Max-Cut Above Spanning Tree is FPT

Jayakrishnan Madathil[1] Saket Saurabh[1,2] Meirav Zehavi[3]

[1]Institute of Mathematical Sciences, HBNI
Chennai, India

[2]Department of Informatics
University of Bergen, Norway

[3]Department of Computer Science
Ben-Gurion University, Israel

CSR, 2018

Outline

Cut of a graph G

Definition

- *A cut* of G is a function

$$f : V(G) \to \{0, 1\}$$

.

Cut of a graph G
Definition

- *A cut* of G is a function
$$f : V(G) \to \{0, 1\}$$
.
- Size of the cut f,
$$||f|| = |\{uv \in E(G) \mid f(u) \neq f(v)\}|$$
.

Max-Cut
Definition

- Max-Cut

  Input: A graph G and a non-negative integer k.

  Question: Does G have a cut of size at least k?

Max-Cut
Definition

- Max-Cut

  Input: A graph G and a non-negative integer k.

  Question: Does G have a cut of size at least k?

- Max-Cut is NP-hard.

Outline

1. Introduction
   - Problem Statement and Results
   - Lower Bounds for Cut Size
   - Parameterizing MAX-CUT
2. FPT Algorithm
3. Polynomial Kernel

Problem Statement and Results

- Max-Cut Above Spanning Tree (Max-Cut-AST)

  Input: A connected n-vertex graph G and a non-negative integer k.

  Parameter: k

  Question: Does G have a cut of size at least $n - 1 + k$?

Problem Statement and Results

- Max-Cut Above Spanning Tree (Max-Cut-AST)

  Input: A connected n-vertex graph G and a non-negative integer k.

  Parameter: k

  Question: Does G have a cut of size at least $n - 1 + k$?
- **Results:**
  - $8^k n^{\mathcal{O}(1)}$ algorithm and $\mathcal{O}(k^5)$ kernel.
  - No $2^{o(k)}$ algorithm unless the Exponential Time Hypothesis fails.

Outline

Lower Bounds for Cut Size

- Graph G, $|V(G)| = n$, $|E(G)| = m$.

Lower Bounds for Cut Size

- Graph G, $|V(G)| = n$, $|E(G)| = m$.
- G has a cut of size at least
  1. $m/2$ [Erdős, 1965].

Lower Bounds for Cut Size

- Graph G, $|V(G)| = n$, $|E(G)| = m$.
- G has a cut of size at least
  1. $m/2$ [Erdős, 1965].
  2. $m/2 + (n-1)/4$ if G is connected [Edwards, 1975] .
     Edwards-Erdős bound.

Lower Bounds for Cut Size

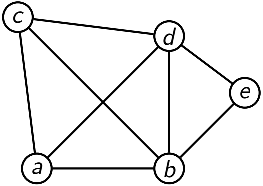
- Graph G, $|V(G)| = n$, $|E(G)| = m$.
- G has a cut of size at least
  1. $m/2$ [Erdős, 1965].
  2. $m/2 + (n-1)/4$ if G is connected [Edwards, 1975] .
     Edwards-Erdős bound.
  3. $n - 1$ if G is connected.
     Spanning tree bound.

Spanning Tree Bound

- Connected graph G, $|V(G)| = n$.
- Cut of size at least $n - 1$.

Spanning Tree Bound

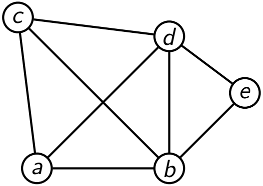
- Connected graph G, $|V(G)| = n$.
- Cut of size at least $n - 1$.



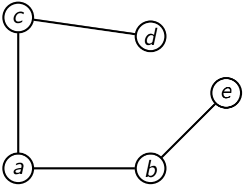
$n = 5$

Spanning Tree Bound

- Connected graph G, $|V(G)| = n$.
- Cut of size at least $n - 1$.



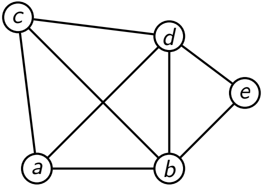
$n = 5$



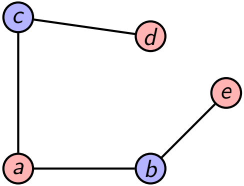
Spanning Tree
$n - 1$ edges

Spanning Tree Bound

- Connected graph G, $|V(G)| = n$
- Cut of size at least $n - 1$



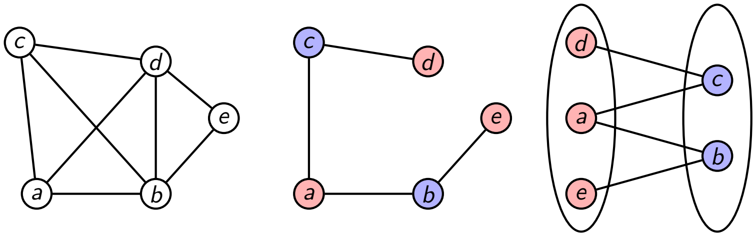
$n = 5$



Spanning tree
$n - 1$ edges
2-colorable

Spanning Tree Bound

- Connected graph G, $|V(G)| = n$.
- Cut of size at least $n - 1$.



$n = 5$

Spanning tree
$n - 1$ edges
2-colorable

Cut of size $n - 1$

Edwards-Erdős Bound vs. Spanning Tree Bound

- Connected graph G, $|V(G)| = n$, $|E(G)| = m$.
- **Edwards-Erdős Bound:** $m/2 + (n-1)/4$.
- **Spanning Tree Bound:** $n - 1$.

Edwards-Erdős Bound vs. Spanning Tree Bound

- Connected graph G, $|V(G)| = n$, $|E(G)| = m$.
- **Edwards-Erdős Bound:** $m/2 + (n-1)/4$.
- **Spanning Tree Bound:** $n - 1$.
- Spanning Tree bound gives a better guarantee for cut size on *sparse graphs*.
- $n - 1 > m/2 + (n-1)/4 \iff (\text{average degree of } G) < 3$.

Outline

1. Introduction
   - Problem Statement and Results
   - Lower Bounds for Cut Size
   - Parameterizing Max-Cut
2. FPT Algorithm
3. Polynomial Kernel

Parameterizing Max-Cut

- Parameterize by cut size?

Parameterizing MAX-CUT

- Parameterize by cut size?

  Input: A graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least k?

Parameterizing MAX-CUT

- Parameterize by cut size?

  Input: A graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least k?
- Trivially FPT

Parameterizing MAX-CUT

- Parameterize by cut size?

  Input: A graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least k?

- Trivially FPT
  - $k \leq m/2 \implies$ yes.

Parameterizing MAX-CUT

- Parameterize by cut size?

  Input: A graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least k?

- Trivially FPT
  - $k \leq m/2 \implies$ yes.
  - $m \leq 2k$. Brute force.

Parameterizing MAX-CUT

- Parameterize by cut size?

  Input: A graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least k?
- Trivially FPT
  - $k \leq m/2 \implies$ yes.
  - $m \leq 2k$. Brute force.
- Parameterize above the cut size [Mahajan and Raman, 1997]

Parameterizing Max-Cut

- Parameterize by cut size?

  Input: A graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least k?
- Trivially FPT
  - $k \leq m/2 \implies$ yes.
  - $m \leq 2k$. Brute force.

- Parameterize above the cut size [Mahajan and Raman, 1997]

  Question: Does G have a cut of size at least $m/2 + k$?

Parameterizing Max-Cut

- Parameterize by cut size?

  Input: A graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least k?

- Trivially FPT
  - $k \leq m/2 \implies$ yes.
  - $m \leq 2k$. Brute force.

- Parameterize above the cut size [Mahajan and Raman, 1997]

  Question: Does G have a cut of size at least $m/2 + k$?

- FPT.
- *Above Guarantee* parameterization.

Max-Cut: Above Guarantee Parameterizations

- Max-Cut Above Edwards-Erdős (Max-Cut-AEE)

Input: A connected graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

Parameter: k

Question: Does G have a cut of size at least $m/2 + (n-1)/4 + k$?

Max-Cut: Above Guarantee Parameterizations

- Max-Cut Above Edwards-Erdős (Max-Cut-AEE)

  Input: A connected graph G and a positive integer k, $|V(G)| = n$, $|E(G)| = m$.

  Parameter: k

  Question: Does G have a cut of size at least $m/2 + (n-1)/4 + k$?

- **Results:**
  - $8^k n^{\mathcal{O}(1)}$ algorithm and $\mathcal{O}(k^5)$ kernel [Crowston et al., 2012]
  - Extended to signed graphs with an $\mathcal{O}(k)$ kernel [Etscheid and Mnich, 2016]

Max-Cut Above Spanning Tree (Max-Cut-AST)

Our Problem

- Max-Cut Above Spanning Tree (Max-Cut-AST)

  Input: A connected graph G and a positive integer k, $|V(G)| = n$.

  Parameter: k

  Question: Does G have a cut of size at least $n - 1 + k$?

Max-Cut Above Spanning Tree (Max-Cut-AST)

Our Problem

- Max-Cut Above Spanning Tree (Max-Cut-AST)

  Input: A connected graph G and a positive integer k, $|V(G)| = n$.

  Parameter: k

  Question: Does G have a cut of size at least $n - 1 + k$?

- **Results:**
  - $8^k n^{\mathcal{O}(1)}$ algorithm and $\mathcal{O}(k^5)$ kernel.
  - No $2^{o(k)}$ algorithm.

Algorithm
Strategy

- In polynomial time

  either conclude that (G, k) is a yes-instance

Algorithm
Strategy

- In polynomial time

  either conclude that (G, k) is a yes-instance
  or find a small set S such that $G - S$ has a nice structure.

Algorithm

Strategy

- In polynomial time

  either conclude that (G, k) is a yes-instance
  or find a small set S such that $G - S$ has a nice structure.
- Guess the optimal partition of S.
- Optimally extend each guess to a partition of $G - S$.

Algorithm

Strategy

- In polynomial time

  either conclude that (G, k) is a yes-instance
  or find a small set S such that $G - S$ has a nice structure.
- Guess the optimal partition of S.
- Optimally extend each guess to a partition of $G - S$.

  Define an auxiliary problem on $G - S$.

Algorithm
Strategy

- In polynomial time
  either conclude that (G, k) is a yes-instance
  or find a small set S such that $G - S$ has a nice structure.
- Guess the optimal partition of S.
- Optimally extend each guess to a partition of $G - S$.
  Define an auxiliary problem on $G - S$.
  Solve it in polynomial time by exploiting $G - S$'s nice structure.

Outline of Algorithm

- Apply a set of one-way reduction rules.

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \to (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \rightarrow (G', k')$ such that
  - (G', k') is a yes-instance $\Longrightarrow$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \rightarrow (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:
  - Apply if [some condition] holds.

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \rightarrow (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:
  - Apply if [some condition] holds.
  - *Delete* a set of vertices X.

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \to (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:
  - Apply if [some condition] holds.
  - *Delete* a set of vertices X.
  - *Mark* a set of vertices A.

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \to (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:
  - Apply if [some condition] holds.
  - *Delete* a set of vertices X.
  - *Mark* a set of vertices A.
  - *Decrement* k appropriately.

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \to (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:
  - Apply if [some condition] holds.
  - *Delete* a set of vertices X.
  - *Mark* a set of vertices A.
  - *Decrement* k appropriately.
- Guarantees:

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \to (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:
  - Apply if [some condition] holds.
  - *Delete* a set of vertices X.
  - *Mark* a set of vertices A.
  - *Decrement* k appropriately.
- Guarantees:
  - G' is connected.

Outline of Algorithm

- Apply a set of one-way reduction rules.
- One-way rule: $(G, k) \to (G', k')$ such that
  - (G', k') is a yes-instance $\implies$ (G, k) is a yes-instance.
  - Converse need not hold.
- Generic Reduction Rule:
  - Apply if [some condition] holds.
  - *Delete* a set of vertices X.
  - *Mark* a set of vertices A.
  - *Decrement* k appropriately.
- Guarantees:
  - G' is connected.
  - $|A| \leq 3$
  - $A \neq \emptyset \implies k$ drops by at least 1.

Outline of Algorithm
Contd.

- Rules apply as long as $G' \neq K_1$.

Outline of Algorithm

Contd.

- Rules apply as long as $G' \neq K_1$.
- If $k' \leq 0$, then (G, k) is a yes-instance.

Outline of Algorithm
Contd.

- Rules apply as long as $G' \neq K_1$.
- If $k' \leq 0$, then (G, k) is a yes-instance.
- Otherwise, at most $3k$ vertices are marked.
  Let $S =$ set of marked vertices.

Outline of Algorithm
Contd.

- Rules apply as long as $G' \neq K_1$.
- If $k' \leq 0$, then (G, k) is a yes-instance.
- Otherwise, at most $3k$ vertices are marked.
  Let S = set of marked vertices.
- $G - S$ is a clique-cycle-forest.

Outline of Algorithm
Contd.

- Rules apply as long as $G' \neq K_1$.
- If $k' \leq 0$, then (G, k) is a yes-instance.
- Otherwise, at most $3k$ vertices are marked.
  Let S = set of marked vertices.
- $G - S$ is a clique-cycle-forest. Every block (2-connected component) of $G - S$ is a clique or a cycle.

Outline of Algorithm
Contd.

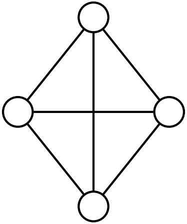
- Rules apply as long as $G' \neq K_1$.
- If $k' \leq 0$, then (G, k) is a yes-instance.
- Otherwise, at most $3k$ vertices are marked.
  Let S = set of marked vertices.
- $G - S$ is a clique-cycle-forest. Every block (2-connected component) of $G - S$ is a clique or a cycle.
- Guess the partition of S. Optimally extend it to $G - S$.

Outline of Algorithm
Contd.

- Rules apply as long as $G' \neq K_1$.
- If $k' \leq 0$, then (G, k) is a yes-instance.
- Otherwise, at most $3k$ vertices are marked.
  Let $S =$ set of marked vertices.
- $G - S$ is a clique-cycle-forest. Every block (2-connected component) of $G - S$ is a clique or a cycle.
- Guess the partition of S. Optimally extend it to $G - S$.

Clique-cycle-forest
Definition

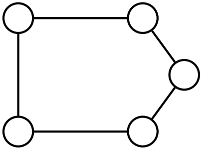
A clique is a clique-cycle-forest.



Clique-cycle-forest

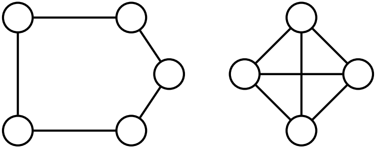
Definition

A cycle is a clique-cycle-forest.



Clique-cycle-forest
Definition

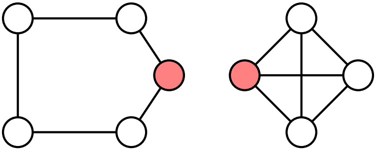
Disjoint union of two clique-cycle-forests is a clique-cycle-forest.



Clique-cycle-forest

Definition

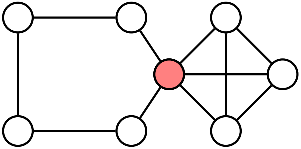
Graph obtained by identifying one vertex each from two different components of a clique-cycle-forest is again a clique-cycle-forest.



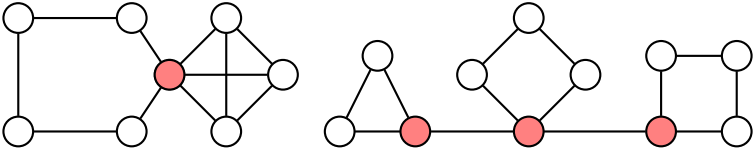
Clique-cycle-forest

Definition

Graph obtained by identifying one vertex each from two different components of a clique-cycle-forest is again a clique-cycle-forest.



Clique-cycle-forest
Example



FPT Algorithm

Reduction Rules

- **Rule 1:**

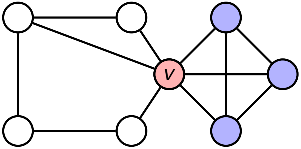
FPT Algorithm

Reduction Rules

- **Rule 1:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a clique.

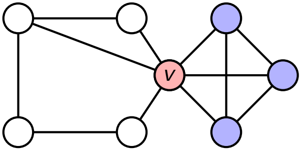
FPT Algorithm
Reduction Rules

- **Rule 1:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a clique.



FPT Algorithm
Reduction Rules

- **Rule 1:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a clique.

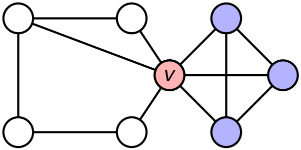


- **Delete:** All vertices in X.

FPT Algorithm

Reduction Rules

- **Rule 1:** Apply if $\exists\, v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a clique.

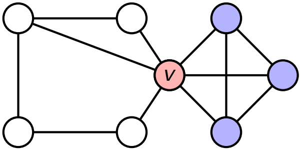


- **Delete:** All vertices in X.
- **Mark:** Nothing.

FPT Algorithm

Reduction Rules

- **Rule 1:** Apply if $\exists\, v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a clique.

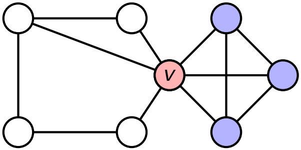


- **Delete:** All vertices in X.
- **Mark:** Nothing.
- **Parameter:** Reduce k' by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.

FPT Algorithm

Reduction Rules

- **Rule 1:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a clique.



- **Delete:** All vertices in X.
- **Mark:** Nothing.
- **Parameter:** Reduce k' by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.

FPT Algorithm

Reduction Rules–Rule 1 contd.

- **Parameter:** Reduce k by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.

FPT Algorithm

Reduction Rules–Rule 1 contd.

- **Parameter:** Reduce k by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.
- **New instance:** (G'', k'').

FPT Algorithm

Reduction Rules–Rule 1 contd.

- **Parameter:** Reduce k by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.
- **New instance:** (G'', k'').
- $|V(G'')| = n' - x$ and $k'' = k' - (x^2/4 - x/2)$

FPT Algorithm

Reduction Rules–Rule 1 contd.

- **Parameter:** Reduce k by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.
- **New instance:** (G'', k'').
- $|V(G'')| = n' - x$ and $k'' = k' - (x^2/4 - x/2)$
- Consider f, a cut of $G' - v$.

FPT Algorithm

Reduction Rules–Rule 1 contd.

- **Parameter:** Reduce k by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.
- **New instance:** (G'', k'').
- $|V(G'')| = n' - x$ and $k'' = k' - (x^2/4 - x/2)$
- Consider f, a cut of $G' - v$.
- Define g, a cut of G':
  - $g = f + X$ partitioned evenly.
  - $||g|| = ||f|| + x^2/4 + x/2$.

FPT Algorithm

Reduction Rules–Rule 1 contd.

- **Parameter:** Reduce k by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.
- **New instance:** (G'', k'').
- $|V(G'')| = n' - x$ and $k'' = k' - (x^2/4 - x/2)$
- Consider f, a cut of $G' - v$.
- Define g, a cut of G':
  - $g = f + X$ partitioned evenly.
  - $||g|| = ||f|| + x^2/4 + x/2$.
- Suppose $||f|| \geq (n - x - 1) + k''$. Then,

$$\begin{aligned} ||g|| &\geq (n - x - 1) + (k' - (x^2/4 - x/2)) + x^2/4 + x/2 \\ &= n - 1 + k'. \end{aligned}$$

FPT Algorithm

Reduction Rules–Rule 1 contd.

- **Parameter:** Reduce k by $\lceil x^2/4 - x/2 \rceil$, where $x = |X|$.
- **New instance:** (G'', k'').
- $|V(G'')| = n' - x$ and $k'' = k' - (x^2/4 - x/2)$
- Consider f, a cut of $G' - v$.
- Define g, a cut of G':
  - $g = f + X$ partitioned evenly.
  - $||g|| = ||f|| + x^2/4 + x/2$.
- Suppose $||f|| \geq (n - x - 1) + k''$. Then,

$$\begin{aligned} ||g|| &\geq (n - x - 1) + (k' - (x^2/4 - x/2)) + x^2/4 + x/2 \\ &= n - 1 + k'. \end{aligned}$$

FPT Algorithm

Reduction Rules

- **Rule 2:**

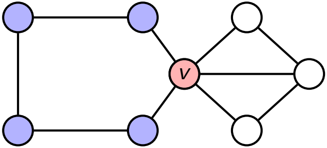
FPT Algorithm

Reduction Rules

- **Rule 2:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a cycle.

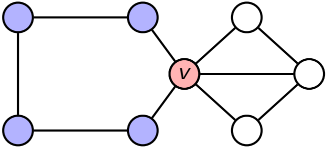
FPT Algorithm
Reduction Rules

- **Rule 2:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a cycle.



FPT Algorithm
Reduction Rules

- **Rule 2:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X \cup \{v\}]$ is a cycle.



- **Delete:** All vertices in X.
- **Mark:** Nothing.
- **Parameter:** Reduce k' by 1 if x is odd, and no change in k' if x is even $(x = |X|)$.

FPT Algorithm
Reduction Rules

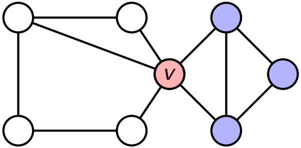
- **Rule 3:**

FPT Algorithm
Reduction Rules

- **Rule 3:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X]$ is a clique.

FPT Algorithm
Reduction Rules

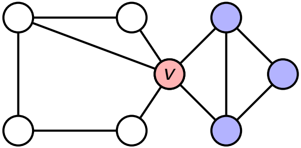
- **Rule 3:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X]$ is a clique.



FPT Algorithm

Reduction Rules

- **Rule 3:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X]$ is a clique.

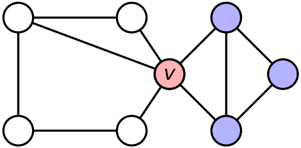


- **Delete:** All vertices in X.

FPT Algorithm

Reduction Rules

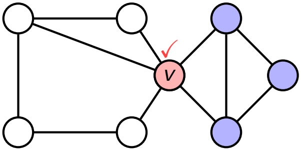
- **Rule 3:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X]$ is a clique.



- **Delete:** All vertices in X.

FPT Algorithm
Reduction Rules

- **Rule 3:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X]$ is a clique.

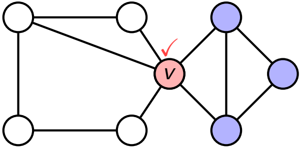


- **Delete:** All vertices in X.
- **Mark:** v.

FPT Algorithm

Reduction Rules

- **Rule 3:** Apply if $\exists\ v \in V(G')$ and $X \subseteq V(G')$ such that
  - $G'[X]$ is a connected component of $G' - \{v\}$,
  - $G'[X]$ is a clique.



- **Delete:** All vertices in X.
- **Mark:** v.
- **Parameter:** Reduce k' by $\lfloor x^2/4 \rfloor + \min\left\{d_{G'[X\cup\{v\}]}(v), \lceil x/2 \rceil\right\} - x$.

FPT Algorithm

Reduction Rules–Rule 3 contd.

- $d_{G'[X \cup \{v\}]}(v) > 1$ and $x > 2$. Otherwise, Rule 1 applies.

FPT Algorithm

Reduction Rules–Rule 3 contd.

- $d_{G'[X\cup\{v\}]}(v) > 1$ and $x > 2$. Otherwise, Rule 1 applies.

$$\begin{aligned}\lfloor x^2/4 \rfloor + \min\left\{d_{G[X\cup\{v\}]}(v), \lceil x/2 \rceil\right\} - x &\geq \lfloor x^2/4 \rfloor + 2 - x \\ &\geq \lfloor 3^2/4 \rfloor + 2 - 3 \\ &= 1.\end{aligned}$$

- Parameter drops.

FPT Algorithm
Reduction Rules

- Good P_3:

FPT Algorithm
Reduction Rules

- Good P_3: Induced P_3 abc such that
  - $G' - \{a, b, c\}$ is connected,

FPT Algorithm
Reduction Rules

- Good P_3: Induced P_3 abc such that
  - $G' - \{a, b, c\}$ is connected,
  - $d(a) > 2$ or $d(b) > 2$ or $d(c) > 2$.

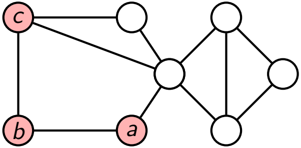
FPT Algorithm

Reduction Rules

- Good P_3: Induced P_3 abc such that
  - $G' - \{a, b, c\}$ is connected,
  - $d(a) > 2$ or $d(b) > 2$ or $d(c) > 2$.
- **Rule 4:** Apply if $\exists$ a good P_3 abc.

FPT Algorithm
Reduction Rules

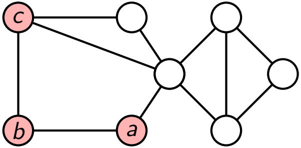
- Good P_3: Induced P_3 abc such that
  - $G' - \{a, b, c\}$ is connected,
  - $d(a) > 2$ or $d(b) > 2$ or $d(c) > 2$.
- **Rule 4:** Apply if $\exists$ a good P_3 abc.



FPT Algorithm

Reduction Rules

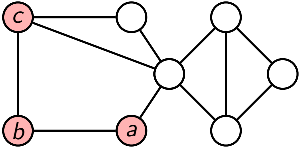
- Good P_3: Induced P_3 abc such that
  - $G' - \{a, b, c\}$ is connected,
  - $d(a) > 2$ or $d(b) > 2$ or $d(c) > 2$.
- **Rule 4:** Apply if $\exists$ a good P_3 abc.



  - **Delete:** Vertices a, b, c.

FPT Algorithm
Reduction Rules

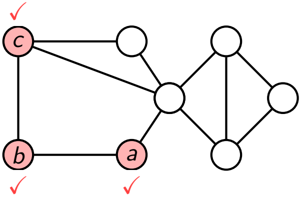
- Good P_3: Induced P_3 abc such that
  - $G' - \{a, b, c\}$ is connected,
  - $d(a) > 2$ or $d(b) > 2$ or $d(c) > 2$.
- **Rule 4:** Apply if $\exists$ a good P_3 abc.



  - **Delete:** Vertices a, b, c.

FPT Algorithm
Reduction Rules

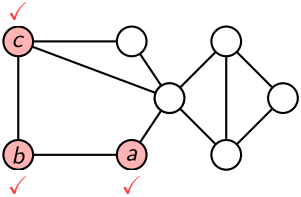
- Good P_3: Induced P_3 *abc* such that
  - $G' - \{a, b, c\}$ is connected,
  - $d(a) > 2$ or $d(b) > 2$ or $d(c) > 2$.
- **Rule 4:** Apply if $\exists$ a good P_3 abc.



  - **Delete:** Vertices a, b, c.
  - **Mark:** a, b, c.

FPT Algorithm
Reduction Rules

- Good P_3: Induced P_3 abc such that
  - $G' - \{a, b, c\}$ is connected,
  - $d(a) > 2$ or $d(b) > 2$ or $d(c) > 2$.
- **Rule 4:** Apply if $\exists$ a good P_3 abc.



- **Delete:** Vertices a, b, c.
- **Mark:** a, b, c.
- **Parameter:** Reduce k' by $\lceil (d(\alpha) - 2)/2 \rceil$, where $\alpha =$ highest degree vertex in $\{a, b, c\}$.

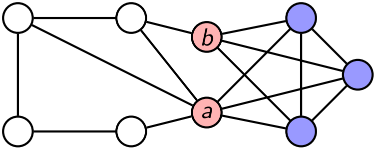
FPT Algorithm

Reduction Rules

- **Rule 5:**

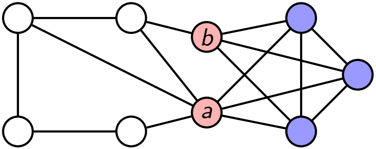
FPT Algorithm
Reduction Rules

- **Rule 5:** Apply if $\exists\ a, b \in V(G')$ and $Y \subseteq V(G')$ such that
  - $ac \notin E(G')$,
  - $G' - \{a, c\}$ has exactly two connected components, X and Y,
  - $|X| \geq 2$,
  - $G'[X \cup \{a\}]\ G'[X \cup \{c\}]$ are cliques.



FPT Algorithm
Reduction Rules

- **Rule 5:** Apply if $\exists$ $a, b \in V(G')$ and $Y \subseteq V(G')$ such that
  - $ac \notin E(G')$,
  - $G' - \{a, c\}$ has exactly two connected components, X and Y,
  - $|X| \geq 2$,
  - $G'[X \cup \{a\}]$ $G'[X \cup \{c\}]$ are cliques.

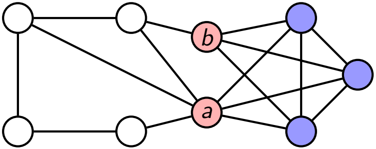


- **Delete:** All vertices in $X \cup \{a, b\}$.

FPT Algorithm

Reduction Rules

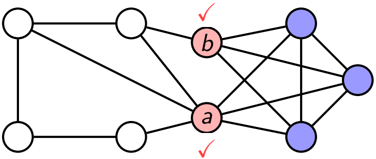
- **Rule 5:** Apply if $\exists$ $a, b \in V(G')$ and $Y \subseteq V(G')$ such that
  - $ac \notin E(G')$,
  - $G' - \{a, c\}$ has exactly two connected components, X and Y,
  - $|X| \geq 2$,
  - $G'[X \cup \{a\}]$ $G'[X \cup \{c\}]$ are cliques.



- **Delete:** All vertices in $X \cup \{a, b\}$.

FPT Algorithm
Reduction Rules

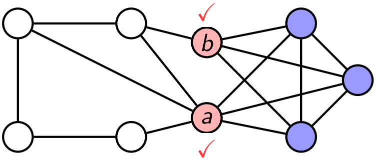
- **Rule 5:** Apply if $\exists$ $a, b \in V(G')$ and $Y \subseteq V(G')$ such that
  - $ac \notin E(G')$,
  - $G' - \{a, c\}$ has exactly two connected components, X and Y,
  - $|X| \geq 2$,
  - $G'[X \cup \{a\}]$ $G'[X \cup \{c\}]$ are cliques.



- **Delete:** All vertices in $X \cup \{a, b\}$.
- **Mark:** a, b.

FPT Algorithm
Reduction Rules

- **Rule 5:** Apply if $\exists$ $a, b \in V(G')$ and $Y \subseteq V(G')$ such that
  - $ac \notin E(G')$,
  - $G' - \{a, c\}$ has exactly two connected components, X and Y,
  - $|X| \geq 2$,
  - $G'[X \cup \{a\}]$ $G'[X \cup \{c\}]$ are cliques.



- **Delete:** All vertices in $X \cup \{a, b\}$.
- **Mark:** a, b.
- **Parameter:** Reduce k' by $\lceil x/2 \rceil \cdot \lfloor x/2 \rfloor + \lceil d_{G[Y \cup \{a\}]}(a)/2 \rceil + \lceil d_{G[Y \cup \{b\}]}(b)/2 \rceil - 2.$

FPT Algorithm

Lemma

Rules 1-5 are one-way safe.

FPT Algorithm

Lemma

Rules 1-5 are one-way safe.

Lemma

If G' has at least one edge then one of Rules 1-5 will apply.

FPT Algorithm

Lemma

Rules 1-5 are one-way safe.

Lemma

If G' has at least one edge then one of Rules 1-5 will apply.

Lemma

If $k' \leq 0$, then (G', k') is a yes-instance, and hence (G, k) is a yes-instance. Otherwise, $|S| \leq 3k$, where S is the set of marked vertices.

FPT Algorithm
Key Lemma

Lemma

$G - S$ is a clique-cycle-forest.

FPT Algorithm

Key Lemma

Lemma

$G - S$ is a clique-cycle-forest.

- Proof by induction on the length of the reduction.

FPT Algorithm
Key Lemma

Lemma

$G - S$ is a clique-cycle-forest.

- Proof by induction on the length of the reduction.
- Let $G = G_0 \to G_1 \to \cdots \to G_\ell = K_1$.

FPT Algorithm

Key Lemma

Lemma

$G - S$ is a clique-cycle-forest.

- Proof by induction on the length of the reduction.
- Let $G = G_0 \to G_1 \to \cdots \to G_\ell = K_1$.
- Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

FPT Algorithm
Key Lemma

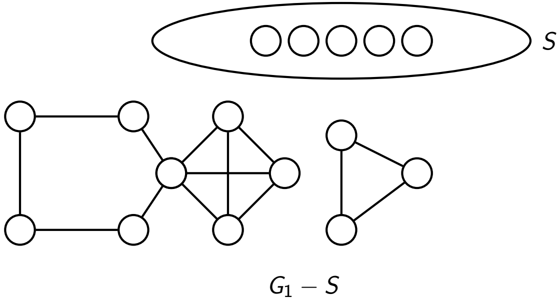
Lemma

$G - S$ is a clique-cycle-forest.

- Proof by induction on the length of the reduction.
- Let $G = G_0 \to G_1 \to \cdots \to G_\ell = K_1$.
- Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

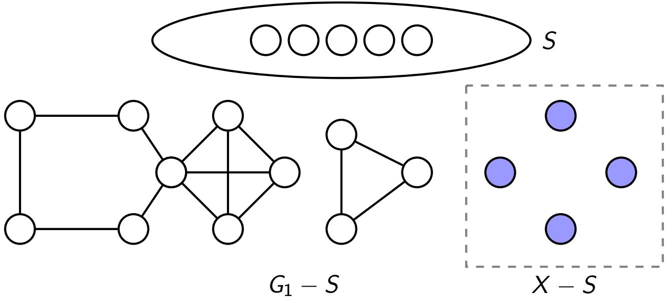
FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.



FPT Algorithm
Key Lemma

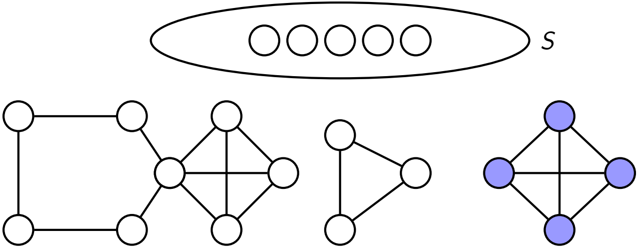
Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.



FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 1: $X \cup \{v\}$ is a clique.

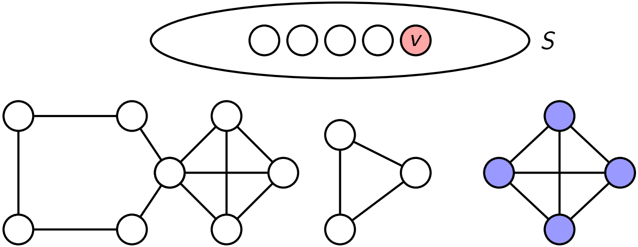


FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 1: $X \cup \{v\}$ is a clique.

Case 1: $v \in S$



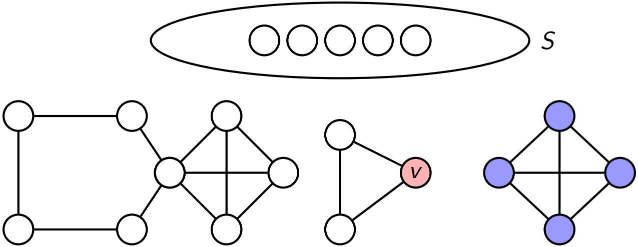
FPT Algorithm

Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 1: $X \cup \{v\}$ is a clique.

Case 2: $v \notin S$.



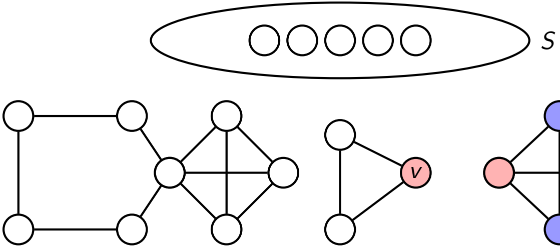
FPT Algorithm

Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 1: $X \cup \{v\}$ is a clique.

Case 2: $v \notin S$.



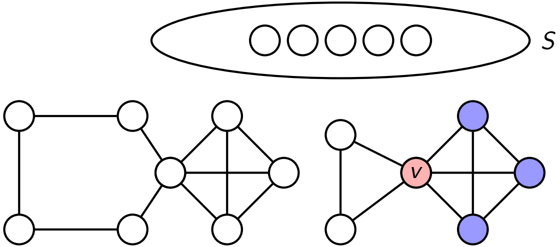
FPT Algorithm

Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 1: $X \cup \{v\}$ is a clique.

Case 2: $v \notin S$.



FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

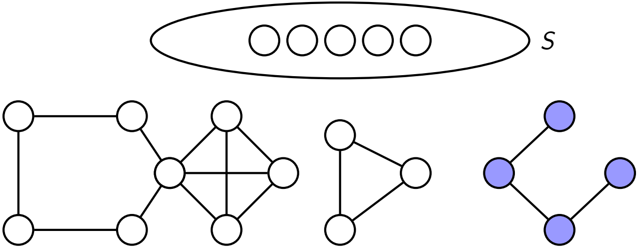
Rule 2: $X \cup \{v\}$ is a cycle.

FPT Algorithm

Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 2: $X \cup \{v\}$ is a cycle. $X - v$ is a path.

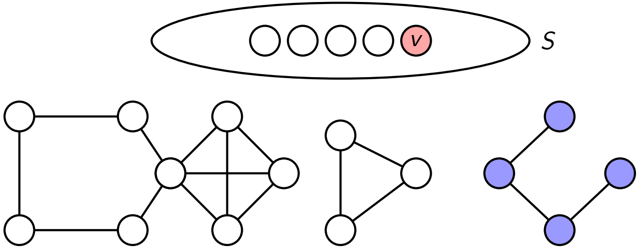


FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 2: $X \cup \{v\}$ is a cycle.

Case 1: $v \in S$. $X - v$ is a path.

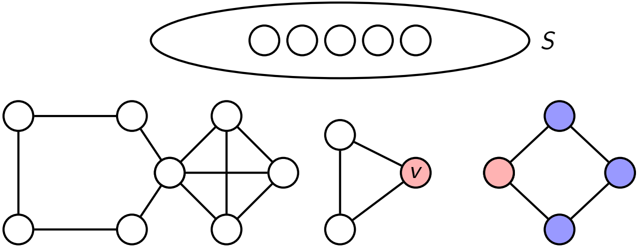


FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 2: $X \cup \{v\}$ is a cycle.

Case 2: $v \notin S$.

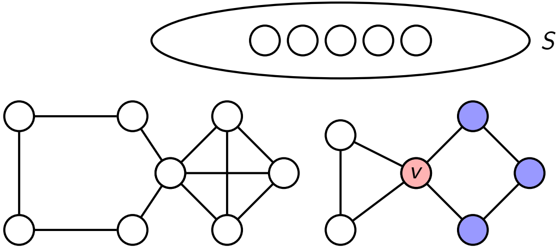


FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 2: $X \cup \{v\}$ is a cycle.

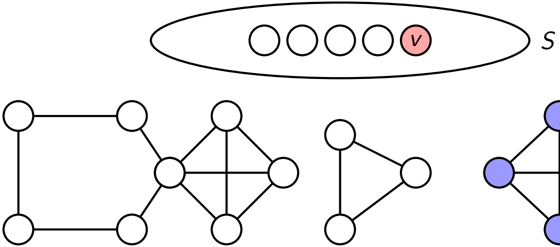
Case 2: $v \notin S$.



FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

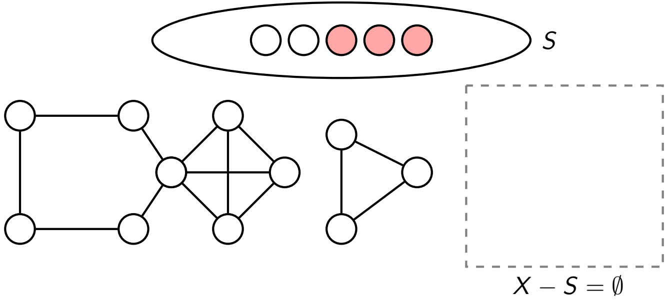
Rule 3: X is a clique. v is marked.



FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

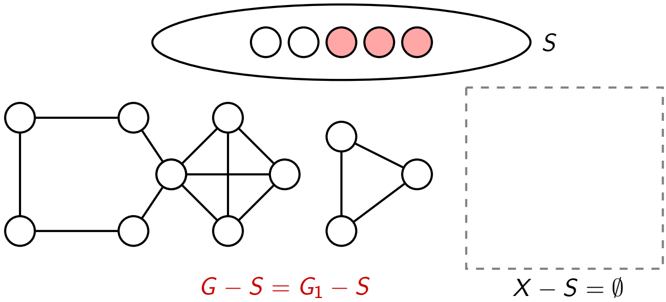
Rule 4: X is a P_3. X is marked.



FPT Algorithm
Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 4: X is a P_3. X is marked.

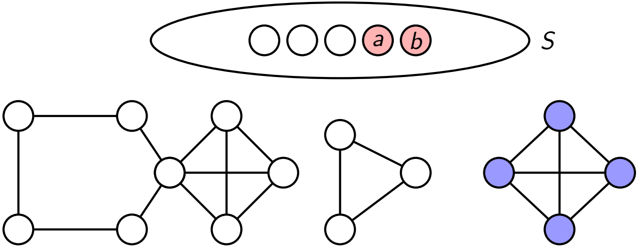


FPT Algorithm

Key Lemma

Ind. hyp.: $G_1 - S$ is a clique-cycle-forest.

Rule 5: $X \setminus \{a, b\}$ is a clique. a, b are marked.



Max-Cut-With-Weighted-Vertices

- **Input:**
  - A graph G, an integer $t \in \mathbb{N}$, and
  - weight functions $w_0 : V(G) \to \mathbb{N} \cup \{0\}$ and
    $w_1 : V(G) \to \mathbb{N} \cup \{0\}$.

MAX-CUT-WITH-WEIGHTED-VERTICES

- **Input:**
  - A graph G, an integer $t \in \mathbb{N}$, and
  - weight functions $w_0 : V(G) \to \mathbb{N} \cup \{0\}$ and
    $w_1 : V(G) \to \mathbb{N} \cup \{0\}$.
- **Objective:** Test if $\exists\ f : V(G) \to \{0, 1\}$ such that
  $\sum_{xy \in E(G)} |f(x) - f(y)| + \sum_{f(x)=0} w_0(x) + \sum_{f(x)=1} w_1(x) \geq t$?

MAX-CUT-WITH-WEIGHTED-VERTICES

- **Input:**
  - A graph G, an integer $t \in \mathbb{N}$, and
  - weight functions $w_0 : V(G) \to \mathbb{N} \cup \{0\}$ and
    $w_1 : V(G) \to \mathbb{N} \cup \{0\}$.
- **Objective:** Test if $\exists\ f : V(G) \to \{0,1\}$ such that
  $\sum_{xy \in E(G)} |f(x) - f(y)| + \sum_{f(x)=0} w_0(x) + \sum_{f(x)=1} w_1(x) \geq t$?

Lemma (Crowston et al.)

MAX-CUT-WITH-WEIGHTED-VERTICES *is polynomial time solvable on clique-forests.*

Max-Cut-With-Weighted-Vertices

- **Input:**
  - A graph G, an integer $t \in \mathbb{N}$, and
  - weight functions $w_0 : V(G) \to \mathbb{N} \cup \{0\}$ and $w_1 : V(G) \to \mathbb{N} \cup \{0\}$.
- **Objective:** Test if $\exists\, f : V(G) \to \{0,1\}$ such that $\sum_{xy \in E(G)} |f(x) - f(y)| + \sum_{f(x)=0} w_0(x) + \sum_{f(x)=1} w_1(x) \geq t$?

Lemma (Crowston et al.)

Max-Cut-With-Weighted-Vertices *is polynomial time solvable on clique-forests.*

Lemma

Max-Cut-With-Weighted-Vertices *is polynomial time solvable on clique-cycle-forests.*

FPT Algorithm for MAX-CUT-AST

- $G - S$ is a clique-cycle-forest.

FPT Algorithm for MAX-CUT-AST

- $G - S$ is a clique-cycle-forest.
- Guess the optimal partition of S.

FPT Algorithm for Max-Cut-AST

- $G - S$ is a clique-cycle-forest.
- Guess the optimal partition of S.
- For each guess $f : S \to \{0, 1\}$, construct a MCWWV instance on $G - S$.

FPT Algorithm for MAX-CUT-AST

- $G - S$ is a clique-cycle-forest.
- Guess the optimal partition of S.
- For each guess $f : S \to \{0, 1\}$, construct a MCWWV instance on $G - S$.
- $2^{|S|} \leq 2^{3k} = 8^k$ such instances.

FPT Algorithm for MAX-CUT-AST

- $G - S$ is a clique-cycle-forest.
- Guess the optimal partition of S.
- For each guess $f : S \to \{0, 1\}$, construct a MCWWV instance on $G - S$.
- $2^{|S|} \leq 2^{3k} = 8^k$ such instances.
- Original MAX-CUT-AST instance is a yes-instance if and only if one of these 8^k instances of MCWWV is a yes-instance.

FPT Algorithm for Max-Cut-AST

- $G - S$ is a clique-cycle-forest.
- Guess the optimal partition of S.
- For each guess $f : S \to \{0, 1\}$, construct a MCWWV instance on $G - S$.
- $2^{|S|} \leq 2^{3k} = 8^k$ such instances.
- Original Max-Cut-AST instance is a yes-instance if and only if one of these 8^k instances of MCWWV is a yes-instance.

FPT Algorithm for MAX-CUT-AST

Fix $f : S \to \{0, 1\}$. Construct an instance of MCWWV.

FPT Algorithm for Max-Cut-AST

Fix $f : S \to \{0, 1\}$. Construct an instance of MCWWV.

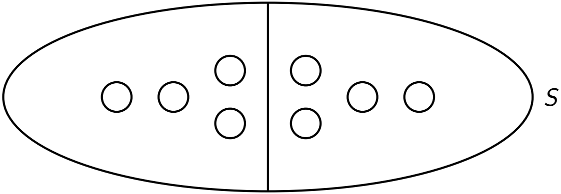
- $\ell =$ no. of edges of $G[S]$ that are satisfied by f.

FPT Algorithm for Max-Cut-AST

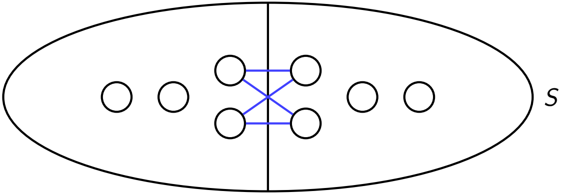
Fix $f : S \to \{0, 1\}$. Construct an instance of MCWWV.

- $\ell =$ no. of edges of $G[S]$ that are satisfied by f.
- For $x \in V(G) - S$,
  - $w_0(x) = |\{s \in S \mid sx \in E(G), \text{ and } f(s) = 1\}|$,
  - $w_1(x) = |\{s \in S \mid sx \in E(G), \text{ and } f(s) = 0\}|$.

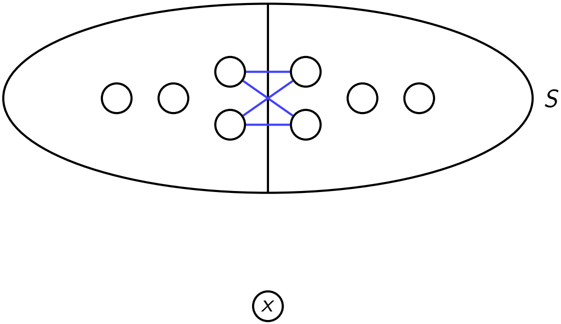
FPT Algorithm for MAX-CUT-AST



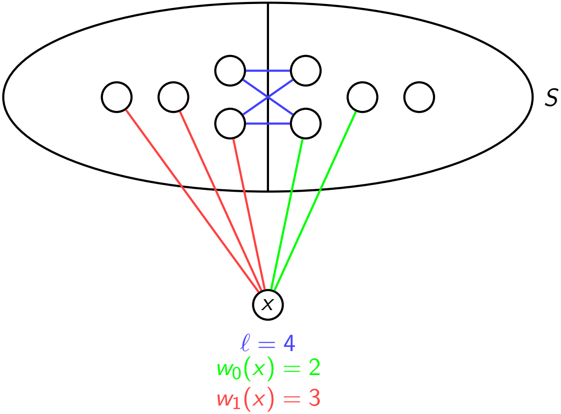
FPT Algorithm for Max-Cut-AST



FPT Algorithm for Max-Cut-AST



FPT Algorithm for Max-Cut-AST



FPT Algorithm for Max-Cut-AST contd.

- Set $t = n - 1 + k - \ell$.

FPT Algorithm for Max-Cut-AST contd.

- Set $t = n - 1 + k - \ell$.
- Let $f' : V(G - S) \to \{0, 1\}$ = optimum solution for MCWWV on $(G - S, w_0, w_1, t)$

FPT Algorithm for Max-Cut-AST contd.

- Define a cut $g : V(G) \rightarrow \{0, 1\}$ of G.

FPT Algorithm for MAX-CUT-AST contd.

- Define a cut $g : V(G) \rightarrow \{0, 1\}$ of G.
- $g(x) = f(x)$ if $x \in S$ and
  $g(x) = f'(x)$ if $x \in V(G - S)$.

FPT Algorithm for Max-Cut-AST contd.

- Define a cut $g : V(G) \to \{0,1\}$ of G.
- $g(x) = f(x)$ if $x \in S$ and
  $g(x) = f'(x)$ if $x \in V(G - S)$.

$$
\begin{aligned}
||g|| = \ell + & \sum_{xy \in E(G-S)} |g(x) - g(y)| \\
& + \sum_{\substack{x \in V(G-S) \\ g(x)=0}} w_0(x) + \sum_{\substack{x \in V(G-S) \\ g(x)=1}} w_1(x) \\
= \ell + ||f'||.
\end{aligned}
$$

FPT Algorithm for Max-Cut-AST contd.

- Define a cut $g : V(G) \to \{0,1\}$ of G.
- $g(x) = f(x)$ if $x \in S$ and
  $g(x) = f'(x)$ if $x \in V(G-S)$.

$$
\begin{aligned}
||g|| = \ell + & \sum_{xy \in E(G-S)} |g(x) - g(y)| \\
& + \sum_{\substack{x \in V(G-S) \\ g(x)=0}} w_0(x) + \sum_{\substack{x \in V(G-S) \\ g(x)=1}} w_1(x) \\
= \ell + ||f'||.
\end{aligned}
$$

- $||g|| \geq n - 1 + k \iff ||f'|| \geq n - 1 + k - \ell = t.$

FPT Algorithm for Max-Cut-AST contd.

- Define a cut $g : V(G) \to \{0,1\}$ of G.
- $g(x) = f(x)$ if $x \in S$ and
  $g(x) = f'(x)$ if $x \in V(G-S)$.

$$\begin{aligned} ||g|| = \ell + & \sum_{xy \in E(G-S)} |g(x) - g(y)| \\ & + \sum_{\substack{x \in V(G-S) \\ g(x)=0}} w_0(x) + \sum_{\substack{x \in V(G-S) \\ g(x)=1}} w_1(x) \\ = \ell + ||f'||. \end{aligned}$$

- $||g|| \geq n - 1 + k \iff ||f'|| \geq n - 1 + k - \ell = t$.

No $2^{o(k)}$ algorithm

- MAX-CUT has no $2^{o(t)}$ algorithm, t =cut size, unless the Exponential Time Hypothesis fails.

No $2^{o(k)}$ algorithm

- Max-Cut has no $2^{o(t)}$ algorithm, $t =$cut size, unless the Exponential Time Hypothesis fails.
- Reduce Max-Cut instance (G, t) to Max-Cut-AST instance (G, k).
- Set $k = t - (n - 1)$.

No $2^{o(k)}$ algorithm

- Max-Cut has no $2^{o(t)}$ algorithm, t =cut size, unless the Exponential Time Hypothesis fails.
- Reduce Max-Cut instance (G, t) to Max-Cut-AST instance (G, k).
- Set $k = t - (n - 1)$.
- $k \leq t$.

No $2^{o(k)}$ algorithm

- Max-Cut has no $2^{o(t)}$ algorithm, t =cut size, unless the Exponential Time Hypothesis fails.
- Reduce Max-Cut instance (G, t) to Max-Cut-AST instance (G, k).
- Set $k = t - (n - 1)$.
- $k \leq t$.
- $2^{o(k)}$ algorithm for Max-Cut-AST will imply $2^{o(t)}$ algorithm for Max-Cut.

Polynomial Kernel

Strategy

Polynomial Kernel

Strategy

- G has an even cycle implies G has a cut of size $n - 1 + 1 = n$.
- One even cycle means one additional edge in the cut.
- If G has k vertex disjoint even cycles, then (G, k) is a yes-instance.
- Cycles need not be vertex disjoint.
- Identify families of cycles such that all edges of all the cycles in the family fall into a cut.

Polynomial Kernel

Strategy contd.

- $G - S$ is a clique-cycle-forest, a forest of blocks.
- Bound the number of components of $G - S$.
- Bound the number of blocks in each component.
- Bound the size of each block.
- $\mathcal{O}(k^5)$ kernel.

Results and Conclusion

- FPT algorithm and $\mathcal{O}(k^5)$ kernel.

- Simple reduction to MAX-CUT-AEE possible?
- $\mathcal{O}(k)$ kernel?
- Extend the results to signed graphs?

Thank You.