Randomized Contractions

Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, Michał Pilipczuk

Update meeting on graph separation problems,
War saw, 9th April 2013
What are randomized contractions?

- A tool for designing FPT algorithms for cut problems.
What are randomized contractions?

- A tool for designing FPT algorithms for cut problems.
- An alternative to important separators, treewidth reduction, etc.

Original inspiration: the algorithm for $k$-way cut of Kawarabayashi and Thorup.

Another hammer in the toolbox.

CCHPP, Designing FPT algorithms for cut problems using randomized contractions, FOCS 2012

Chitnis, Cygan, Hajiaghayi, Pilipczuk
What are randomized contractions?

- A tool for designing FPT algorithms for cut problems.
- An alternative to important separators, treewidth reduction, etc.
- Can solve an orthogonal subset of problems, give better/worse running times.

Original inspiration: the algorithm for $k$-way cut of Kawarabayashi and Thorup.

Another hammer in the toolbox.

CCHPP, Designing FPT algorithms for cut problems using randomized contractions, FOCS 2012

Chitnis, Cygan, Hajiaghayi, Pilipczuk

Randomized Contractions

2/21
What are randomized contractions?

- A tool for designing FPT algorithms for cut problems.
- An alternative to important separators, treewidth reduction, etc.
- Can solve an orthogonal subset of problems, give better/worse running times.

**Original inspiration**: the algorithm for $k$-way cut of Kawarabayashi and Thorup.
What are randomized contractions?

- A tool for designing FPT algorithms for cut problems.
- An alternative to important separators, treewidth reduction, etc.
- Can solve an orthogonal subset of problems, give better/worse running times.

**Original inspiration**: the algorithm for $k$-way cut of Kawarabayashi and Thorup.

- Another hammer in the toolbox.
What are randomized contractions?

- A tool for designing FPT algorithms for cut problems.
- An alternative to important separators, treewidth reduction, etc.
- Can solve an orthogonal subset of problems, give better/worse running times.
- **Original inspiration**: the algorithm for $k$-way cut of Kawarabayashi and Thorup.
- Another hammer in the toolbox.
- **CCHPP**, *Designing FPT algorithms for cut problems using randomized contractions*, FOCS 2012
Exemplary problem: ULC

- Let $\Sigma$ be a finite alphabet of labels.
Exemplary problem: ULC

- Let $\Sigma$ be a finite alphabet of labels.
- A $\Sigma$-labeled graph consists of:
Exemplary problem: ULC

- Let $\Sigma$ be a finite alphabet of labels.
- A $\Sigma$-labeled graph consists of:
  - a set of vertices $V$;
Exemplary problem: ULC

- Let $\Sigma$ be a finite alphabet of labels.
- A $\Sigma$-labeled graph consists of:
  - a set of vertices $V$;
  - a set of constraints $E$ (called edges) of form $((v, w), \varphi_{(v, w)})$ such that $\varphi_{(v, w)}$ is a permutation of $\Sigma$. 
Exemplary problem: ULC

- Let $\Sigma$ be a finite alphabet of labels.
- A $\Sigma$-labeled graph consists of:
  - a set of vertices $V$;
  - a set of constraints $E$ (called edges) of form $((v, w), \varphi_{(v, w)})$ such that $\varphi_{(v, w)}$ is a permutation of $\Sigma$.
- A labeling $\Lambda : V \rightarrow \Sigma$ is consistent if $(\Lambda(v), \Lambda(w)) \in \varphi_{(v, w)}$ for each constraint $((v, w), \varphi_{(v, w)}) \in E$. 
Example
Figures from Wikipedia under Creative Commons BY-SA 3.0, created by Thore Husfeldt.
Example

Figures from Wikipedia under Creative Commons BY-SA 3.0, created by Thore Husfeldt.
Figures from Wikipedia under Creative Commons BY-SA 3.0, created by Thore Husfeldt.
Figures from Wikipedia under Creative Commons BY-SA 3.0, created by Thore Husfeldt.
**Observation**: existence of a consistent labeling is polynomial-time checkable.
**ULC: definition**

**Observation**: existence of a consistent labeling is polynomial-time checkable.

- For every connected component, try all labelings of an arbitrarily chosen vertex, and propagate.
**ULC**: definition

- **Observation**: existence of a consistent labeling is polynomial-time checkable.
  - For every connected component, try all labelings of an arbitrarily chosen vertex, and propagate.

- What if we want to minimize the number of unsatisfied constraints?
**ULC: definition**

- **Observation**: existence of a consistent labeling is polynomial-time checkable.
  - For every connected component, try all labelings of an arbitrarily chosen vertex, and propagate.

- What if we want to minimize the number of unsatisfied constraints?

---

**Unique Label Cover**

**Input**: a $\Sigma$-labeled graph $G$ and an integer $k$

**Question**: Is there a labeling disrespecting at most $k$ constraints?
ULC: definition

- **Observation**: existence of a consistent labeling is polynomial-time checkable.
  - For every connected component, try all labelings of an arbitrarily chosen vertex, and propagate.
- What if we want to minimize the number of unsatisfied constraints?

**Unique Label Cover**

- **Input**: a $\Sigma$-labeled graph $G$ and an integer $k$
- **Question**: Is there a labeling disrespecting at most $k$ constraints?

- We show an algorithm working in time $O^*(2^{O(k^2 \log |\Sigma|)})$. 

Chitnis, Cygan, Hajiaghayi, Pilipczuk \times 2
Why ULC?

- Generalizes many graph separation problems:
Why ULC?

- Generalizes many graph separation problems:
  - Edge Bipartization;

Hardness of robust approximation for ULC is the base of the Unique Games Conjecture.
Why ULC?

- Generalizes many graph separation problems:
  - **Edge Bipartization**;
  - **Edge Multiway Cut**;

Hardness of robust approximation for ULC is the base of the Unique Games Conjecture.
Why ULC?

- Generalizes many graph separation problems:
  - Edge Bipartization;
  - Edge Multiway Cut;
  - Group Feedback Edge Set...
Why ULC?

- Generalizes many graph separation problems:
  - Edge Bipartization;
  - Edge Multiway Cut;
  - Group Feedback Edge Set...

- Hardness of robust approximation for ULC is the base of the Unique Games Conjecture.
Ingredients

- Ingredients:
  
  - A sound notion of an edge contraction;
  - A robust divide step on small separators;
  - High connectivity helps.

Strategy:
If there is a nice separator, perform divide-and-conquer on it, otherwise, exploit the high-connectivity structure of the graph to solve the problem directly.
Ingredients:

- sound notion of an edge contraction;
Ingredients:

- sound notion of an edge contraction;
- robust divide step on small separators;
Ingredients:

- sound notion of an edge contraction;
- robust divide step on small separators;
- high connectivity helps.
Ingredients

- **Ingredients:**
  - sound notion of an edge contraction;
  - robust divide step on small separators;
  - high connectivity helps.

- **Strategy:**
Ingredients:

- sound notion of an edge contraction;
- robust divide step on small separators;
- high connectivity helps.

Strategy:

- If there is a *nice* separator, perform divide-and-conquer on it,
Ingredients:
- sound notion of an edge contraction;
- robust divide step on small separators;
- high connectivity helps.

Strategy:
- If there is a *nice* separator, perform divide-and-conquer on it,
- otherwise, exploit the high-connectivity structure of the graph
to solve the problem directly.
Observation

Labelings of $G$ that respect constraint $(u, v, \phi_{uv})$ correspond one-to-one to labelings of $G/uv$, where the correspondence retains the set of disrespected constraints.

Corollary
If we infer that $uv$ is not contained in some optimum solution, then it is safe to contract $uv$.
Contraction

Observation

Labelings of $G$ that respect constraint $((u, v), \phi_{uv})$ correspond one-to-one to labelings of $G/uv$, where the correspondence retains the set of disrespected constraints.

Corollary

If we infer that $uv$ is not contained in some optimum solution, then it is safe to contract $uv$.
Observation

Labelings of $G$ that respect constraint $((u, v), \varphi_{uv})$ correspond one-to-one to labelings of $G/uv$, where the correspondence retains the set of disrespected constraints.
Contraction

Observation
Labelings of $G$ that respect constraint $((u, v), \varphi_{uv})$ correspond one-to-one to labelings of $G/uv$, where the correspondence retains the set of disrespected constraints.

Corollary
If we infer that $uv$ is not contained in some optimum solution, then it is safe to contract $uv$. 
Assume that there are $2k + 1$ edge-disjoint paths from $u$ to $v$. 

Majority of paths are for sure not hit... so a majority candidate is always correct: We can infer $\Lambda(v)$ even if we do not know which constraints are not respected!
Assume that there are $2k + 1$ edge-disjoint paths from $u$ to $v$. Suppose that we know $\Lambda(u)$. A majority of paths are for sure not hit... so a majority candidate is always correct: We can infer $\Lambda(v)$ even if we do not know which constraints are not respected!
Assume that there are $2k + 1$ edge-disjoint paths from $u$ to $v$.
Suppose that we know $\Lambda(u)$.
Each path gives a candidate $\varphi_{P_i}(\Lambda(u))$ for $\Lambda(v)$, correct assuming the path is not hit by a disrespected constraint.
Assume that there are $2k + 1$ edge-disjoint paths from $u$ to $v$.
Suppose that we know $\Lambda(u)$.
Each path gives a candidate $\varphi_{P_i}(\Lambda(u))$ for $\Lambda(v)$, correct assuming the path is not hit by a disrespected constraint.
Majority of paths are for sure not hit...
High connectivity lemma

Assume that there are $2k + 1$ edge-disjoint paths from $u$ to $v$.
Suppose that we know $\Lambda(u)$.
Each path gives a candidate $\varphi_{P_i}(\Lambda(u))$ for $\Lambda(v)$, correct assuming the path is not hit by a disrespected constraint.
Majority of paths are for sure not hit...

so a majority candidate is always correct:

We can infer $\Lambda(v)$ even if we do not know which constraints are not respected!
Assume that we have a $2k$-edge separator $S$, and suppose both sides are connected and of size larger than $q(k)$.
Look at one side, iterate through all labelings of endpoints of $S$. 
For each labeling, mark some optimum solution of size $\leq k$ (or nothing if there is no such).
Divide and conquer

For each labeling, mark some optimum solution of size $\leq k$ (or nothing if there is no such).
For each labeling, mark some optimum solution of size $\leq k$ (or nothing if there is no such).
Divide and conquer

Recall that we had both of the sides.

$S > q(k)$
Claim. Each unmarked edge is not contained in some optimum solution.
Contract all the unmarked edges; if $q(k) \geq k \cdot |\Sigma|^{2k} + 1$, something gets contracted.
Border problem

- **Problem**: we iterate through the labelings of the border.
Border problem

- **Problem**: we iterate through the labelings of the border.
  - We cannot afford a recursive call for every labeling.
Border problem

Problem: we iterate through the labelings of the border.
- We cannot afford a recursive call for every labeling.
- How do we control growth of the border?
Border problem

- **Problem**: we iterate through the labelings of the border.
  - We cannot afford a recursive call for every labeling.
  - How do we control growth of the border?

- **Idea**: generalize the problem — incorporate the border in the definition.
Border problem

**Problem**: we iterate through the labelings of the border.
- We cannot afford a recursive call for every labeling.
- How do we control growth of the border?

**Idea**: generalize the problem — incorporate the border in the definition.

**Border ULC**

**Input**: a $\Sigma$-labeled graph $G$, an integer $k$, and a set $T$ of at most $4k$ terminals

**Output**: for every labeling $\Lambda_0$ of $T$, an optimum set of edges $F$ after removing which $\Lambda_0$ can be extended on $G$, or $\perp$ if no such $F$ of cardinality $\leq k$ exists.
Assume that we have a $2k$-edge separator $S$, and suppose both sides are connected and of size larger than $q(k)$. 
Recursive understanding

One of the sides has at most $2k$ terminals. (assume it is the left one)
Recursive understanding

Do the marking by a recursive call. The border becomes also terminals.
Recursive understanding

Do the marking by a recursive call. The border becomes also terminals.
Recall that we had both of the sides.
Recursive understanding

Contract all the unmarked edges.
The whole algorithm

Is there a good separation?

Chitnis, Cygan, Hajiaghayi, Pilipczuk ×2
Randomized Contractions 13/21
The whole algorithm

Is there a good separation?

$(q, 2k)$-good separation
The whole algorithm

Is there a good separation?

\((q, 2k)\)-good separation

Recurse to mark relevant edges
The whole algorithm

Is there a good separation?

(q, 2k)-good separation

Recursion to mark relevant edges

Contract unmarked edges

Time analysis

We either solve the problem completely, or spend $O^*(2^{O(k \log q)})$ time to detect some irrelevant edges. Total $O^*(2^{O(k \log q)})$ running time follows.
The whole algorithm

Is there a good separation?

- (q, 2k)-good separation
  - Recurse to mark relevant edges
  - Contract unmarked edges

- No (q, 2k)-good separation

High-connectivity phase, solve directly

Time analysis

We either solve the problem completely, or spend $O^*(2^{O(k \log q)})$ time to detect some irrelevant edges. Total $O^*(2^{O(k \log q)})$ running time follows.
The whole algorithm

1. Is there a good separation?
   - (q, 2k)-good separation
     - Recurse to mark relevant edges
     - Contract unmarked edges
   - No (q, 2k)-good separation
     - High-connectivity phase, solve directly

Time analysis
We either solve the problem completely, or spend $O^\ast(2^{O(k \log q)})$ time to detect some irrelevant edges. Total $O^\ast(2^{O(k \log q)})$ running time follows.
The whole algorithm

Is there a good separation?

\(O^*(2^{O(k \log q)})\) time

\((q, 2k)\)-good separation

Recurse to mark relevant edges

Contract unmarked edges

\((q, 2k)\)-good separation

No \((q, 2k)\)-good separation

High-connectivity phase, solve directly

Time analysis

We either solve the problem completely, or spend \(O^*(2^{O(k \log q)})\) time to detect some irrelevant edges.

Total \(O^*(2^{O(k \log q)})\) running time follows.
The whole algorithm

Is there a good separation?

- \((q, 2k)\)-good separation
  - Recurse to mark relevant edges
  - Contract unmarked edges

- No \((q, 2k)\)-good separation
  - High-connectivity phase, solve directly

\(O^*(2^{O(k \log q)})\) time

Time analysis

We either solve the problem completely, or spend \(O^*(2^{O(k \log q)})\) time to detect some irrelevant edges. Total \(O^*(2^{O(k \log q)})\) running time follows.
The whole algorithm

Is there a good separation?

\(O^*(2^{O(k \log q)})\) time

\((q, 2k)\)-good separation

Recurs to mark relevant edges

Contract unmarked edges

\(O^*(2^{O(k \log q)})\) time

No \((q, 2k)\)-good separation

High-connectivity phase, solve directly

Time analysis

We either solve the problem completely, or spend \(O^*(2^{O(k \log q)})\) time to detect some irrelevant edges. Total \(O^*(2^{O(k \log q)})\) running time follows.
The whole algorithm

Is there a good separation?

- (q, 2k)-good separation
  - Recurse to mark relevant edges
  - Contract unmarked edges

- No (q, 2k)-good separation
  - High-connectivity phase, solve directly

Time analysis

We either solve the problem completely,
The whole algorithm

Is there a good separation?

- *(q, 2k)-good separation
  - Recurse to mark relevant edges
  - Contract unmarked edges

- No *(q, 2k)-good separation
  - High-connectivity phase, solve directly

Time analysis

We either solve the problem completely, or spend \(O^*(2^{O(k \log q)})\) time to detect some irrelevant edges.
The whole algorithm

Is there a good separation?

$(q, 2k)$-good separation
- Recurse to mark relevant edges
- Contract unmarked edges

No $(q, 2k)$-good separation
- High-connectivity phase, solve directly

$O^*(2^{O(k \log q)})$ time

Time analysis
We either solve the problem completely, or spend $O^*(2^{O(k \log q)})$ time to detect some irrelevant edges. Total $O^*(2^{O(k \log q)})$ running time follows.
The whole algorithm

We either solve the problem completely, or spend $O^*(2^{O(k \log q)})$ time to detect some irrelevant edges. Total $O^*(2^{O(k \log q)})$ running time follows.
Recall: no \((q, k)\)-good edge separation could be found.
Recall: no \((q, k)\)-good edge separation could be found.

Hence, for every two disjoint connected sets \(X, Y\), such that \(|X|, |Y| > q\), there are \(2k + 1\) edge-disjoint paths between \(X\) and \(Y\).
**Recall**: no \((q, k)\)-good edge separation could be found.

Hence, for every two disjoint connected sets \(X, Y\), such that \(|X|, |Y| > q\), there are \(2k + 1\) edge-disjoint paths between \(X\) and \(Y\).

**Goal**: Use this property to apply the high-connectivity lemma.
Recall: no \((q, k)\)-good edge separation could be found. Hence, for every two disjoint connected sets \(X, Y\), such that \(|X|, |Y| > q\), there are \(2k + 1\) edge-disjoint paths between \(X\) and \(Y\).

Goal: Use this property to apply the high-connectivity lemma.

For simplicity assume that there are no terminals.
High-connectivity phase

- **Recall**: no \((q, k)\)-good edge separation could be found.
- Hence, for every two disjoint connected sets \(X, Y\), such that \(|X|, |Y| > q\), there are \(2k + 1\) edge-disjoint paths between \(X\) and \(Y\).
- **Goal**: Use this property to apply the high-connectivity lemma.
- For simplicity assume that there are no terminals.
- Fix an optimum solution \(F\) of size \(\leq k\), and examine the graph after removing \(F\).
High-connectivity phase

- **Recall:** no \((q, k)\)-good edge separation could be found.
- Hence, for every two disjoint connected sets \(X, Y\), such that \(|X|, |Y| > q\), there are \(2k + 1\) edge-disjoint paths between \(X\) and \(Y\).
- **Goal:** Use this property to apply the high-connectivity lemma.
- For simplicity assume that there are no terminals.
- Fix an optimum solution \(F\) of size \(\leq k\), and examine the graph after removing \(F\).
- It can contain at most \(k\) small connected components of size at most \(q\), and at most one big of arbitrarily large size.
Colour coding

- For every edge of the graph, independently toss a coin.
Colour coding

- For every edge of the graph, independently toss a coin.
- With probability $\frac{1}{2}$ it becomes red, and with probability $\frac{1}{2}$ blue.
For every edge of the graph, independently toss a coin.

- With probability $\frac{1}{2}$ it becomes red, and with probability $\frac{1}{2}$ blue.
- **We aim at the event that:**

  For every small component, some of its spanning tree becomes blue; for every endpoint $v$ of an edge from $F$ in the big component, we have a blue tree on $q+1$ vertices adjacent to $v$ (anchor).
For every edge of the graph, independently toss a coin.
With probability $\frac{1}{2}$ it becomes red, and with probability $\frac{1}{2}$ blue.
We aim at the event that:
- the whole $F$ becomes red;
Colour coding

- For every edge of the graph, independently toss a coin.
- With probability $\frac{1}{2}$ it becomes red, and with probability $\frac{1}{2}$ blue.
- We aim at the event that:
  - the whole $F$ becomes red;
  - for every small component, some its spanning tree becomes blue;
For every edge of the graph, independently toss a coin.

With probability $\frac{1}{2}$ it becomes red, and with probability $\frac{1}{2}$ blue.

We aim at the event that:

- the whole $F$ becomes red;
- for every small component, some its spanning tree becomes blue;
- for every endpoint $v$ of an edge from $F$ in the big component, we have a blue tree on $q + 1$ vertices adjacent to $v$ (anchor).
Colour coding

$C_0$

$C_1$

$C_2$

$C_3$

$C_4$

$C_5$
Colour coding

$C_0$

$C_1$

$C_2$

$C_3$

$C_4$

$C_5$
We request something about $O(qk)$ edges, so with $2^{-O(qk)}$ probability we have a correct colouring.
Colour coding: analysis

- We request something about $O(qk)$ edges, so with $2^{-O(qk)}$ probability we have a correct colouring.
- Can be boosted to $2^{-O(k \log q)}$ and derandomized. With $O^*(2^{O(k \log q)})$ overhead we have a correct colouring.
We request something about $O(qk)$ edges, so with $2^{-O(qk)}$ probability we have a correct colouring.

Can be boosted to $2^{-O(k \log q)}$ and derandomized. With $O^*(2^{O(k \log q)})$ overhead we have a correct colouring.

The interesting objects are connected components of the blue edges.
We request something about $O(qk)$ edges, so with $2^{-O(qk)}$ probability we have a correct colouring.

Can be boosted to $2^{-O(k \log q)}$ and derandomized. With $O^*(2^{O(k \log q)})$ overhead we have a correct colouring.

The interesting objects are connected components of the blue edges.

- Recall that blue edges cannot be in $F$. 

Recall that blue edges cannot be in $F$. 

Stains: A stain is large if it has $> q$ vertices, and small otherwise.
Colour coding: analysis

- We request something about $O(qk)$ edges, so with $2^{-O(qk)}$ probability we have a correct colouring.
- Can be boosted to $2^{-O(k \log q)}$ and derandomized. With $O^*(2^{O(k \log q)})$ overhead we have a correct colouring.
- The interesting objects are connected components of the blue edges.
  - Recall that blue edges cannot be in $F$.
- Such components are called stains:
Colour coding: analysis

- We request something about $O(qk)$ edges, so with $2^{-O(qk)}$ probability we have a correct colouring.
- Can be boosted to $2^{-O(k \log q)}$ and derandomized. With $O^*(2^{O(k \log q)})$ overhead we have a correct colouring.
- The interesting objects are connected components of the blue edges.
  - Recall that blue edges cannot be in $F$.
- Such components are called stains:
  - A stain is large if it has $> q$ vertices, and small otherwise.
Colour coding: analysis

- We request something about $O(qk)$ edges, so with $2^{-O(qk)}$ probability we have a correct colouring.
- Can be boosted to $2^{-O(k \log q)}$ and derandomized. With $O^*(2^{O(k \log q)})$ overhead we have a correct colouring.
- The interesting objects are connected components of the blue edges.
  - Recall that blue edges cannot be in $F$.
- Such components are called **stains**:
  - A stain is **large** if it has $> q$ vertices, and **small** otherwise.
  - All small components become small stains, while anchors are contained in large stains.
Take any two large stains $S_1, S_2$. 

There are $2k+1$ edge-disjoint paths between them!

Guess labeling of any vertex of any large stain ($|\Sigma|$ choices), propagate it to its stain via blue edges, and to all the other large stains using the high-connectivity lemma.

At a cost of $|\Sigma|$ overhead, we have all the large stains labeled!
Take any two large stains $S_1, S_2$.

There are $2k + 1$ edge-disjoint paths between them!
Take any two large stains $S_1, S_2$.
There are $2k + 1$ edge-disjoint paths between them!

- Guess labeling of any vertex of any large stain ($|\Sigma|$ choices),
Take any two large stains $S_1, S_2$.

There are $2k + 1$ edge-disjoint paths between them!

- Guess labeling of any vertex of any large stain ($|\Sigma|$ choices),
- propagate it to its stain via blue edges,
Large Stains

- Take any two large stains $S_1, S_2$.
- There are $2k + 1$ edge-disjoint paths between them!
  - Guess labeling of any vertex of any large stain ($|\Sigma|$ choices),
  - propagate it to its stain via blue edges,
  - and to all the other large stains using the high-connectivity lemma.

At a cost of $|\Sigma|$ overhead, we have all the large stains labeled!
Large stains

- Take any two large stains $S_1, S_2$.
- There are $2k + 1$ edge-disjoint paths between them!
  - Guess labeling of any vertex of any large stain ($|\Sigma|$ choices),
  - propagate it to its stain via blue edges,
  - and to all the other large stains using the high-connectivity lemma.
- At a cost of $|\Sigma|$ overhead, we have all the large stains labeled!
Goal:
determine which red edges are in the solution, and which not.
Small stains

Edges with both endpoints in large stains: easy
Small stains

**Group of stains:**
connected component of $G \setminus$ large stains
Small stains

Anchors:
every group either goes fully into $F$, or fully out of $F$. 

Chitnis, Cygan, Hajiaghayi, Pilipczuk × 2
Randomized Contractions
Every group on which we cannot extend must go into $F$. 
Every group on which we cannot extend **must** go into $F$. Every group on which we can extend **can** go out of $F$. 
What else?

- **Steiner Cut:** delete \( k \) edges to get \( \geq \ell \) components with terminals.
What else?

- **Steiner Cut**: delete $k$ edges to get $\geq \ell$ components with terminals.
  - A polynomial-time knapsack DP in high-connectivity phase.
What else?

- **Steiner Cut**: delete $k$ edges to get $\geq \ell$ components with terminals.
  - A polynomial-time knapsack DP in high-connectivity phase.

- **Multiway Cut-Uncut**: delete $k$ edges to separate groups of terminals, but **do not** separate within a group.

Much more technically involved. We need a second type of separations. Branching after colour-coding.
What else?

- **Steiner Cut**: delete $k$ edges to get $\geq \ell$ components with terminals.
  - A polynomial-time knapsack DP in high-connectivity phase.

- **Multiway Cut-Uncut**: delete $k$ edges to separate groups of terminals, but **do not** separate within a group.
  - More complicated border problem.
What else?

- **Steiner Cut**: delete $k$ edges to get $\geq \ell$ components with terminals.
  - A polynomial-time knapsack DP in high-connectivity phase.
- **Multiway Cut-Uncut**: delete $k$ edges to separate groups of terminals, but **do not** separate within a group.
  - More complicated border problem.
- Node-deletion versions of MwC-U and ULC.
What else?

- **Steiner Cut**: delete $k$ edges to get $\geq \ell$ components with terminals.
  - A polynomial-time knapsack DP in high-connectivity phase.

- **Multiway Cut-Uncut**: delete $k$ edges to separate groups of terminals, but **do not** separate within a group.
  - More complicated border problem.

- Node-deletion versions of MwC-U and ULC.
  - Much more technically involved.
What else?

- **Steiner Cut**: delete \( k \) edges to get \( \geq \ell \) components with terminals.
  - A polynomial-time knapsack DP in high-connectivity phase.

- **Multiway Cut-Uncut**: delete \( k \) edges to separate groups of terminals, but **do not** separate within a group.
  - More complicated border problem.

- **Node-deletion versions of MwC-U and ULC**.
  - Much more technically involved.
  - *We need second type of separations.*
What else?

- **Steiner Cut**: delete $k$ edges to get $\geq \ell$ components with terminals.
  - A polynomial-time knapsack DP in high-connectivity phase.

- **Multiway Cut-Uncut**: delete $k$ edges to separate groups of terminals, but do not separate within a group.
  - More complicated border problem.

- Node-deletion versions of MwC-U and ULC.
  - Much more technically involved.
  - We need second type of separations.
  - Branching after colour-coding.
We have shown a new technique for cut problems that is an alternative to other tools.
Conclusions

- We have shown a new technique for cut problems that is an alternative to other tools.
- **Typical running time:** $O^*(2^{O(k^2 \log k)})$
Conclusions

- We have shown a new technique for cut problems that is an alternative to other tools.
- Typical running time: $O^*(2^{O(k^2 \log k)})$
- Can this be made better?
Conclusions

- We have shown a new technique for cut problems that is an alternative to other tools.
- Typical running time: $O^*(2^{O(k^2 \log k)})$
- Can this be made better?
- **Thank you for attention!**