Two-sets cut-uncut on planar graphs

Tuukka Korhonen

UNIVERSITY OF BERGEN

based on joint work with Matthias Bentert, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach

University of Warsaw Algorithms Seminar

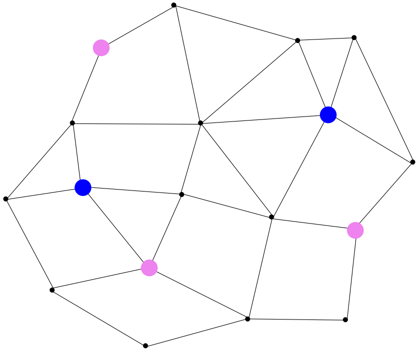
19 May 2023

Two-Sets Cut-Uncut

TWO-SETS CUT-UNCUT

Input: Undirected graph and two sets of vertices S and T.

Output: Smallest number of edges to delete so that S is separated from T, but vertices in S, (resp. in T), stay in the same component.

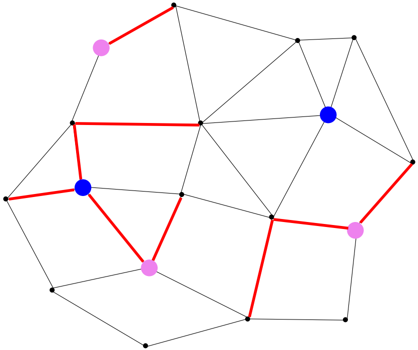


Two-Sets Cut-Uncut

TWO-SETS CUT-UNCUT

Input: Undirected graph and two sets of vertices S and T.

Output: Smallest number of edges to delete so that S is separated from T, but vertices in S, (resp. in T), stay in the same component.

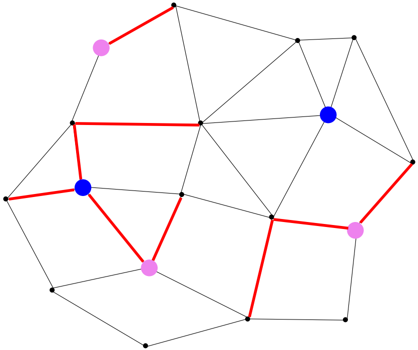


Two-Sets Cut-Uncut

TWO-SETS CUT-UNCUT

Input:	Undirected graph and two sets of vertices S and T.
Output:	Smallest number of edges to delete so that S is separated from T, but vertices in S, (resp. in T), stay in the same component.

- Equivalently, a smallest S-T cut that is a minimal cut



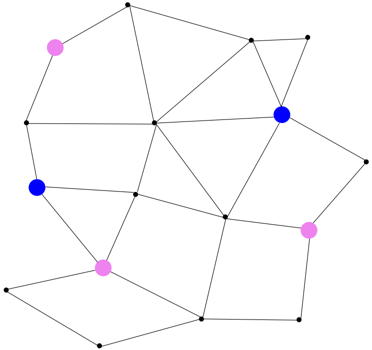
Two-Sets Cut-Uncut

TWO-SETS CUT-UNCUT

Input: Undirected graph and two sets of vertices S and T.

Output: Smallest number of edges to delete so that S is separated from T, but vertices in S, (resp. in T), stay in the same component.

- Equivalently, a smallest S-T cut that is a minimal cut
- Possible that no solution exists!



Previous works

Previous works

- The decision version: Two-Disjoint Connected Subgraphs

Previous works

- The decision version: Two-Disjoint Connected Subgraphs
  - NP-hard on planar graphs [Gray, Kammer, Löffler, Silveira '12]

Previous works

- The decision version: Two-Disjoint Connected Subgraphs
  - NP-hard on planar graphs [Gray, Kammer, Löffler, Silveira '12]
  - NP-hard on general graphs even when $|S| = 2$ [van't Hof, Paulusma, Woeginger '09]

Previous works

- The decision version: Two-Disjoint Connected Subgraphs
  - NP-hard on planar graphs [Gray, Kammer, Löffler, Silveira '12]
  - NP-hard on general graphs even when $|S| = 2$ [van't Hof, Paulusma, Woeginger '09]
  - Faster than 2^n exponential-time algorithms [Cygan, Pilipczuk, Pilipczuk, Wojtaszczyk '14], [Telle & Villanger '13]

Previous works

- The decision version: Two-Disjoint Connected Subgraphs
  - NP-hard on planar graphs [Gray, Kammer, Löffler, Silveira '12]
  - NP-hard on general graphs even when $|S| = 2$ [van't Hof, Paulusma, Woeginger '09]
  - Faster than 2^n exponential-time algorithms [Cygan, Pilipczuk, Pilipczuk, Wojtaszczyk '14], [Telle & Villanger '13]
- Parameterized by solution size k: $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ time (in very general setting) [Chitnis, Cygan, Hajiaghayi, Pilipczuk, Pilipczuk '16]

Previous works

- The decision version: Two-Disjoint Connected Subgraphs
  - NP-hard on planar graphs [Gray, Kammer, Löffler, Silveira '12]
  - NP-hard on general graphs even when $|S| = 2$ [van't Hof, Paulusma, Woeginger '09]
  - Faster than 2^n exponential-time algorithms [Cygan, Pilipczuk, Pilipczuk, Wojtaszczyk '14], [Telle & Villanger '13]
- Parameterized by solution size k: $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ time (in very general setting) [Chitnis, Cygan, Hajiaghayi, Pilipczuk, Pilipczuk '16]
- Polynomial-time algorithms on planar graphs:

Previous works

- The decision version: Two-Disjoint Connected Subgraphs
  - NP-hard on planar graphs [Gray, Kammer, Löffler, Silveira '12]
  - NP-hard on general graphs even when $|S| = 2$ [van't Hof, Paulusma, Woeginger '09]
  - Faster than 2^n exponential-time algorithms [Cygan, Pilipczuk, Pilipczuk, Wojtaszczyk '14], [Telle & Villanger '13]
- Parameterized by solution size k: $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ time (in very general setting) [Chitnis, Cygan, Hajiaghayi, Pilipczuk, Pilipczuk '16]
- Polynomial-time algorithms on planar graphs:
  - When $|S| = 1$ and $|T| = 2$ [Duan & Xu '14]

Previous works

- The decision version: Two-Disjoint Connected Subgraphs
  - NP-hard on planar graphs [Gray, Kammer, Löffler, Silveira '12]
  - NP-hard on general graphs even when $|S| = 2$ [van't Hof, Paulusma, Woeginger '09]
  - Faster than 2^n exponential-time algorithms [Cygan, Pilipczuk, Pilipczuk, Wojtaszczyk '14], [Telle & Villanger '13]
- Parameterized by solution size k: $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$ time (in very general setting) [Chitnis, Cygan, Hajiaghayi, Pilipczuk, Pilipczuk '16]
- Polynomial-time algorithms on planar graphs:
  - When $|S| = 1$ and $|T| = 2$ [Duan & Xu '14]
  - When $|S| = 1$ [Bezáková & Langley '14]

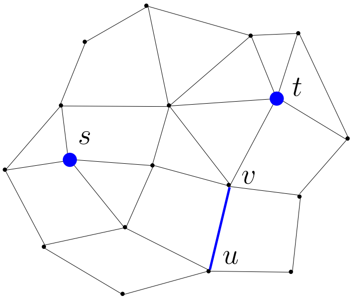
Special case: Network Diversion

Special case: Network Diversion

NETWORK DIVERSION

Input: Undirected graph, vertices s and t, edge uv.

Output: Smallest number of edges to delete so that uv becomes an s-t-bridge.



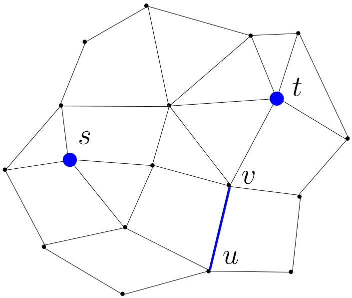
Special case: Network Diversion

NETWORK DIVERSION

Input: Undirected graph, vertices s and t, edge uv.

Output: Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:



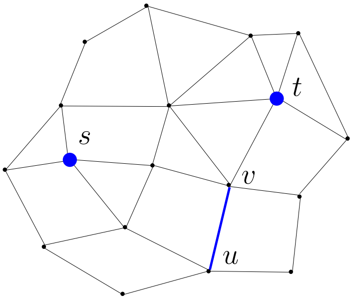
Special case: Network Diversion

NETWORK DIVERSION

Input:	Undirected graph, vertices s and t, edge uv.
Output:	Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:

- Try with $S = \{s, v\}$, $T = \{t, u\}$



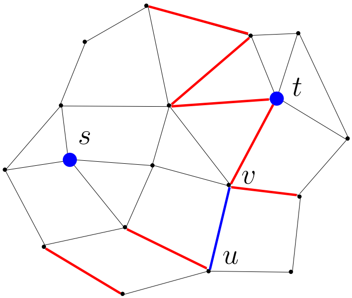
Special case: Network Diversion

NETWORK DIVERSION

Input:	Undirected graph, vertices s and t, edge uv.
Output:	Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:

- Try with $S = \{s, v\}$, $T = \{t, u\}$



Special case: Network Diversion

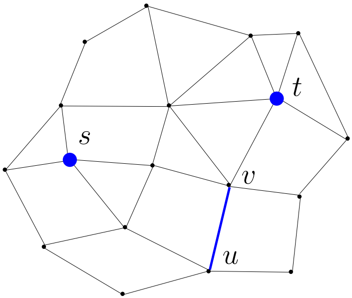
NETWORK DIVERSION

Input: Undirected graph, vertices s and t, edge uv.

Output: Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:

- Try with $S = \{s, v\}$, $T = \{t, u\}$ and with $S = \{s, u\}$, $T = \{t, v\}$



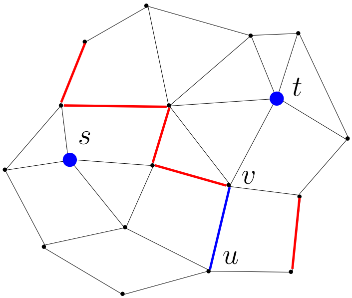
Special case: Network Diversion

NETWORK DIVERSION

Input:	Undirected graph, vertices s and t, edge uv.
Output:	Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:

- Try with $S = \{s, v\}$, $T = \{t, u\}$ and with $S = \{s, u\}$, $T = \{t, v\}$



Special case: Network Diversion

NETWORK DIVERSION

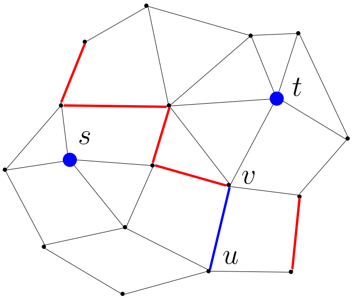
Input: Undirected graph, vertices s and t, edge uv.

Output: Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:

- Try with $S = \{s, v\}$, $T = \{t, u\}$ and with $S = \{s, u\}$, $T = \{t, v\}$

Previous works on planar graphs:



Special case: Network Diversion

NETWORK DIVERSION

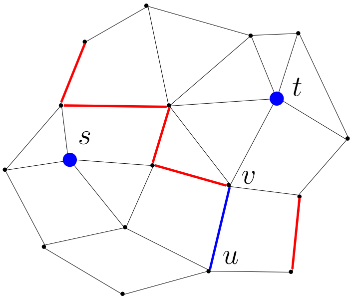
Input:	Undirected graph, vertices s and t, edge uv.
Output:	Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:

- Try with $S = \{s, v\}$, $T = \{t, u\}$ and with $S = \{s, u\}$, $T = \{t, v\}$

Previous works on planar graphs:

- Polytime when s and t on the same face [Cullenbine, Wood, Newman '13]



Special case: Network Diversion

NETWORK DIVERSION

Input: Undirected graph, vertices s and t, edge uv.

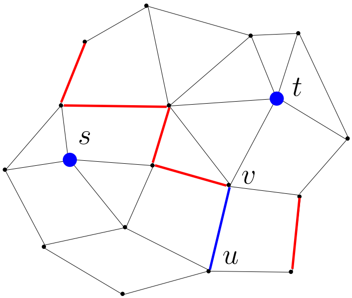
Output: Smallest number of edges to delete so that uv becomes an s-t-bridge.

Reduction to Two-Sets Cut-Uncut with $|S| = 2$ and $|T| = 2$:

- Try with $S = \{s, v\}$, $T = \{t, u\}$ and with $S = \{s, u\}$, $T = \{t, v\}$

Previous works on planar graphs:

- Polytime when s and t on the same face [Cullenbine, Wood, Newman '13]
- Incorrect claim of polytime on planar graphs [Duan, Jafarian, Al-Shaer, Xu '14]



Our results

Our results

Theorem

Two-Sets Cut-Uncut can be solved on planar graphs in $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time by randomized algorithm.

Our results

Theorem

Two-Sets Cut-Uncut can be solved on planar graphs in $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time by randomized algorithm.

- Implies polynomial-time algorithm for Network Diversion on planar graphs

Our results

Theorem

Two-Sets Cut-Uncut can be solved on planar graphs in $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time by randomized algorithm.

- Implies polynomial-time algorithm for Network Diversion on planar graphs
- Implies FPT algorithms for Generalized Network Diversion and Location Constrained Shortest Path

Our results

Theorem

Two-Sets Cut-Uncut can be solved on planar graphs in $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time by randomized algorithm.

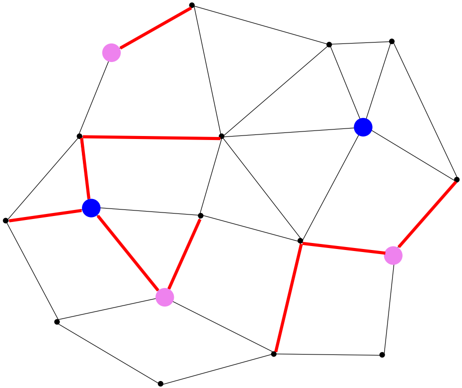
- Implies polynomial-time algorithm for Network Diversion on planar graphs
- Implies FPT algorithms for Generalized Network Diversion and Location Constrained Shortest Path

Theorem

Two-Sets Cut-Uncut can be solved on plane graphs in $4^{r+\mathcal{O}(\sqrt{r})}n^{\mathcal{O}(1)}$ time, where r is the minimum number of faces to cover $S \cup T$.

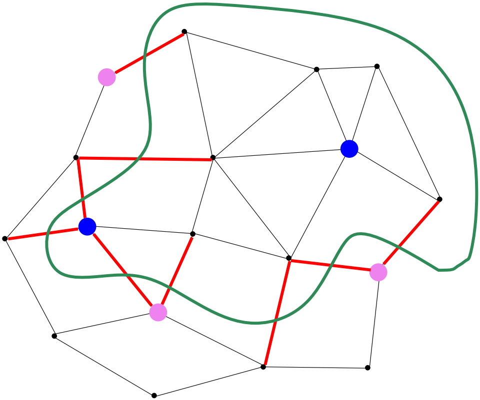
The algorithm

Minimal cuts in planar graphs



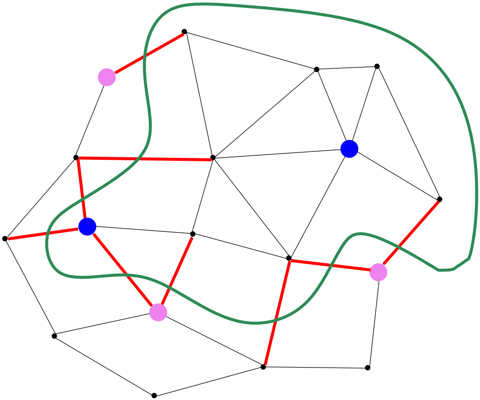
Minimal cuts in planar graphs

- A minimal cut corresponds to a cycle in the dual graph



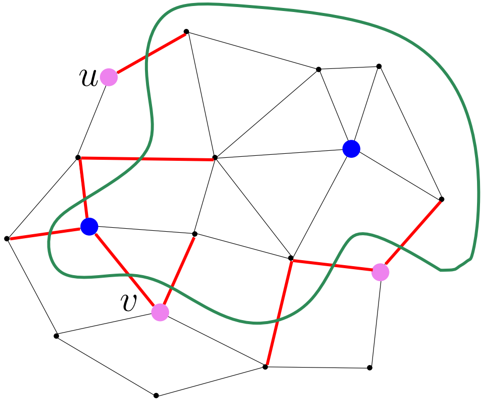
Minimal cuts in planar graphs

- A minimal cut corresponds to a cycle in the dual graph
- Plan: Phrase the problem as a problem about finding a cycle in the dual



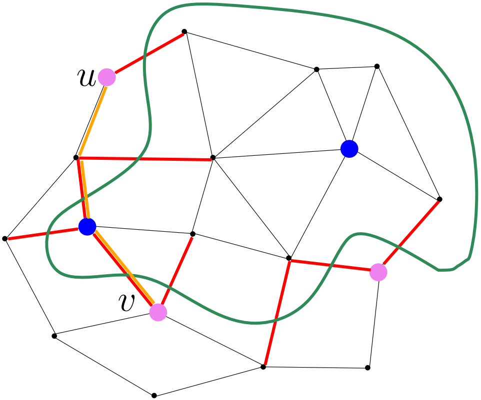
Minimal cuts in planar graphs

- A minimal cut corresponds to a cycle in the dual graph
- Plan: Phrase the problem as a problem about finding a cycle in the dual
- How to understand if *u* and *v* are on the same side of the cut?



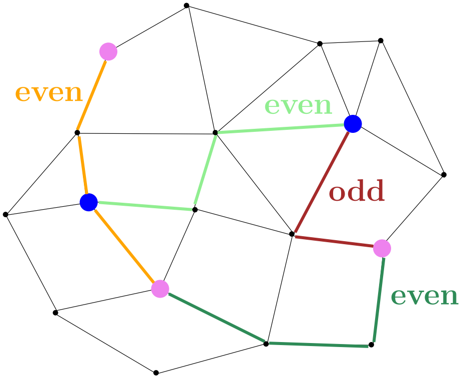
Minimal cuts in planar graphs

- A minimal cut corresponds to a cycle in the dual graph
- Plan: Phrase the problem as a problem about finding a cycle in the dual
- How to understand if u and v are on the same side of the cut?
  - Pick any u-v-path and count the parity of how many times it is cut



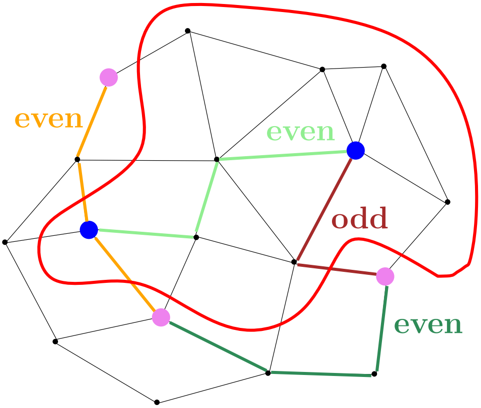
Minimal cuts in planar graphs

- A minimal cut corresponds to a cycle in the dual graph
- Plan: Phrase the problem as a problem about finding a cycle in the dual
- How to understand if u and v are on the same side of the cut?
  - Pick any u-v-path and count the parity of how many times it is cut
- Reduction to shortest cycle under $|S| + |T| - 1$ parity constraints



Minimal cuts in planar graphs

- A minimal cut corresponds to a cycle in the dual graph
- Plan: Phrase the problem as a problem about finding a cycle in the dual
- How to understand if *u* and *v* are on the same side of the cut?
  - Pick any *u*-*v*-path and count the parity of how many times it is cut
- Reduction to shortest cycle under $|S| + |T| - 1$ parity constraints



Shortest paths in group-labeled graphs

Boolean Group-Labeled Shortest Path

Input: Undirected graph whose edges are labeled by elements of the Boolean group $(\mathbb{Z}_2^d, +)$, two vertices s and t, and an element $c \in \mathbb{Z}_2^d$.

Output: Shortest s-t-path whose edge labels sum up to c.

Shortest paths in group-labeled graphs

BOOLEAN GROUP-LABELED SHORTEST PATH

Input: Undirected graph whose edges are labeled by elements of the Boolean group $(\mathbb{Z}_2^d, +)$, two vertices s and t, and an element $c \in \mathbb{Z}_2^d$.

Output: Shortest s-t-path whose edge labels sum up to c.

- Reduction from Two-Sets Cut-Uncut to Boolean Group-Labeled Shortest Path with $d = |S| + |T| - 1$

Shortest paths in group-labeled graphs

BOOLEAN GROUP-LABELED SHORTEST PATH

Input: Undirected graph whose edges are labeled by elements of the Boolean group $(\mathbb{Z}_2^d, +)$, two vertices s and t, and an element $c \in \mathbb{Z}_2^d$.

Output: Shortest s-t-path whose edge labels sum up to c.

- Reduction from Two-Sets Cut-Uncut to Boolean Group-Labeled Shortest Path with $d = |S| + |T| - 1$

Theorem

There is a $2^d n^{\mathcal{O}(1)}$ time randomized algorithm for Boolean Group-Labeled Shortest Path.

Shortest paths in group-labeled graphs

BOOLEAN GROUP-LABELED SHORTEST PATH

Input: Undirected graph whose edges are labeled by elements of the Boolean group $(\mathbb{Z}_2^d, +)$, two vertices s and t, and an element $c \in \mathbb{Z}_2^d$.

Output: Shortest s-t-path whose edge labels sum up to c.

- Reduction from Two-Sets Cut-Uncut to Boolean Group-Labeled Shortest Path with $d = |S| + |T| - 1$

Theorem

There is a $2^d n^{\mathcal{O}(1)}$ time randomized algorithm for Boolean Group-Labeled Shortest Path.

- Generalizes the result of [Derigs'85] for $d = 1$

Shortest paths in group-labeled graphs

Boolean Group-Labeled Shortest Path

Input: Undirected graph whose edges are labeled by elements of the Boolean group $(\mathbb{Z}_2^d, +)$, two vertices s and t, and an element $c \in \mathbb{Z}_2^d$.

Output: Shortest s-t-path whose edge labels sum up to c.

- Reduction from Two-Sets Cut-Uncut to Boolean Group-Labeled Shortest Path with $d = |S| + |T| - 1$

Theorem

There is a $2^d n^{\mathcal{O}(1)}$ time randomized algorithm for Boolean Group-Labeled Shortest Path.

- Generalizes the result of [Derigs'85] for $d = 1$
- Generalizes the result of [Björklund, Husfeldt, Taslaman '12] for finding a shortest cycle through T specified vertices in time $2^{|T|} n^{\mathcal{O}(1)}$

The algorithm

The Algorithm for Boolean Group-Labeled Shortest Path

Solving problems by Schwartz–Zippel

Plan:

Solving problems by Schwartz–Zippel

Plan:

For integer ℓ, define a multivariate polynomial P_ℓ of degree ℓ over the field $\mathrm{GF}(2^{\lceil \log \ell \rceil + 2})$ so that:

Solving problems by Schwartz–Zippel

Plan:

For integer ℓ, define a multivariate polynomial P_ℓ of degree ℓ over the field $GF(2^{\lceil \log \ell \rceil + 2})$ so that:

1. if no solutions of length $\leq \ell$ exists, then P_ℓ identically zero

Solving problems by Schwartz–Zippel

Plan:

For integer ℓ, define a multivariate polynomial P_ℓ of degree ℓ over the field GF($2^{\lceil \log \ell \rceil + 2}$) so that:

1. if no solutions of length $\leq \ell$ exists, then P_ℓ identically zero
2. if a solution of length ℓ exists, then P_ℓ non-zero

Solving problems by Schwartz–Zippel

Plan:

For integer ℓ, define a multivariate polynomial P_ℓ of degree ℓ over the field GF($2^{\lceil \log \ell \rceil + 2}$) so that:

1. if no solutions of length $\leq \ell$ exists, then P_ℓ identically zero
2. if a solution of length ℓ exists, then P_ℓ non-zero
3. the value of P_ℓ can be evaluated in time $2^d n^{\mathcal{O}(1)}$

Solving problems by Schwartz–Zippel

Plan:

For integer ℓ, define a multivariate polynomial P_ℓ of degree ℓ over the field $\mathrm{GF}(2^{\lceil \log \ell \rceil + 2})$ so that:

1. if no solutions of length $\leq \ell$ exists, then P_ℓ identically zero
2. if a solution of length ℓ exists, then P_ℓ non-zero
3. the value of P_ℓ can be evaluated in time $2^d n^{\mathcal{O}(1)}$

$\Rightarrow$ By applying the Schwartz–Zippel lemma, the shortest solution can be found in randomized time $2^d n^{\mathcal{O}(1)}$

The polynomial

- An s-t-walk of length ℓ is a sequence $(s = v_0, v_1, v_2, \ldots, v_\ell = t)$ that is like a path, but vertices can repeat

The polynomial

- An s-t-walk of length ℓ is a sequence $(s = v_0, v_1, v_2, \ldots, v_\ell = t)$ that is like a path, but vertices can repeat
- An s-t-walk $(v_0, v_1, v_2, \ldots, v_\ell)$ of length ℓ is feasible if its edge labels sum up to the element $c \in \mathbb{Z}_2^d$

The polynomial

- An s-t-walk of length ℓ is a sequence $(s = v_0, v_1, v_2, \ldots, v_\ell = t)$ that is like a path, but vertices can repeat
- An s-t-walk $(v_0, v_1, v_2, \ldots, v_\ell)$ of length ℓ is feasible if its edge labels sum up to the element $c \in \mathbb{Z}_2^d$
- Let $\mathcal{W}_\ell$ be the set of feasible s-t-walks of length ℓ

The polynomial

- An s-t-walk of length ℓ is a sequence $(s = v_0, v_1, v_2, \ldots, v_\ell = t)$ that is like a path, but vertices can repeat
- An s-t-walk $(v_0, v_1, v_2, \ldots, v_\ell)$ of length ℓ is feasible if its edge labels sum up to the element $c \in \mathbb{Z}_2^d$
- Let $\mathcal{W}_\ell$ be the set of feasible s-t-walks of length ℓ
- We have a variable $\mathbf{x}(uv)$ for every edge $uv \in E(G)$

The polynomial

- An s-t-walk of length ℓ is a sequence $(s = v_0, v_1, v_2, \ldots, v_\ell = t)$ that is like a path, but vertices can repeat
- An s-t-walk $(v_0, v_1, v_2, \ldots, v_\ell)$ of length ℓ is feasible if its edge labels sum up to the element $c \in \mathbb{Z}_2^d$
- Let $\mathcal{W}_\ell$ be the set of feasible s-t-walks of length ℓ
- We have a variable $\mathbf{x}(uv)$ for every edge $uv \in E(G)$
- $P_\ell = \sum_{(v_0, v_1, \ldots, v_\ell) \in \mathcal{W}_\ell} \prod_{i=1}^{\ell} \mathbf{x}(v_{i-1} v_i)$

The polynomial

- An s-t-walk of length ℓ is a sequence $(s = v_0, v_1, v_2, \ldots, v_\ell = t)$ that is like a path, but vertices can repeat
- An s-t-walk $(v_0, v_1, v_2, \ldots, v_\ell)$ of length ℓ is feasible if its edge labels sum up to the element $c \in \mathbb{Z}_2^d$
- Let $\mathcal{W}_\ell$ be the set of feasible s-t-walks of length ℓ
- We have a variable $\mathbf{x}(uv)$ for every edge $uv \in E(G)$
- $P_\ell = \sum_{(v_0, v_1, \ldots, v_\ell) \in \mathcal{W}_\ell} \prod_{i=1}^{\ell} \mathbf{x}(v_{i-1} v_i)$ (over $\mathrm{GF}(2^{\lceil \log \ell \rceil + 2})$)

The polynomial

- $P_\ell = \sum_{(v_0, v_1, \ldots, v_\ell) \in \mathcal{W}_\ell} \prod_{i=1}^{\ell} \mathbf{x}(v_{i-1} v_i)$ (over $GF(2^{\lceil \log \ell \rceil + 2})$)

The polynomial

- $P_\ell = \sum_{(v_0, v_1, \ldots, v_\ell) \in \mathcal{W}_\ell} \prod_{i=1}^{\ell} \mathbf{x}(v_{i-1} v_i)$ (over $GF(2^{\lceil \log \ell \rceil + 2})$)
- Can evaluate in $2^d n^{\mathcal{O}(1)}$ time by dynamic programming over walks

The polynomial

- $P_\ell = \sum_{(v_0, v_1, \ldots, v_\ell) \in \mathcal{W}_\ell} \prod_{i=1}^{\ell} \mathbf{x}(v_{i-1} v_i)$ (over $GF(2^{\lceil \log \ell \rceil + 2})$)
- Can evaluate in $2^d n^{\mathcal{O}(1)}$ time by dynamic programming over walks
- Non-zero if a solution of length ℓ exists, because a solution gives monomial corresponding to exactly one walk

The polynomial

- $P_\ell = \sum_{(v_0, v_1, \ldots, v_\ell) \in \mathcal{W}_\ell} \prod_{i=1}^{\ell} \mathbf{x}(v_{i-1} v_i)$ (over $\mathrm{GF}(2^{\lceil \log \ell \rceil + 2})$)
- Can evaluate in $2^d n^{\mathcal{O}(1)}$ time by dynamic programming over walks
- Non-zero if a solution of length ℓ exists, because a solution gives monomial corresponding to exactly one walk
- Remains to show that if no solutions of length $\leq \ell$ exists, then each monomial appears an even number of times

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices

$$\mathbf{sabcdbcefbt}$$

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence

$$\mathrm{sa}\underline{\mathrm{bcdbcefb}}\mathrm{t}$$

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence

$$\mathbf{sa}\underline{\mathbf{bcdbcefb}}\mathbf{t} \Rightarrow \mathbf{sa}\underline{\mathbf{bfecbdcb}}\mathbf{t}$$

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable

sabcbadbebhefget

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable

s<u>abcba</u>dbebhefget

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable

sabcbadbebhefget

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable

sabcbadbebhefget

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable

sabcbadbebhefget

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable

sabcbadbebhefget

Cancellation of monomials

Goal: Define a function $\phi\colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable

$$\mathrm{sabcbadbebh}\underline{\mathrm{efge}}\mathrm{t} \Rightarrow \mathrm{sabcbadbebh}\underline{\mathrm{egfe}}\mathrm{t}$$

Cancellation of monomials

Goal: Define a function $\phi \colon \mathcal{W}_\ell \to \mathcal{W}_\ell$ so that for all $W \in \mathcal{W}_\ell$

1. $\phi(W)$ has the same multiset of edges as W
2. $\phi(W) \neq W$
3. $\phi(\phi(W)) = W$

- Think W as a string of vertices
- Find the first repeating vertex and reverse between its first and last occurrence
- Sometimes reversal does not work because of palindromes, but fixable
  - Here we use that the group is $(\mathbb{Z}_2^d, +)$

$$\mathrm{sabcbadbebh\underline{efge}t} \Rightarrow \mathrm{sabcbadbebh\underline{egfe}t}$$

Conclusion

- $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time algorithm for Two-Sets Cut-Uncut on planar graphs

Conclusion

- $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time algorithm for Two-Sets Cut-Uncut on planar graphs
- The case of $|S| = |T| = 2$ solves the open problem of Network Diversion on planar graphs

Conclusion

- $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time algorithm for Two-Sets Cut-Uncut on planar graphs
- The case of $|S| = |T| = 2$ solves the open problem of Network Diversion on planar graphs
- Implementable in practice

Conclusion

- $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time algorithm for Two-Sets Cut-Uncut on planar graphs
- The case of $|S| = |T| = 2$ solves the open problem of Network Diversion on planar graphs
- Implementable in practice
- Open problems:

Conclusion

- $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time algorithm for Two-Sets Cut-Uncut on planar graphs
- The case of $|S| = |T| = 2$ solves the open problem of Network Diversion on planar graphs
- Implementable in practice
- Open problems:
  - Network Diversion in general graphs: Polynomial-time or NP-complete?

Conclusion

- $2^{|S|+|T|}n^{\mathcal{O}(1)}$ time algorithm for Two-Sets Cut-Uncut on planar graphs
- The case of $|S| = |T| = 2$ solves the open problem of Network Diversion on planar graphs
- Implementable in practice
- Open problems:
  - Network Diversion in general graphs: Polynomial-time or NP-complete?
  - Three-Sets Cut-Uncut on planar graphs: Open even for three sets of sizes $2, 1, 1$

Thank you!

Thank you!