On the Parameterized Complexity of Deletion to $\mathcal{H}$-free Strong Components

Rian Neogi[1], M. S. Ramanujan[2], Saket Saurabh[1] and Roohani Sharma[1]

[1]Institute of Mathematical Sciences, and [2]University of Warwick

MFCS 2020

Table of Contents

Directed FVS and related problems

DIRECTED FEEDBACK VERTEX SET
Input: Directed graph D, integer k
Output: Does there exist a set S of size at most k such that $D - S$ is acyclic?

Best known FPT algorithm: $O^*(k!4^k)$ (here O^* notation suppresses polynomial factors)

Improving this is a big open problem in parameterized complexity.

Recent work by Göke et al. [CIAC 2019] designs FPT algorithms for related problems.

Directed FVS and related problems

1-Out Regular Vertex Deletion
Input: Directed graph D, integer k
Output: Does there exist a set S of size at most k such that every strong component of $D - S$ has every vertex of degree at most 1?

Directed FVS and related problems

1-OUT REGULAR VERTEX DELETION **Input:** Directed graph D, integer k **Output:** Does there exist a set S of size at most k such that every strong component of $D - S$ has every vertex of degree at most 1?

BOUNDED SIZE STRONG COMPONENT VERTEX DELETION **Input:** Directed graph D, integers k, s **Output:** Does there exist a set S of size at most k such that every strong component of $D - S$ contains at most s vertices?

Directed FVS and related problems

1-Out Regular Vertex Deletion
Input: Directed graph D, integer k
Output: Does there exist a set S of size at most k such that every strong component of $D - S$ has every vertex of degree at most 1?

Bounded Size Strong Component Vertex Deletion
Input: Directed graph D, integers k, s
Output: Does there exist a set S of size at most k such that every strong component of $D - S$ contains at most s vertices?

Göke et al. gave a $2^{O(k^3)}n^{O(1)}$ algorithm for the first problem and a $4^k(ks+k+s)!n^{O(1)}$ algorithm for the second one.

Our problem

We generalize these problems to a more unified framework.

$\mathcal{H}$-FREE STRONG CONNECTED COMPONENT DELETION
Input: Directed graph D, integer k, finite family of graphs $\mathcal{H}$
Output: Does there exist a set S of at most k vertices such that every strong component of $D - S$ does not have a subgraph isomorphic to any graph $\mathcal{H}$

Our problem

We generalize these problems to a more unified framework.

$\mathcal{H}$-free Strong Connected Component Deletion
Input: Directed graph D, integer k, finite family of graphs $\mathcal{H}$
Output: Does there exist a set S of at most k vertices such that every strong component of $D - S$ does not have a subgraph isomorphic to any graph $\mathcal{H}$

Directed Feedback Vertex Set. Here $\mathcal{H}$ is an independent set on two vertices.

Our problem

We generalize these problems to a more unified framework.

$\mathcal{H}$-FREE STRONG CONNECTED COMPONENT DELETION
Input: Directed graph D, integer k, finite family of graphs $\mathcal{H}$
Output: Does there exist a set S of at most k vertices such that every strong component of $D - S$ does not have a subgraph isomorphic to any graph $\mathcal{H}$

DIRECTED FEEDBACK VERTEX SET. Here $\mathcal{H}$ is an independent set on two vertices.

1-OUT REGULAR VERTEX DELETION. Here $\mathcal{H}$ is a star with 2 leaves.

Our problem

We generalize these problems to a more unified framework.

$\mathcal{H}$-FREE STRONG CONNECTED COMPONENT DELETION
Input: Directed graph D, integer k, finite family of graphs $\mathcal{H}$
Output: Does there exist a set S of at most k vertices such that every strong component of $D - S$ does not have a subgraph isomorphic to any graph $\mathcal{H}$

DIRECTED FEEDBACK VERTEX SET. Here $\mathcal{H}$ is an independent set on two vertices.

1-OUT REGULAR VERTEX DELETION. Here $\mathcal{H}$ is a star with 2 leaves.

BOUNDED SIZE STRONG COMPONENT VERTEX DELETION. Here $\mathcal{H}$ is an independent set on $s + 1$ vertices.

Our Results

- A $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph in $\mathcal{H}$ has a 'rooted' property. Here h is the maximum size amognst all graphs in $\mathcal{H}$.

Our Results

- A $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph in $\mathcal{H}$ has a ‘rooted’ property. Here h is the maximum size amognst all graphs in $\mathcal{H}$.
- A $2^{O(k^3 \log k)} n^{O(1)}$ algorithm for the case when $\mathcal{H}$ contains a path.

Our Results

- A $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph in $\mathcal{H}$ has a ‘rooted’ property. Here h is the maximum size amognst all graphs in $\mathcal{H}$.
- A $2^{O(k^3 \log k)} n^{O(1)}$ algorithm for the case when $\mathcal{H}$ contains a path.
- A $2^{O(k \log k)} n^{O(1)}$ algorithm for 1-Out Regular Vertex Deletion.

Our Results

- A $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph in $\mathcal{H}$ has a 'rooted' property. Here h is the maximum size amogmst all graphs in $\mathcal{H}$.
- A $2^{O(k^3 \log k)} n^{O(1)}$ algorithm for the case when $\mathcal{H}$ contains a path.
- A $2^{O(k \log k)} n^{O(1)}$ algorithm for 1-Out Regular Vertex Deletion.
- A $2^{O(k(\log k + \log s))} n^{O(1)}$ algorithm for Bounded Size Strong Component Vertex Deletion.

Our Results

- A $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph in $\mathcal{H}$ has a 'rooted' property. Here h is the maximum size amognst all graphs in $\mathcal{H}$.
- A $2^{O(k^3 \log k)} n^{O(1)}$ algorithm for the case when $\mathcal{H}$ contains a path.
- A $2^{O(k \log k)} n^{O(1)}$ algorithm for 1-Out Regular Vertex Deletion.
- A $2^{O(k(\log k + \log s))} n^{O(1)}$ algorithm for Bounded Size Strong Component Vertex Deletion.

Our Results

- A $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph in $\mathcal{H}$ has a 'rooted' property. Here h is the maximum size amognst all graphs in $\mathcal{H}$.
- A $2^{O(k^3 \log k)} n^{O(1)}$ algorithm for the case when $\mathcal{H}$ contains a path.
- A $2^{O(k \log k)} n^{O(1)}$ algorithm for 1-Out Regular Vertex Deletion.
- A $2^{O(k(\log k + \log s))} n^{O(1)}$ algorithm for Bounded Size Strong Component Vertex Deletion.

Last two results improve on the bounds given by Göke et al.

This talk

In this talk, we will covering the $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph has a special 'rooted' property.

This talk

In this talk, we will covering the $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph has a special 'rooted' property.

The $2^{O(k^3 \log k)} n^{O(1)}$ algorithm for when $\mathcal{H}$ contains a path is based upon a reduction to the rooted case.

This talk

In this talk, we will covering the $2^{O(k^3 \log k)} n^{O(h)}$ algorithm for when each graph has a special 'rooted' property.

The $2^{O(k^3 \log k)} n^{O(1)}$ algorithm for when $\mathcal{H}$ contains a path is based upon a reduction to the rooted case.

Algorithms for 1-Out Regular Vertex Deletion and Bounded Size Strong Component Vertex Deletion are based upon similar ideas.

Table of Contents

Preliminaries

For an S-T separator C, we define $R(S, C)$ to be the set of vertices reachable from S after the deletion of C.

Preliminaries

For an S-T separator C, we define $R(S, C)$ to be the set of vertices reachable from S after the deletion of C.

Important property. There is a unique minimum 'closest' separator i.e. there is a unique minimum S-T separator C such that $R(S, C) \subseteq R(S, C')$ for all other minimum S-T separators C'.

Preliminaries

For an S-T separator C, we define $R(S, C)$ to be the set of vertices reachable from S after the deletion of C.

Important property. There is a unique minimum 'closest' separator i.e. there is a unique minimum S-T separator C such that $R(S, C) \subseteq R(S, C')$ for all other minimum S-T separators C'.

Symmetrically, there is a unique minimum 'furthest' separator i.e. separator C such that $R(S, C) \supseteq R(S, C')$ for all other minimum S-T separators C'.

Preliminaries

A minimum S-T separator C is said to *cover* another minimum S-T separator C' if $R(S, C) \supseteq R(S, C')$.

Preliminaries

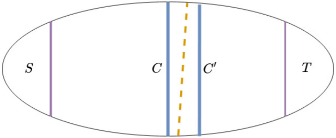
A minimum S-T separator C is said to *cover* another minimum S-T separator C' if $R(S, C) \supseteq R(S, C')$.

A minimum S-T separator C is said to *tightly cover* another minimum S-T separator C' if $R(S, C) \supseteq R(S, C')$ and there is no other minimum S-T separator C'' such that $R(S, C) \supseteq R(S, C'') \supseteq R(S, C')$.

Preliminaries

A minimum S-T separator C is said to *cover* another minimum S-T separator C' if $R(S, C) \supseteq R(S, C')$.

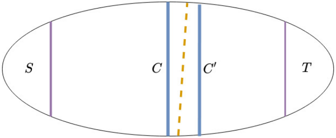
A minimum S-T separator C is said to *tightly cover* another minimum S-T separator C' if $R(S, C) \supseteq R(S, C')$ and there is no other minimum S-T separator C'' such that $R(S, C) \supseteq R(S, C'') \supseteq R(S, C')$.



Preliminaries

A minimum S-T separator C is said to *cover* another minimum S-T separator C' if $R(S, C) \supseteq R(S, C')$.

A minimum S-T separator C is said to *tightly cover* another minimum S-T separator C' if $R(S, C) \supseteq R(S, C')$ and there is no other minimum S-T separator C'' such that $R(S, C) \supseteq R(S, C'') \supseteq R(S, C')$.



Lemma (Pushing Routine)

Given a minimum S-T separator C, in polynomial one can either

- Compute a minimum S-T separator C' that tightly covers C
- Conclude that there is no such C', i.e. C is the unique furthest minimum separator.

Important Separators

Important Separators

For a graph D and subsets $S, T \subseteq V(G)$, an S-T separator C is said to be *important* if there is no other S-T separator C' such that $|C'| \leq |C|$ and $R(S, C') \supseteq R(S, C)$.

Important Separators

Important Separators

For a graph D and subsets $S, T \subseteq V(G)$, an S-T separator C is said to be *important* if there is no other S-T separator C' such that $|C'| \leq |C|$ and $R(S, C') \supseteq R(S, C)$.

I.e. important separators are separators can cannot be 'pushed' further without increasing its size.

Important Separators

Important Separators

For a graph D and subsets $S, T \subseteq V(G)$, an S-T separator C is said to be *important* if there is no other S-T separator C' such that $|C'| \leq |C|$ and $R(S, C') \supseteq R(S, C)$.

I.e. important separators are separators can cannot be 'pushed' further without increasing its size.

Lemma

There are at most 4^k important separators of size at most k, and they can be enumerated in $O^*(4^k)$ time.

Fundamental lemma for designing FPT algorithms for cut problems.

Important Separators

Important Separators

For a graph D and subsets $S, T \subseteq V(G)$, an S-T separator C is said to be *important* if there is no other S-T separator C' such that $|C'| \leq |C|$ and $R(S, C') \supseteq R(S, C)$.

I.e. important separators are separators can cannot be 'pushed' further without increasing its size.

Lemma

There are at most 4^k important separators of size at most k, and they can be enumerated in $O^*(4^k)$ time.

Fundamental lemma for designing FPT algorithms for cut problems.

Prove something of the form "If there exists a solution, then there is a solution that contains an important separator", then branch on the 4^k important separators.

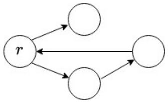
Rooted property

Graph is *rooted* if there exists a vertex r such that every other vertex in the graph can be reached from r.

Rooted property

Graph is *rooted* if there exists a vertex r such that every other vertex in the graph can be reached from r.

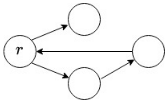
Here r is called the root of the graph.



Rooted property

Graph is *rooted* if there exists a vertex r such that every other vertex in the graph can be reached from r.

Here r is called the root of the graph.

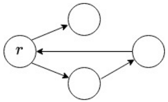


Turns out that this property is very helpful in designing algorithms for the $\mathcal{H}$-FREE SCC DELETION problem.

Rooted property

Graph is *rooted* if there exists a vertex r such that every other vertex in the graph can be reached from r.

Here r is called the root of the graph.



Turns out that this property is very helpful in designing algorithms for the $\mathcal{H}$-FREE SCC DELETION problem.

We will look at the special case when each graph in $\mathcal{H}$ is rooted.

Table of Contents

The Problem

Rooted $\mathcal{H}$-free Strong Connected Component Deletion
Input: Graph D, integer k, finite family of graphs $\mathcal{H}$ where every graph is *rooted*
Output: Does there exist a set S of at most k vertices such that every strong component of $D - S$ does not have a subgraph isomorphic to any graph $\mathcal{H}$

The Problem

ROOTED $\mathcal{H}$-FREE STRONG CONNECTED COMPONENT DELETION
Input: Graph D, integer k, finite family of graphs $\mathcal{H}$ where every graph is *rooted*
Output: Does there exist a set S of at most k vertices such that every strong component of $D - S$ does not have a subgraph isomorphic to any graph $\mathcal{H}$

Using the technique of Iterative Compression, it suffices to solve the disjoint version of the problem.

The Problem

ROOTED $\mathcal{H}$-FREE STRONG CONNECTED COMPONENT DELETION
Input: Graph D, integer k, finite family of graphs $\mathcal{H}$ where every graph is *rooted*
Output: Does there exist a set S of at most k vertices such that every strong component of $D - S$ does not have a subgraph isomorphic to any graph $\mathcal{H}$

Using the technique of Iterative Compression, it suffices to solve the disjoint version of the problem.

DISJOINT ROOTED $\mathcal{H}$-FREE SCC DELETION
Input: Graph D, integer k, finite family of graphs $\mathcal{H}$ where every graph is *rooted*, and solution $W \subseteq V(D)$ of size $k+1$
Output: Does there exist a solution of size k that is disjoint from W

Structure of solution X. After contracting all strongly connected components of $D - X$, we get a DAG and thus a topological order.

Structure of solution X. After contracting all strongly connected components of $D - X$, we get a DAG and thus a topological order.

This topological order induces an ordered partition on the vertices of W.

Structure of solution X. After contracting all strongly connected components of $D - X$, we get a DAG and thus a topological order.

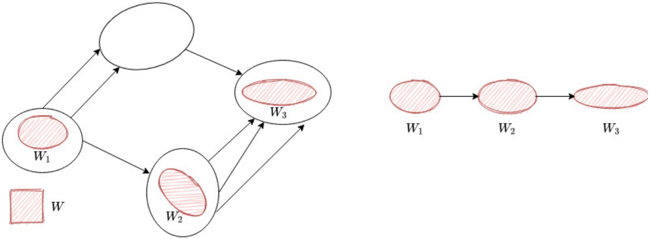
This topological order induces an ordered partition on the vertices of W.

I.e. Vertices of W in the same strongly connected components $D - X$ correspond to vertices in the same partition, and a partition come before another partition in the ordering if that strong component comes before the other in the topological ordering of $D - X$.

Structure of solution X. After contracting all strongly connected components of $D - X$, we get a DAG and thus a topological order.

This topological order induces an ordered partition on the vertices of W.

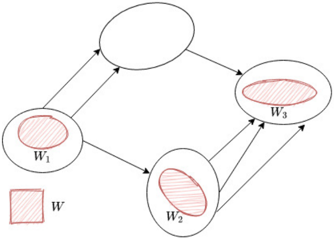
I.e. Vertices of W in the same strongly connected components $D - X$ correspond to vertices in the same partition, and a partition come before another partition in the ordering if that strong component comes before the other in the topological ordering of $D - X$.

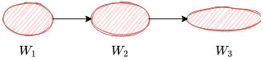


Structure of solution X. After contracting all strongly connected components of $D - X$, we get a DAG and thus a topological order.

This topological order induces an ordered partition on the vertices of W.

I.e. Vertices of W in the same strongly connected components $D - X$ correspond to vertices in the same partition, and a partition come before another partition in the ordering if that strong component comes before the other in the topological ordering of $D - X$.





We start by *guessing* this ordered partition on W!

Now the problem reduces to the following: Given an ordered partition on $W = (W_1, \ldots, W_q)$, find a solution X that is disjoint from W and of size k such that the aforementioned topological ordering of $D - X$ induces same ordered partition on W.

Now the problem reduces to the following: Given an ordered partition on $W = (W_1, \ldots, W_q)$, find a solution X that is disjoint from W and of size k such that the aforementioned topological ordering of $D - X$ induces same ordered partition on W.

That is, if W_i and W_j are sets in the partition with $i > j$ then we want to kill all paths from W_i to W_j *and* deal with subgraphs isomorphic to a graph in $\mathcal{H}$ that we encounter.

Now the problem reduces to the following: Given an ordered partition on $W = (W_1, \ldots, W_q)$, find a solution X that is disjoint from W and of size k such that the aforementioned topological ordering of $D - X$ induces same ordered partition on W.

That is, if W_i and W_j are sets in the partition with $i > j$ then we want to kill all paths from W_i to W_j *and* deal with subgraphs isomorphic to a graph in $\mathcal{H}$ that we encounter.

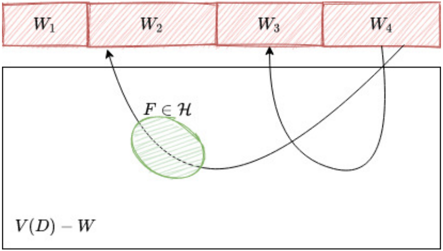


Table of Contents

We want to kill all paths from W_q to $W_1 \cup \ldots \cup W_{q-1}$. So our solution must contain an S-T separator where $S = W_q$ and $T = W_1 \cup \ldots \cup W_{q-1}$.

We want to kill all paths from W_q to $W_1 \cup \ldots \cup W_{q-1}$. So our solution must contain an S-T separator where $S = W_q$ and $T = W_1 \cup \ldots \cup W_{q-1}$.

New task. Given S, T: Kill all S-T paths and deal with forbidden subgraphs $F \in \mathcal{H}$ that are in the same strongly connected component of S.

We want to kill all paths from W_q to $W_1 \cup \ldots \cup W_{q-1}$. So our solution must contain an S-T separator where $S = W_q$ and $T = W_1 \cup \ldots \cup W_{q-1}$.

New task. Given S, T: Kill all S-T paths and deal with forbidden subgraphs $F \in \mathcal{H}$ that are in the same strongly connected component of S.

We will start with the unique closest minimum S-T separator C and iteratively try to push C further and further away from S.

We want to kill all paths from W_q to $W_1 \cup \ldots \cup W_{q-1}$. So our solution must contain an S-T separator where $S = W_q$ and $T = W_1 \cup \ldots \cup W_{q-1}$.

New task. Given S, T: Kill all S-T paths and deal with forbidden subgraphs $F \in \mathcal{H}$ that are in the same strongly connected component of S.

We will start with the unique closest minimum S-T separator C and iteratively try to push C further and further away from S.

While we push, we want to break all the forbidden subgraphs in $\mathcal{H}$ that we encounter.

We want to kill all paths from W_q to $W_1 \cup \ldots \cup W_{q-1}$. So our solution must contain an S-T separator where $S = W_q$ and $T = W_1 \cup \ldots \cup W_{q-1}$.

New task. Given S, T: Kill all S-T paths and deal with forbidden subgraphs $F \in \mathcal{H}$ that are in the same strongly connected component of S.

We will start with the unique closest minimum S-T separator C and iteratively try to push C further and further away from S.

While we push, we want to break all the forbidden subgraphs in $\mathcal{H}$ that we encounter.

At every branch, we either drop k or increase λ: the minimum S-T separator size.

We want to kill all paths from W_q to $W_1 \cup \ldots \cup W_{q-1}$. So our solution must contain an S-T separator where $S = W_q$ and $T = W_1 \cup \ldots \cup W_{q-1}$.

New task. Given S, T: Kill all S-T paths and deal with forbidden subgraphs $F \in \mathcal{H}$ that are in the same strongly connected component of S.

We will start with the unique closest minimum S-T separator C and iteratively try to push C further and further away from S.

While we push, we want to break all the forbidden subgraphs in $\mathcal{H}$ that we encounter.

At every branch, we either drop k or increase λ: the minimum S-T separator size.

Eventually when $\lambda = k$, either k must drop or we can conclude that is a NO-instance once $\lambda > k$, since every solution must contain an S-T separator.

Initially we find the closest S-T separator C and try to 'clean' $R(S, C)$ by breaking any forbidden subgraphs $F \in \mathcal{H}$ such that the root of F is in $R(S, C)$.

Initially we find the closest S-T separator C and try to 'clean' $R(S, C)$ by breaking any forbidden subgraphs $F \in \mathcal{H}$ such that the root of F is in $R(S, C)$.

'Breaking' a forbidden subgraph F involves branching into the following cases:

Initially we find the closest S-T separator C and try to 'clean' $R(S, C)$ by breaking any forbidden subgraphs $F \in \mathcal{H}$ such that the root of F is in $R(S, C)$.

'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution.

Initially we find the closest S-T separator C and try to 'clean' $R(S, C)$ by breaking any forbidden subgraphs $F \in \mathcal{H}$ such that the root of F is in $R(S, C)$.

'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution.
2. All vertices of F should not be reachable from S, which is equivalent to killing all S-$\{r\}$ paths (where r is the root of F). This can be achieved by adding r to T and recursing.

Initially we find the closest S-T separator C and try to 'clean' $R(S, C)$ by breaking any forbidden subgraphs $F \in \mathcal{H}$ such that the root of F is in $R(S, C)$.

'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution.
2. All vertices of F should not be reachable from S, which is equivalent to killing all S-$\{r\}$ paths (where r is the root of F). This can be achieved by adding r to T and recursing.
3. Kill all $\{u\}$-S paths for some vertex $u \in F$, we can achieve this by branching on all $\{u\}$-S important separators.

Initially we find the closest S-T separator C and try to 'clean' $R(S, C)$ by breaking any forbidden subgraphs $F \in \mathcal{H}$ such that the root of F is in $R(S, C)$.

'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution.
2. All vertices of F should not be reachable from S, which is equivalent to killing all S-$\{r\}$ paths (where r is the root of F). This can be achieved by adding r to T and recursing.
3. Kill all $\{u\}$-S paths for some vertex $u \in F$, we can achieve this by branching on all $\{u\}$-S important separators.

In the first and third case, we reduce k.

Initially we find the closest S-T separator C and try to 'clean' $R(S, C)$ by breaking any forbidden subgraphs $F \in \mathcal{H}$ such that the root of F is in $R(S, C)$.

'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution.
2. All vertices of F should not be reachable from S, which is equivalent to killing all S-$\{r\}$ paths (where r is the root of F). This can be achieved by adding r to T and recursing.
3. Kill all $\{u\}$-S paths for some vertex $u \in F$, we can achieve this by branching on all $\{u\}$-S important separators.

In the first and third case, we reduce k.

In the second case, the minimum S-T separator size λ increases because we add a vertex to T

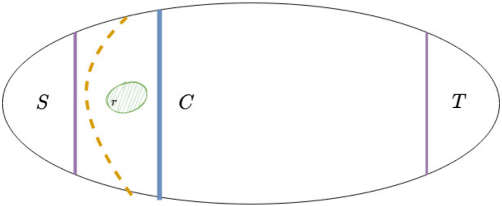
Let λ be the minimum S-T separator size. We will prove that λ increases when we add r to T.

Let λ be the minimum S-T separator size. We will prove that λ increases when we add r to T.

Recall that we add the vertex $r \in R(S, C)$ to T.

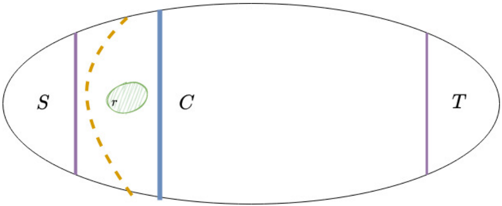
Let λ be the minimum S-T separator size. We will prove that λ increases when we add r to T.

Recall that we add the vertex $r \in R(S, C)$ to T.



Let λ be the minimum S-T separator size. We will prove that λ increases when we add r to T.

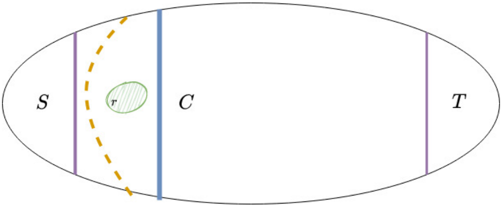
Recall that we add the vertex $r \in R(S, C)$ to T.



Since C is the closest minimum S-T separator, every other minimum S-T separator must cover C.

Let λ be the minimum S-T separator size. We will prove that λ increases when we add r to T.

Recall that we add the vertex $r \in R(S, C)$ to T.

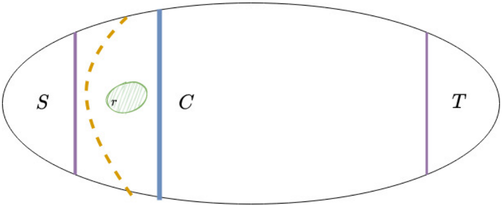


Since C is the closest minimum S-T separator, every other minimum S-T separator must cover C.

However any S-$(T \cup \{r\})$ separator *cannot* cover C since $r \in R(S, C)$.

Let λ be the minimum S-T separator size. We will prove that λ increases when we add r to T.

Recall that we add the vertex $r \in R(S, C)$ to T.



Since C is the closest minimum S-T separator, every other minimum S-T separator must cover C.

However any S-$(T \cup \{r\})$ separator *cannot* cover C since $r \in R(S, C)$.

Thus the minimum S-$(T \cup \{r\})$ separator size must be greater than λ.

Lemma (Pushing Routine)

Given a minimum S-T separator C, in polynomial one can either

- Compute a minimum S-T separator C' that tightly covers C
- Conclude that there is no such C', i.e. C is the unique furthest minimum separator.

Lemma (Pushing Routine)

Given a minimum S-T separator C, in polynomial one can either

- Compute a minimum S-T separator C' that tightly covers C
- Conclude that there is no such C', i.e. C is the unique furthest minimum separator.

We have a separator C and we try to 'push' to a new separator C' that tightly covers C.

Lemma (Pushing Routine)

Given a minimum S-T separator C, in polynomial one can either

- Compute a minimum S-T separator C' that tightly covers C
- Conclude that there is no such C', i.e. C is the unique furthest minimum separator.

We have a separator C and we try to 'push' to a new separator C' that tightly covers C.

When we push, we want to make sure that $R(S, C')$ is 'clean'. We do that by breaking all forbidden subgraphs $F \in \mathcal{H}$ such that its root r is in $R(S, C')$.

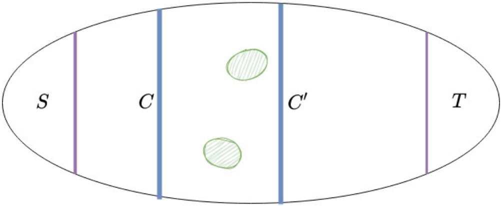
Lemma (Pushing Routine)

Given a minimum S-T separator C, in polynomial one can either

- Compute a minimum S-T separator C' that tightly covers C
- Conclude that there is no such C', i.e. C is the unique furthest minimum separator.

We have a separator C and we try to 'push' to a new separator C' that tightly covers C.

When we push, we want to make sure that $R(S, C')$ is 'clean'. We do that by breaking all forbidden subgraphs $F \in \mathcal{H}$ such that its root r is in $R(S, C')$.



'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution
2. Adding the root r to T and recursing
3. Branching on all $\{u\}$-S important separators, for every vertex $u \in F$

Again, the parameter k drops for cases 1 and 3.

'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution
2. Adding the root r to T and recursing
3. Branching on all $\{u\}$-S important separators, for every vertex $u \in F$

Again, the parameter k drops for cases 1 and 3.

However, this time, for case 2, its not so clear how we make progress.

'Breaking' a forbidden subgraph F involves branching into the following cases:

1. Picking a vertex $v \in F$ into our solution
2. Adding the root r to T and recursing
3. Branching on all $\{u\}$-S important separators, for every vertex $u \in F$

Again, the parameter k drops for cases 1 and 3.

However, this time, for case 2, its not so clear how we make progress.

Turns out by guessing which vertices of C that are reachable or unreachable in the final solution, we gain enough information to make progress in case 2 also.

Conclusions and further work

- We give FPT algorithms when every graph in $\mathcal{H}$ is rooted and when $\mathcal{H}$ contains a path of arbitrary length

Conclusions and further work

- We give FPT algorithms when every graph in $\mathcal{H}$ is rooted and when $\mathcal{H}$ contains a path of arbitrary length
- What about algorithms for other families $\mathcal{H}$? Is it possible to design an FPT algorithm for every such $\mathcal{H}$?

Conclusions and further work

- We give FPT algorithms when every graph in $\mathcal{H}$ is rooted and when $\mathcal{H}$ contains a path of arbitrary length
- What about algorithms for other families $\mathcal{H}$? Is it possible to design an FPT algorithm for every such $\mathcal{H}$?
- What about infinite families?

Conclusions and further work

- We give FPT algorithms when every graph in $\mathcal{H}$ is rooted and when $\mathcal{H}$ contains a path of arbitrary length
- What about algorithms for other families $\mathcal{H}$? Is it possible to design an FPT algorithm for every such $\mathcal{H}$?
- What about infinite families?
- Recent result by Göke, Marx and Mnich [ICALP 2020] shows that one can design an FPT algorithm for when $\mathcal{H}$ is the set of cycles of length greater than some integer s.

Thank You