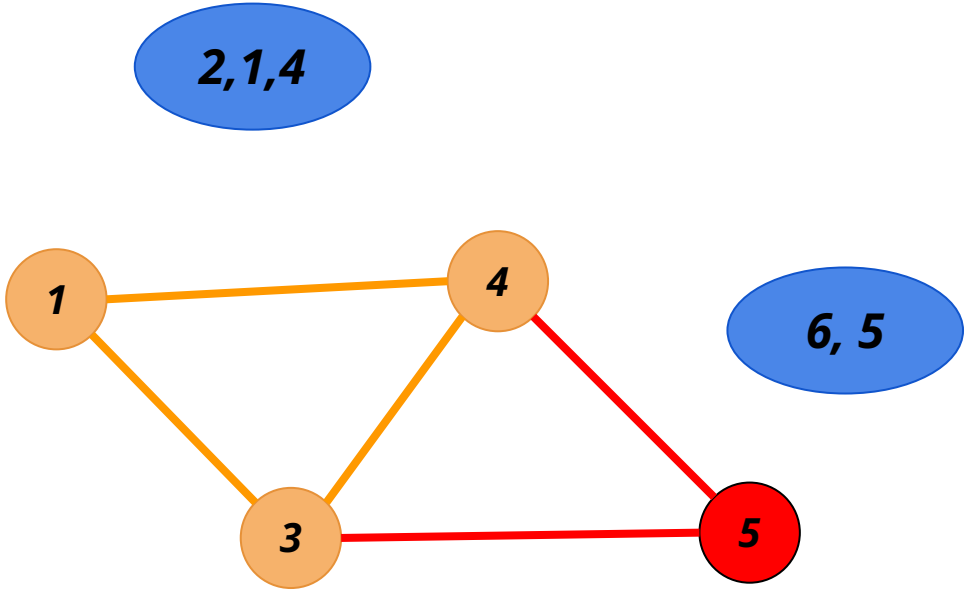

2
1
4
3
5
6, 5

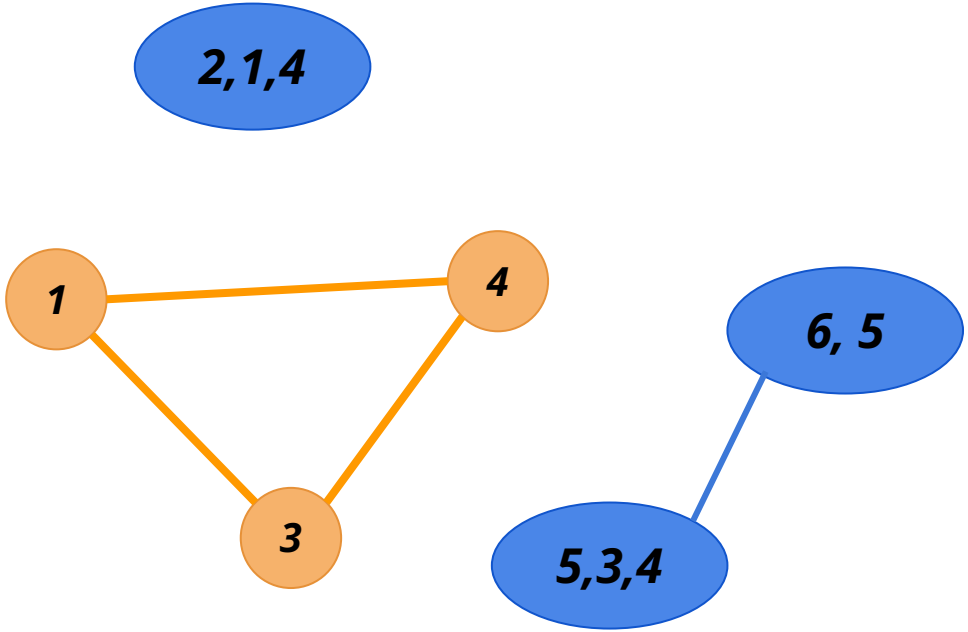
Min Degree Node: 2 ⟶ (1,4)



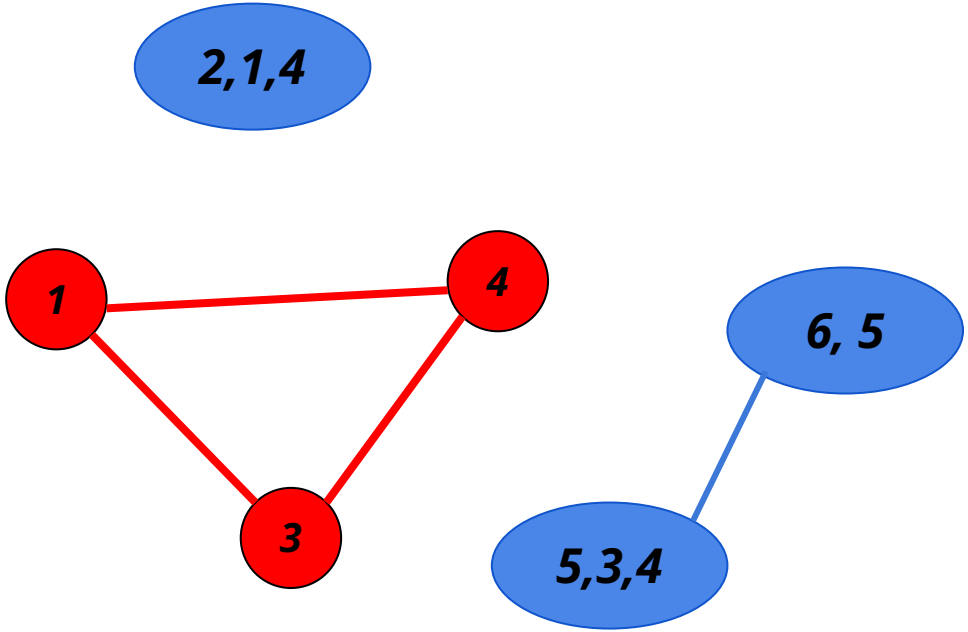

2,1,4
1
4
6, 5
3
5

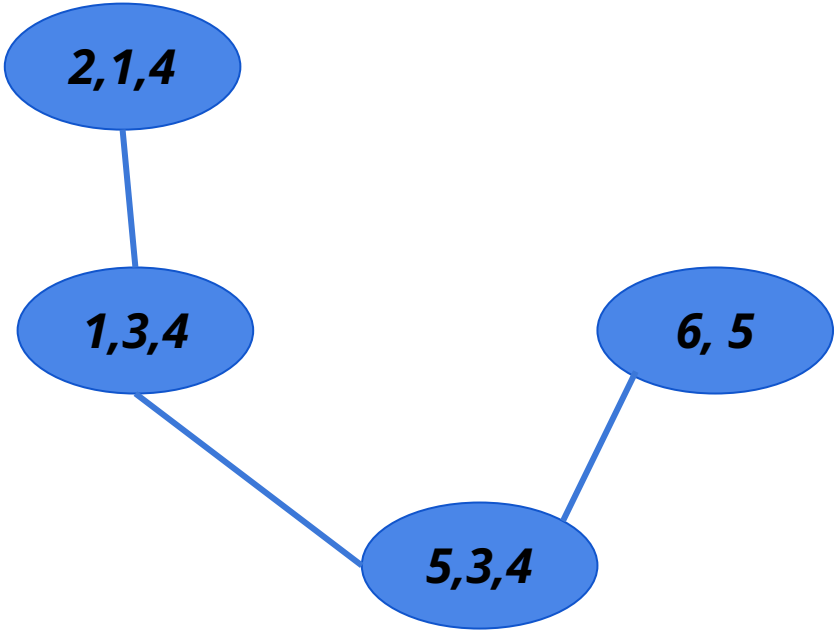
Min Degree Node : 5 ⟶ (3,4)



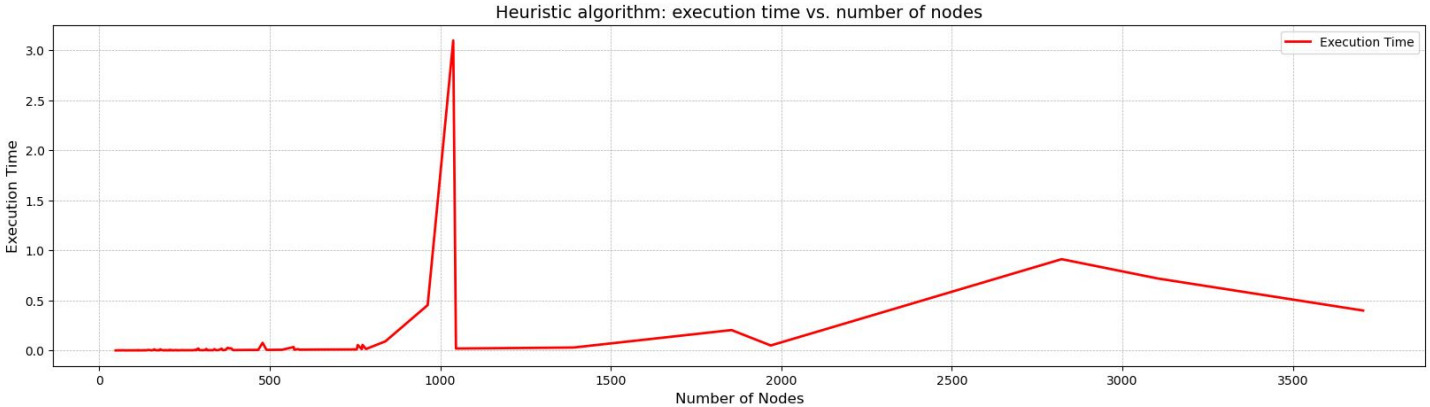

2,1,4
1
4
3
6, 5
5,3,4

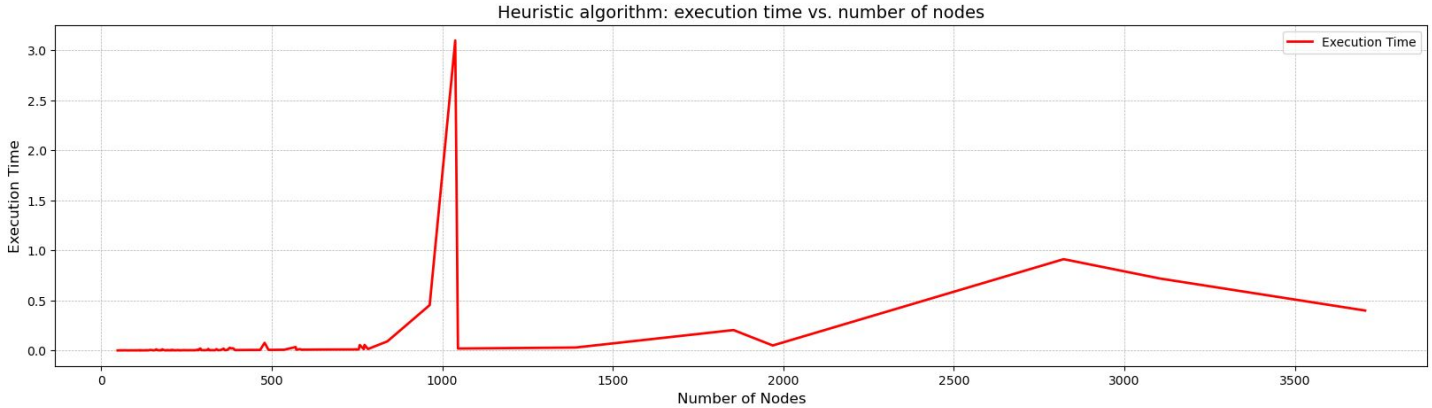
Termination : Complete Graph Found

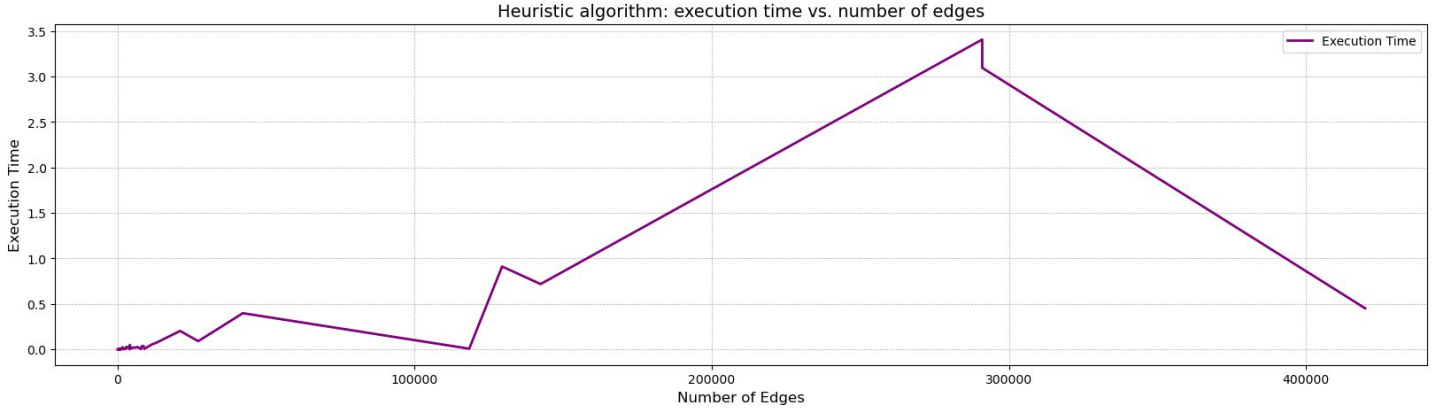


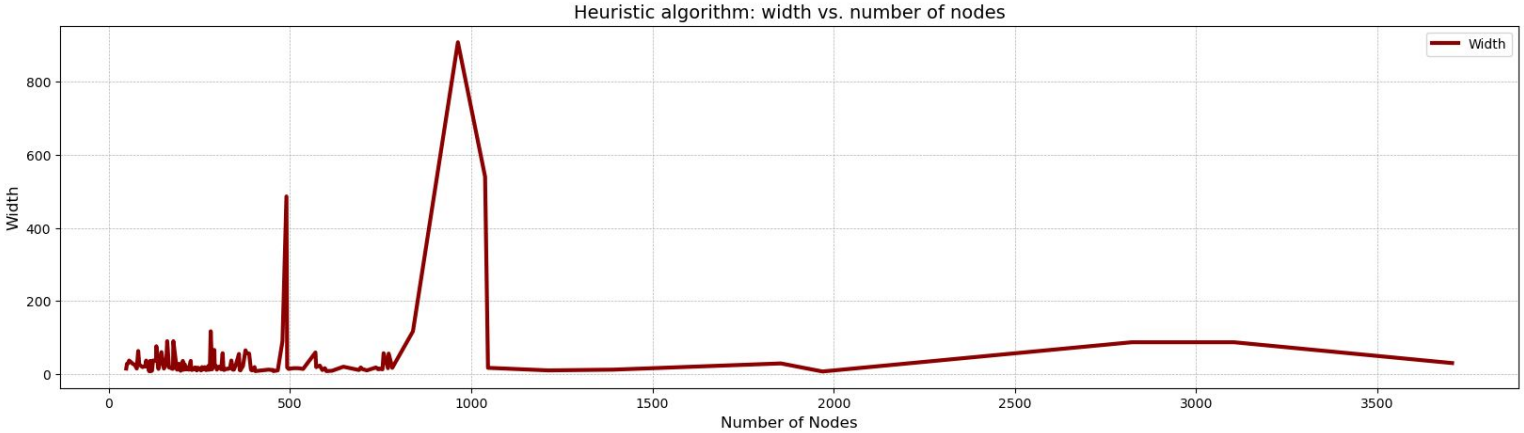

2,1,4
1,3,4
6, 5
5,3,4

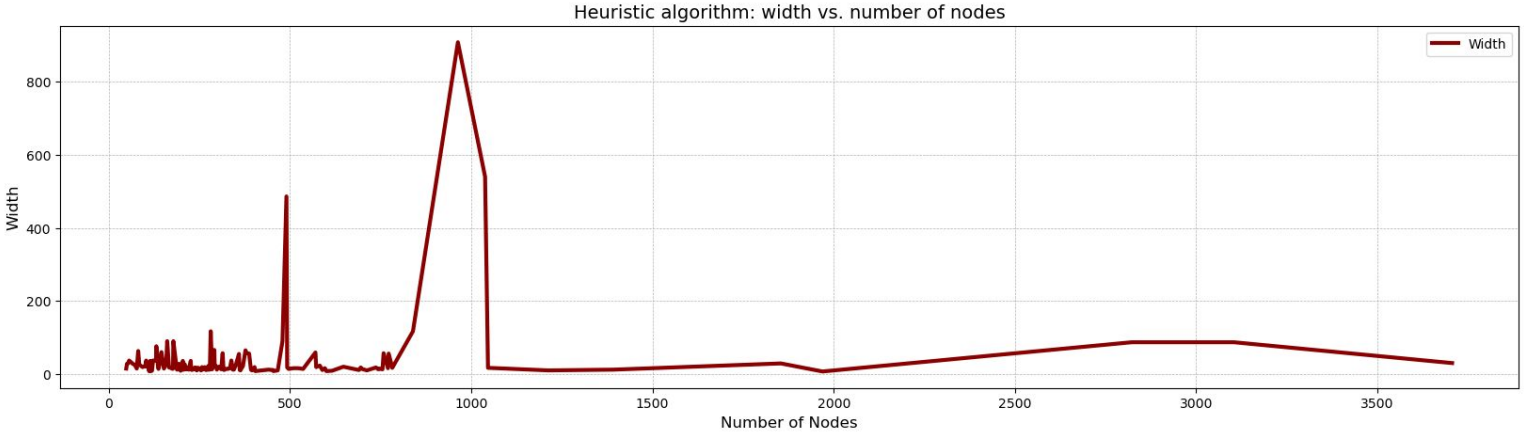
Analysis of Min Degree Heuristics

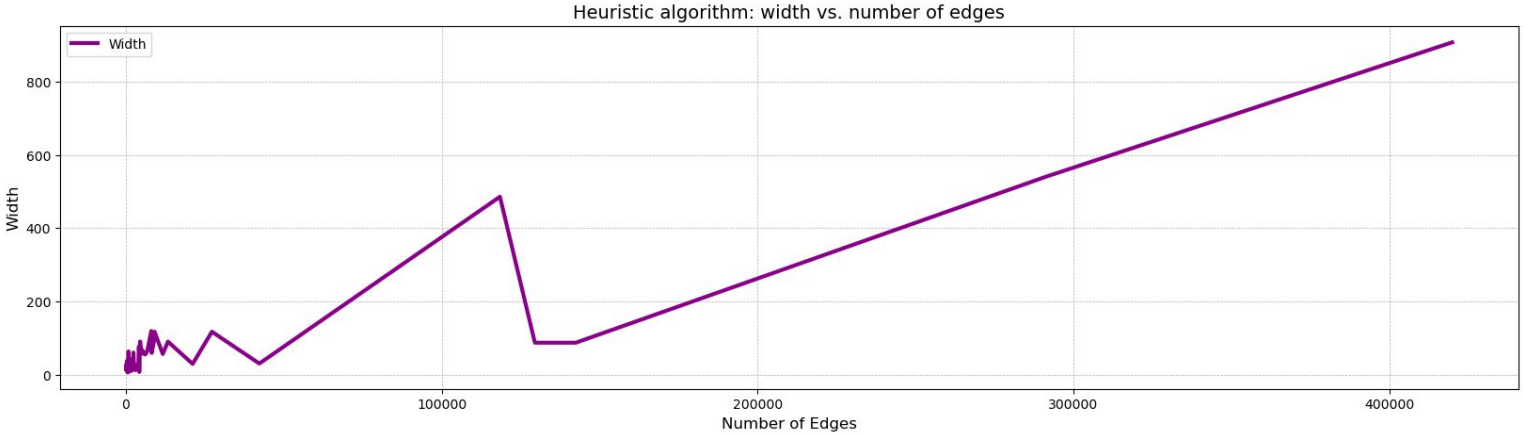

Heuristic algorithm: execution time vs. number of nodes
Execution Time
Execution Time
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000
3500
Number of Nodes


Heuristic algorithm: execution time vs. number of nodes
Execution Time
Number of Nodes
0
500
1000
1500
2000
2500
3000
3500
0.0
0.5
1.0
1.5
2.0
2.5
3.0

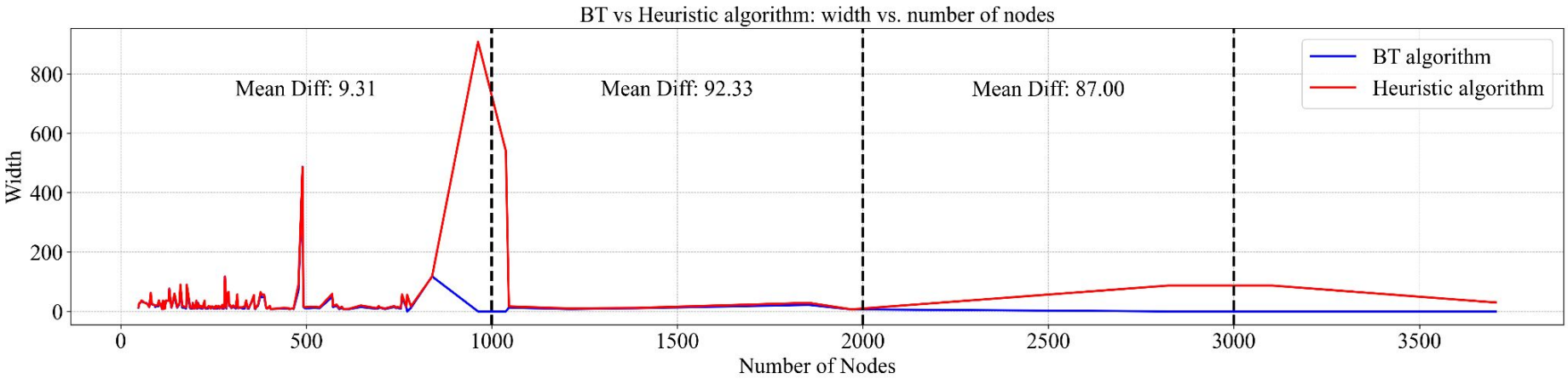

Heuristic algorithm: execution time vs. number of edges
Execution Time
Number of Edges
0
100000
200000
300000
400000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

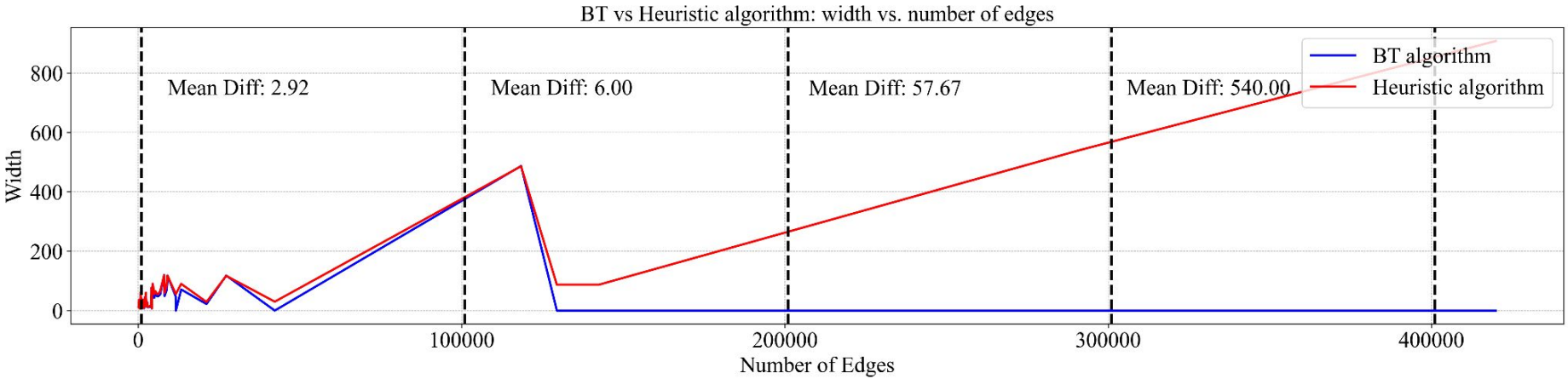

Heuristic algorithm: width vs. number of nodes
Width
Width
800
600
400
200
0
0
500
1000
1500
2000
2500
3000
3500
Number of Nodes


Heuristic algorithm: width vs. number of nodes
Width
Width
800
600
400
200
0
0
500
1000
1500
2000
2500
3000
3500
Number of Nodes


Heuristic algorithm: width vs. number of edges
Width
Width
800
600
400
200
0
0
100000
200000
300000
400000
Number of Edges

How Good It Performs?

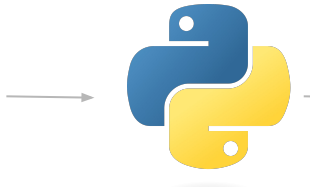

BT vs Heuristic algorithm: width vs. number of nodes
Mean Diff: 9.31
Mean Diff: 92.33
Mean Diff: 87.00
BT algorithm
Heuristic algorithm
Width
Number of Nodes
0
200
400
600
800
0
500
1000
1500
2000
2500
3000
3500

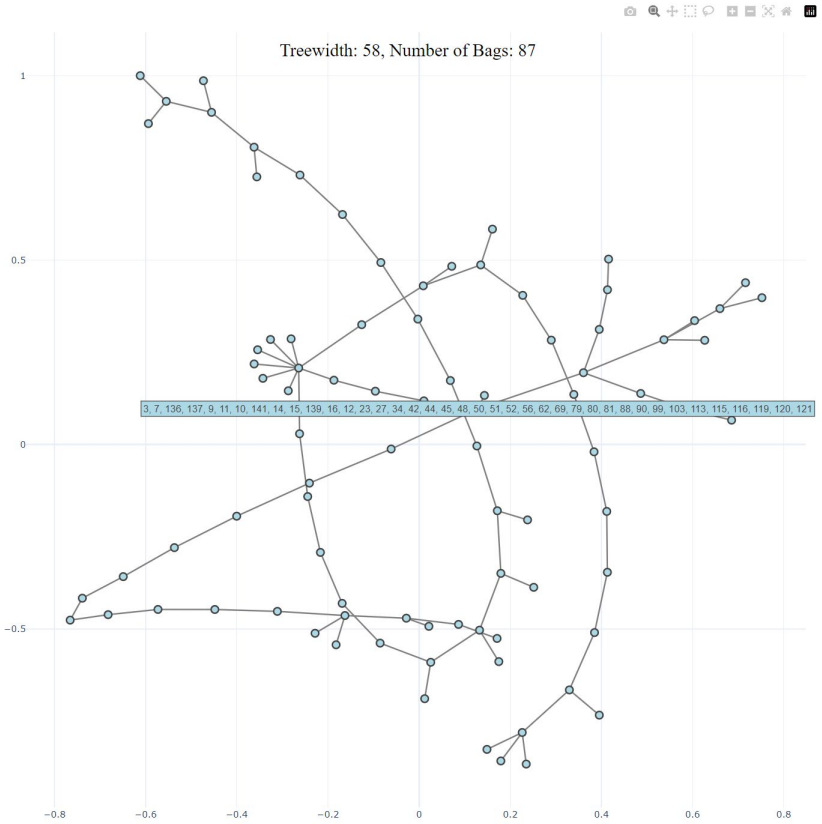

BT vs Heuristic algorithm: width vs. number of edges
Mean Diff: 2.92
Mean Diff: 6.00
Mean Diff: 57.67
Mean Diff: 540.00
BT algorithm
Heuristic algorithm
Width
Number of Edges
0
200
400
600
800
0
100000
200000
300000
400000

Simulation in Code

sample.gr

```
p tw 145 2368
26 129
128 129
26 128
5 126
4 126
126 132
51 126
68 126
12 126
13 126
3 126
56 126
23 126
118 126
126 127
```





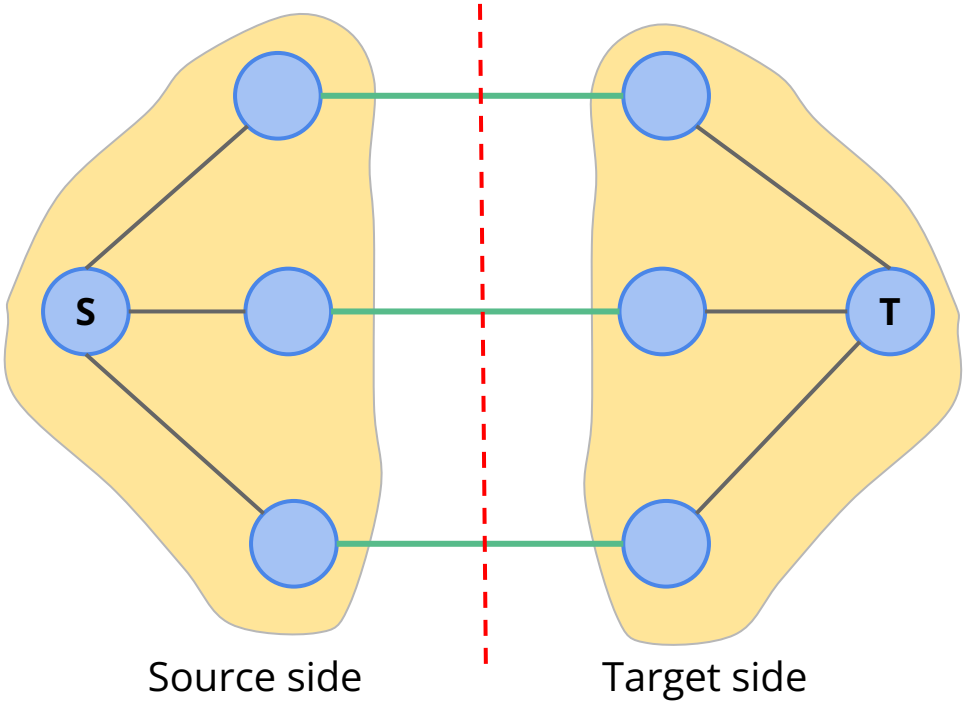
Flow-Cutter: Heuristic algorithm

1805008 - Abdur Rafi

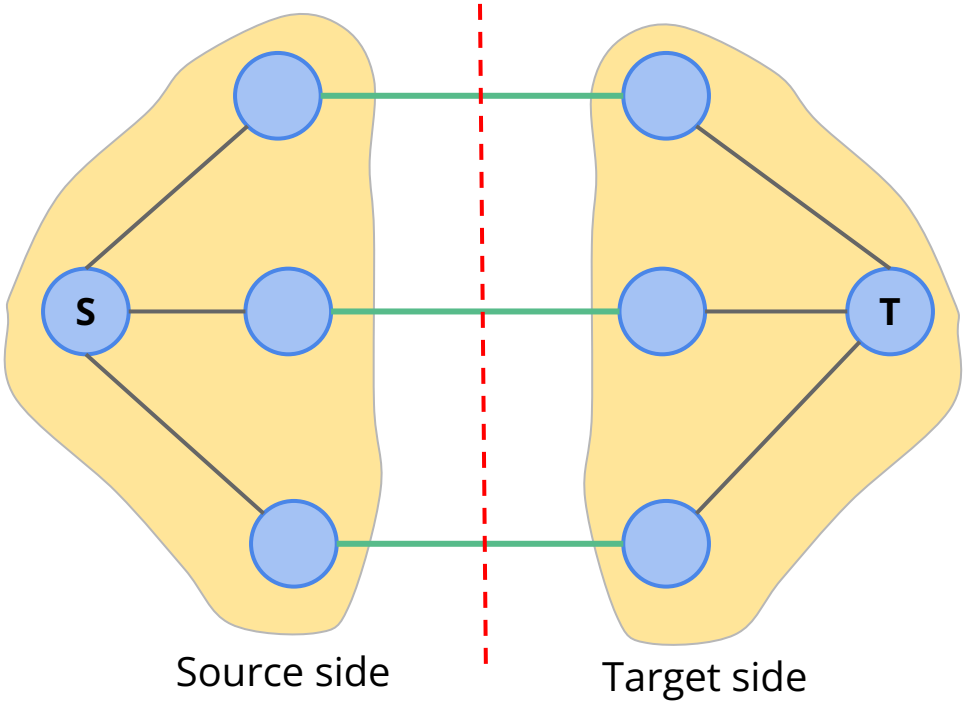
Core Flow-Cutter Algorithm

- A novel method to compute balanced st cuts with small 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

st-cut

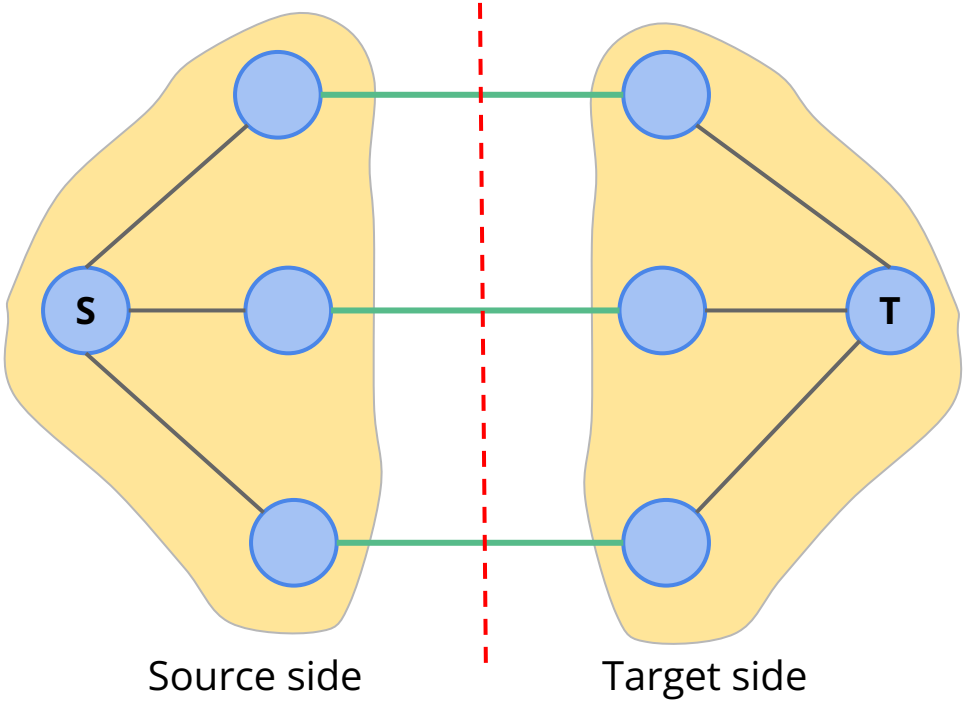


Balanced st-cut



Minimize |nodes in source side - nodes in target side|

Small balanced st-cut

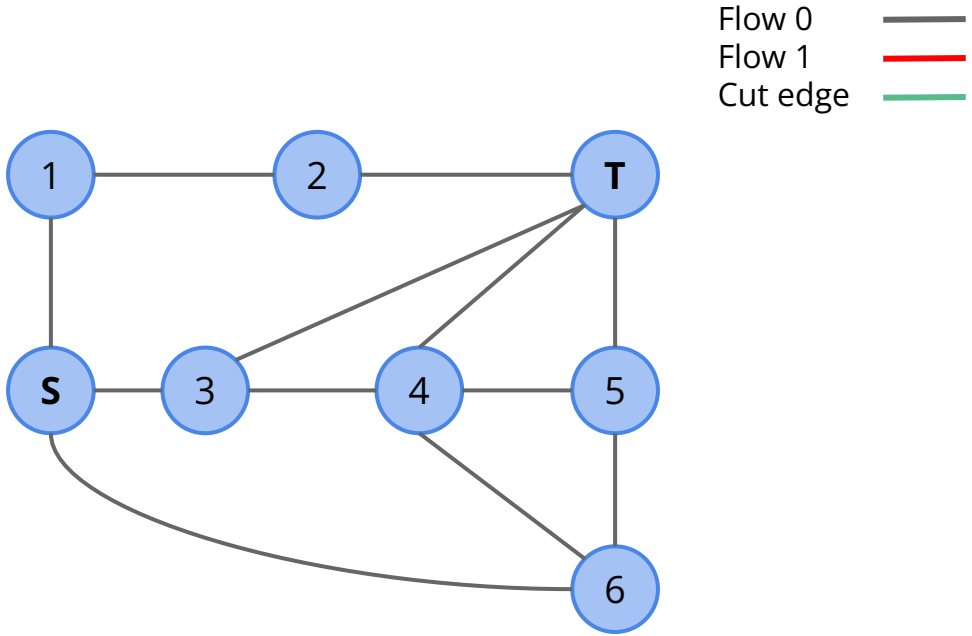


Minimize #cut edges

Finding balanced st-cut

Steps

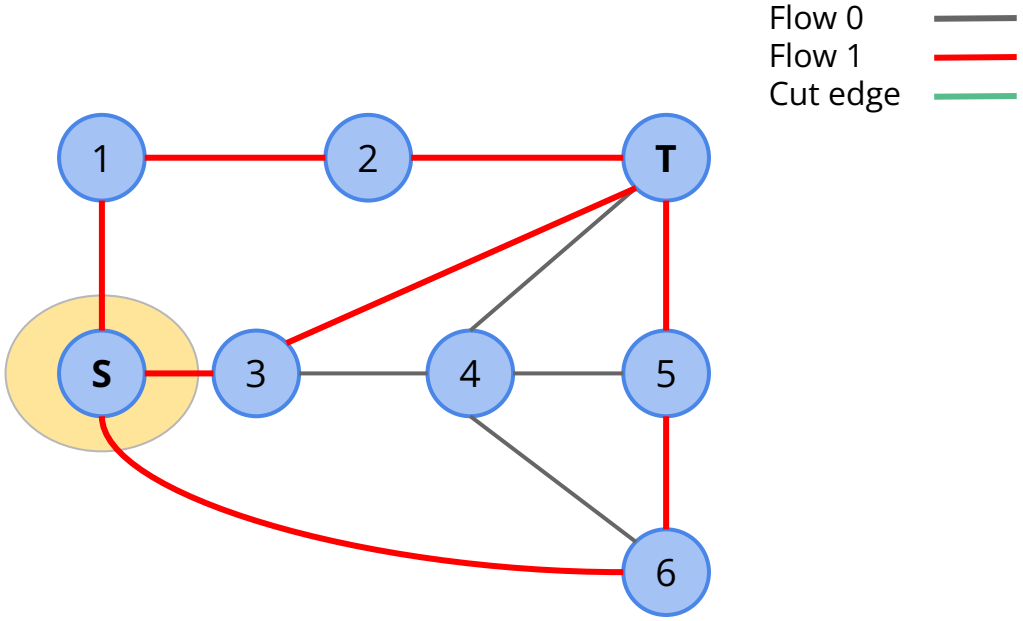
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

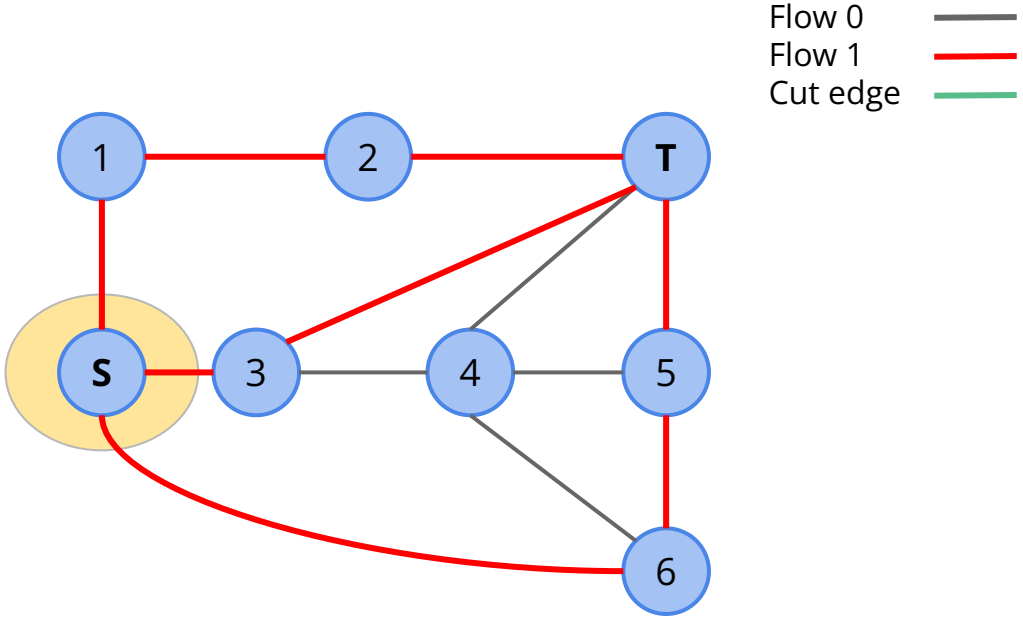
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

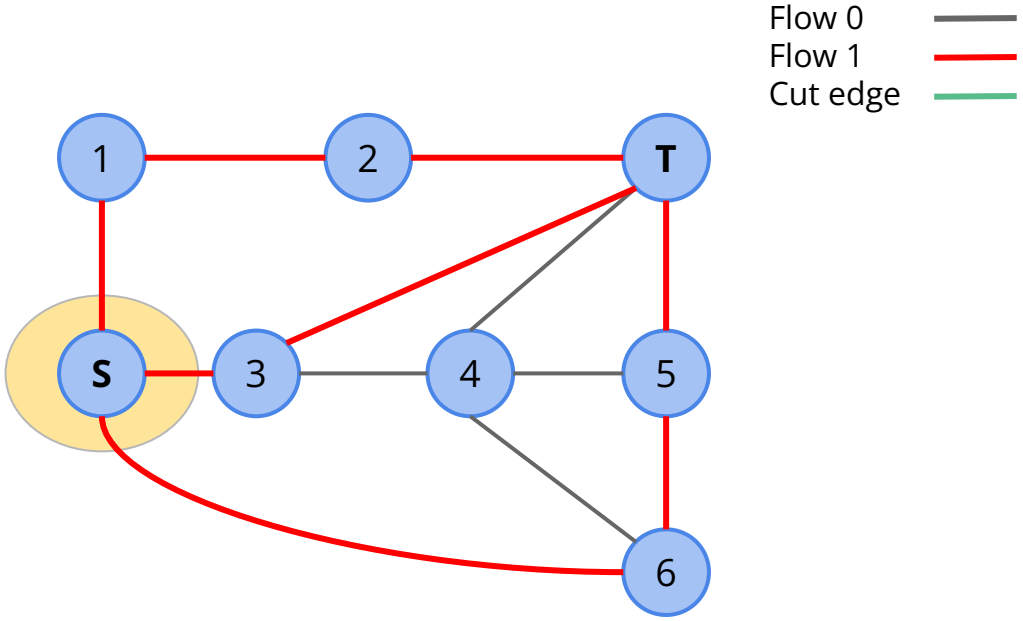
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

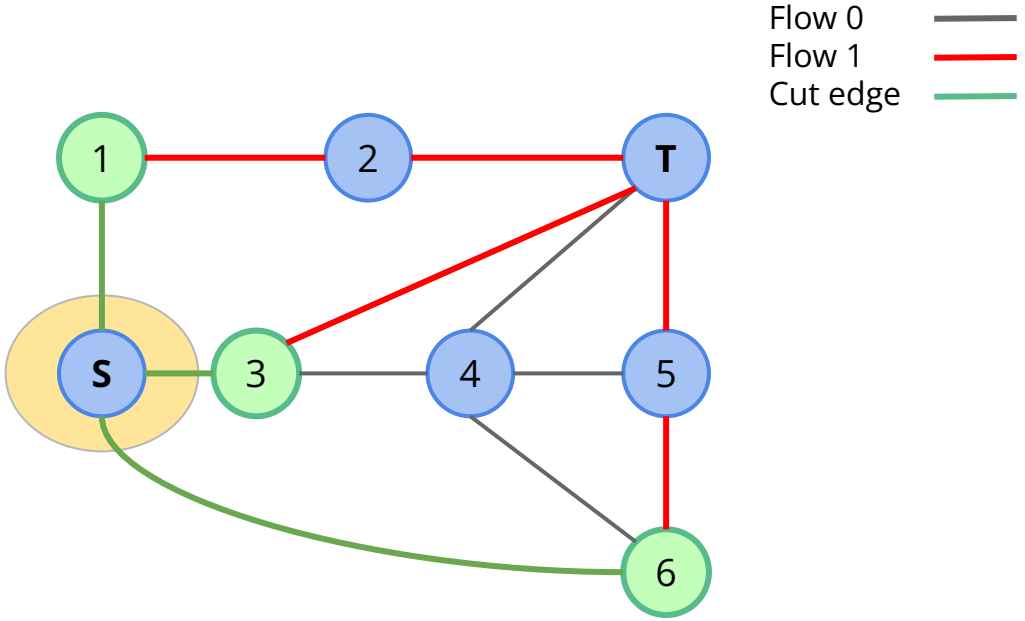
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

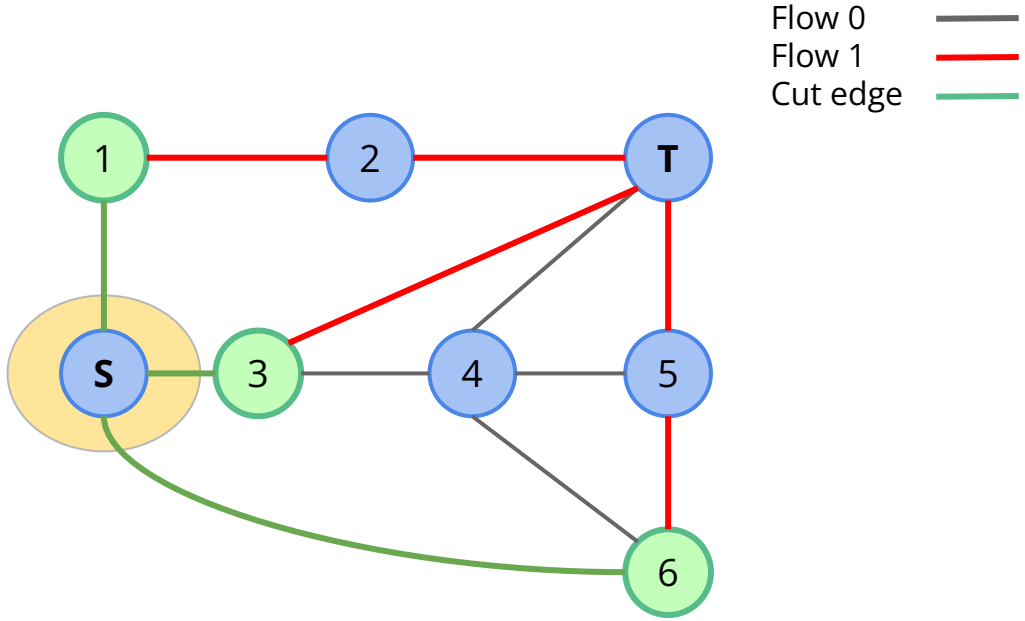
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1

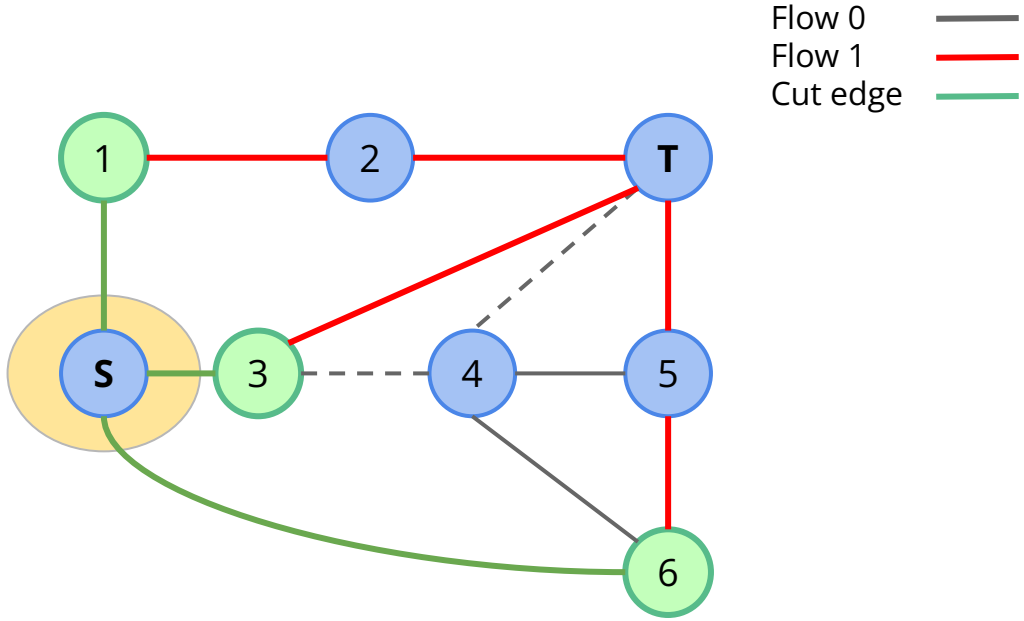


Primary Heuristic: Pick a node with no path of black edges to any target

Finding balanced st-cut

Steps

1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1

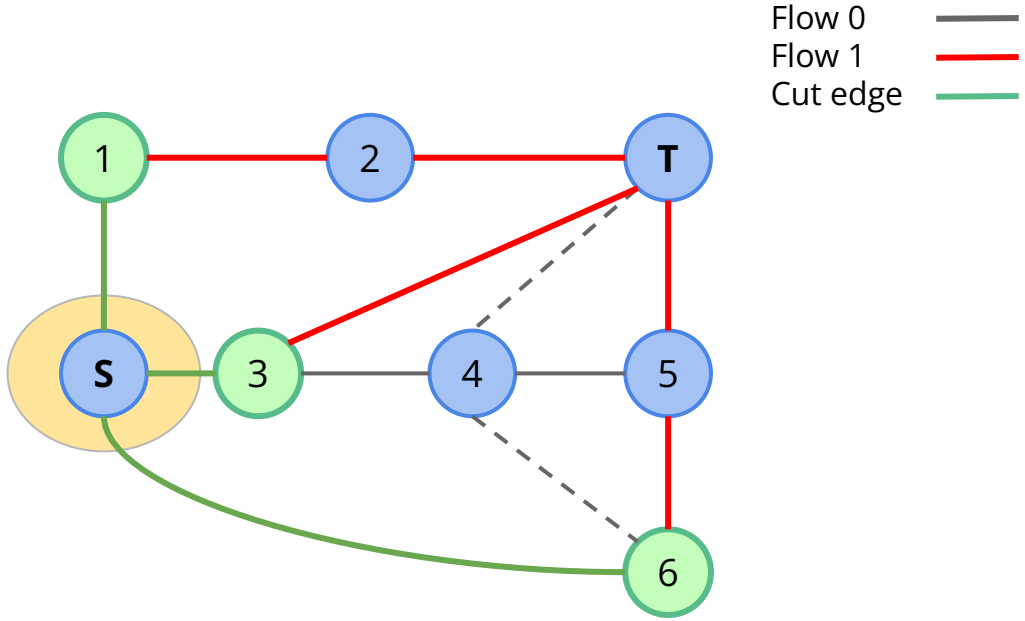


Primary Heuristic: Pick a node with no path of black edges to any target

Finding balanced st-cut

Steps

1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1

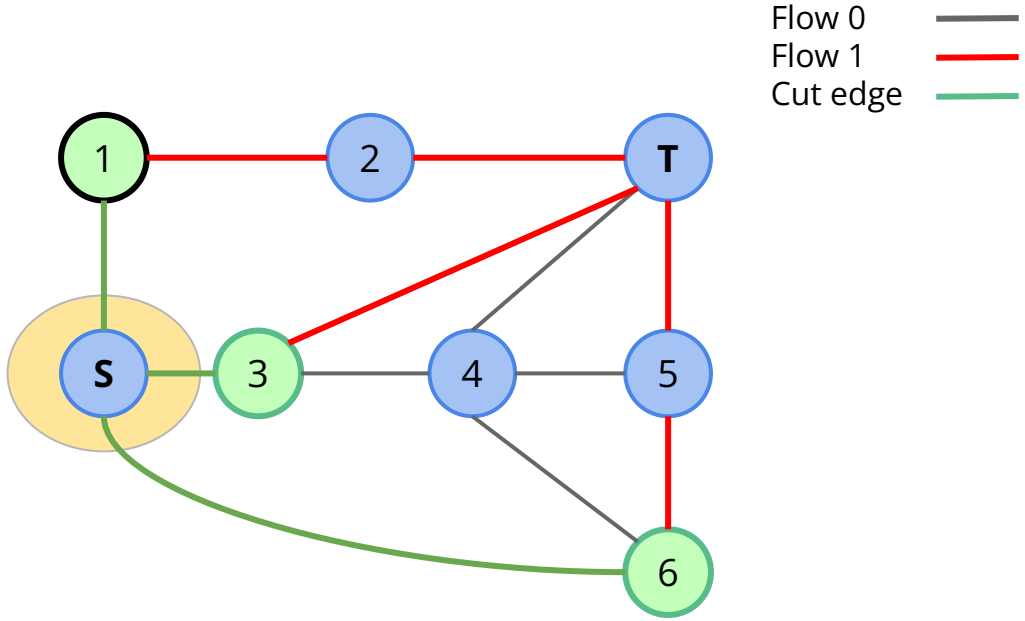


Primary Heuristic: Pick a node with no path of black edges to any target

Finding balanced st-cut

Steps

1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1

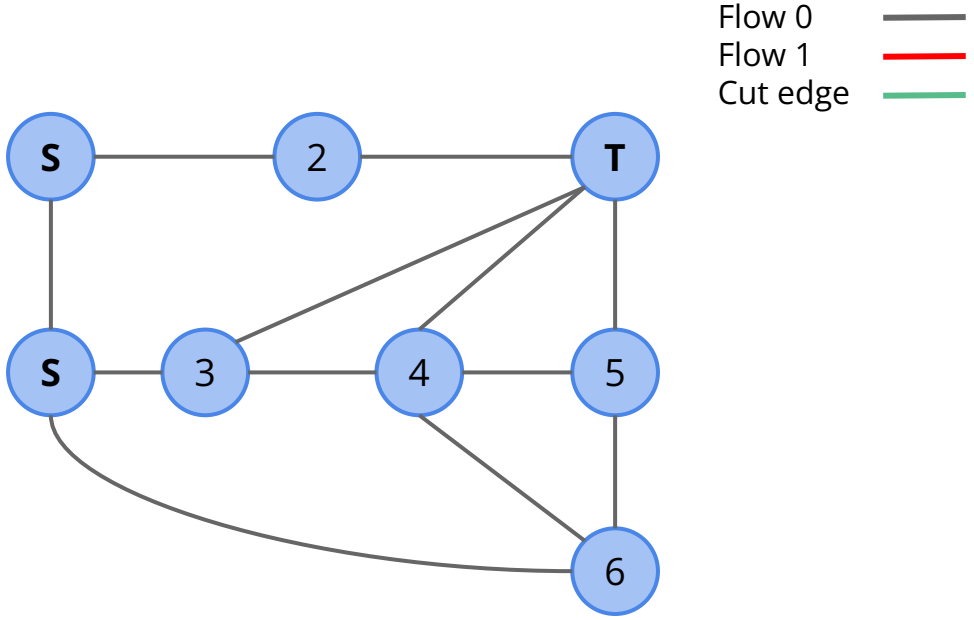


Primary Heuristic: Pick a node with no path of black edges to any target

Finding balanced st-cut

Steps

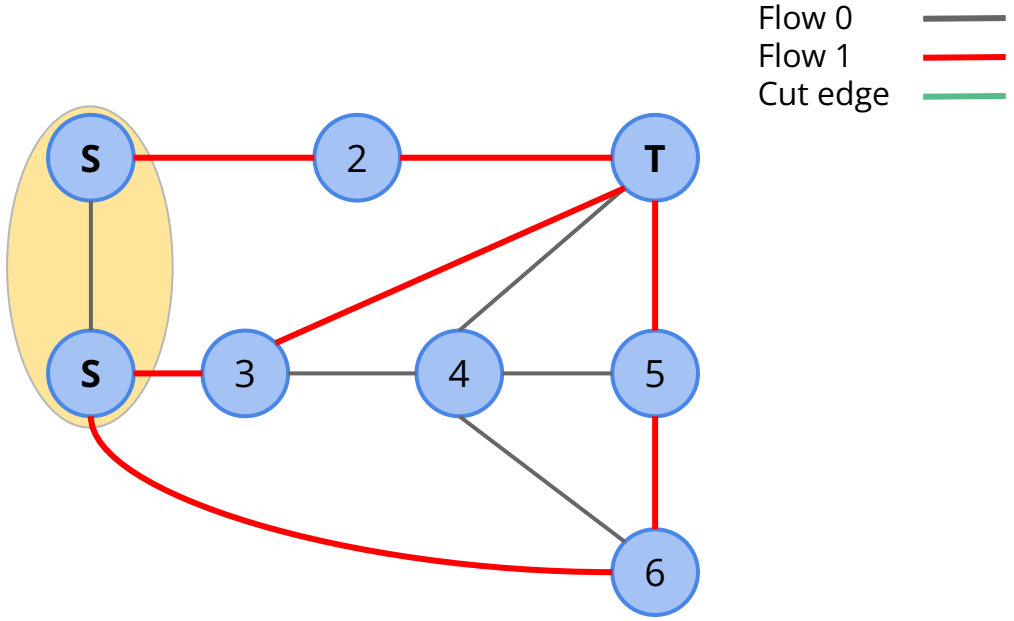
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

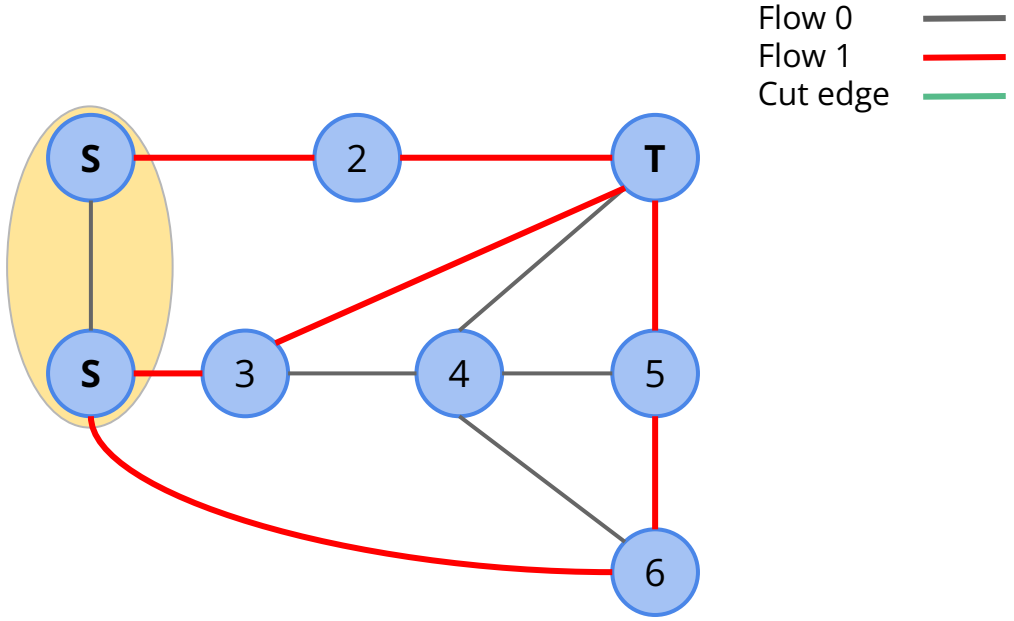
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

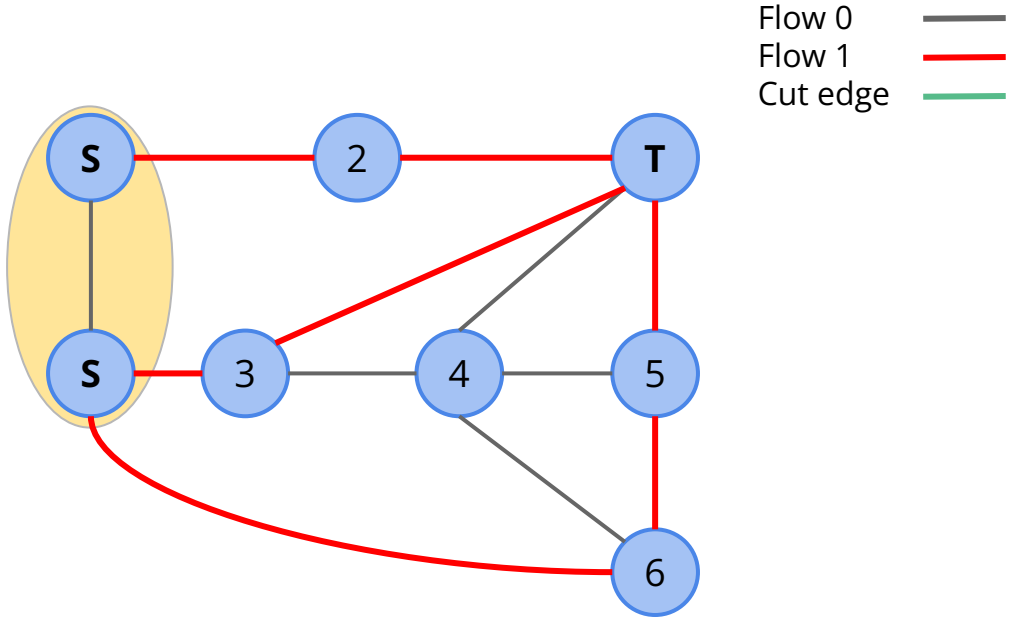
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

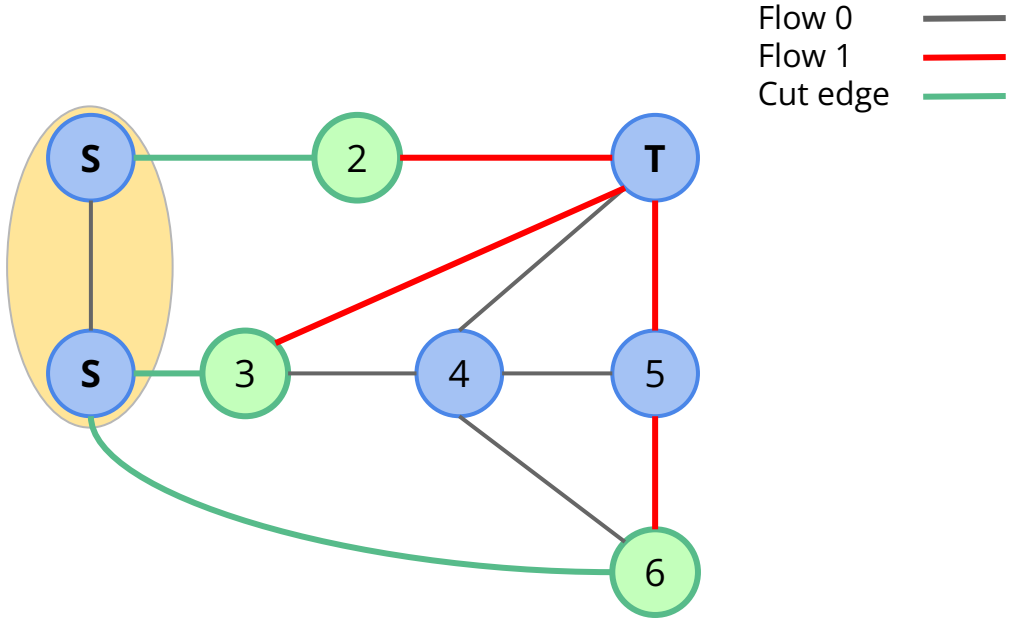
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

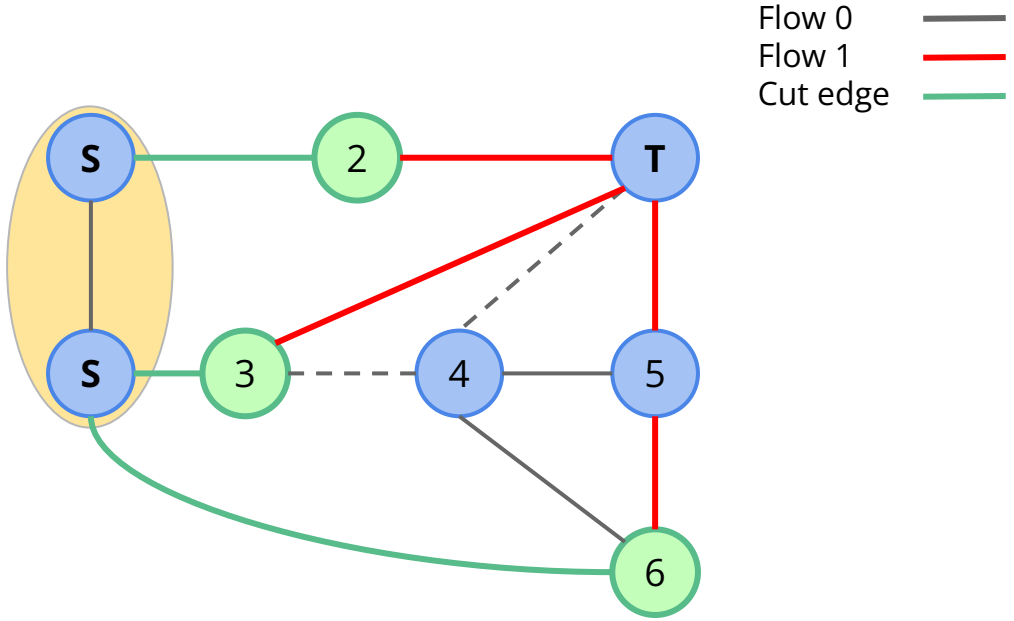
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

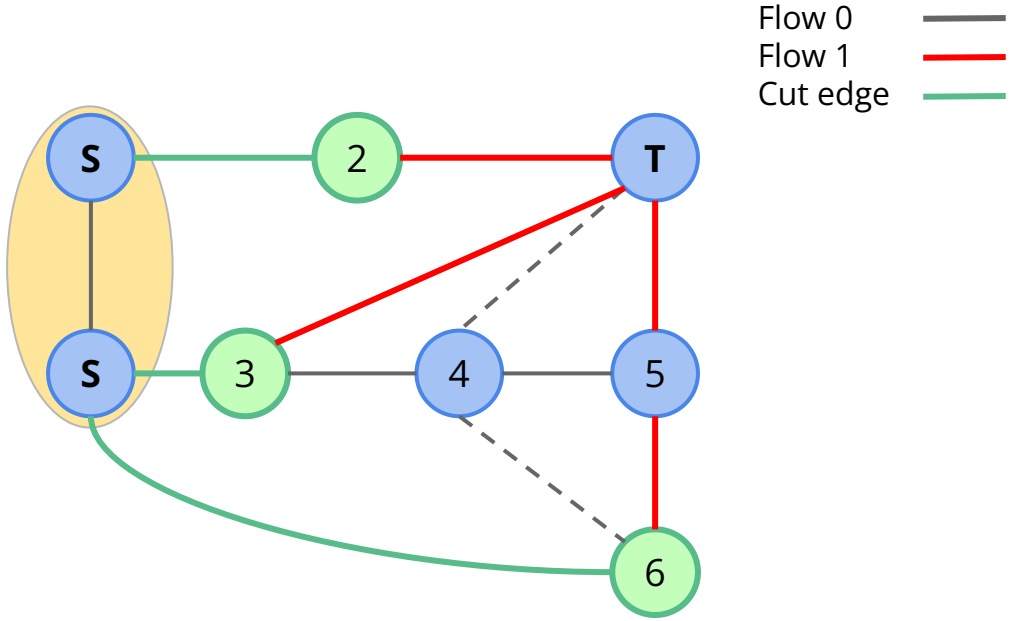
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

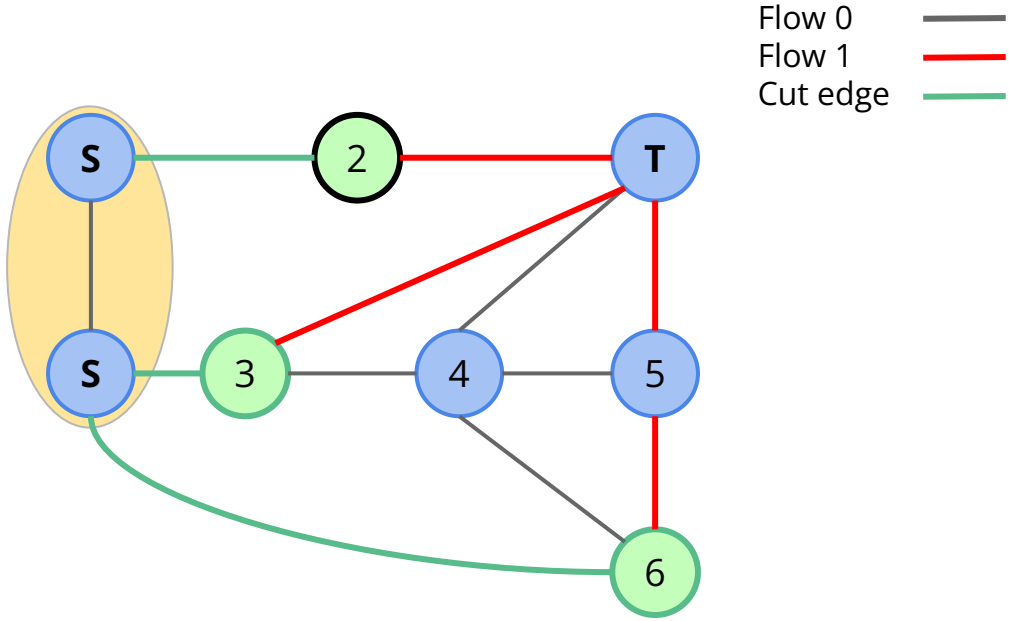
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

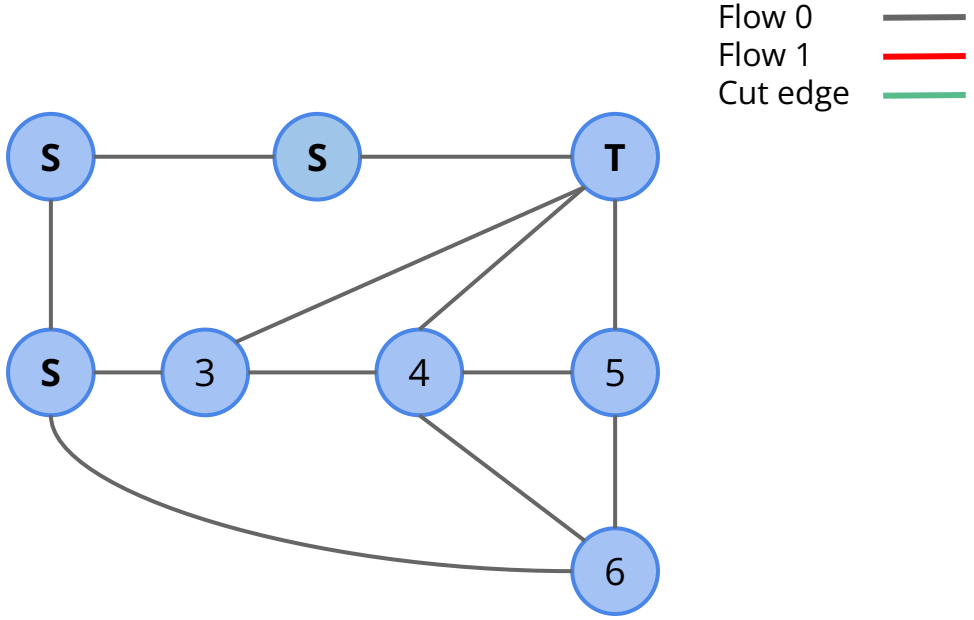
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

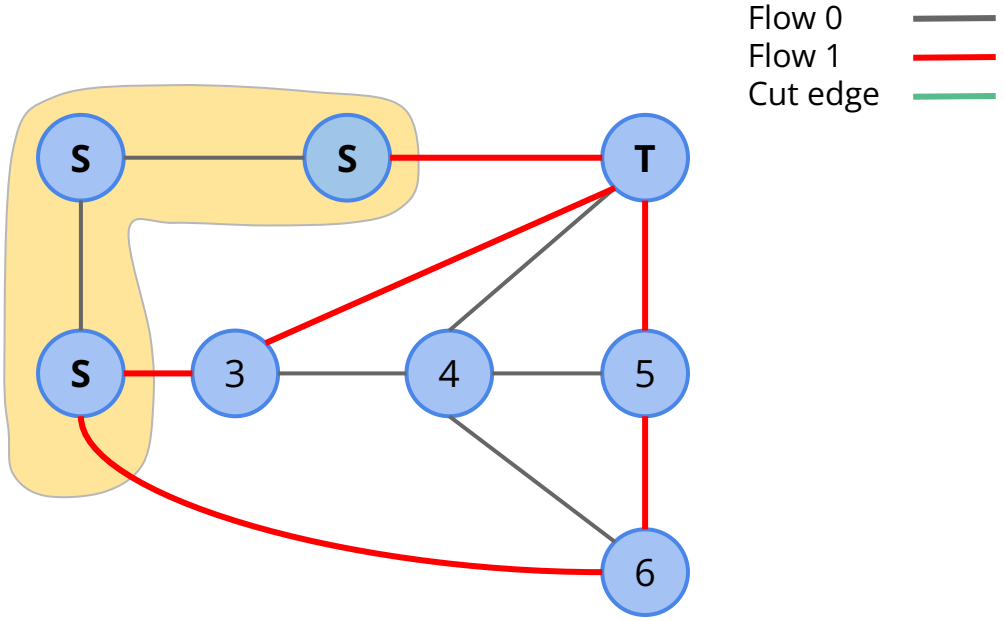
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

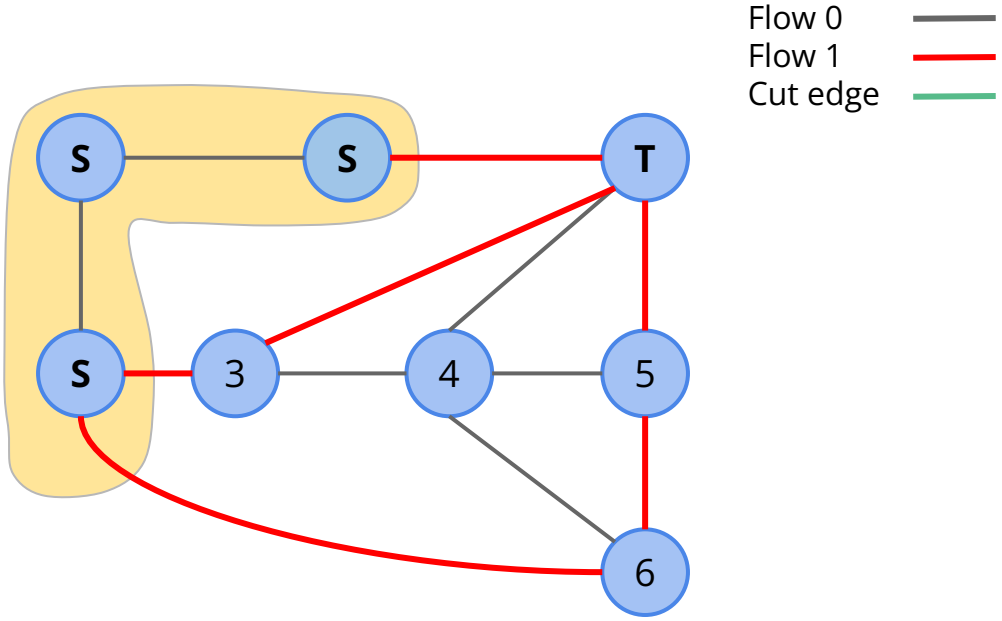
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

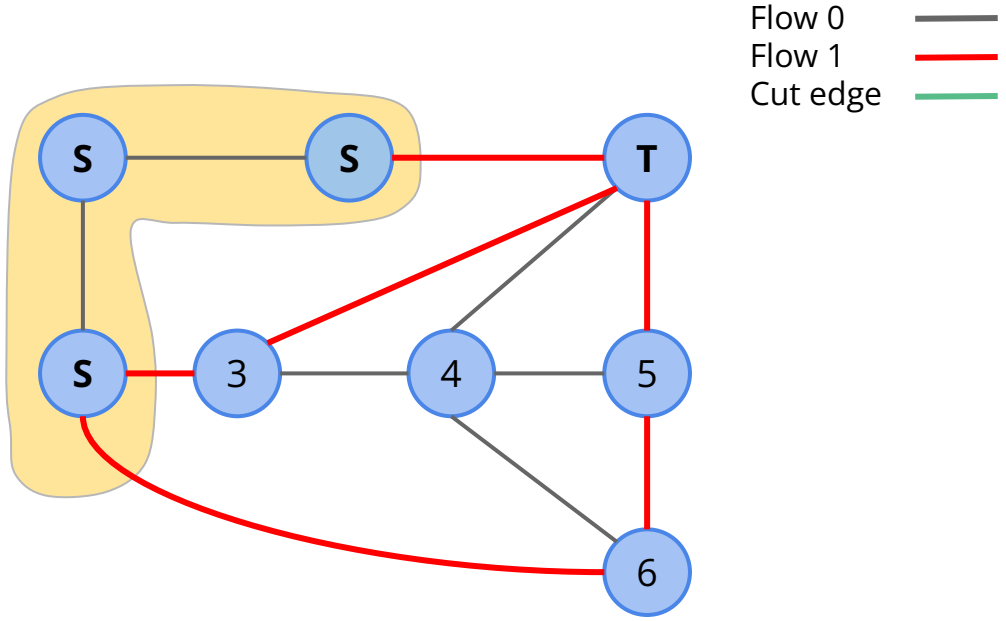
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

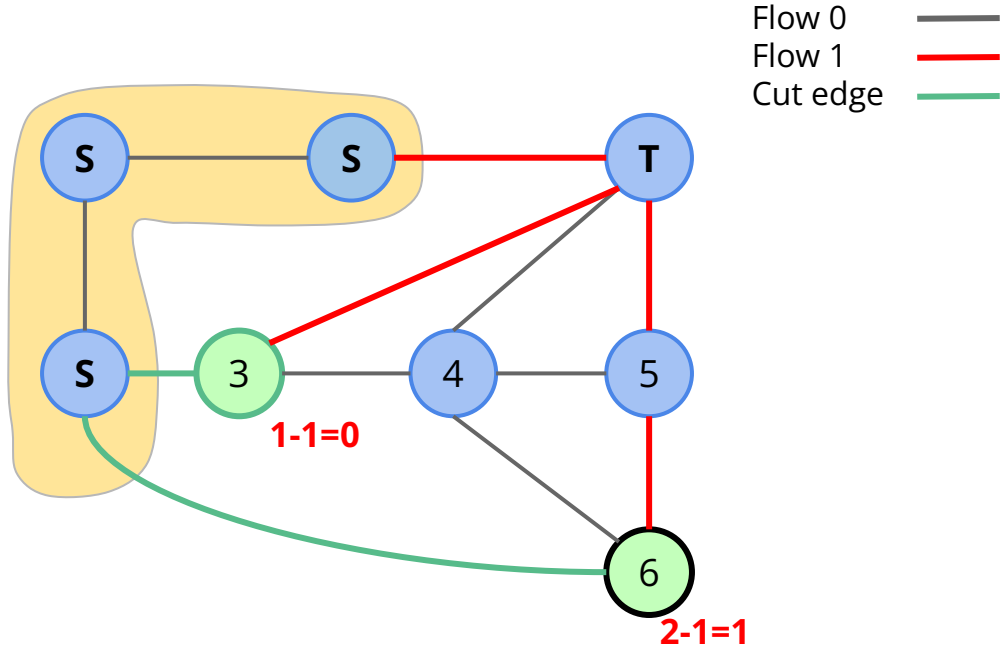
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1

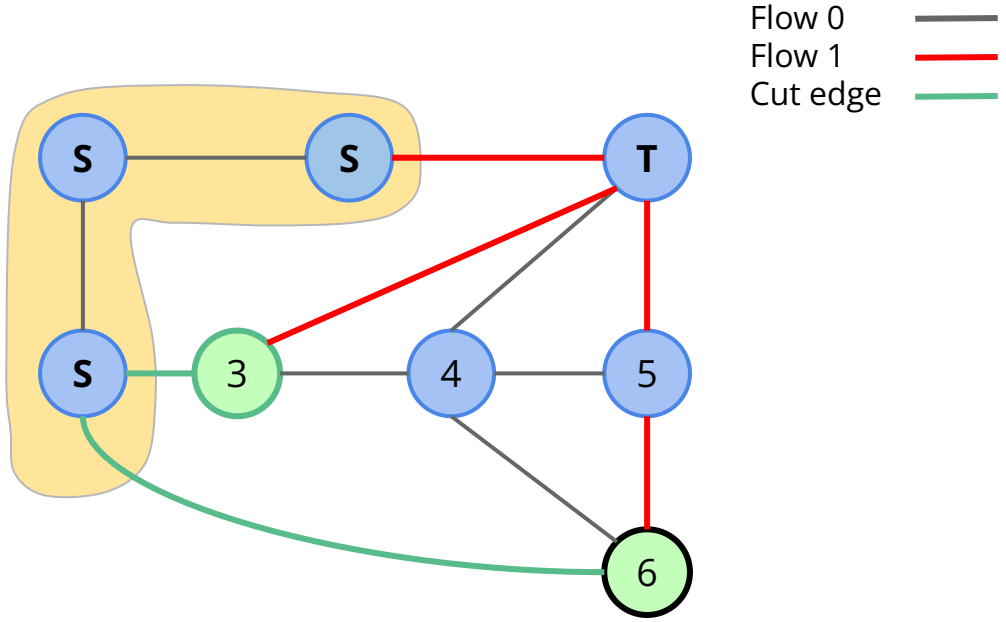


Secondary Heuristic: Pick a node with highest value of (distance from original target - distance from original source)

Finding balanced st-cut

Steps

1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
   a. Make all source side nodes source nodes
   b. Make a piercing node a source node
4. Go to step 1

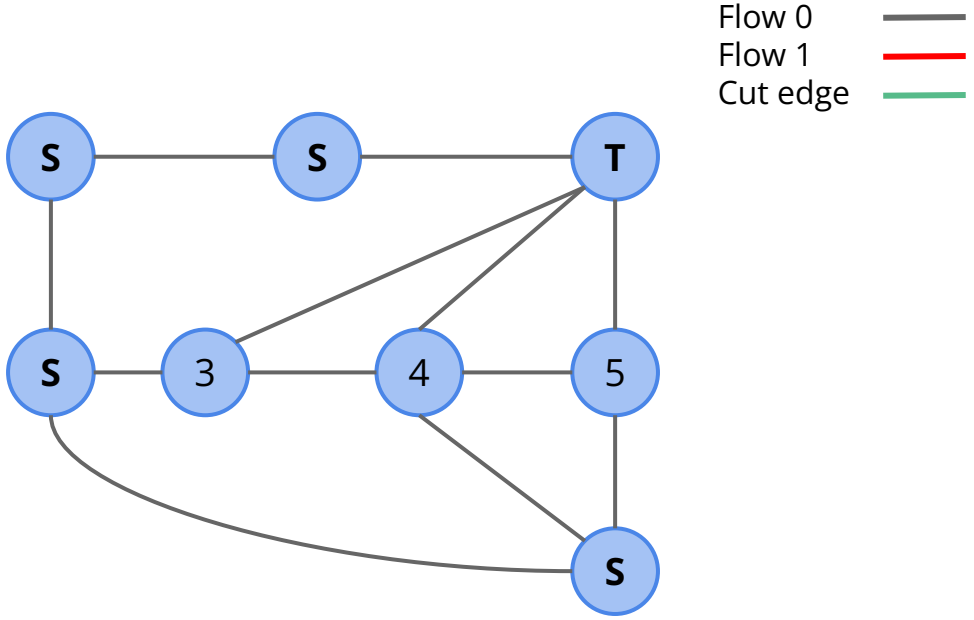


Secondary Heuristic: Pick a node with highest value of (distance from original target - distance from original source)

Finding balanced st-cut

Steps

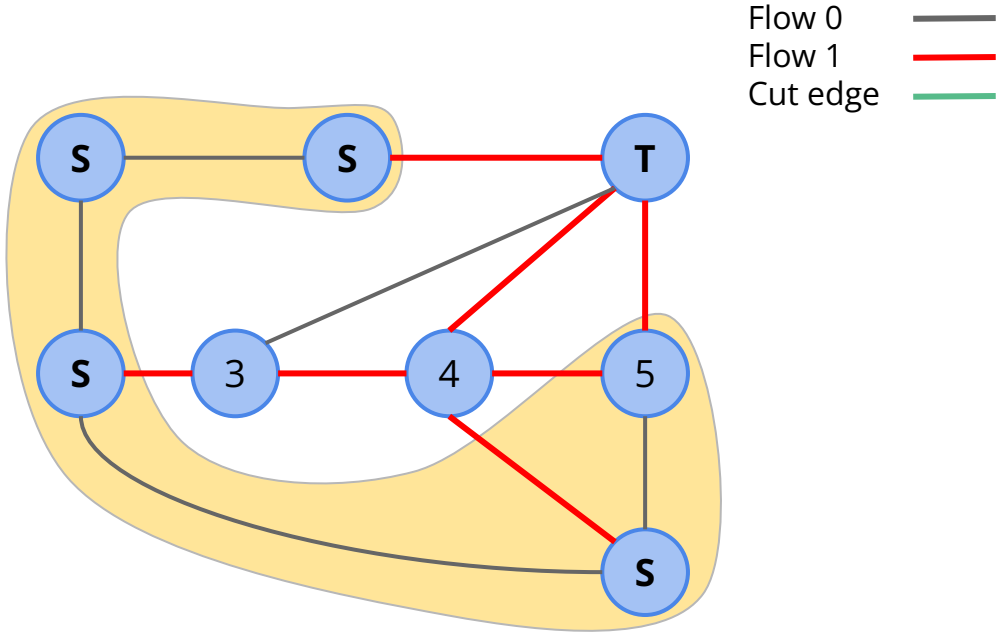
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

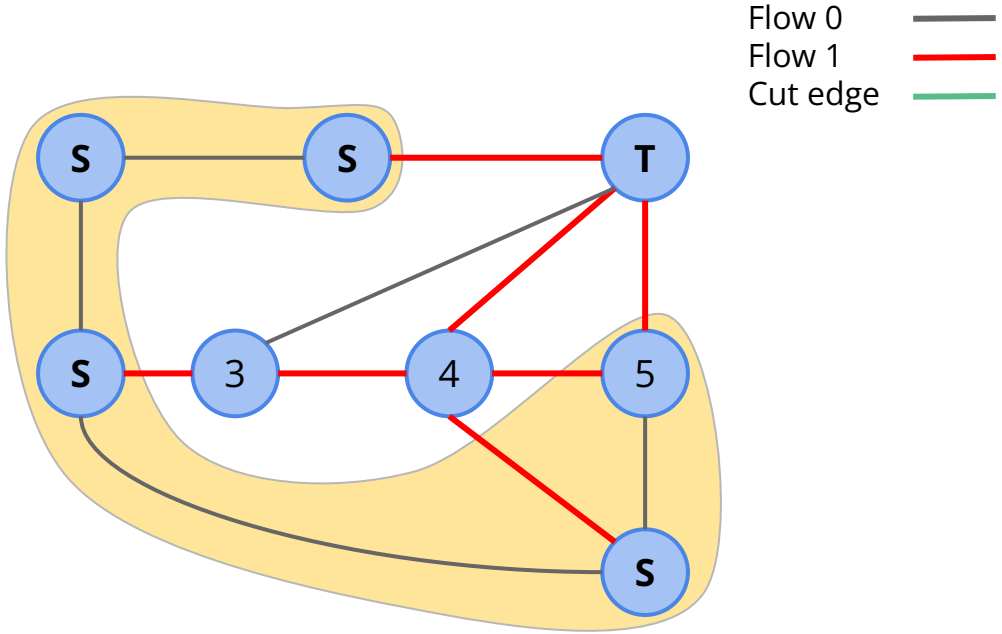
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

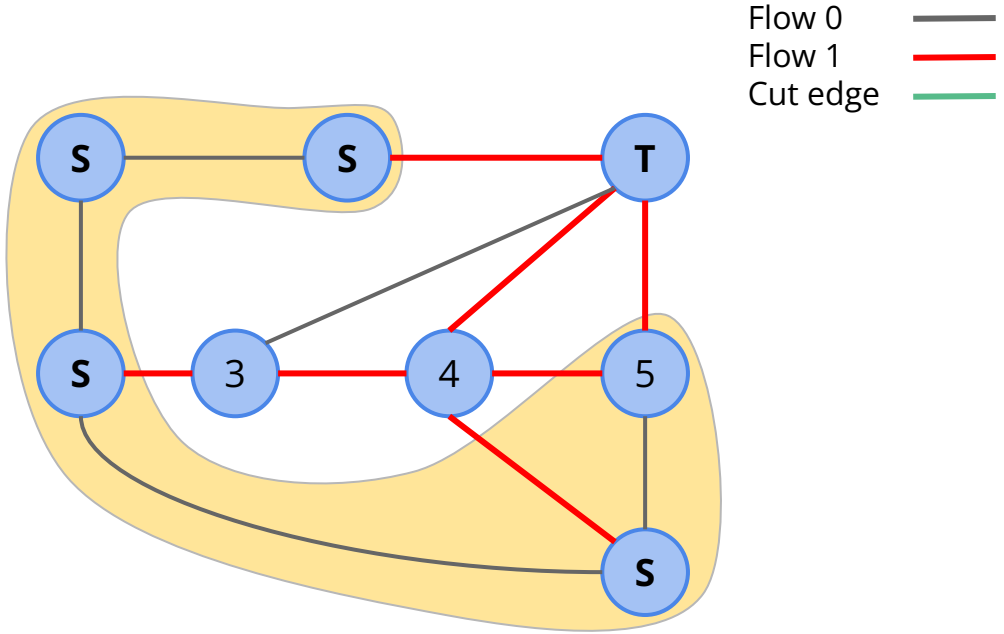
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise suppose source side has lesser nodes. Increase number of sources in source side
4. Go to step 1



Finding balanced st-cut

Steps

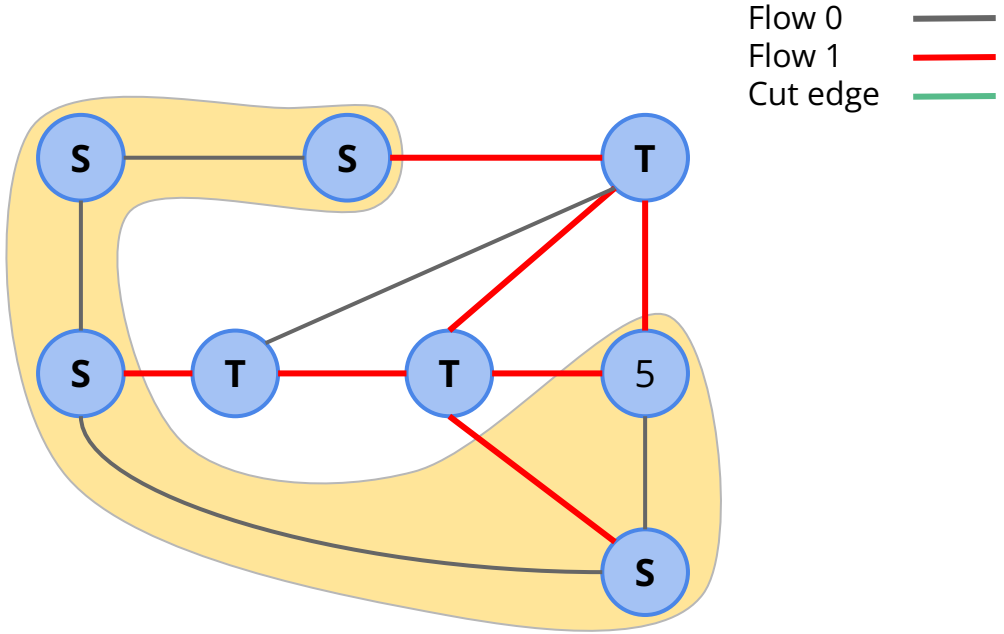
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise, this time target side has lesser nodes. Increase number of targets in target side
   a. Make all target side nodes target nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

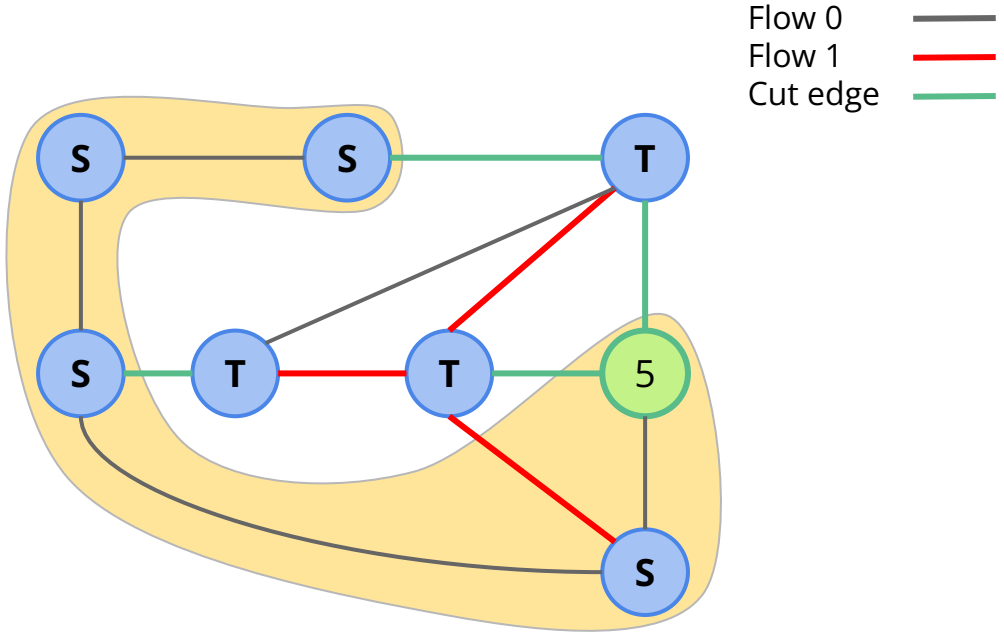
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise, this time target side has lesser nodes. Increase number of targets in target side
   a. Make all target side nodes target nodes
   b. Make a piercing node a source node
4. Go to step 1



Finding balanced st-cut

Steps

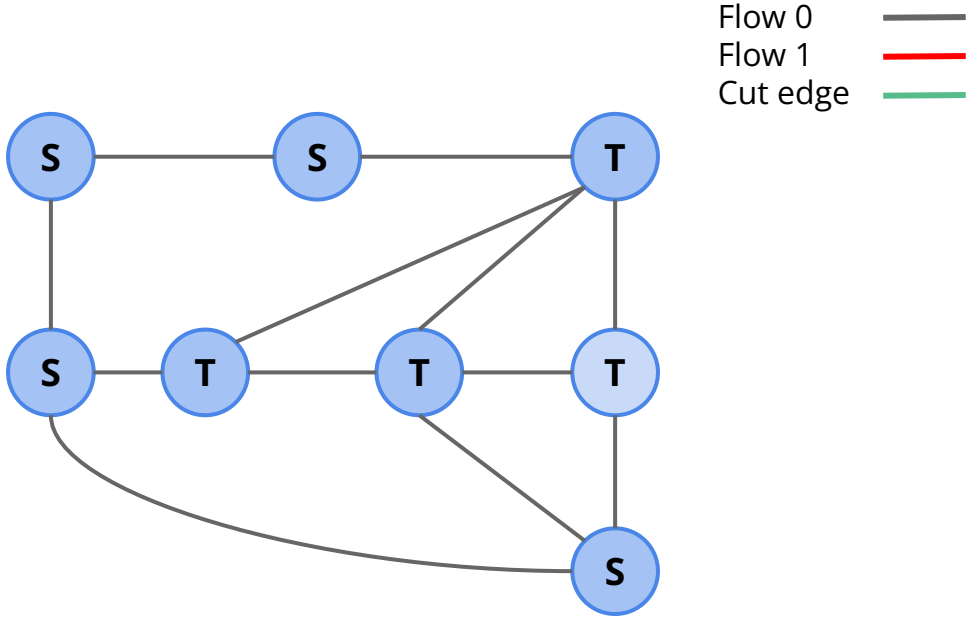
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise, this time target side has lesser nodes. Increase number of targets in target side
   a. Make all target side nodes target nodes
   b. Make a piercing node a target node
4. Go to step 1



Finding balanced st-cut

Steps

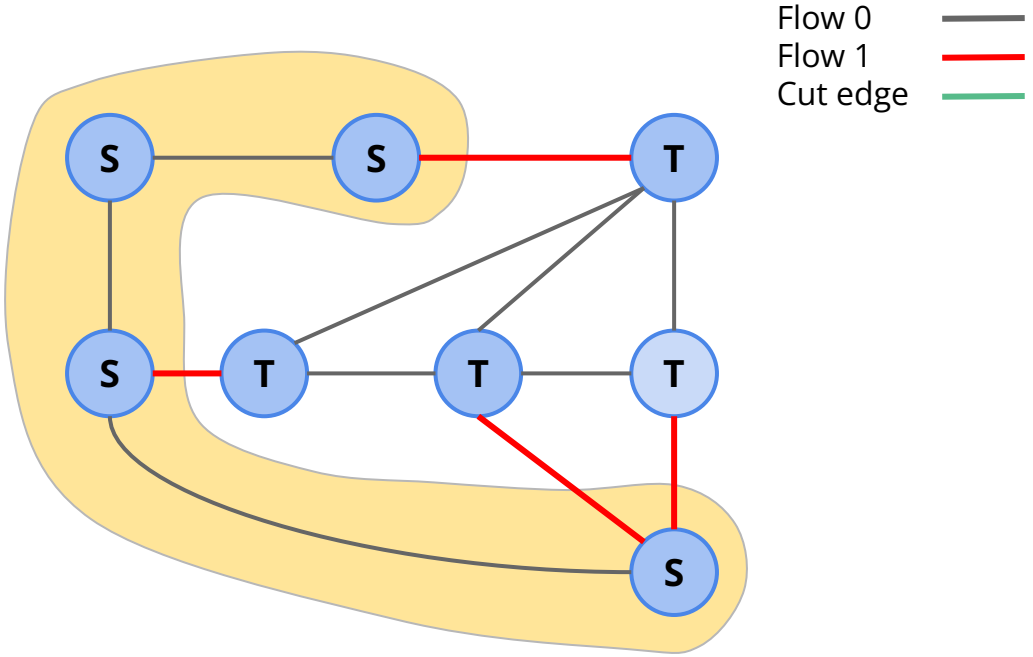
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise, this time target side has lesser nodes. Increase number of targets in target side
4. Go to step 1



Finding balanced st-cut

Steps

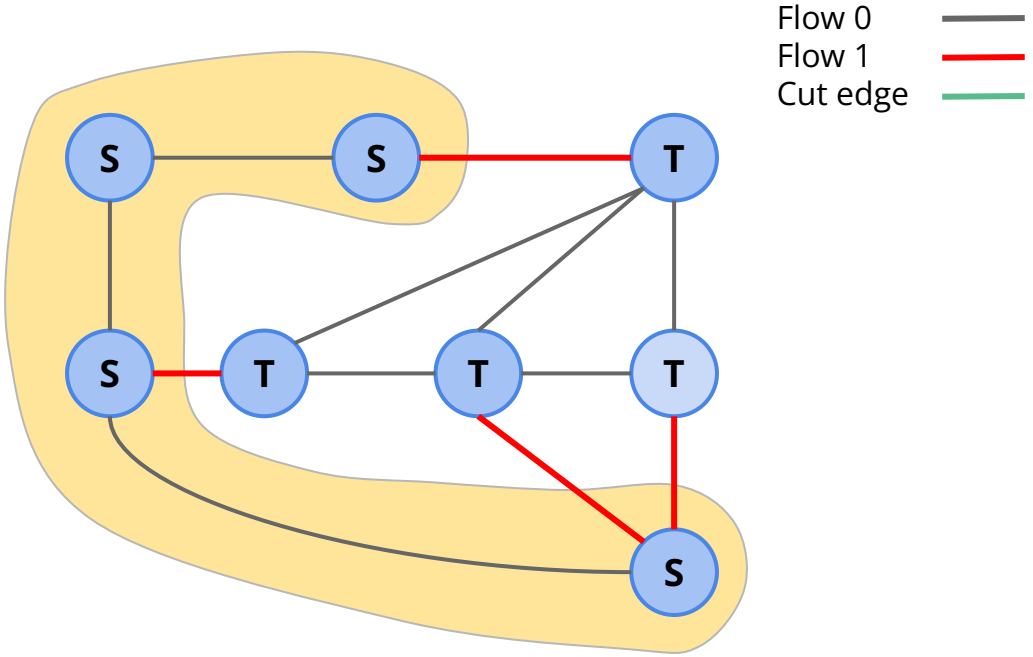
1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise, this time target side has lesser nodes. Increase number of targets in target side
4. Go to step 1



Finding balanced st-cut

Steps

1. Compute a cut using max flow min cut theorem
2. If the cut is balanced or all nodes have become either source or target, then stop.
3. Otherwise, this time target side has lesser nodes. Increase number of targets in target side
4. Go to step 1

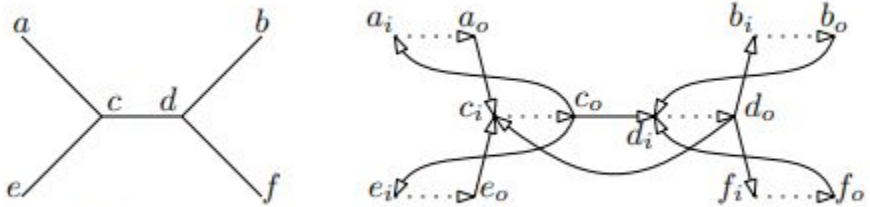


Finding General cut

- Run Flow-Cutter q times with st pairs picked uniformly at random
- High probability of finding a good cut with enough q
- The paper recommends q = 20

Finding Node Separator

- Generate a directed graph from the given undirected graph
- Main idea is to represent a vertex by an edge in the directed graph
- Node separator is computed from the cut of the directed graph



(a) Input graph (b) Expanded graph

Tree Decomposition Using Nested Dissection Method

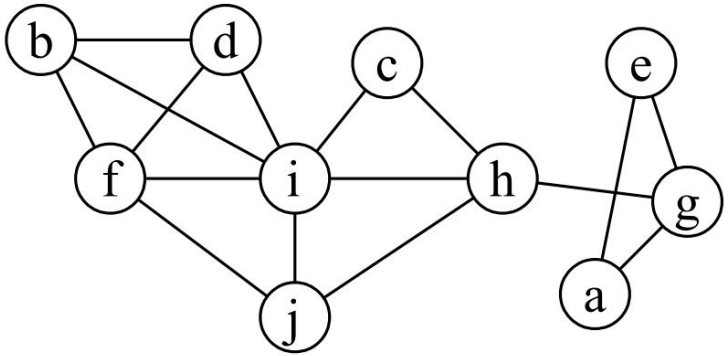
1805010 - Anwarul Bashir Shuaib

Terminologies

- Perfect elimination ordering
- Small balanced node separator
- Chordal supergraph
- Maximum Spanning Tree

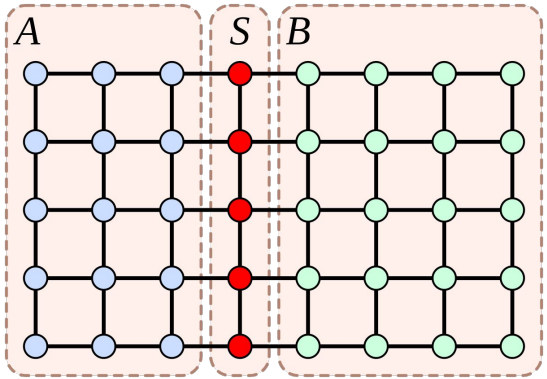
Perfect Elimination Ordering

- Ordering of the vertices of the graph such that, for each vertex **v**, **v** and the neighbors of **v** that occur after **v** in the order form a clique.
  - The vertices in the graph have an ordering
  - For each vertex ($\mathbf{v_i}$), construct $\mathbf{S} = \{\mathbf{v_i}\} \cup \mathbf{N_j}(\mathbf{v})$
  - $\mathbf{N_j}$ is the neighbours of **v** with position in the ordering after **i**
  - **S** is a clique
  - Example figure ordering: a,b,c,d,e,f,g,h,i,j



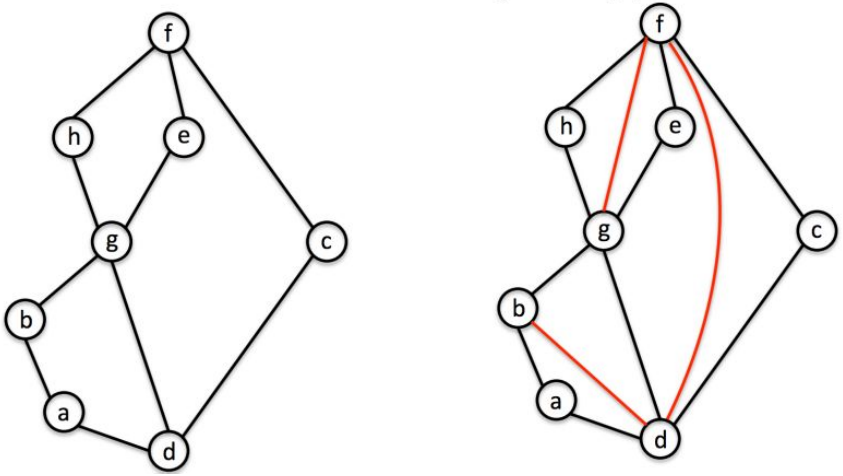
Small Balanced Node Separator

- Subset of vertices in a graph whose removal results in dividing the graph into two or more disconnected subgraphs
  - These subgraphs are approximately equal in size
  - "small" ⇒ desire to minimize the number of nodes in the separator
  - NP-hard for general graphs
  - A graph of treewidth **k** has a balanced separator with at most **k+1** vertices.



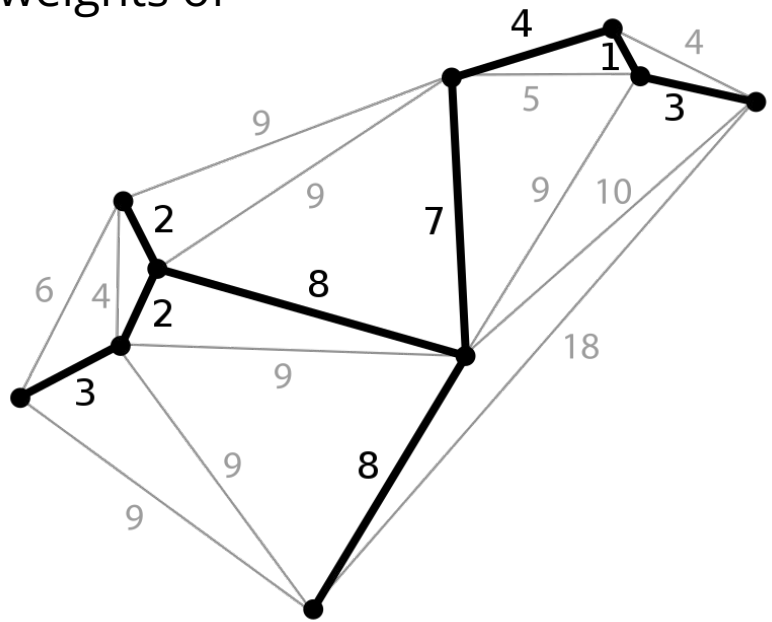
Chordal supergraph

- Transforming a non-chordal graph into a chordal graph involves adding the minimum number of edges necessary to ensure that all cycles of four or more vertices have a chord (triangulated)
- This transformed graph is called a "chordal supergraph" of the original graph

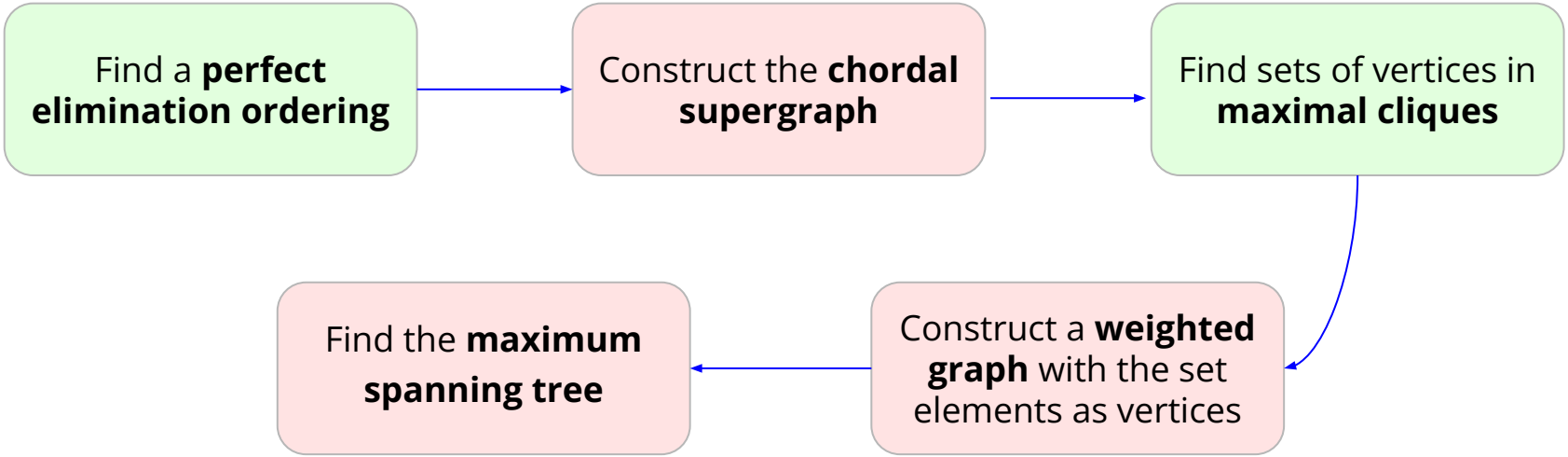


Maximum Spanning Tree

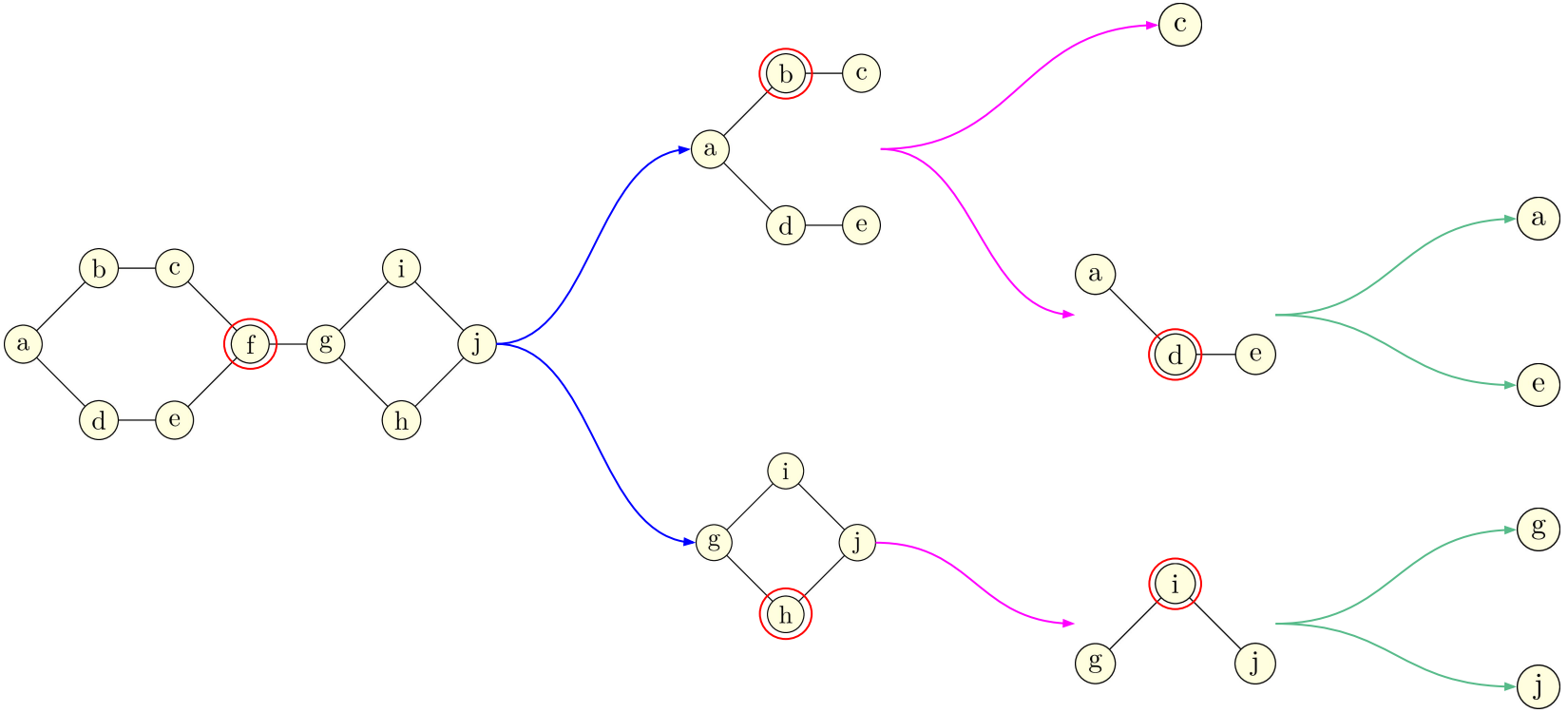
- A Maximum Spanning Tree (MST) of a weighted graph is a spanning tree in which the sum of the weights of the edges in the tree is maximized.



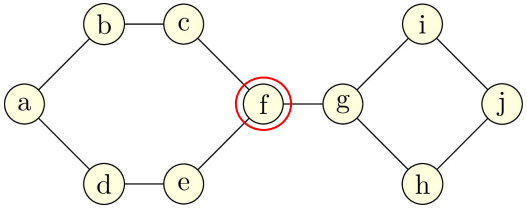
Overview of the Nested Dissection Method



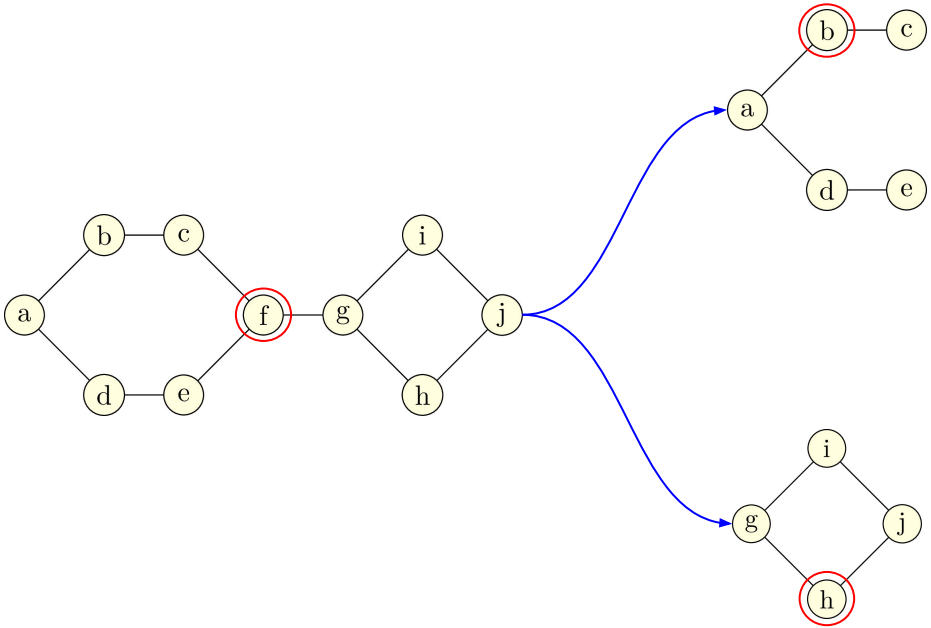
Finding A Perfect Elimination Ordering



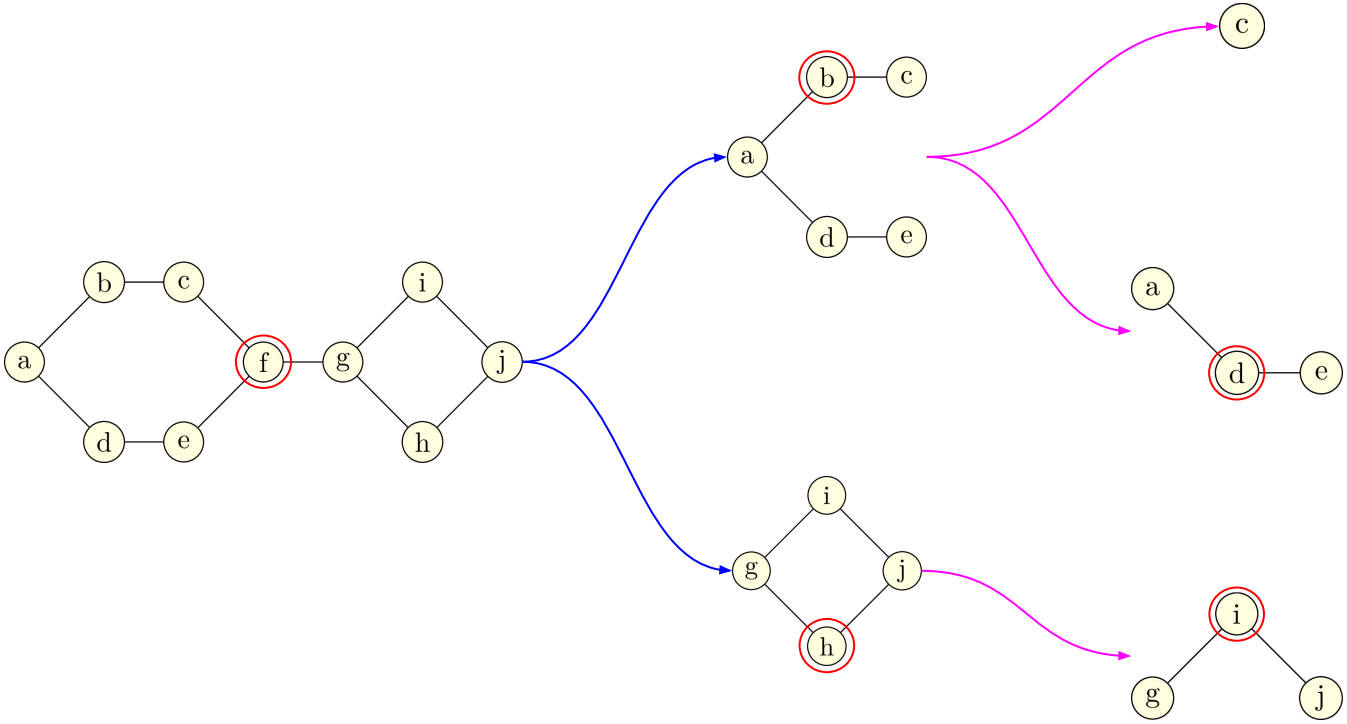
Finding A Perfect Elimination Ordering



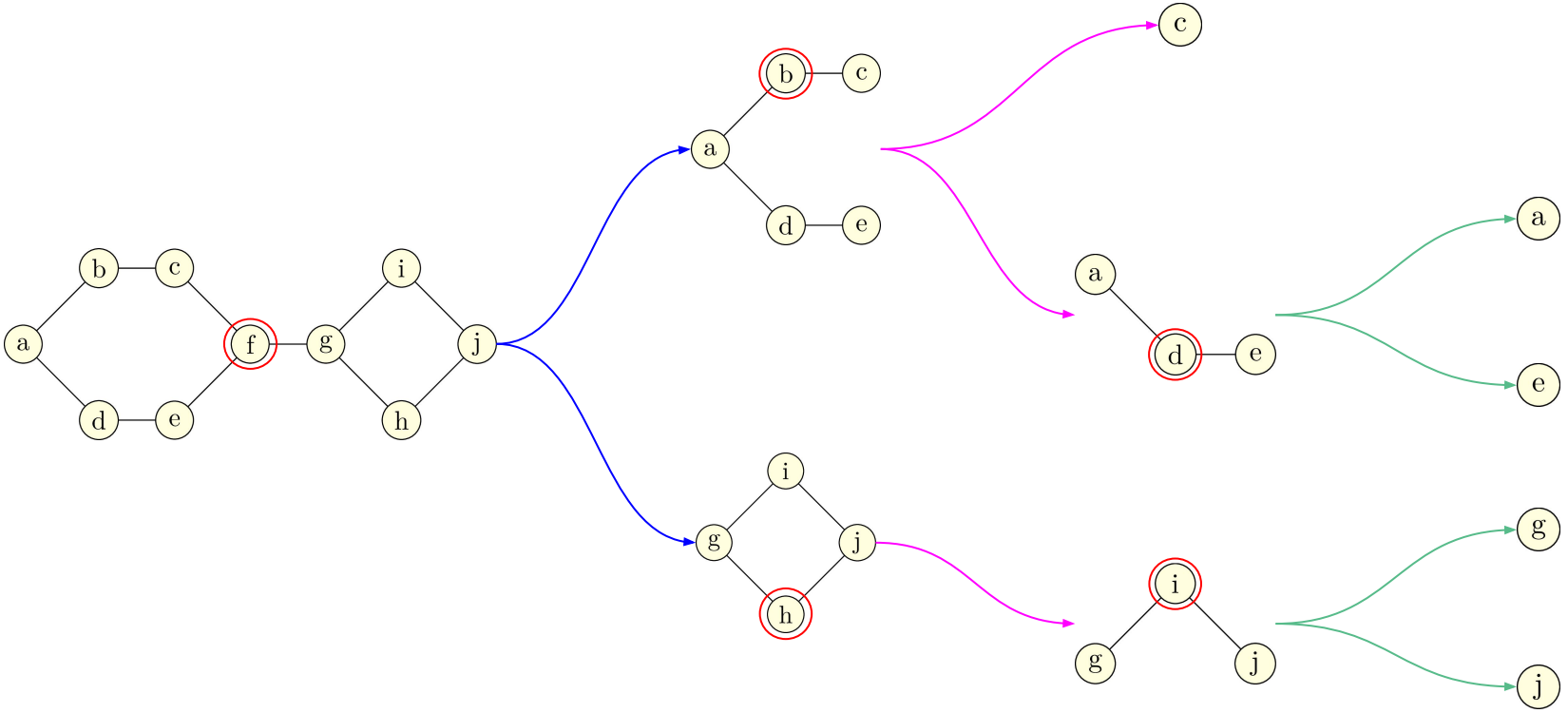
Step 1. Finding A Perfect Elimination Ordering



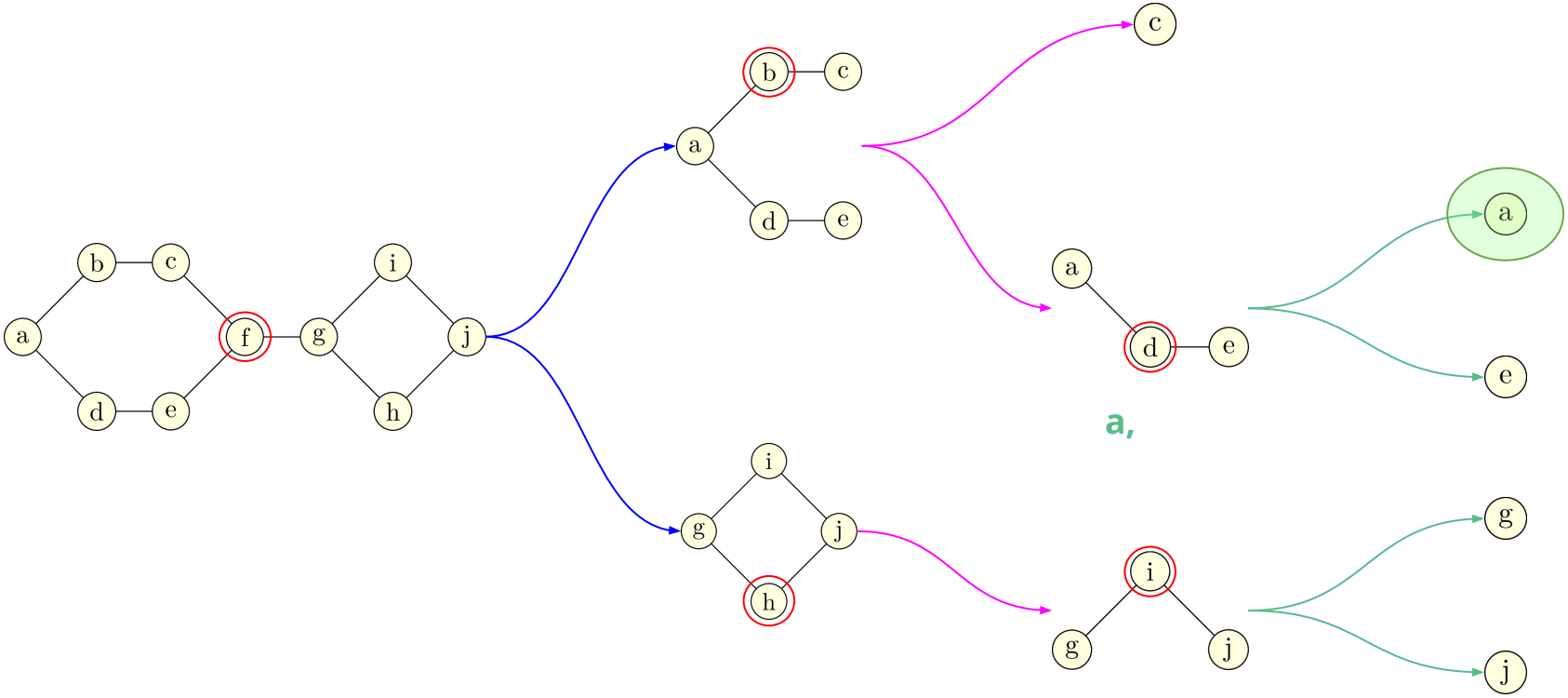
Finding A Perfect Elimination Ordering



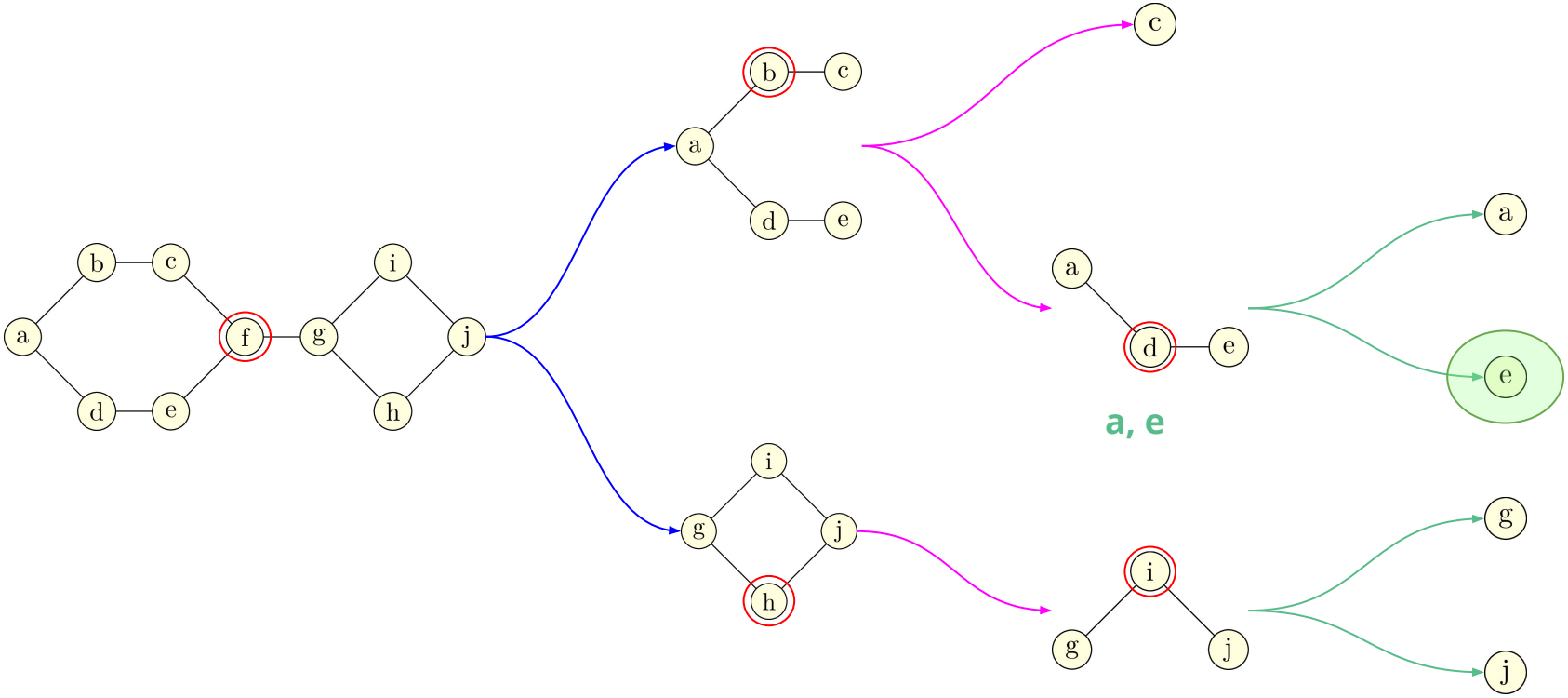
Finding A Perfect Elimination Ordering



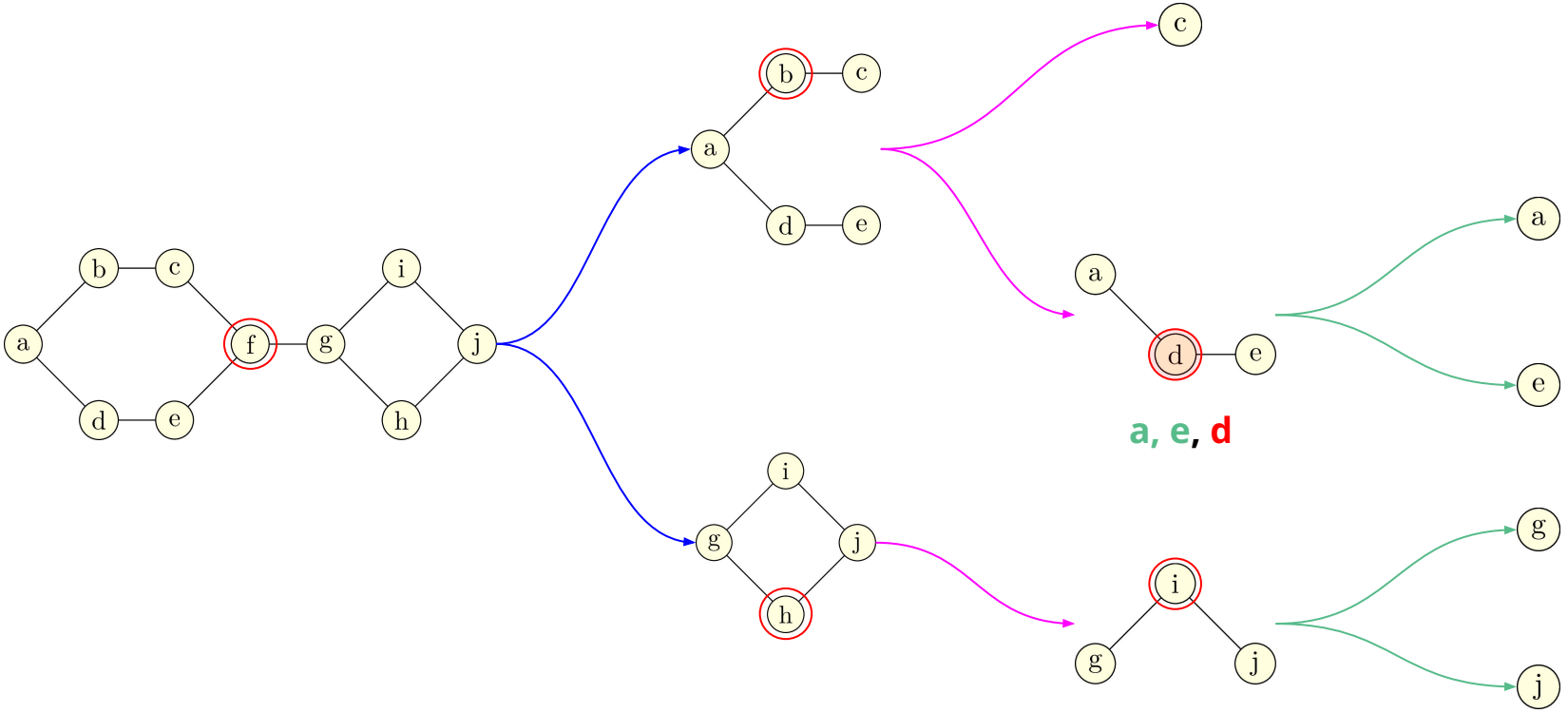
Finding A Perfect Elimination Ordering



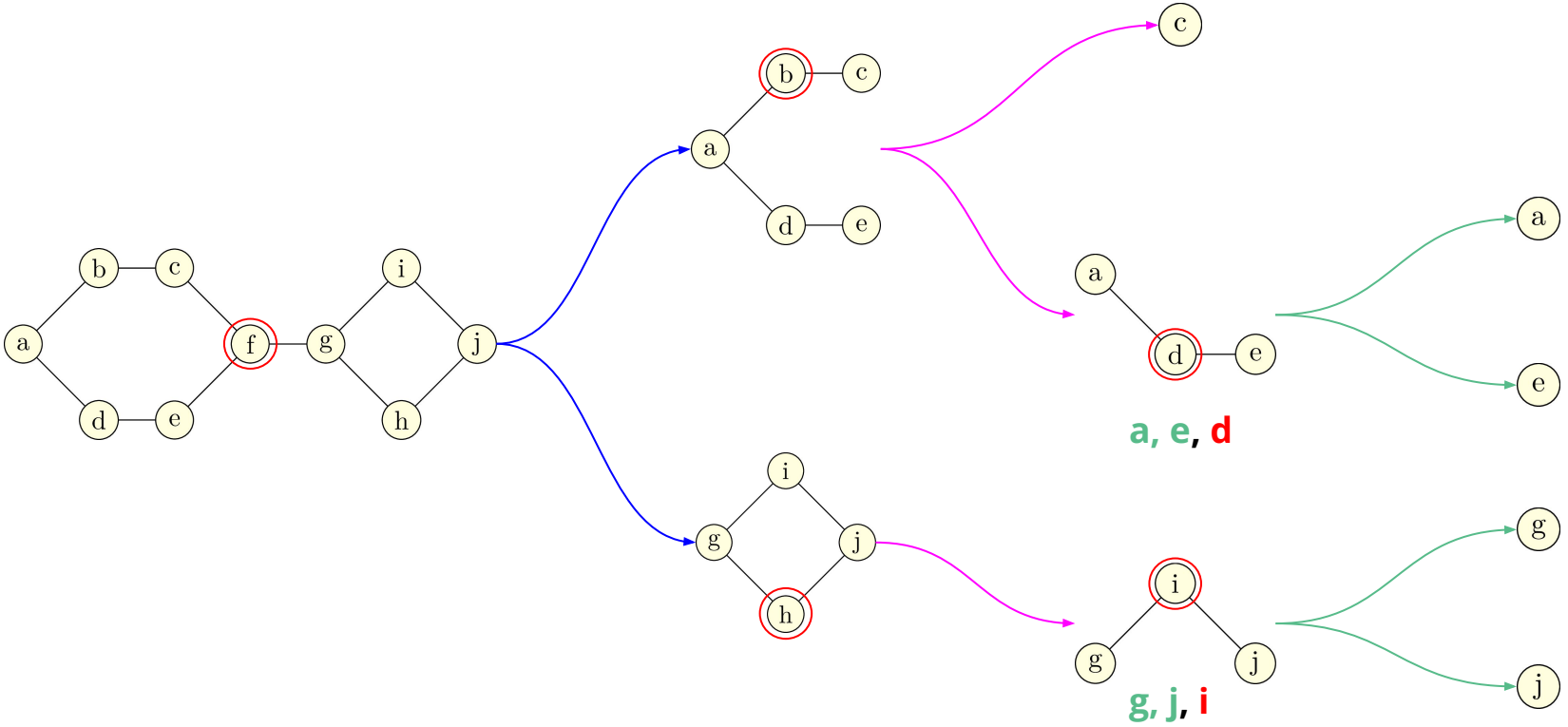
Finding A Perfect Elimination Ordering



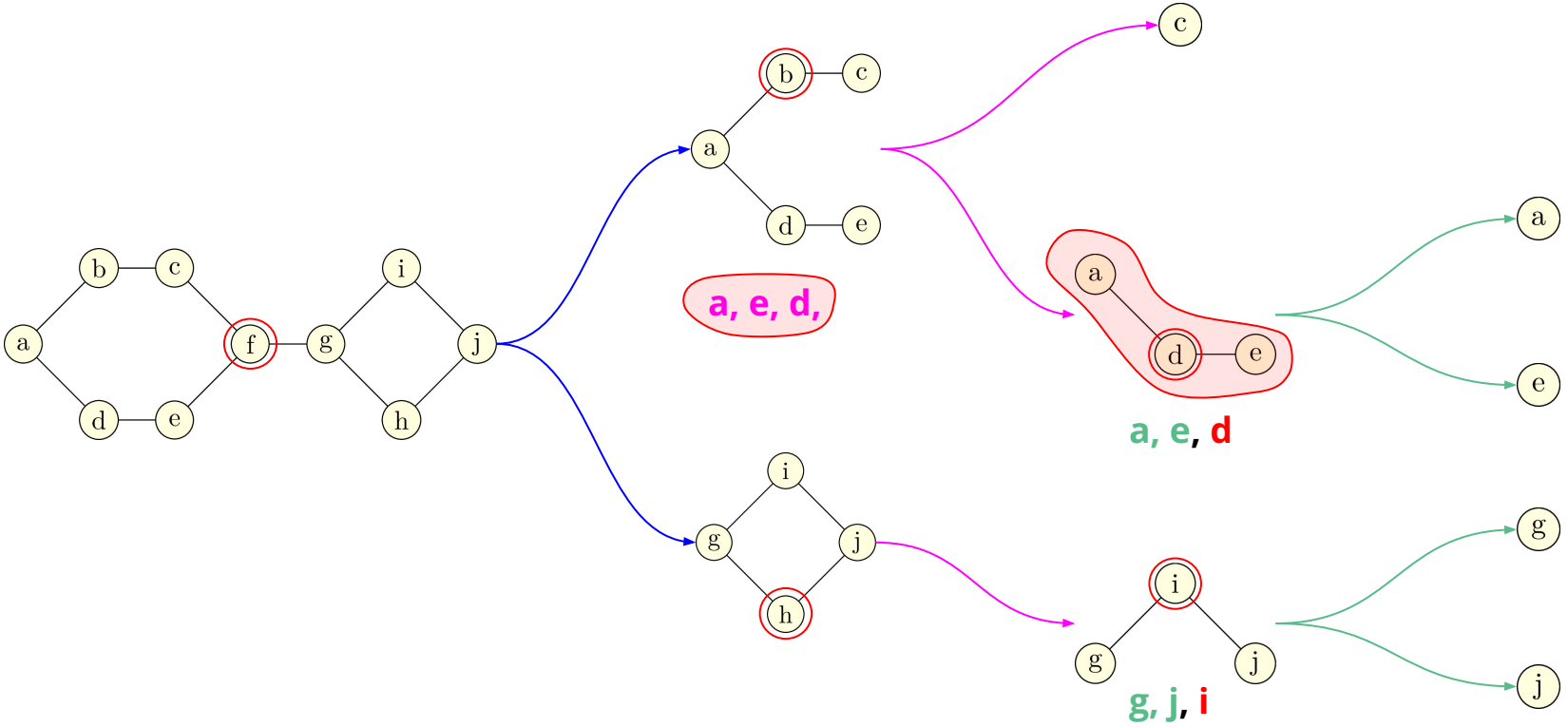
Finding A Perfect Elimination Ordering



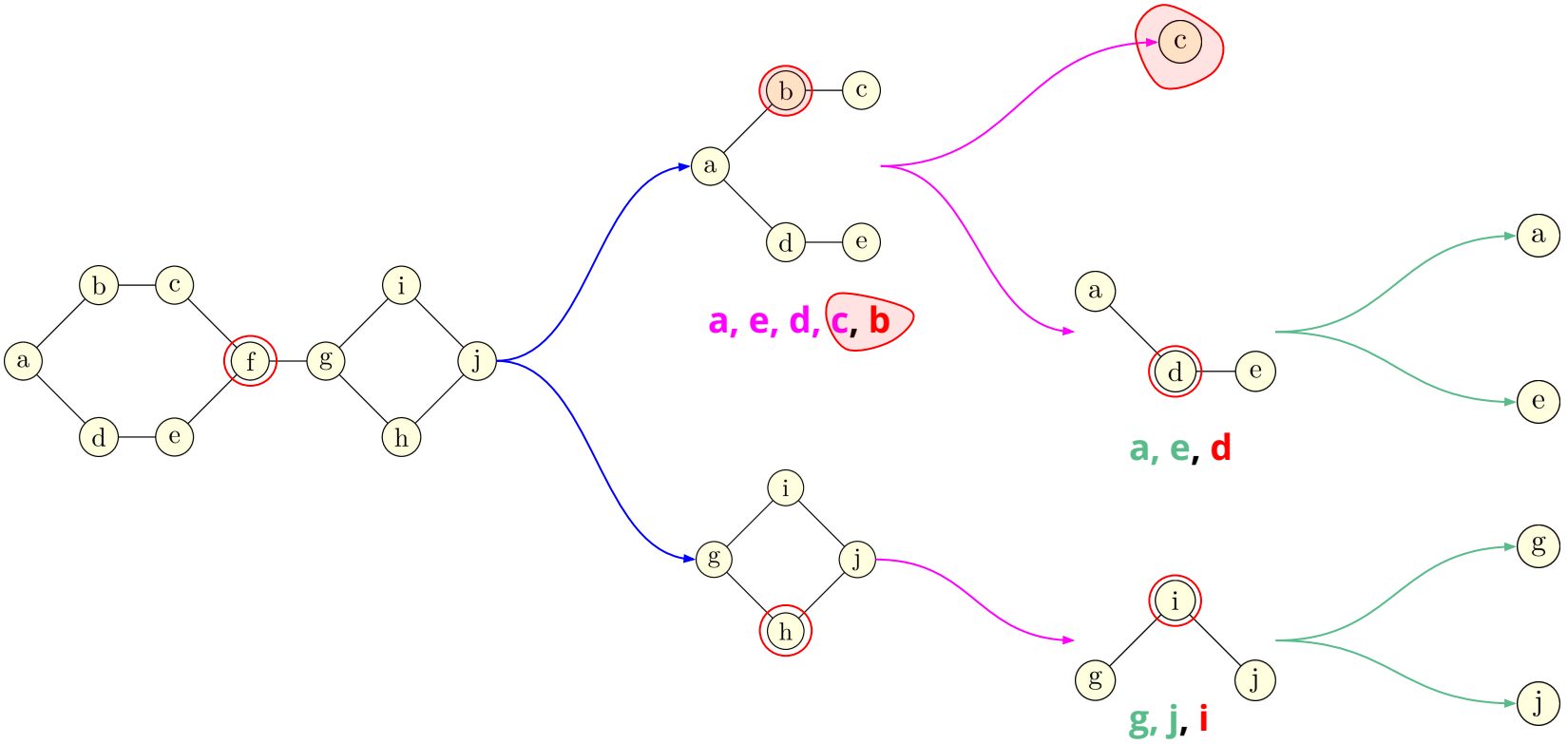
Finding A Perfect Elimination Ordering



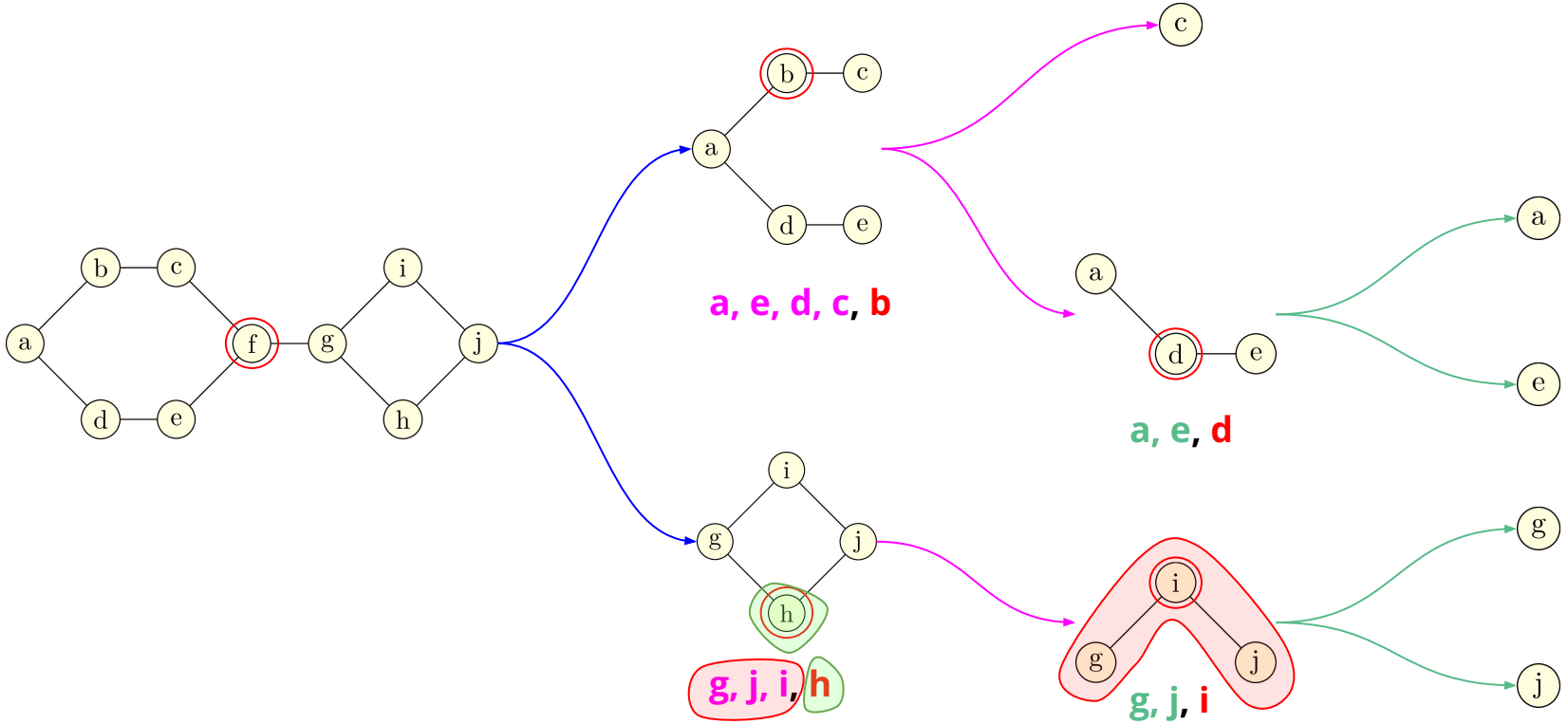
Finding A Perfect Elimination Ordering



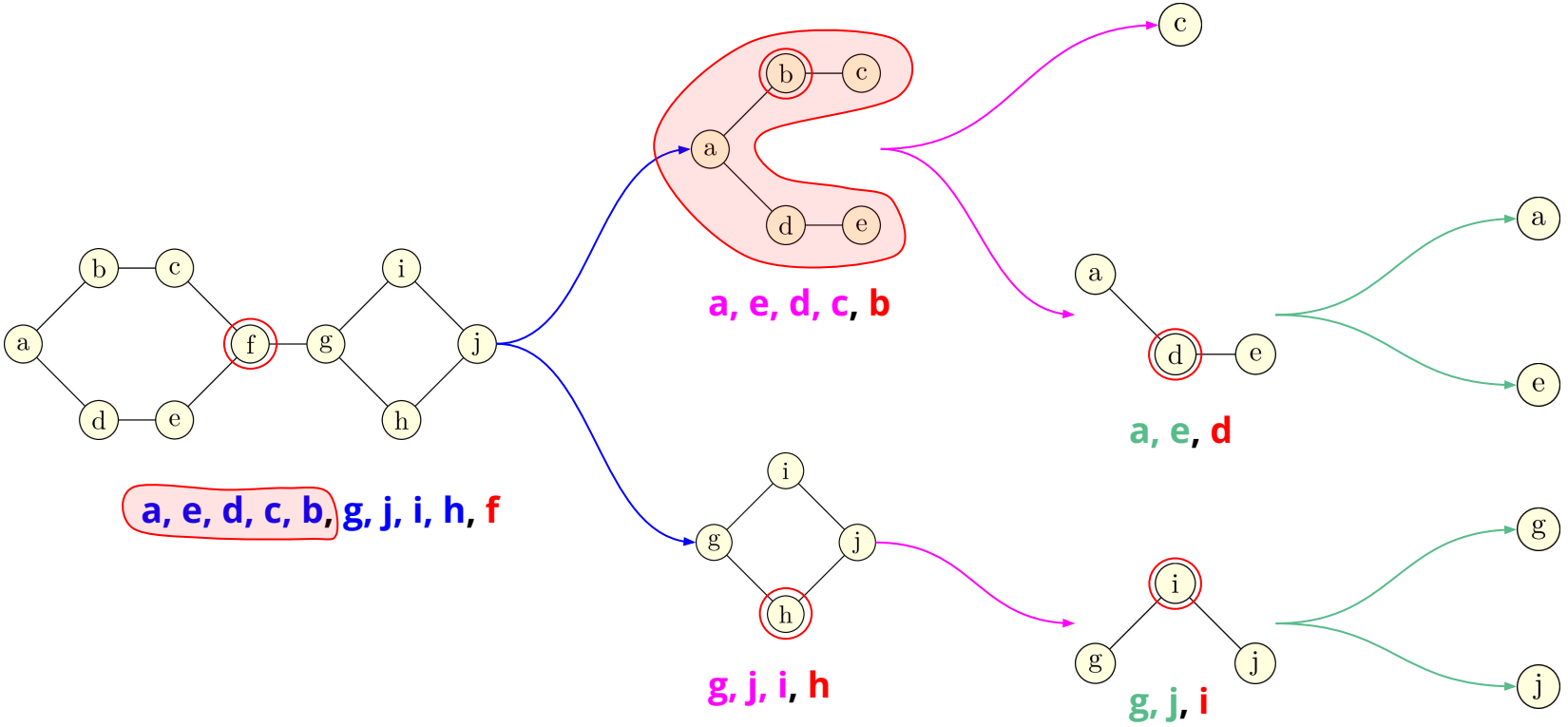
Finding A Perfect Elimination Ordering



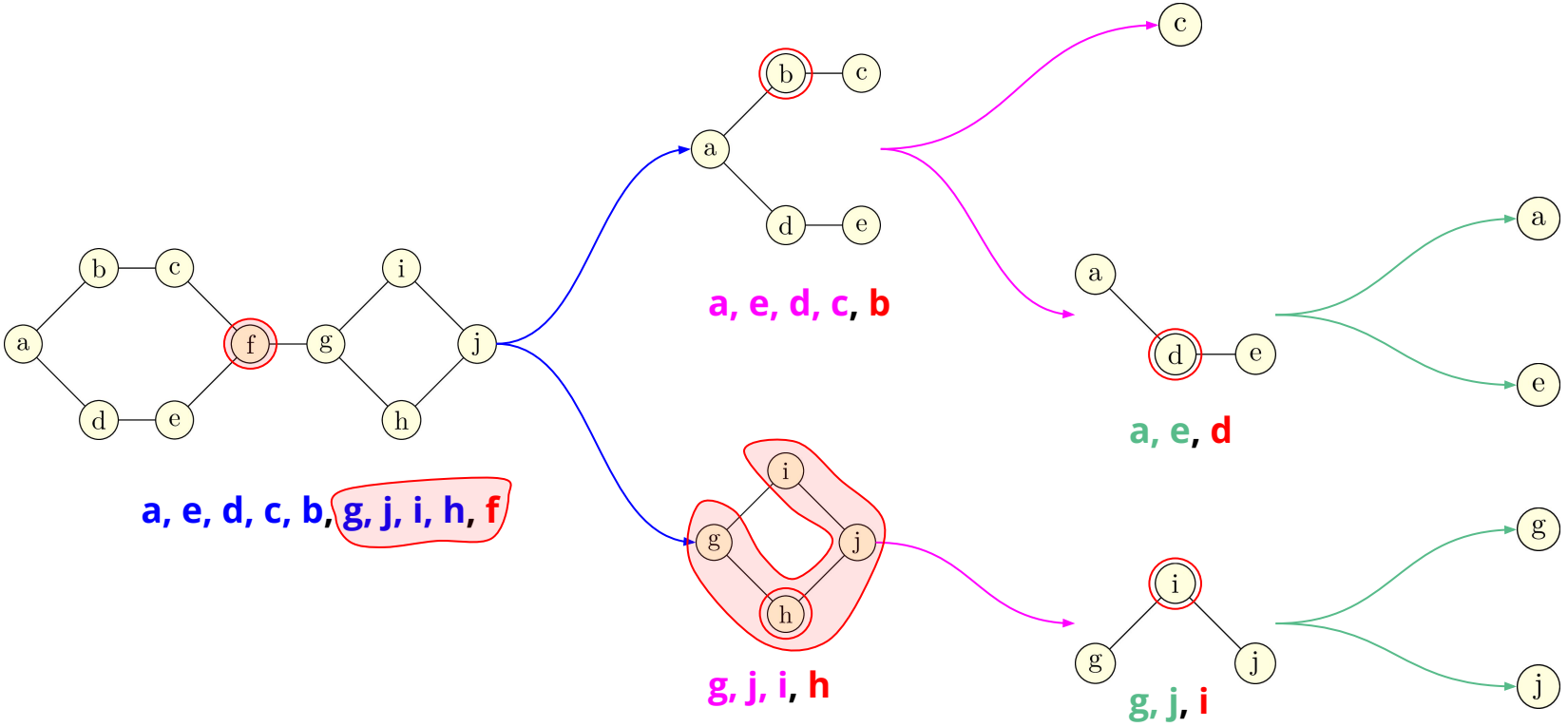
Finding A Perfect Elimination Ordering



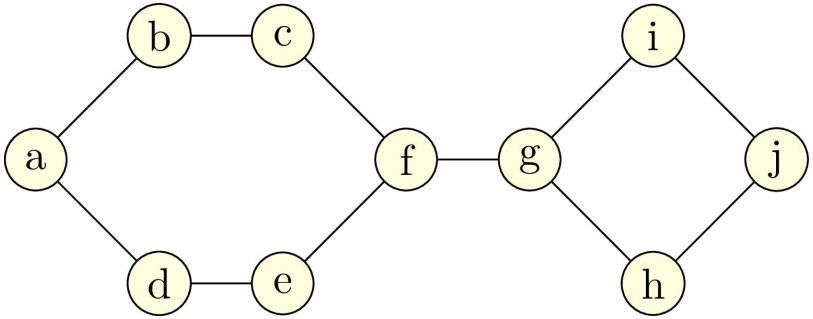
Finding A Perfect Elimination Ordering



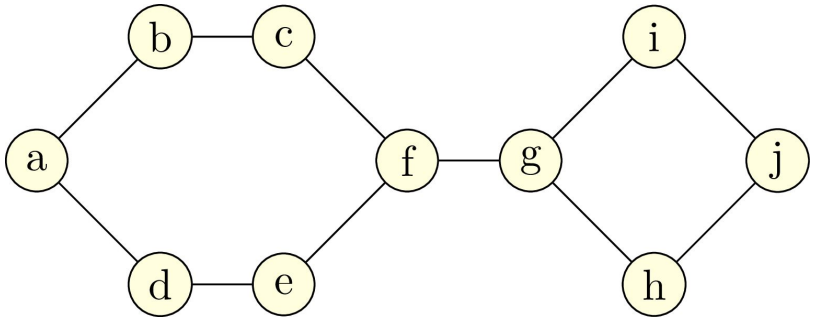
Finding A Perfect Elimination Ordering



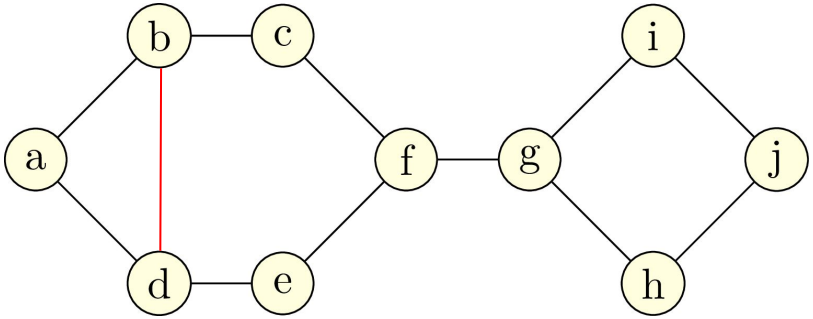
Step 2. Constructing Chordal Supergraph



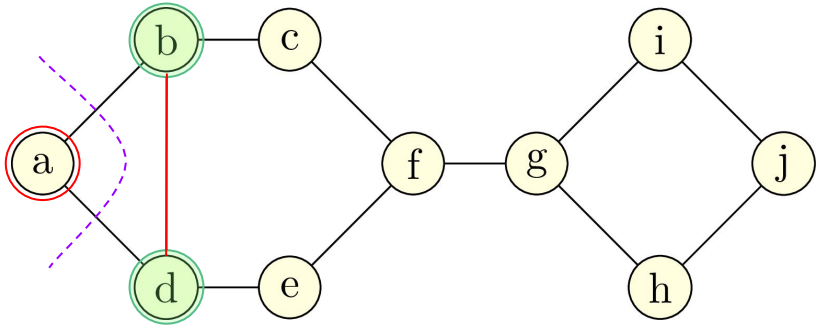
a, e, d, c, b, g, j, i, h, f



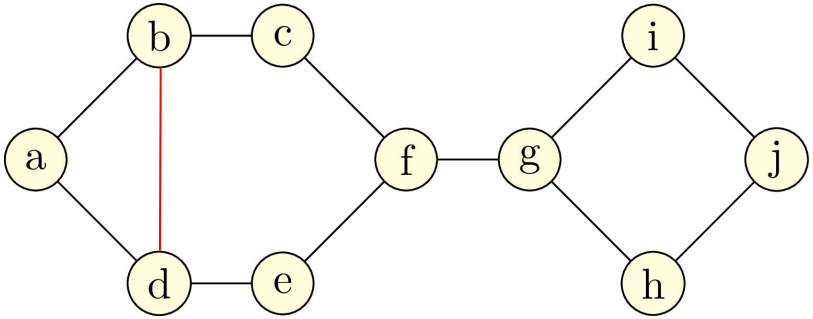
Constructing Chordal Supergraph



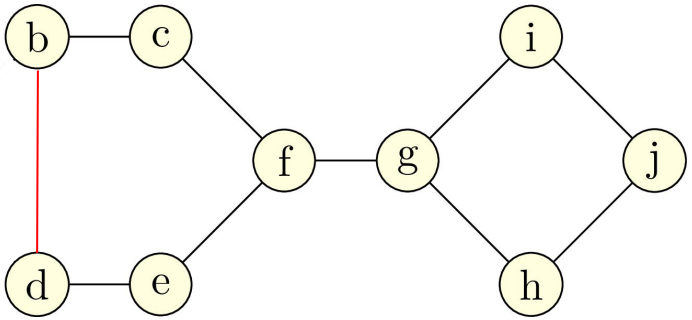
a, e, d, c, b, g, j, i, h, f



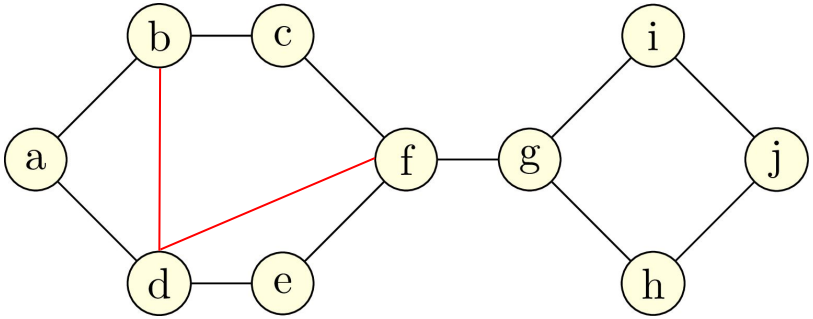
Constructing Chordal Supergraph



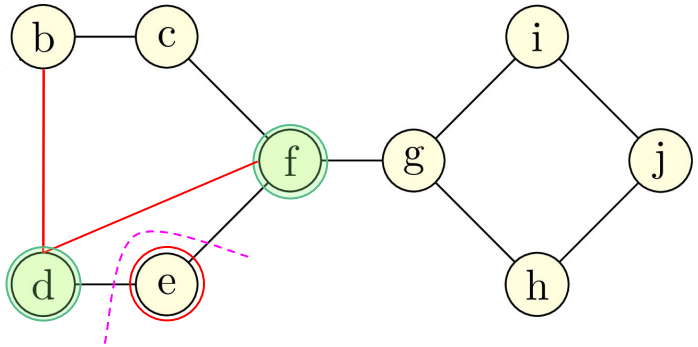
a, e, d, c, b, g, j, i, h, f



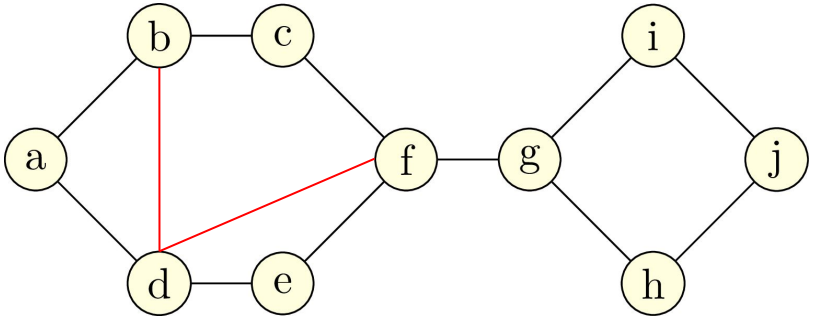
Constructing Chordal Supergraph



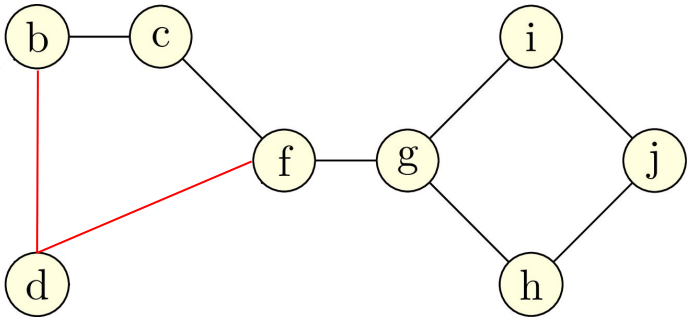
a, **e,** d, c, b, g, j, i, h, f



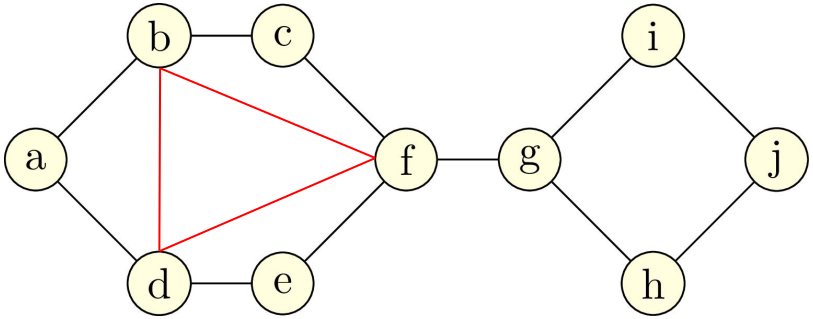
Constructing Chordal Supergraph



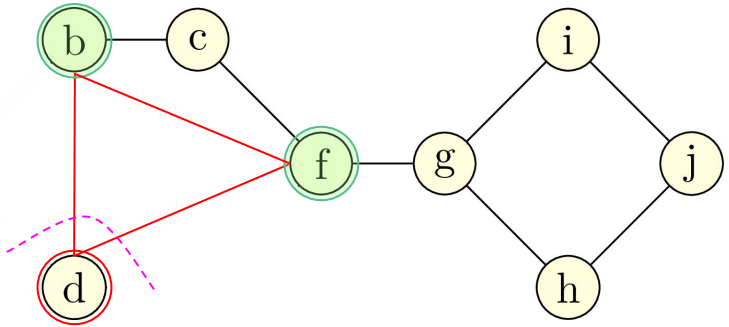
a, **e,** d, c, b, g, j, i, h, f



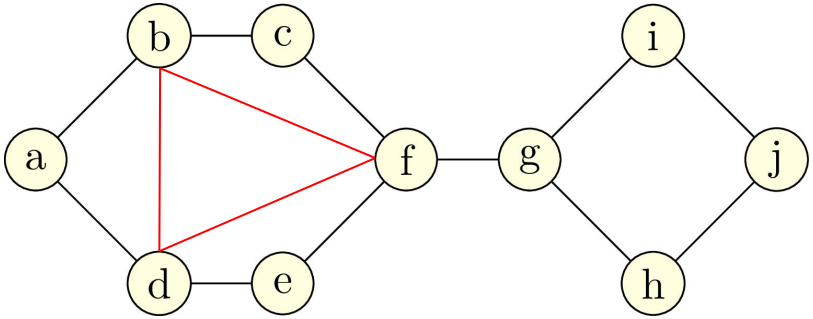
Constructing Chordal Supergraph



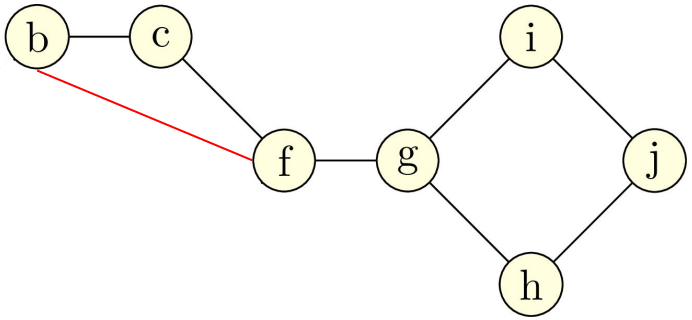
a, e, **d,** c, b, g, j, i, h, f



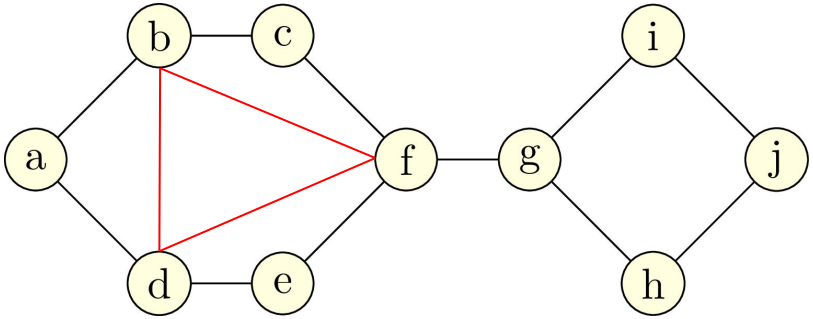
Constructing Chordal Supergraph



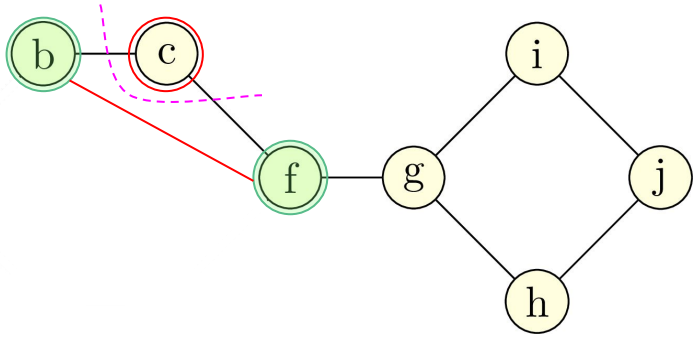
a, e, **d,** c, b, g, j, i, h, f



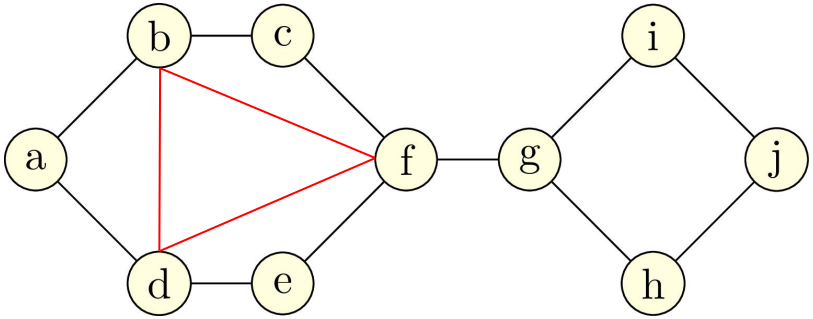
Constructing Chordal Supergraph



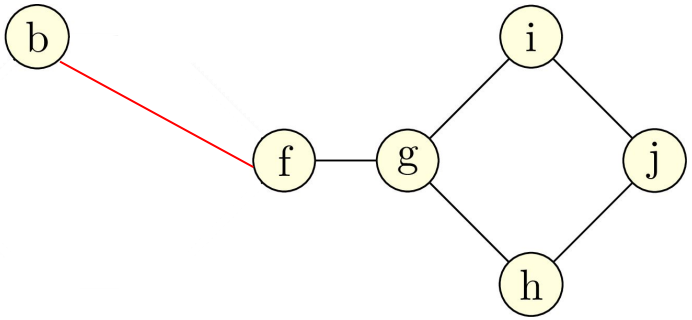
a, e, d, **c,** b, g, j, i, h, f



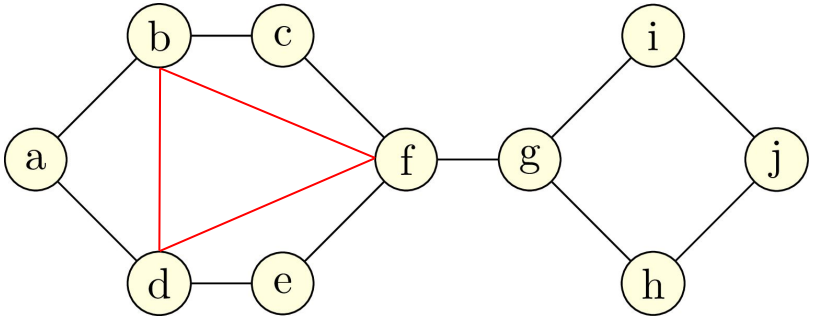
Constructing Chordal Supergraph



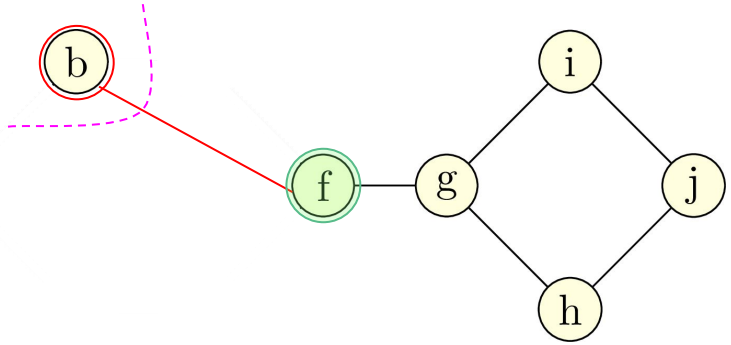
a, e, d, **c,** b, g, j, i, h, f



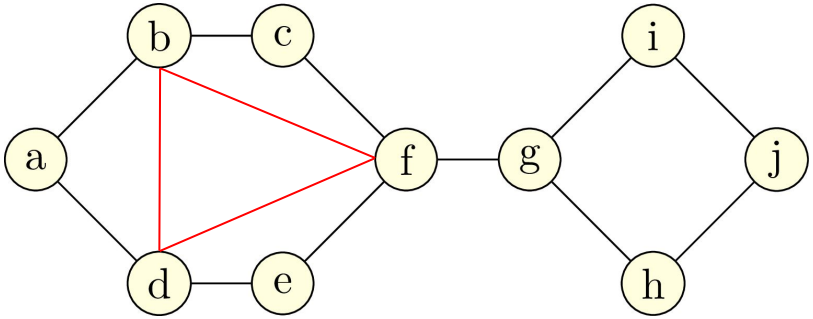
Constructing Chordal Supergraph



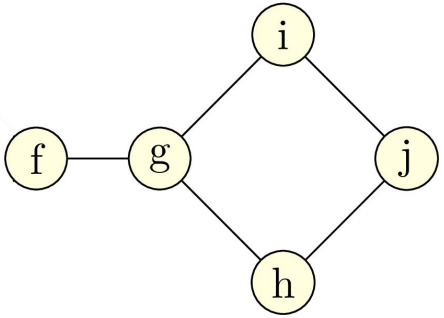
a, e, d, c, **b,** g, j, i, h, f



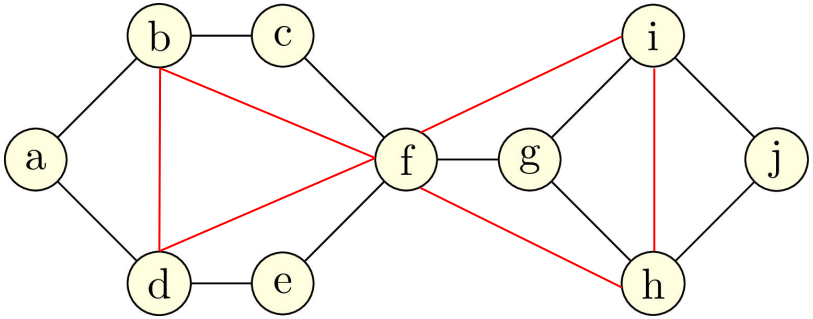
Constructing Chordal Supergraph



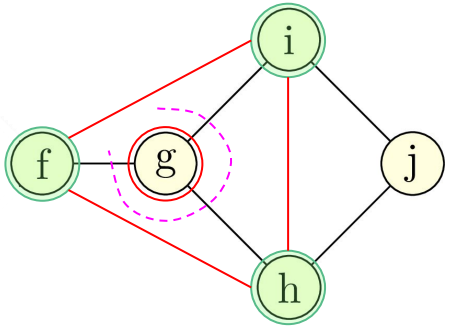
a, e, d, c, **b,** g, j, i, h, f



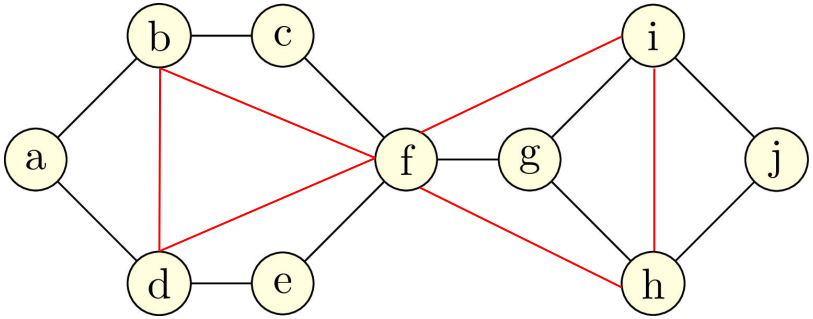
Constructing Chordal Supergraph



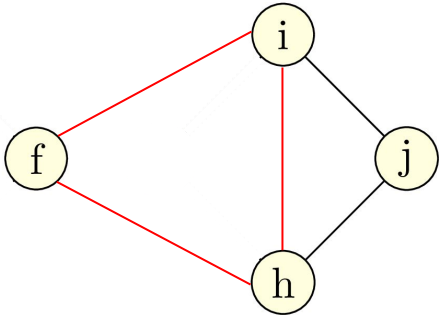
a, e, d, c, b, **g,** j, i, h, f



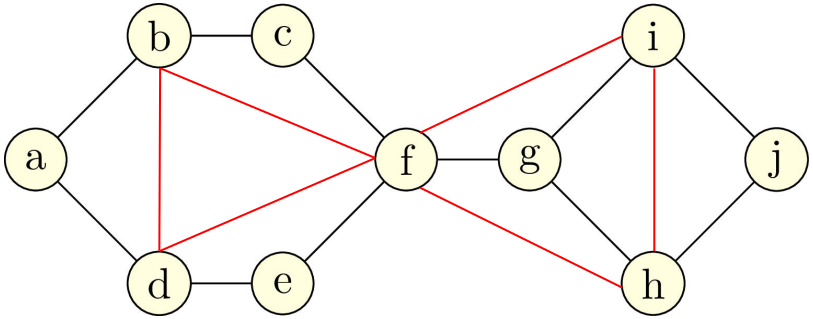
Constructing Chordal Supergraph



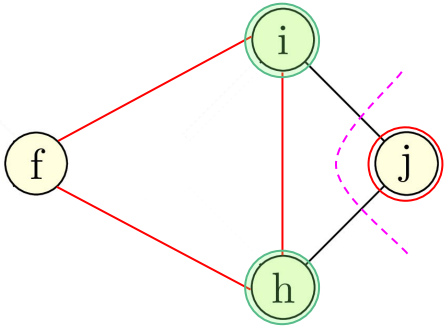
a, e, d, c, b, **g,** j, i, h, f



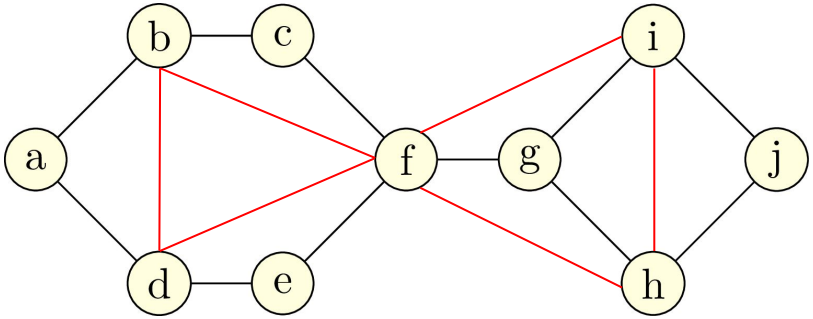
Constructing Chordal Supergraph



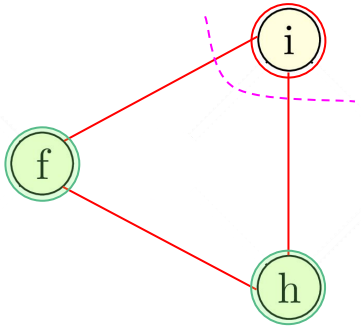
a, e, d, c, b, g, **j,** i, h, f



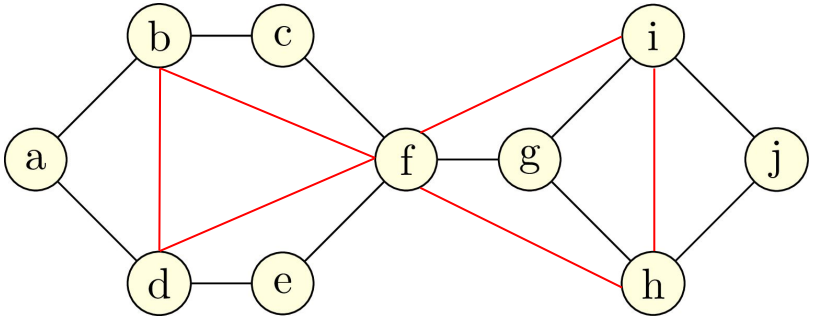
Constructing Chordal Supergraph



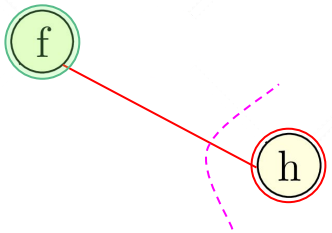
a, e, d, c, b, g, j, **i,** h, f



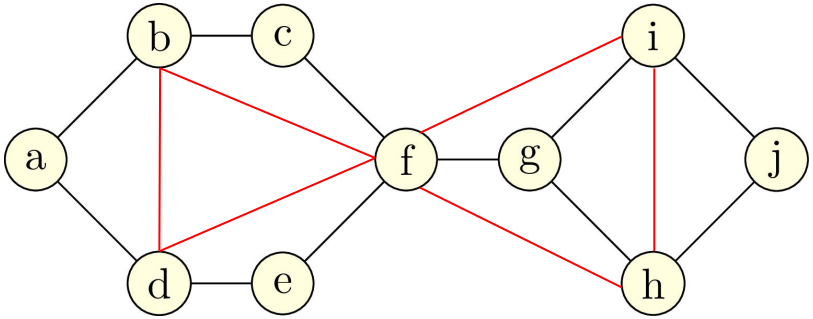
Constructing Chordal Supergraph



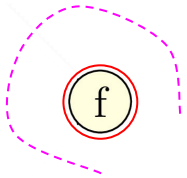
a, e, d, c, b, g, j, i, **h,** f



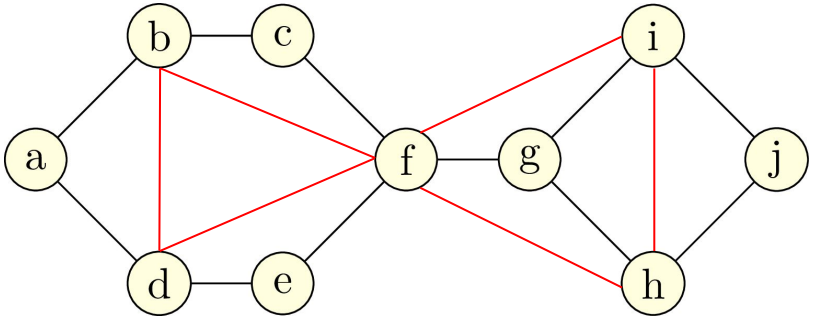
Constructing Chordal Supergraph



a, e, d, c, b, g, j, i, h, **f**



Step 3. Finding Maximal Cliques



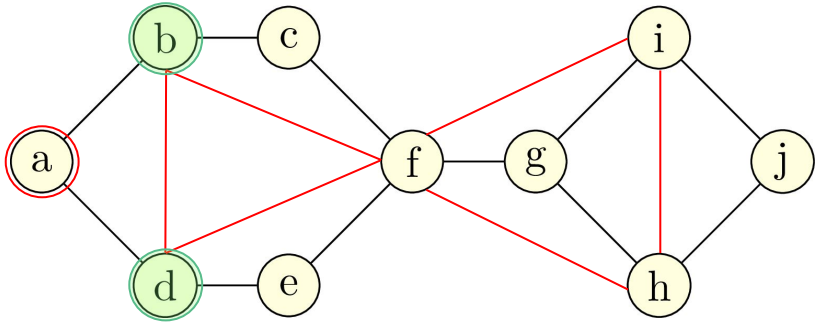
a, e, d, c, b, g, j, i, h, f

For each vertex ***v*** in the elimination ordering:

Find all neighbors N_v to its right

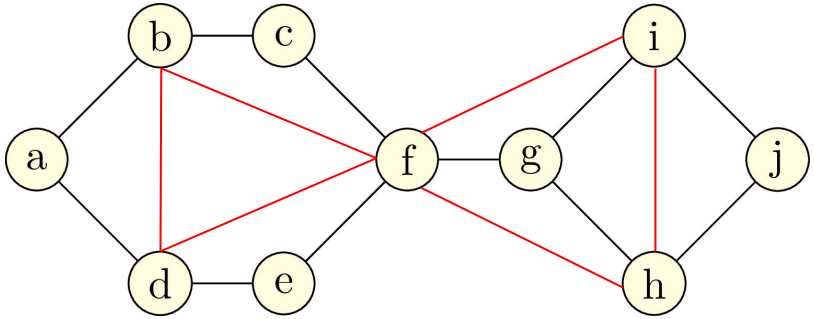
Clique = {**v** U $\mathbf{N_v}$}

Finding Maximal Cliques

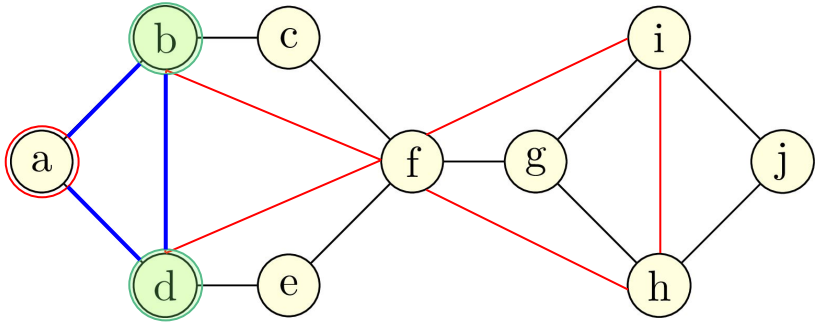


a, e, d, c, b, g, j, i, h, f

{}

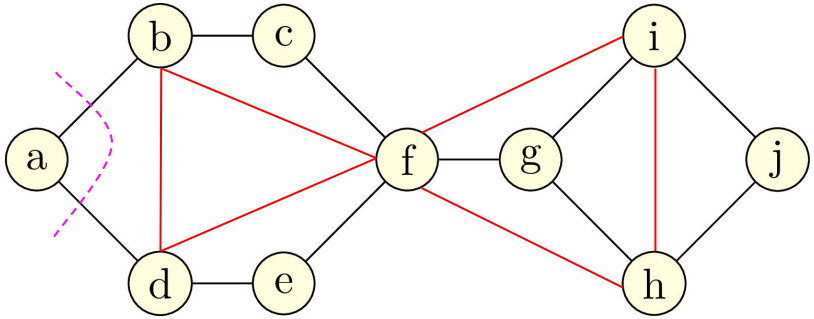


Finding Maximal Cliques

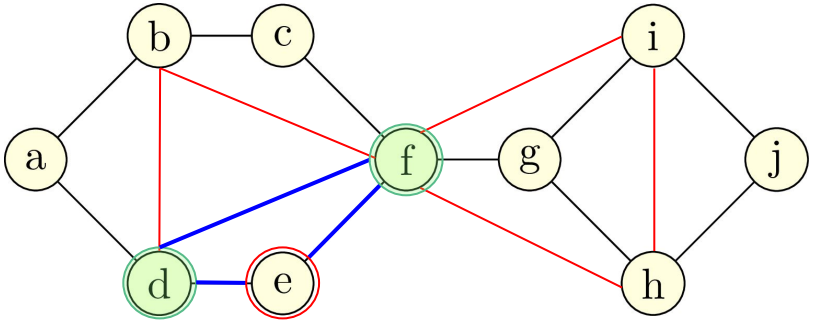


a, e, d, c, b, g, j, i, h, f

{abd, }

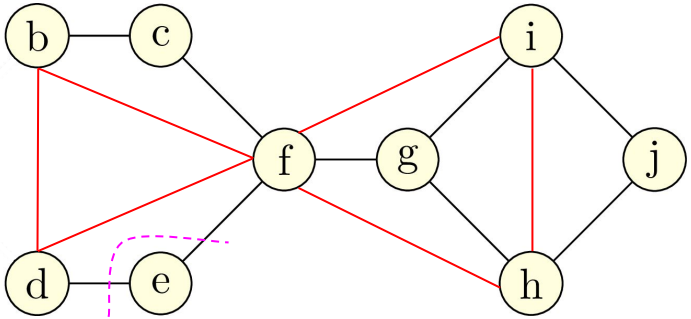


Finding Maximal Cliques

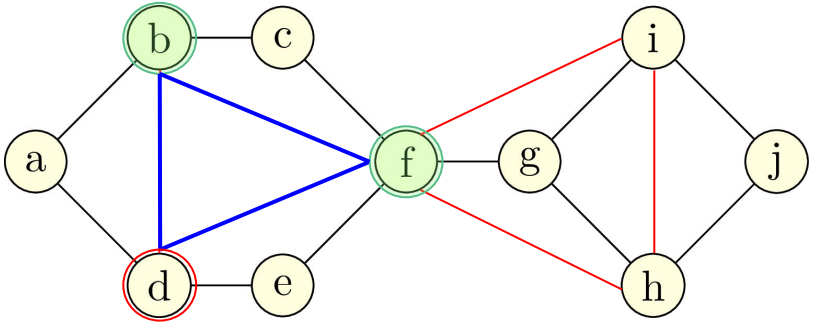


a, e, d, c, b, g, j, i, h, f

{abd, edf, }

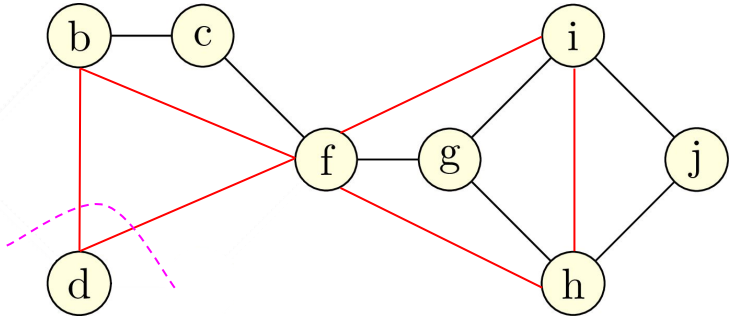


Finding Maximal Cliques

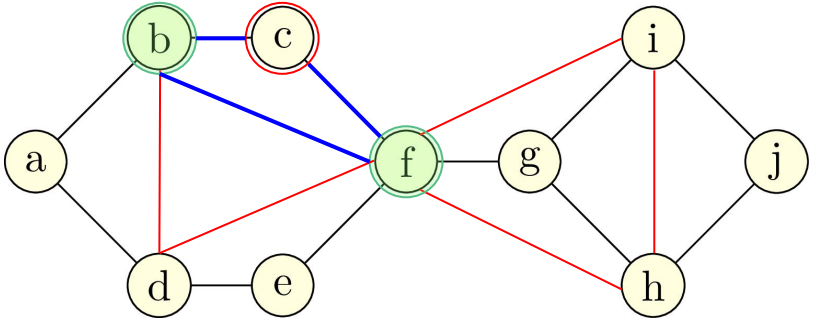


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, }

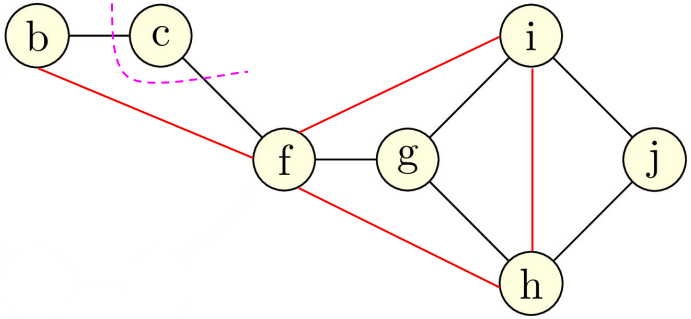


Finding Maximal Cliques

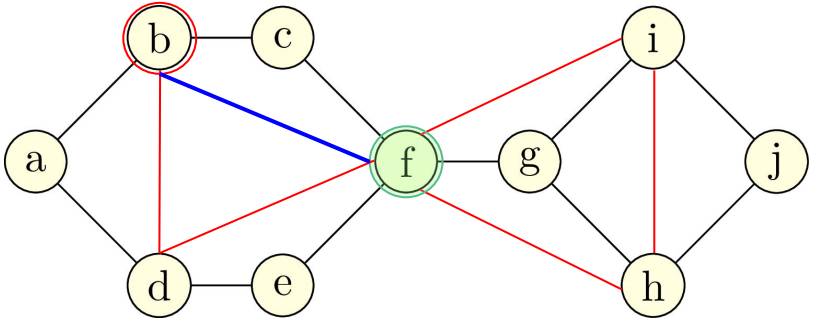


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, }

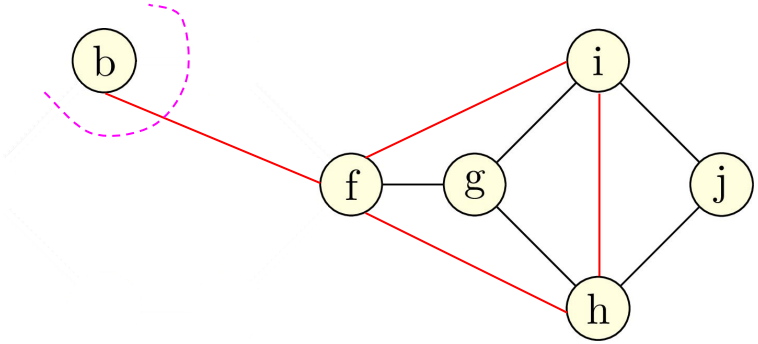


Finding Maximal Cliques

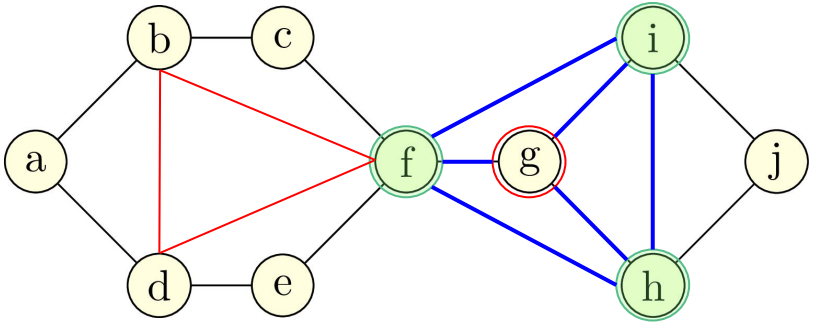


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, bf, }

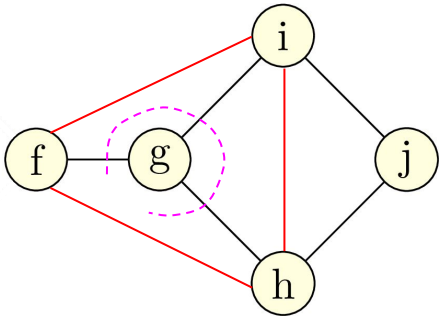


Finding Maximal Cliques

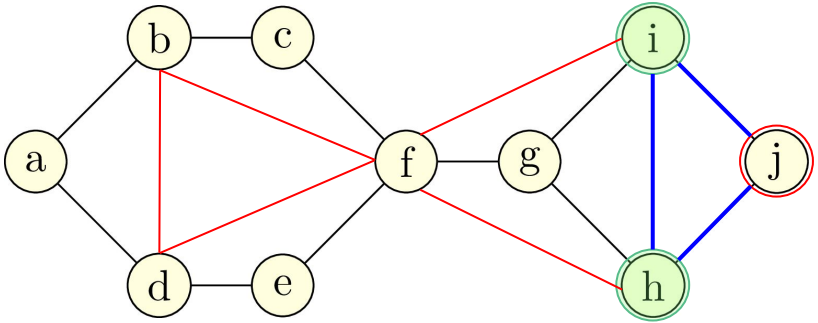


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, bf, ghif, }

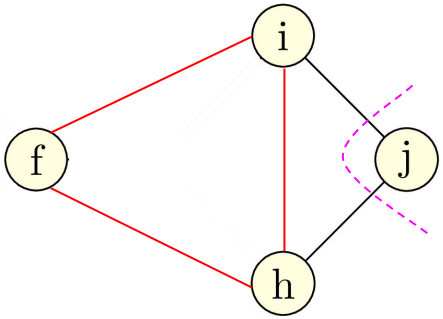


Finding Maximal Cliques

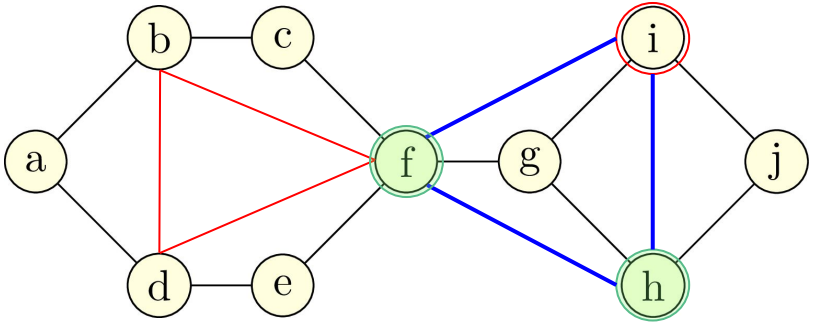


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, bf, ghif, jih, }

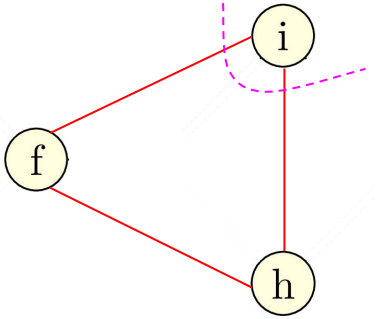


Finding Maximal Cliques

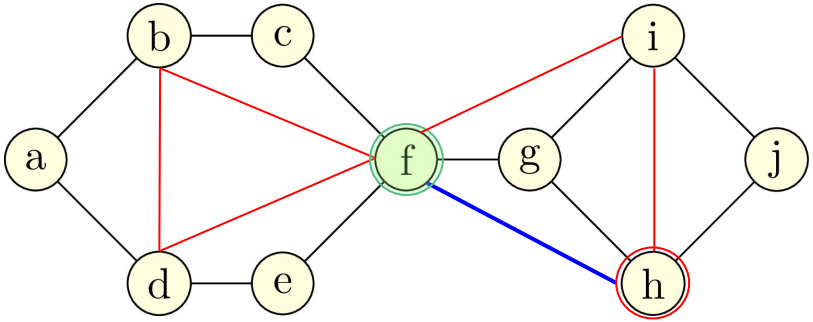


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, bf, ghif, jih, ihf, }

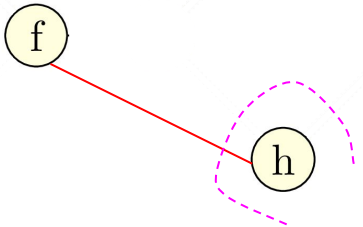


Finding Maximal Cliques

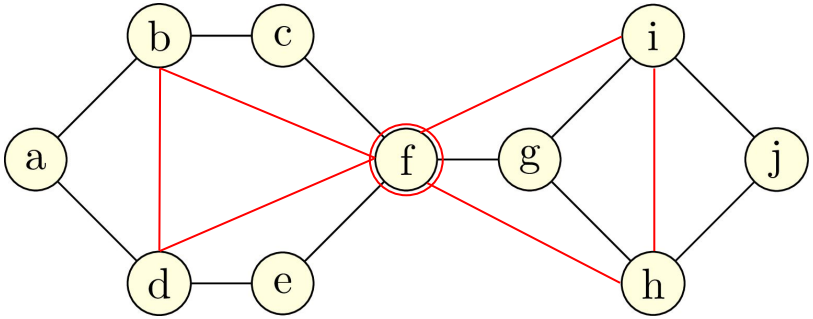


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, }

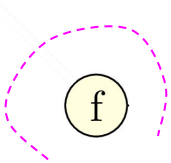


Finding Maximal Cliques

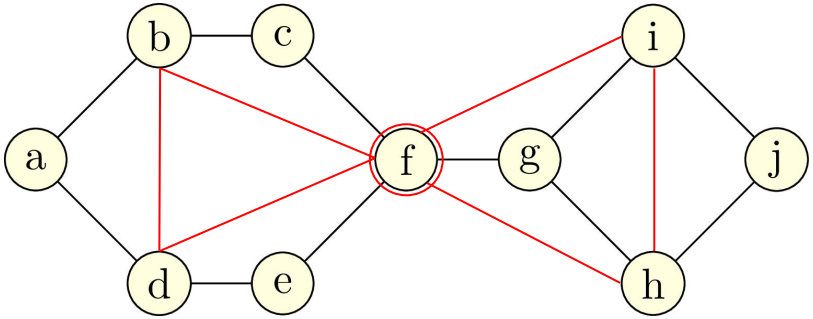


a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, }



Finding Maximal Cliques



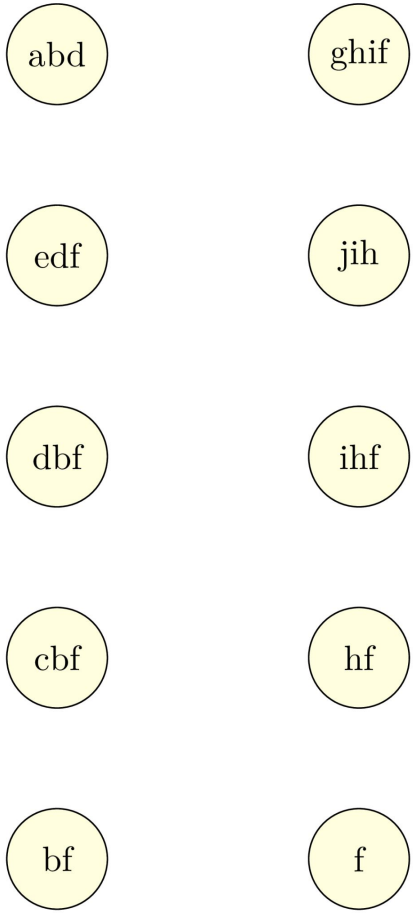
a, e, d, c, b, g, j, i, h, f

{abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }

Step 4. Building the Graph

- {abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }

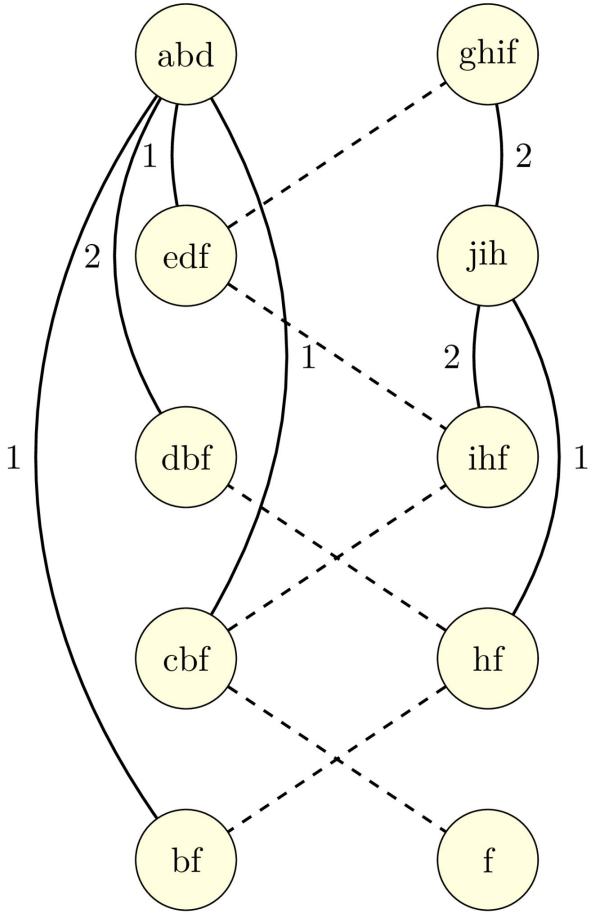
- For each element in the set:
  - Add a new vertex **v** to the graph
- For any pair of vertices (**u,v**):
  - Edge between **u** and **v** is added with weight **w** if **u** and **v** share **w** common characters



Building the Graph

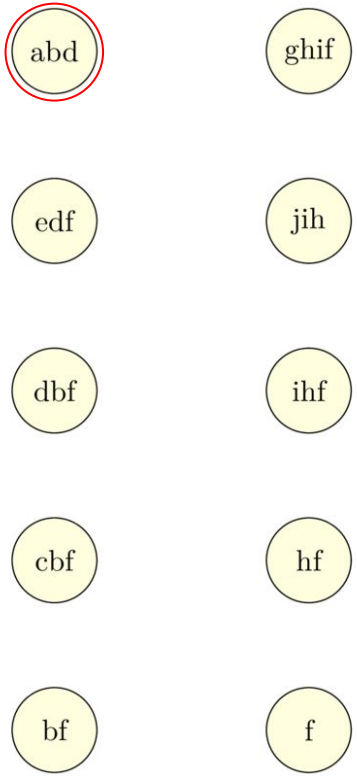
- {abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }

- For each element in the set:
  - Add a new vertex **v** to the graph
- For any pair of vertices (**u,v**):
  - Edge between **u** and **v** is added with weight **w** if **u** and **v** share **w** common characters



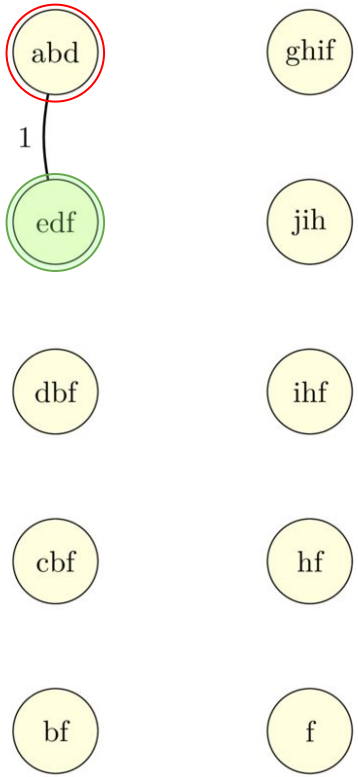
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



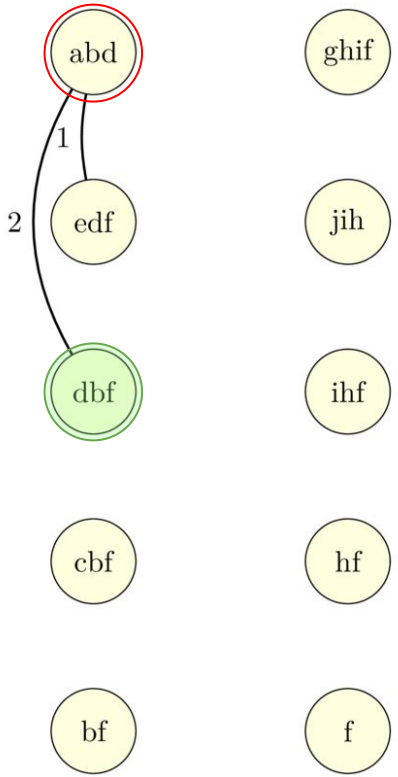
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



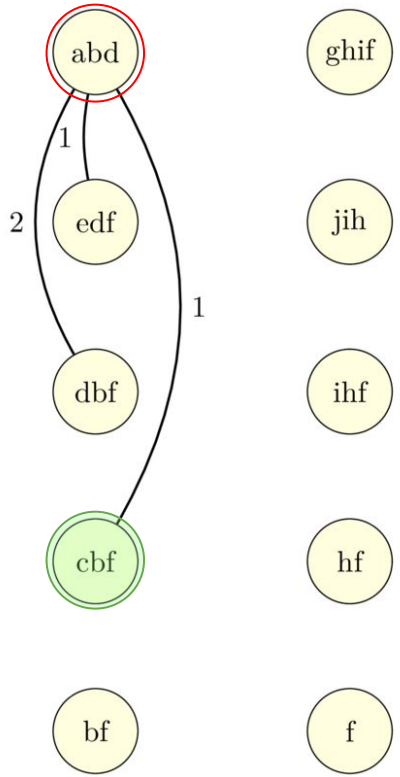
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



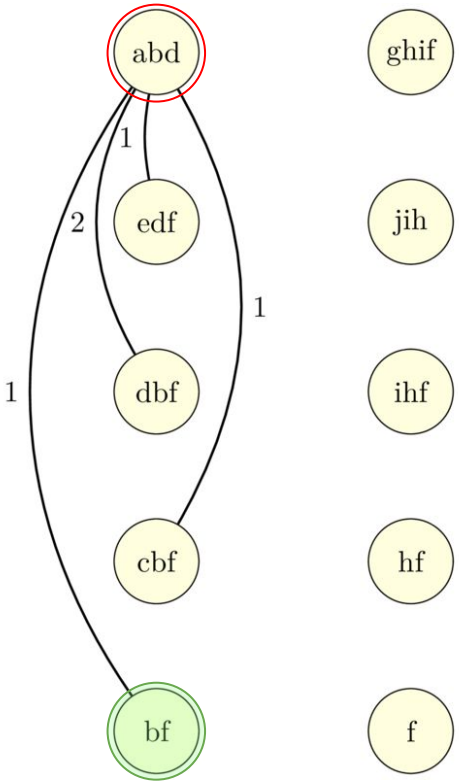
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



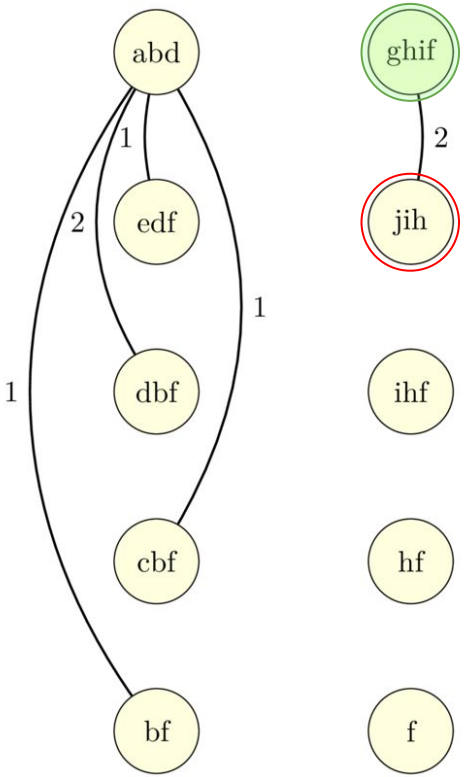
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



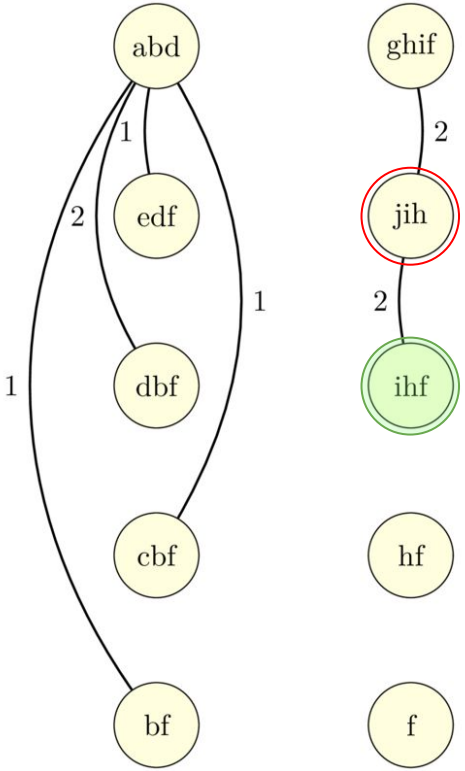
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



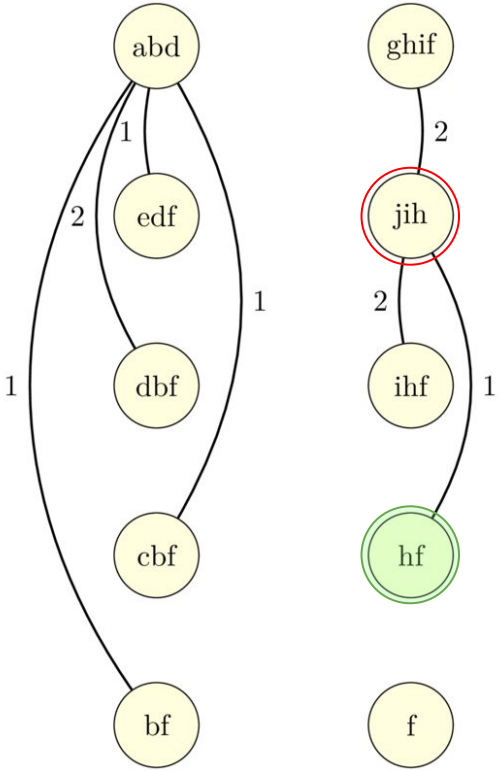
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



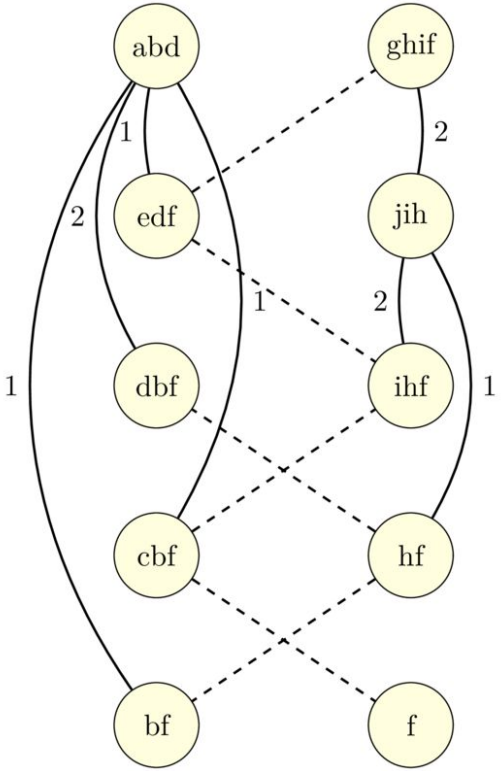
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



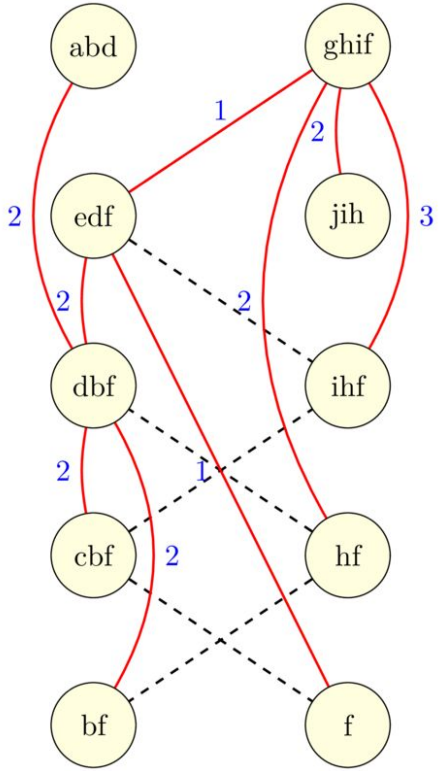
Building the Graph

{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }

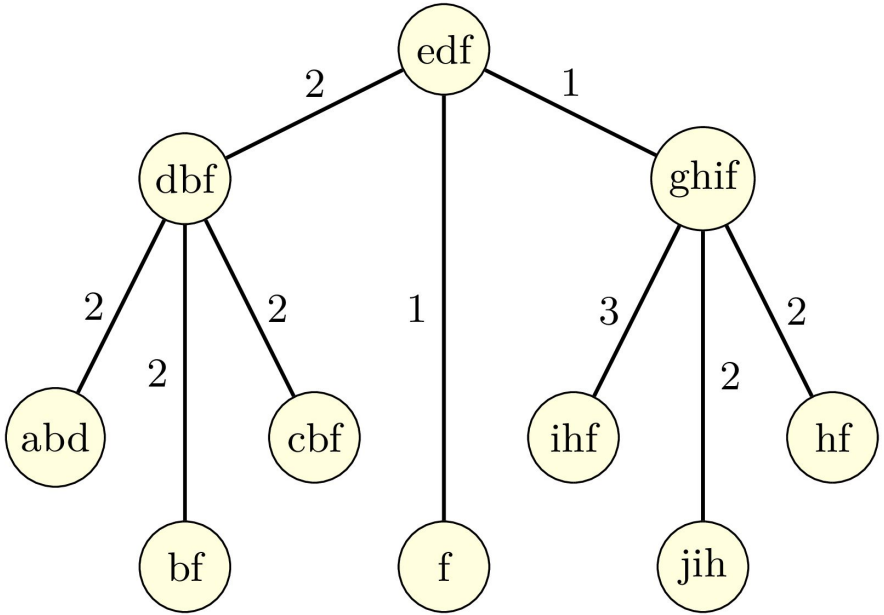


Step 5. Maximum Spanning Tree

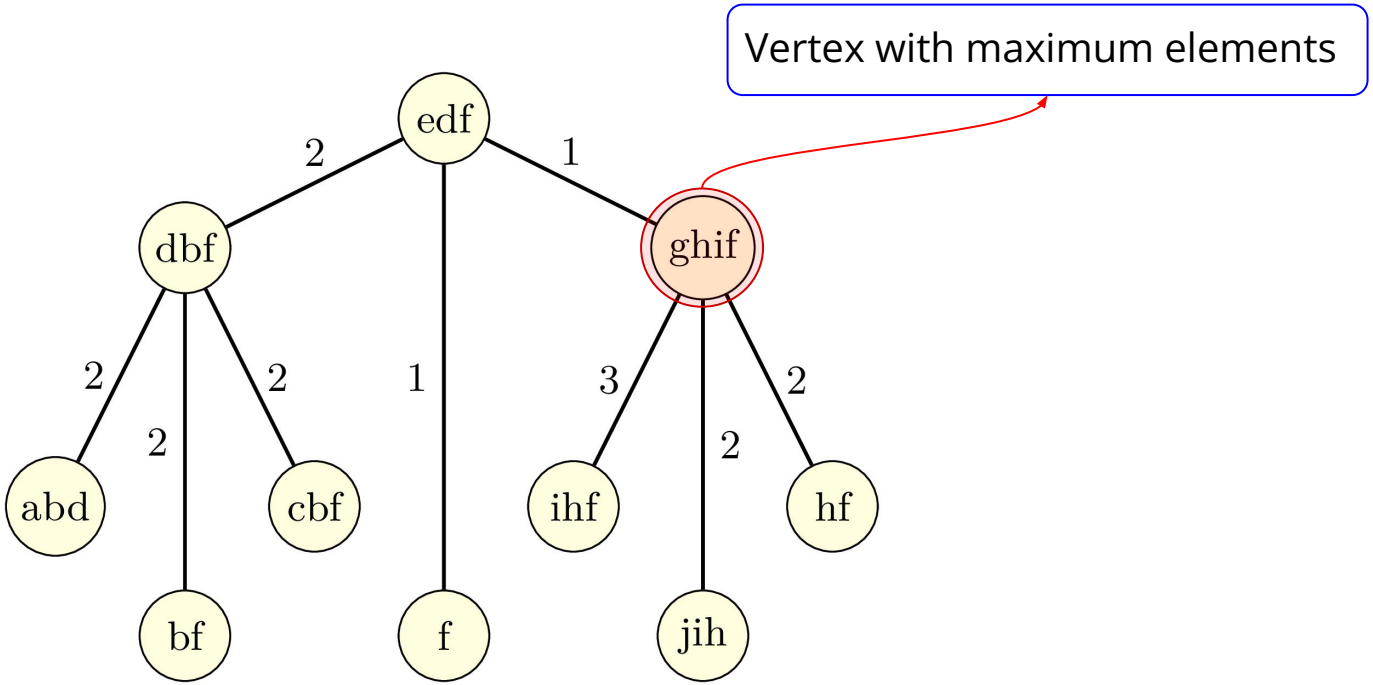
{ abd, edf, dbf, cbf, bf, ghif, jih, ihf, hf, f }



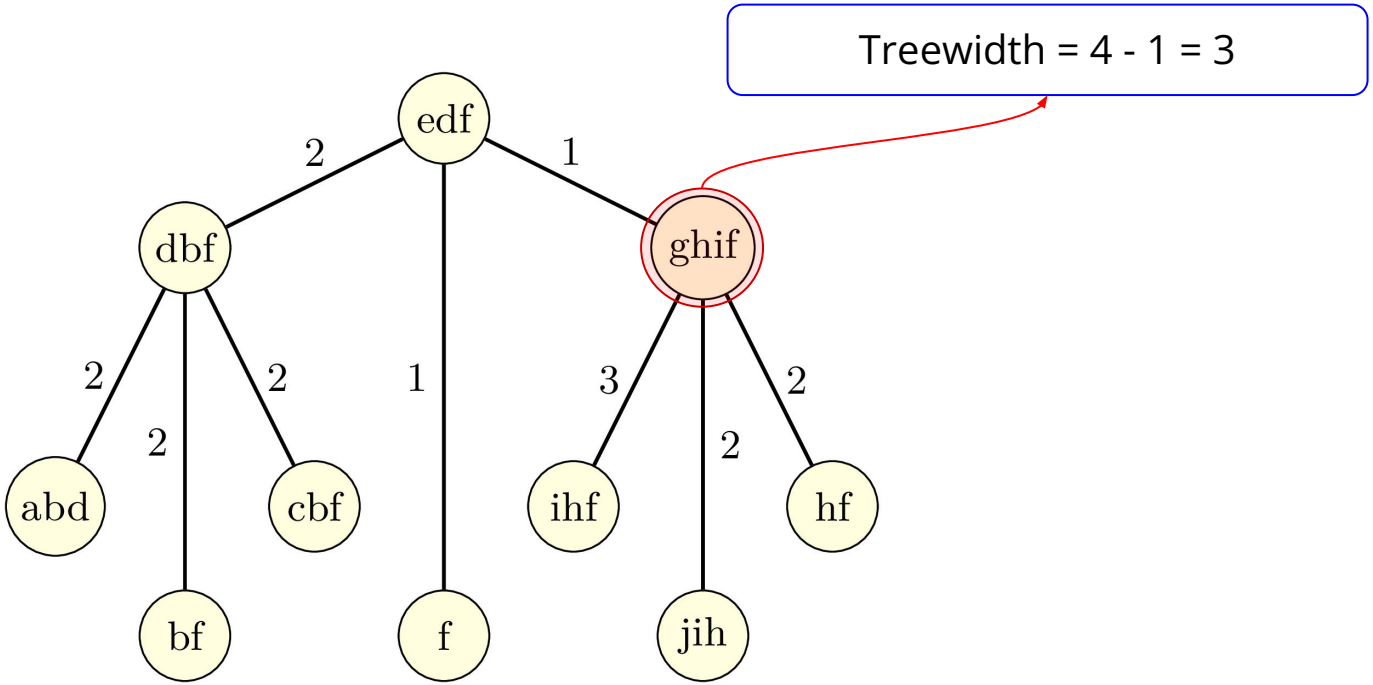
Maximum Spanning Tree



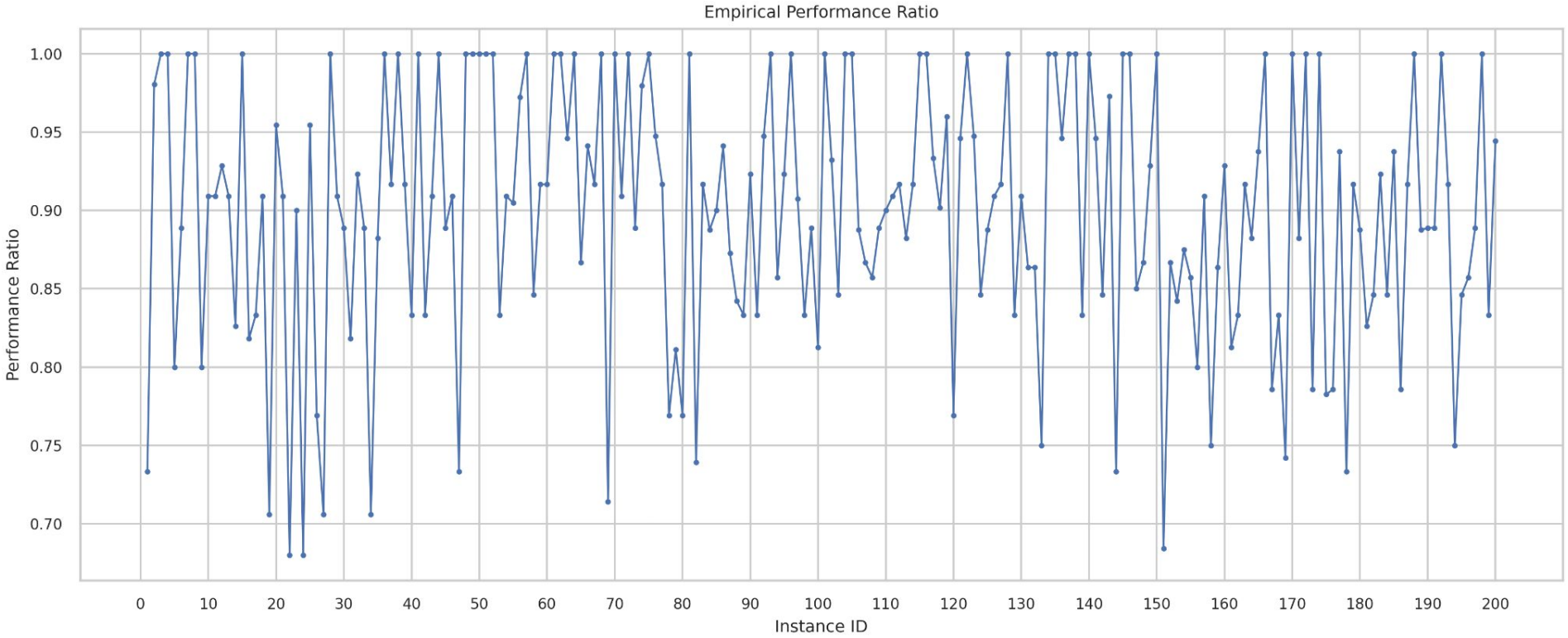
Computing the Treewidth



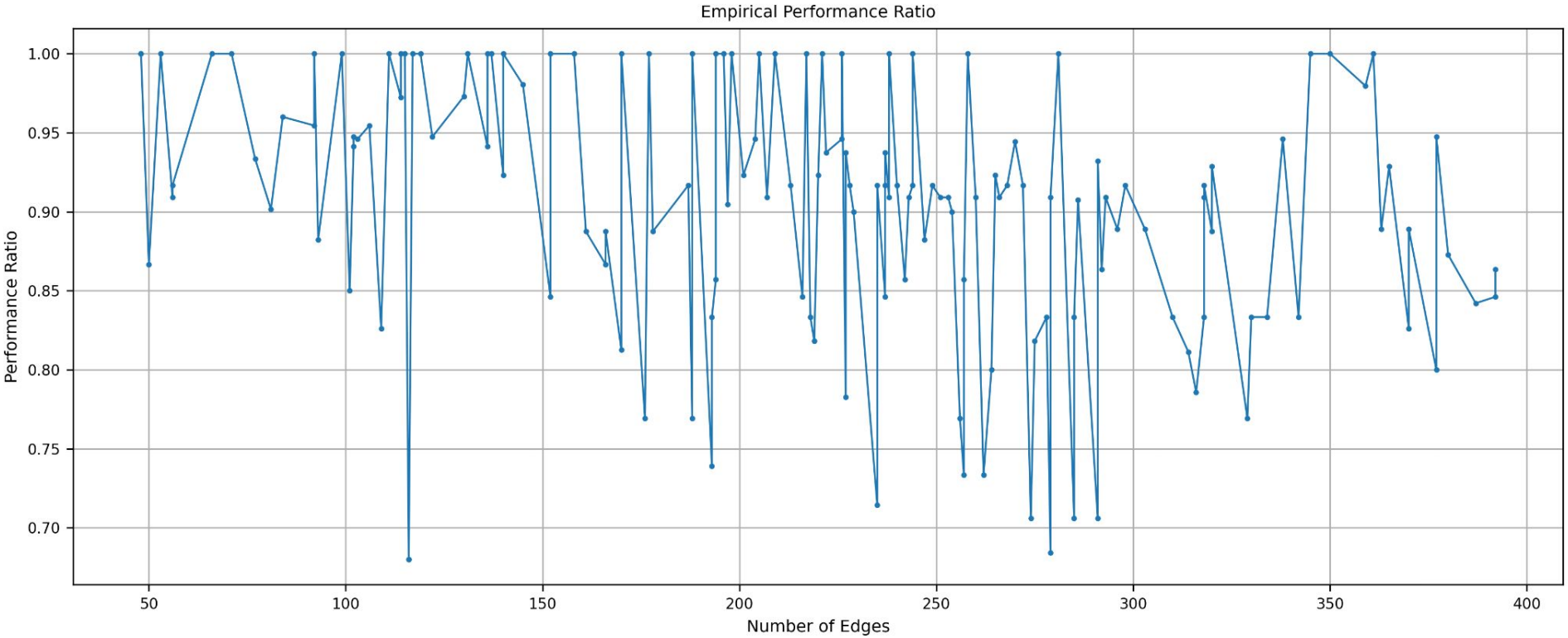
Computing the Treewidth



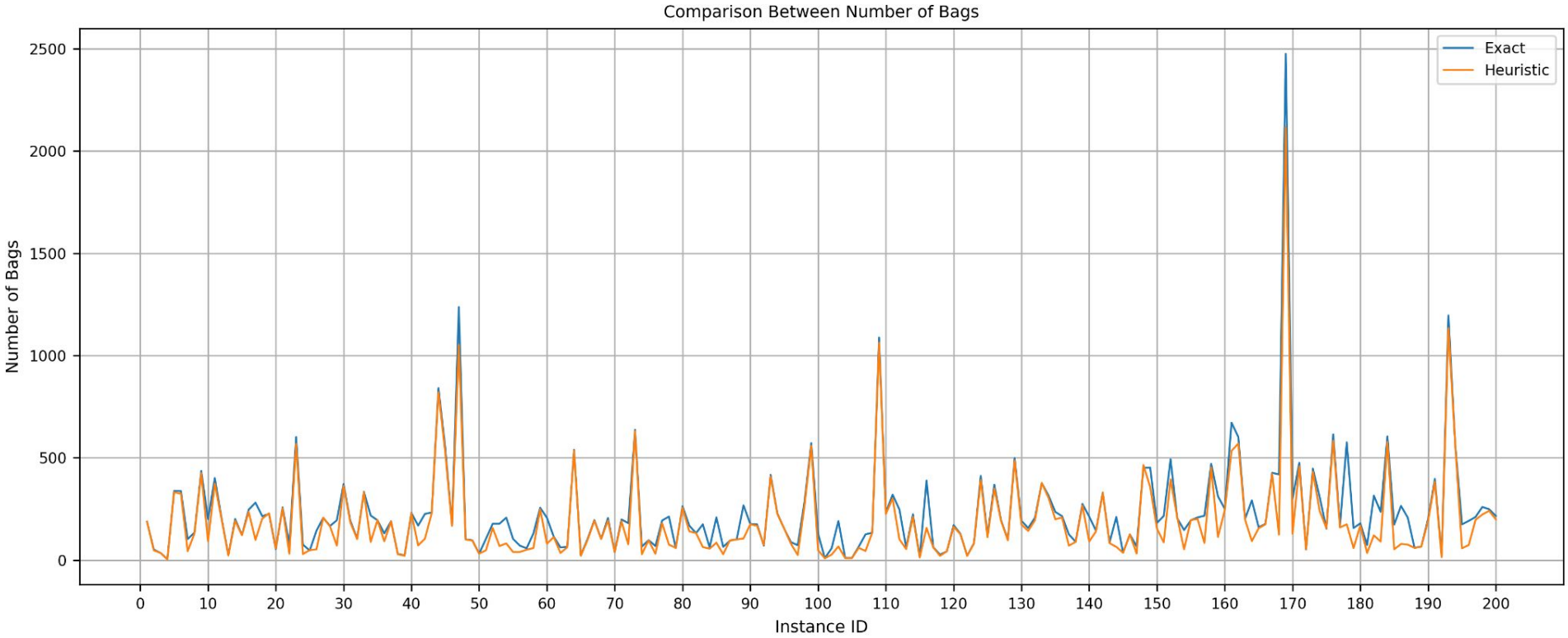
Results


Empirical Performance Ratio
Performance Ratio
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
Instance ID

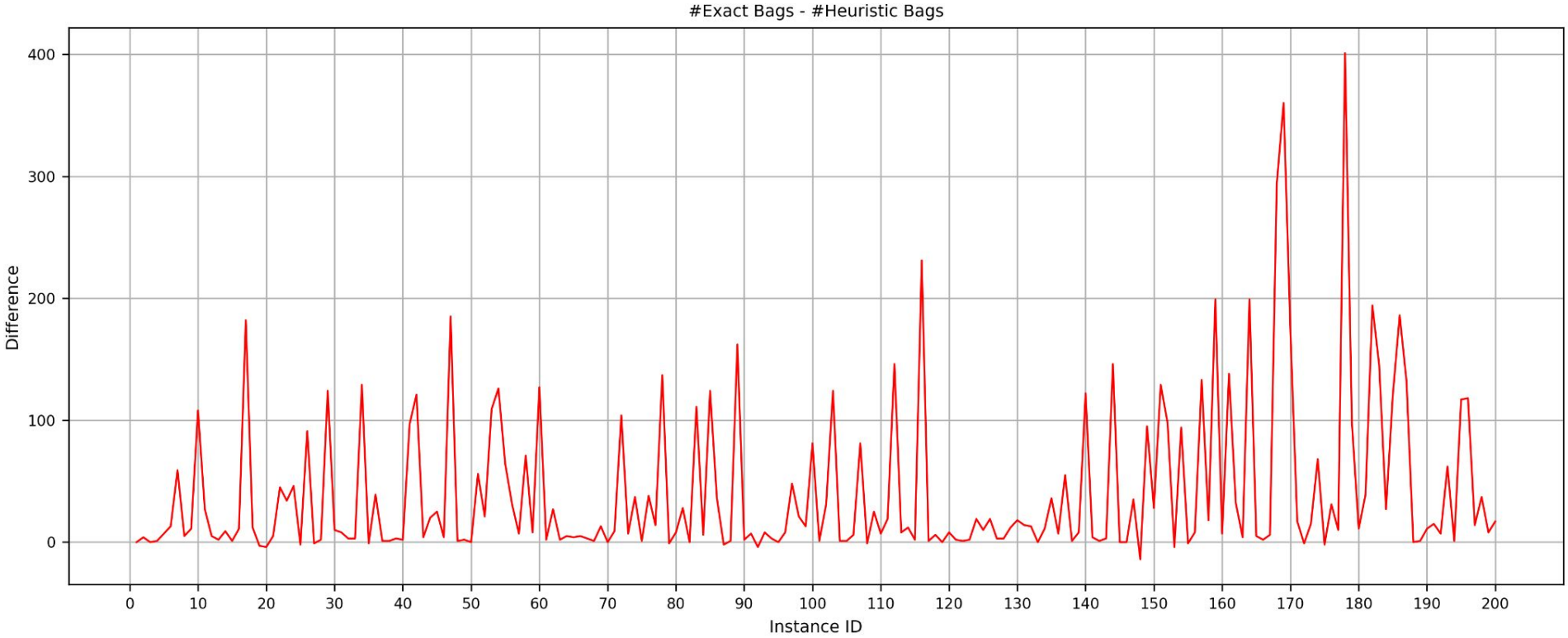
Results


Empirical Performance Ratio
Performance Ratio
1.00
0.95
0.90
0.85
0.80
0.75
0.70
50
100
150
200
250
300
350
400
Number of Edges

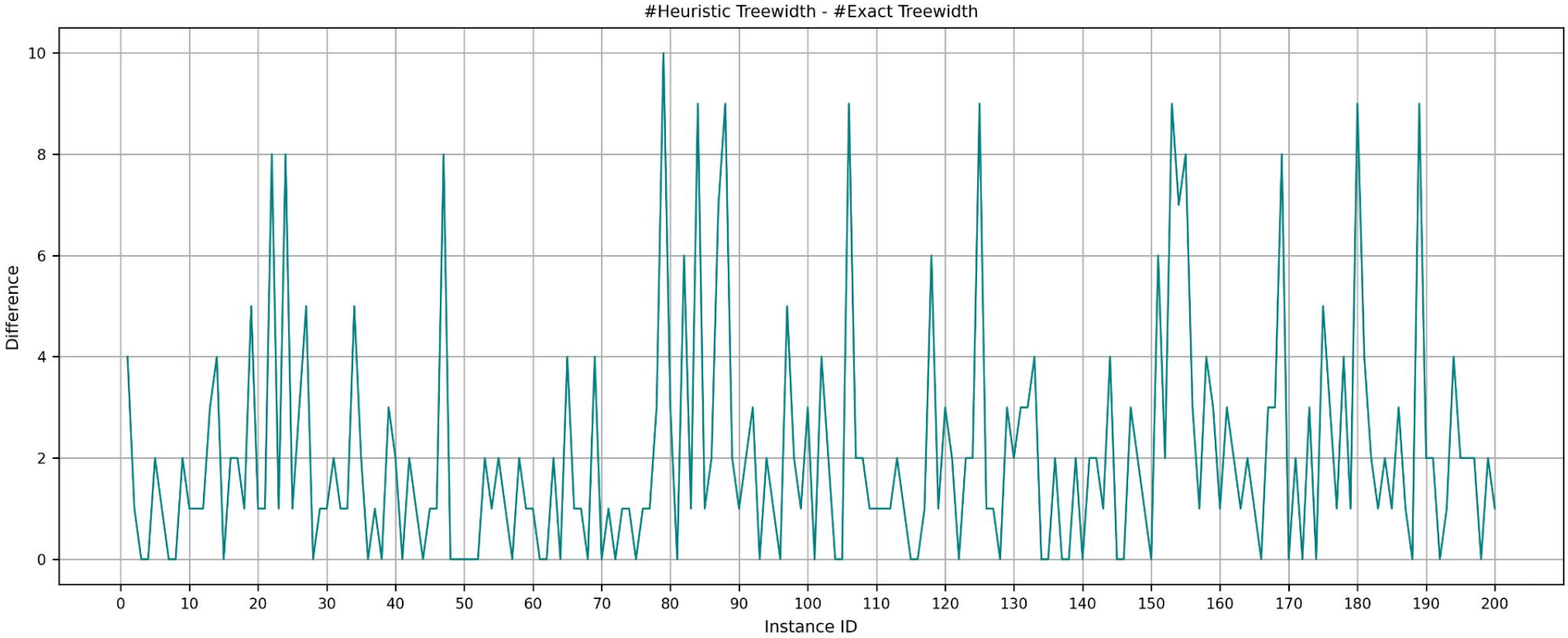
Results


Comparison Between Number of Bags
Exact
Heuristic
Number of Bags
Instance ID
0
500
1000
1500
2000
2500
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200

Results


#Exact Bags - #Heuristic Bags
Difference
Instance ID
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200

Results


#Heuristic Treewidth - #Exact Treewidth
Difference
Instance ID
0
2
4
6
8
10
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200

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
4. *“Parameterized Algorithms” (Textbook)* by Marek Cygan et al.