Shortest Cycles With Monotone Submodular Costs

Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen,
Daniel Lokshtanov[1], and Giannos Stamoulis[2]

UNIVERSITY OF BERGEN

[1]University of California Santa Barbara [2] LIRMM, Universite de Montpellier, CNRS

SODA 2023

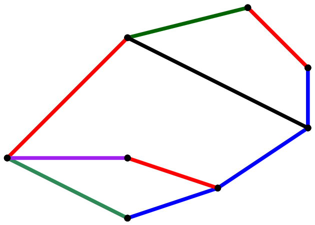
23 January 2023

The Minimum Color Cycle Problem

Minimum Color Cycle

Input: Edge-colored undirected graph.

Output: Cycle with the minimum number of different colors.

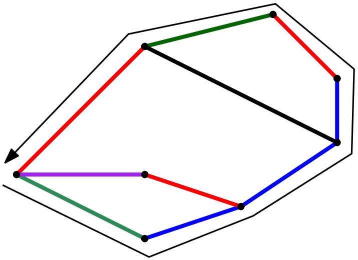


The Minimum Color Cycle Problem

Minimum Color Cycle

Input: Edge-colored undirected graph.

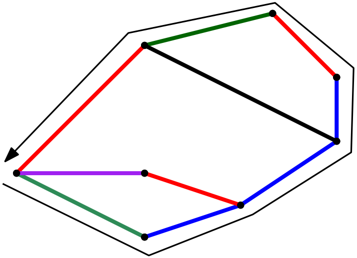
Output: Cycle with the minimum number of different colors.



The Minimum Color Cycle Problem

MINIMUM COLOR CYCLE

Input: Edge-colored undirected graph.
Output: Cycle with the minimum number of different colors.

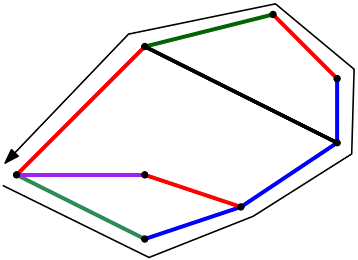


- Polynomial-time or NP-hard?

The Minimum Color Cycle Problem

MINIMUM COLOR CYCLE

Input: Edge-colored undirected graph.
Output: Cycle with the minimum number of different colors.



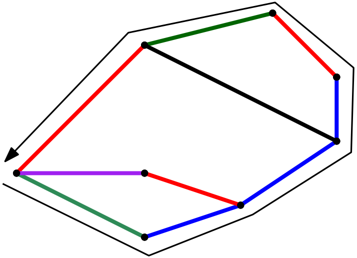
- Polynomial-time or NP-hard?
- If all edges have different colors, equivalent to shortest cycle

The Minimum Color Cycle Problem

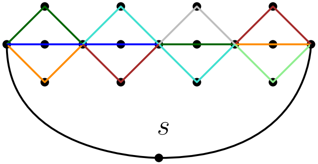
MINIMUM COLOR CYCLE

Input: Edge-colored undirected graph.

Output: Cycle with the minimum number of different colors.



- Polynomial-time or NP-hard?
- If all edges have different colors, equivalent to shortest cycle
- If we require the cycle to include a specified vertex *s*, then NP-hard [Broersma, Li, Woeginger, Zhang '05]

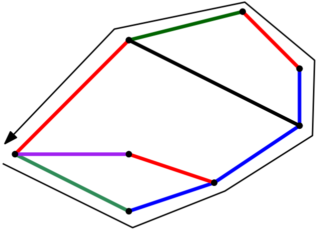


Minimum Color Cycle: Our results

Minimum Color Cycle

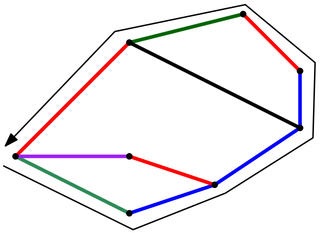
Input: Edge-colored undirected graph.

Output: Cycle with the minimum number of different colors.



Our results for minimum color cycle:

Minimum Color Cycle: Our results



MINIMUM COLOR CYCLE

Input: Edge-colored undirected graph.

Output: Cycle with the minimum number of different colors.

Our results for minimum color cycle:

Theorem

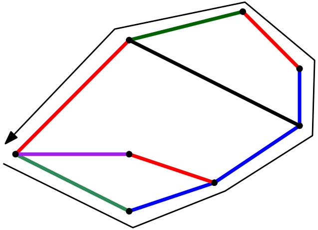
There is $n^{\mathcal{O}(\log \mathrm{OPT})}$ time algorithm for minimum color cycle.

Minimum Color Cycle: Our results

Minimum Color Cycle

Input: Edge-colored undirected graph.

Output: Cycle with the minimum number of different colors.



Our results for minimum color cycle:

Theorem

There is $n^{\mathcal{O}(\log \mathrm{OPT})}$ time algorithm for minimum color cycle.

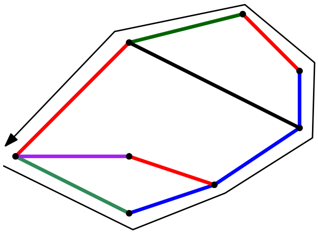
Quasipolynomial-time $\rightarrow$ probably not NP-hard!

Minimum Color Cycle: Our results

MINIMUM COLOR CYCLE

Input: Edge-colored undirected graph.

Output: Cycle with the minimum number of different colors.



Our results for minimum color cycle:

Theorem

There is $n^{\mathcal{O}(\log \mathrm{OPT})}$ time algorithm for minimum color cycle.

Quasipolynomial-time $\rightarrow$ probably not NP-hard!

Theorem

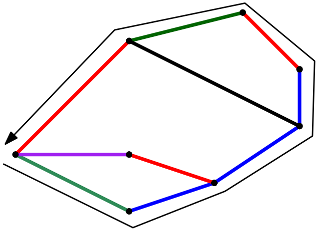
There is $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1+\varepsilon)$-approximation algorithm for minimum color cycle.

Minimum Color Cycle: Our results

Minimum Color Cycle

Input: Edge-colored undirected graph.

Output: Cycle with the minimum number of different colors.



Our results for minimum color cycle:

Theorem

There is $n^{\mathcal{O}(\log \mathrm{OPT})}$ time algorithm for minimum color cycle.

Quasipolynomial-time $\rightarrow$ probably not NP-hard!

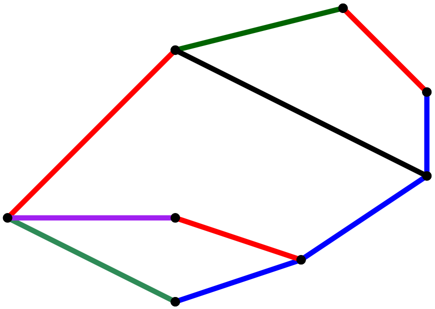
Theorem

There is $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1+\varepsilon)$-approximation algorithm for minimum color cycle.

Note: The approximation scheme implies the $n^{\mathcal{O}(\log \mathrm{OPT})}$ algorithm

Algorithm

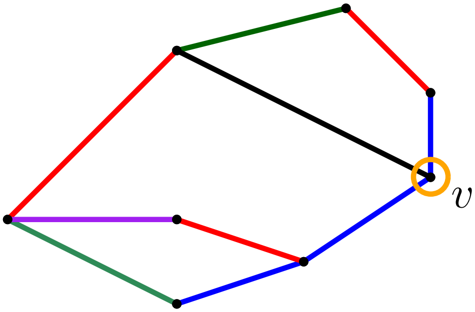
Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

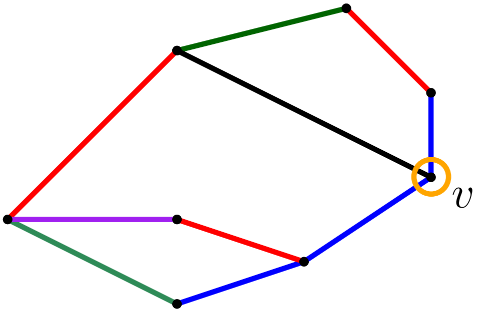
1. Guess a vertex v that is included in the cycle



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

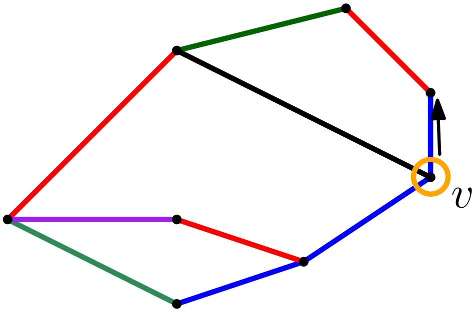
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

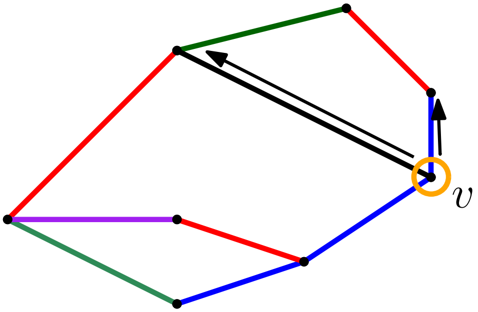
1. Guess a vertex *v* that is included in the cycle
2. Enumerate all paths with at most 1 color starting from *v*



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathsf{OPT})$ recursion levels

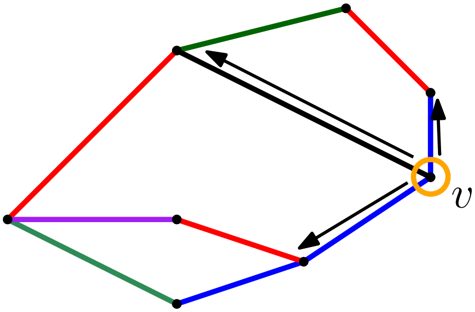
1. Guess a vertex *v* that is included in the cycle
2. Enumerate all paths with at most 1 color starting from *v*



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

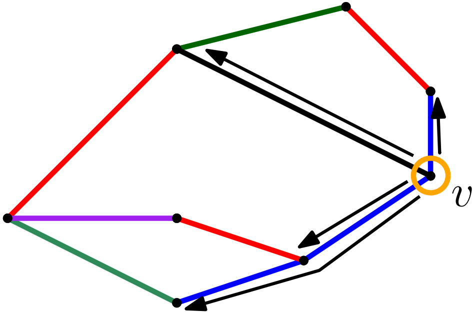
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

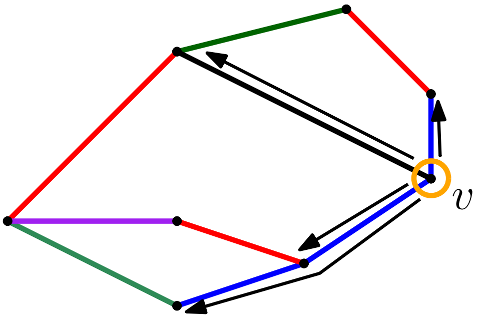
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

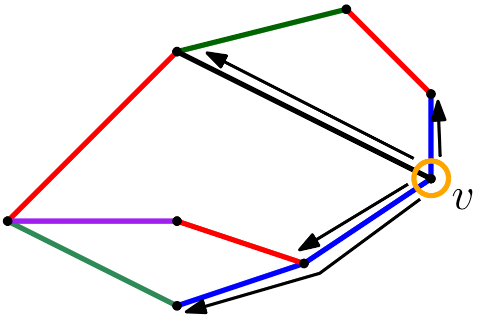
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

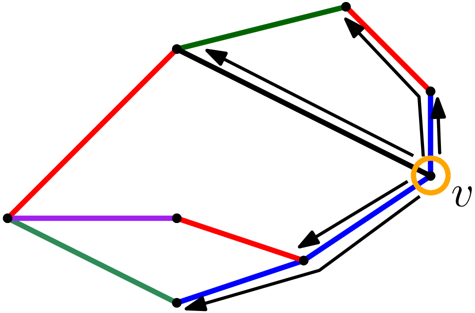
1. Guess a vertex *v* that is included in the cycle
2. Enumerate all paths with at most 1 color starting from *v*
3. Enumerate all paths with at most 2 colors . . . all paths with at most *k* colors



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

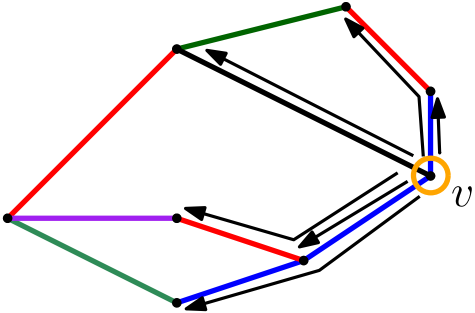
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors . . . all paths with at most k colors



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

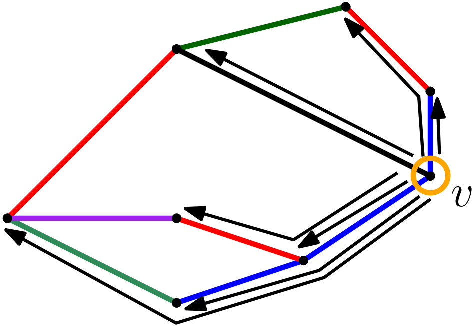
1. Guess a vertex *v* that is included in the cycle
2. Enumerate all paths with at most 1 color starting from *v*
3. Enumerate all paths with at most 2 colors . . . all paths with at most *k* colors



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

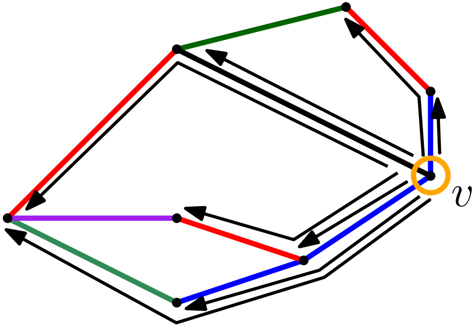
1. Guess a vertex *v* that is included in the cycle
2. Enumerate all paths with at most 1 color starting from *v*
3. Enumerate all paths with at most 2 colors . . . all paths with at most *k* colors



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

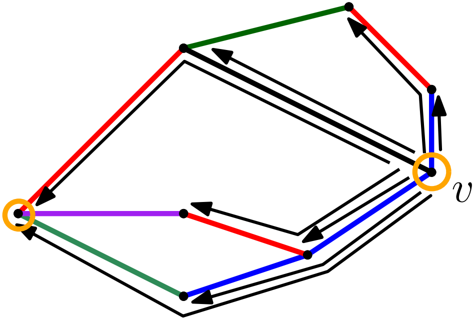
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors . . . all paths with at most k colors



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

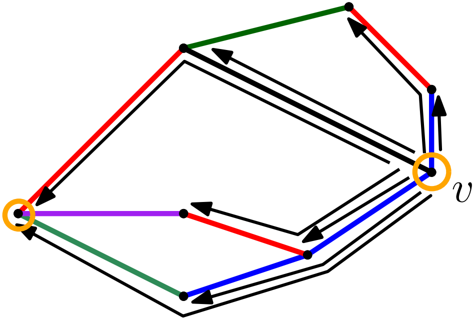
1. Guess a vertex *v* that is included in the cycle
2. Enumerate all paths with at most 1 color starting from *v*
3. Enumerate all paths with at most 2 colors . . . all paths with at most *k* colors
4. Until two different paths to a same vertex



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

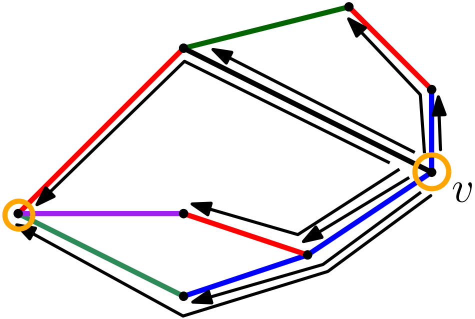
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors . . . all paths with at most k colors
4. Until two different paths to a same vertex
   - Now, $k \leq \mathrm{OPT} \leq 2k$



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

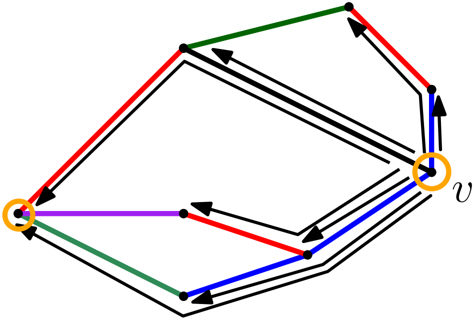
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors . . . all paths with at most k colors
4. Until two different paths to a same vertex
   - Now, $k \leq \mathrm{OPT} \leq 2k$
   - $\mathrm{poly}(n)$ sets of k colors that can be obtained by a path starting from v



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

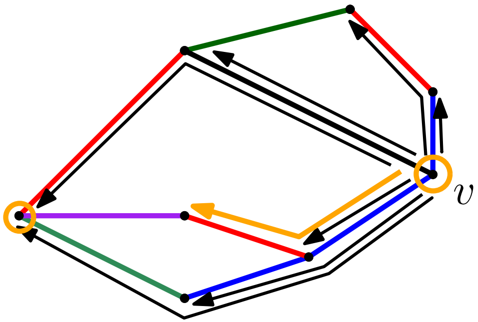
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors . . . all paths with at most k colors
4. Until two different paths to a same vertex
    - Now, $k \leq \mathrm{OPT} \leq 2k$
    - $\mathrm{poly}(n)$ sets of k colors that can be obtained by a path starting from v
5. Branch on which set of colors to select



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

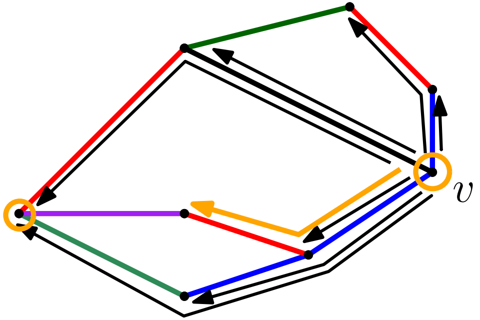
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors . . . all paths with at most k colors
4. Until two different paths to a same vertex
   - Now, $k \leq \mathrm{OPT} \leq 2k$
   - $\mathrm{poly}(n)$ sets of k colors that can be obtained by a path starting from v
5. Branch on which set of colors to select



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

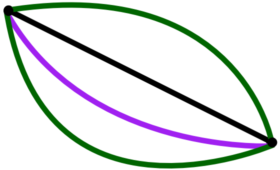
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors ... all paths with at most k colors
4. Until two different paths to a same vertex
   - Now, $k \leq \mathrm{OPT} \leq 2k$
   - $\mathrm{poly}(n)$ sets of k colors that can be obtained by a path starting from v
5. Branch on which set of colors to select
6. Contract the colors and recurse



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

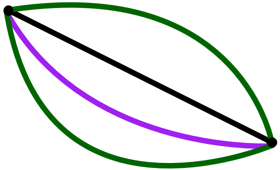
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors ... all paths with at most k colors
4. Until two different paths to a same vertex
   - Now, $k \leq \mathrm{OPT} \leq 2k$
   - $\mathrm{poly}(n)$ sets of k colors that can be obtained by a path starting from v
5. Branch on which set of colors to select
6. Contract the colors and recurse



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

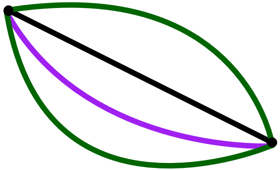
1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors ... all paths with at most k colors
4. Until two different paths to a same vertex
   - Now, $k \leq \mathrm{OPT} \leq 2k$
   - $\mathrm{poly}(n)$ sets of k colors that can be obtained by a path starting from v
5. Branch on which set of colors to select
6. Contract the colors and recurse
7. Each recursion level gets half of the remaining solution $\rightarrow \log_2 \mathrm{OPT}$ levels



Algorithm

Goal: Recursive algorithm with $\mathcal{O}(\log \mathrm{OPT})$ recursion levels

1. Guess a vertex v that is included in the cycle
2. Enumerate all paths with at most 1 color starting from v
3. Enumerate all paths with at most 2 colors . . . all paths with at most k colors
4. Until two different paths to a same vertex
   - Now, $k \leq \mathrm{OPT} \leq 2k$
   - $\mathrm{poly}(n)$ sets of k colors that can be obtained by a path starting from v
5. Branch on which set of colors to select
6. Contract the colors and recurse
7. Each recursion level gets half of the remaining solution $\rightarrow \log_2 \mathrm{OPT}$ levels
   - $n^{\mathcal{O}(\log \mathrm{OPT})}$ time



Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function

Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function
  - Submodular: for all $A, B \subseteq E(G)$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$

Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function
  - Submodular: for all $A, B \subseteq E(G)$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
  - Monotone: for all $A \subseteq B \subseteq E(G)$ we have $f(A) \leq f(B)$

Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function
  - Submodular: for all $A, B \subseteq E(G)$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
  - Monotone: for all $A \subseteq B \subseteq E(G)$ we have $f(A) \leq f(B)$

SHORTEST SUBMODULAR CYCLE

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cycle $C \subseteq E(G)$ of G that minimizes $f(C)$.

Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function
  - Submodular: for all $A, B \subseteq E(G)$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
  - Monotone: for all $A \subseteq B \subseteq E(G)$ we have $f(A) \leq f(B)$

SHORTEST SUBMODULAR CYCLE

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cycle $C \subseteq E(G)$ of G that minimizes $f(C)$.

Theorem

There is a $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1 + \varepsilon)$-approximation algorithm for shortest submodular cycle

Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function
  - Submodular: for all $A, B \subseteq E(G)$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
  - Monotone: for all $A \subseteq B \subseteq E(G)$ we have $f(A) \leq f(B)$

SHORTEST SUBMODULAR CYCLE

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cycle $C \subseteq E(G)$ of G that minimizes $f(C)$.

Theorem

There is a $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1 + \varepsilon)$-approximation algorithm for shortest submodular cycle

In contrast:

Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function
  - Submodular: for all $A, B \subseteq E(G)$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
  - Monotone: for all $A \subseteq B \subseteq E(G)$ we have $f(A) \leq f(B)$

Shortest submodular cycle

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cycle $C \subseteq E(G)$ of G that minimizes $f(C)$.

Theorem

There is a $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1+\varepsilon)$-approximation algorithm for shortest submodular cycle

In contrast:

- Exponential number of queries required to $\mathcal{O}(n^{2/3-\varepsilon})$-approximate minimum submodular cycle through specified vertex [Goel, Karande, Tripathi, Wang '09]

Generalization to monotone submodular functions

- Let $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$ be a monotone submodular function
  - Submodular: for all $A, B \subseteq E(G)$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$
  - Monotone: for all $A \subseteq B \subseteq E(G)$ we have $f(A) \leq f(B)$

SHORTEST SUBMODULAR CYCLE

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cycle $C \subseteq E(G)$ of G that minimizes $f(C)$.

Theorem

There is a $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1+\varepsilon)$-approximation algorithm for shortest submodular cycle

In contrast:

- Exponential number of queries required to $\mathcal{O}(n^{2/3-\varepsilon})$-approximate minimum submodular cycle through specified vertex [Goel, Karande, Tripathi, Wang '09]
- Similar lower bounds also for perfect matching, spanning tree, and (s,t)-cut [Goel, Karande, Tripathi, Wang '09; Jegelka and Bilmes '09]

Tightness

Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

Tightness

Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

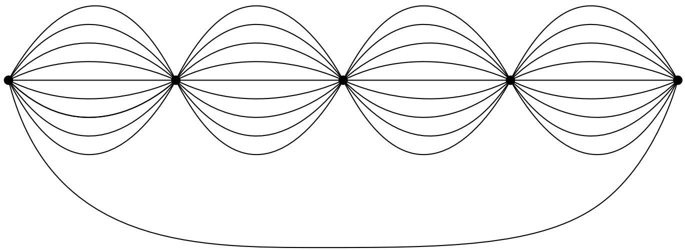
Tightness

Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:



Tightness

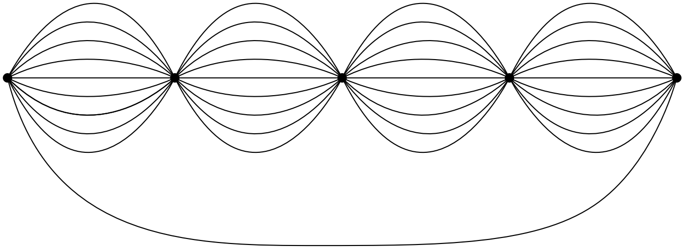
Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:

- Let $k = \lfloor \log_2(1/\varepsilon) \rfloor - 1$, and let the number of pumpkins be k (here $k = 4$)



Tightness

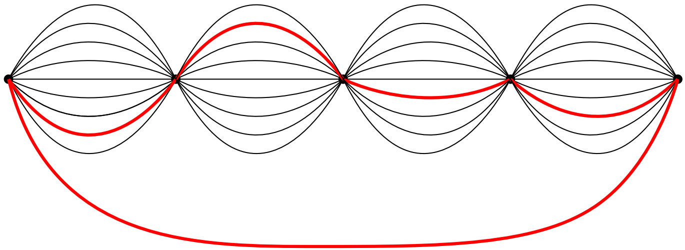
Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:

- Let $k = \lfloor \log_2(1/\varepsilon) \rfloor - 1$, and let the number of pumpkins be k (here $k = 4$)
- One long cycle of cost $2^k - 2 = 14$ and all other cycles cost $2^k - 1 = 15$



Tightness

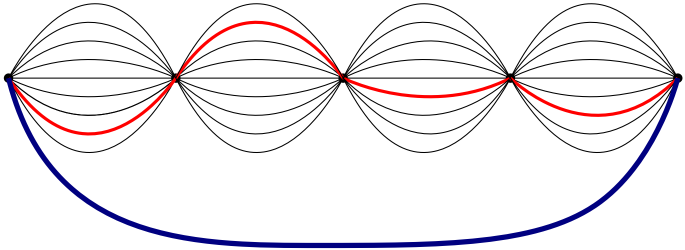
Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:

- Let $k = \lfloor \log_2(1/\varepsilon) \rfloor - 1$, and let the number of pumpkins be k (here $k = 4$)
- One long cycle of cost $2^k - 2 = 14$ and all other cycles cost $2^k - 1 = 15$



Tightness

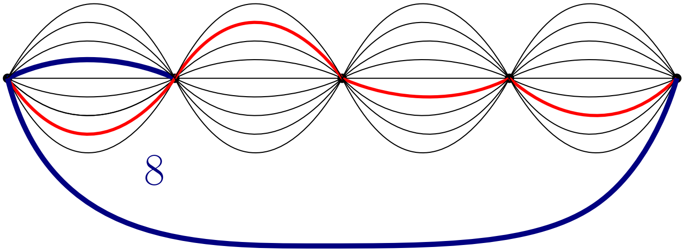
Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:

- Let $k = \lfloor \log_2(1/\varepsilon) \rfloor - 1$, and let the number of pumpkins be k (here $k = 4$)
- One long cycle of cost $2^k - 2 = 14$ and all other cycles cost $2^k - 1 = 15$
- Adding i:th edge increases value by 2^{k-i}



Tightness

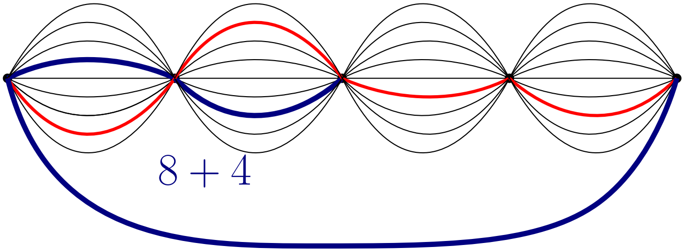
Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:

- Let $k = \lfloor \log_2(1/\varepsilon) \rfloor - 1$, and let the number of pumpkins be k (here $k = 4$)
- One long cycle of cost $2^k - 2 = 14$ and all other cycles cost $2^k - 1 = 15$
- Adding i:th edge increases value by 2^{k-i}



Tightness

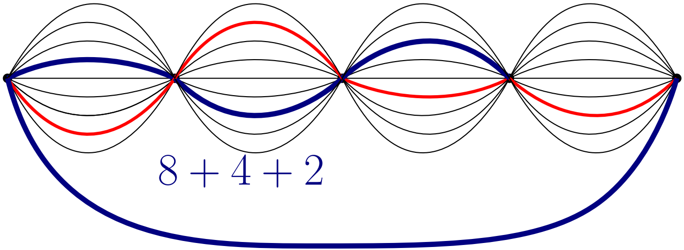
Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:

- Let $k = \lfloor \log_2(1/\varepsilon) \rfloor - 1$, and let the number of pumpkins be k (here $k = 4$)
- One long cycle of cost $2^k - 2 = 14$ and all other cycles cost $2^k - 1 = 15$
- Adding i:th edge increases value by 2^{k-i}



Tightness

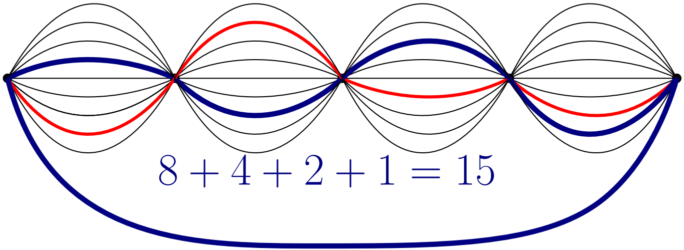
Theorem

For every ε, at least $n^{\log_2(1/\varepsilon)-\mathcal{O}(1)}$ queries are required to $(1+\varepsilon)$-approximate shortest submodular cycle.

(i.e., our algorithm is tight for every value of ε)

Proof idea:

- Let $k = \lfloor \log_2(1/\varepsilon) \rfloor - 1$, and let the number of pumpkins be k (here $k = 4$)
- One long cycle of cost $2^k - 2 = 14$ and all other cycles cost $2^k - 1 = 15$
- Adding i:th edge increases value by 2^{k-i}
- We learn if we made right choices only at the end



Connection to min-cut

EDGE-SUBMODULAR MIN-CUT

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cut $C \subseteq E(G)$ that minimizes $f(C)$.

Connection to min-cut

EDGE-SUBMODULAR MIN-CUT

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cut $C \subseteq E(G)$ that minimizes $f(C)$.

- Duality: Cuts in a planar graph $\Leftrightarrow$ cycles in its dual

Connection to min-cut

EDGE-SUBMODULAR MIN-CUT

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.
Output: Cut $C \subseteq E(G)$ that minimizes $f(C)$.

- Duality: Cuts in a planar graph $\Leftrightarrow$ cycles in its dual

Corollary

For planar graphs, there is a $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1 + \varepsilon)$-approximation algorithm for edge-submodular min-cut.

Connection to min-cut

Edge-submodular min-cut

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cut $C \subseteq E(G)$ that minimizes $f(C)$.

- Duality: Cuts in a planar graph $\Leftrightarrow$ cycles in its dual

Corollary

For planar graphs, there is a $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1+\varepsilon)$-approximation algorithm for edge-submodular min-cut.

- On planar graphs, this generalizes the $n^{\mathcal{O}(\log 1/\varepsilon)}$-time $1+\varepsilon$-approximation of [Ghaffari, Karger & Panigrahi; SODA'17] for colored min-cut

Connection to min-cut

EDGE-SUBMODULAR MIN-CUT

Input: Graph G and a monotone submodular function $f : 2^{E(G)} \to \mathbb{R}_{\geq 0}$.

Output: Cut $C \subseteq E(G)$ that minimizes $f(C)$.

- Duality: Cuts in a planar graph $\Leftrightarrow$ cycles in its dual

Corollary

For planar graphs, there is a $n^{\mathcal{O}(\log 1/\varepsilon)}$ time $(1 + \varepsilon)$-approximation algorithm for edge-submodular min-cut.

- On planar graphs, this generalizes the $n^{\mathcal{O}(\log 1/\varepsilon)}$-time $1 + \varepsilon$-approximation of [Ghaffari, Karger & Panigrahi; SODA'17] for colored min-cut
    - Which was proven to be optimal (for general graphs only!) by [Jaffke, Lima, Masarik, Pilipczuk & Souza; SODA'23]

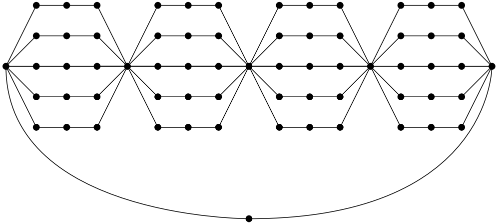
Open Problems

Open Problems

1. Is there a polynomial-time algorithm for minimum color cycle?

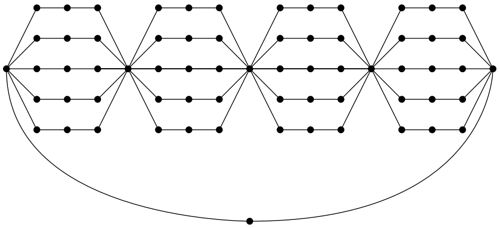
Open Problems

1. Is there a polynomial-time algorithm for minimum color cycle?
   - Open even for “pumpkin graphs”



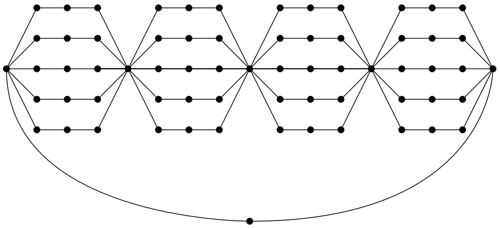
Open Problems

1. Is there a polynomial-time algorithm for minimum color cycle?
   - Open even for "pumpkin graphs"
2. Can the number of *minimal partial solutions* be superpolynomial?



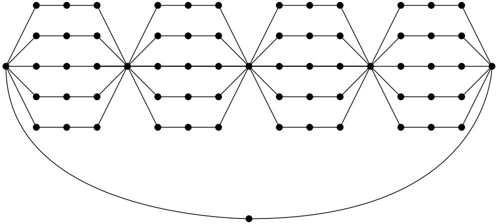
Open Problems

1. Is there a polynomial-time algorithm for minimum color cycle?
   - Open even for "pumpkin graphs"
2. Can the number of *minimal partial solutions* be superpolynomial?
3. Is there $f(\mathrm{OPT}) \cdot n^{\mathcal{O}(1)}$ time algorithm for minimum color cycle?



Open Problems

1. Is there a polynomial-time algorithm for minimum color cycle?
   - Open even for "pumpkin graphs"
2. Can the number of *minimal partial solutions* be superpolynomial?
3. Is there $f(\mathrm{OPT}) \cdot n^{\mathcal{O}(1)}$ time algorithm for minimum color cycle?



Thank you!