Shortest Cycles with Monotone Submodular Costs

FEDOR V. FOMIN, PETR A. GOLOVACH, and TUUKKA KORHONEN, Department of Informatics, University of Bergen, Norway
DANIEL LOKSHTANOV, Department of Computer Science, University of California, Santa Barbara, USA
GIANNOS STAMOULIS, LIRMM, University of Montpellier, CNRS, France

We introduce the following submodular generalization of the Shortest Cycle problem. For a nonnegative monotone submodular cost function $f$ defined on the edges (or the vertices) of an undirected graph $G$, we seek for a cycle $C$ in $G$ of minimum cost $\text{OPT} = f(C)$. We give an algorithm that given an $n$-vertex graph $G$, parameter $\varepsilon > 0$, and the function $f$ represented by an oracle, in time $n^{O(\log 1/\varepsilon)}$ finds a cycle $C$ in $G$ with $f(C) \leq (1 + \varepsilon) \cdot \text{OPT}$. This is in sharp contrast with the non-approximability of the closely related Monotone Submodular Shortest (s, t)-Path problem, which requires exponentially many queries to the oracle for finding an $n^{2/3 - \varepsilon}$-approximation Goel et al. [7], FOCS 2009. We complement our algorithm with a matching lower bound. We show that for every $\varepsilon > 0$, obtaining a $(1 + \varepsilon)$-approximation requires at least $n^{\Omega(\log 1/\varepsilon)}$ queries to the oracle.

When the function $f$ is integer-valued, our algorithm yields that a cycle of cost $\text{OPT}$ can be found in time $n^{O(\log \text{OPT})}$. In particular, for $\text{OPT} = n^{O(1)}$ this gives a quasipolynomial-time algorithm computing a cycle of minimum submodular cost. Interestingly, while a quasipolynomial-time algorithm often serves as a good indication that a polynomial time complexity could be achieved, we show a lower bound that $n^{O(\log n)}$ queries are required even when $\text{OPT} = O(n)$.

We also consider special cases of monotone submodular functions, corresponding to the number of different color classes needed to cover a cycle in an edge-colored multigraph $G$. For special cases of the corresponding minimization problem, we obtain fixed-parameter tractable algorithms and polynomial-time algorithms, when restricted to certain classes of inputs.

CCS Concepts: • Mathematics of computing → Graph algorithms;

Additional Key Words and Phrases: Submodular functions, shortest cycle, polynomial-time approximation schemes

The results of this paper appeared in the Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023).

The research leading to these results has received funding from the Research Council of Norway via the project BWCA (grant no. 314528). Giannis Stamoulis acknowledges support by the ANR project ESIGMA (ANR-17-CE23-0010) and the French-German Collaboration ANR/DFG Project UTMA (ANR-20-CE92-0027).

Authors’ addresses: F. V. Fomin, P. A. Golovach, and T. Korhonen, Department of Informatics, University of Bergen, PB 7803, Bergen, 5020, Norway; e-mails: fedor.fomin@uib.no, petr.golovach@uib.no, tuukka.korhonen@uib.no; D. Lokshtanov, Department of Computer Science, University of California, 2104 Harold Frank Hall, Santa Barbara, California, 93106-5110, USA; e-mail: daniello@ucsb.edu; G. Stamoulis, LIRMM, Univ Montpellier, CNRS, 161 rue Ada,34392 Montpellier cedex 5, France; e-mail: giannos.stamoulis@lirmm.fr.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1549-6325/2023/11-ART2 $15.00
https://doi.org/10.1145/3626824

ACM Transactions on Algorithms, Vol. 20, No. 1, Article 2. Publication date: November 2023.
1 INTRODUCTION

Submodular function minimization is a fundamental problem in combinatorial optimization. This problem is solvable in (strongly) polynomial time [2, 9, 10, 12, 16]. However, the problem becomes intractable even with straightforward additional cardinality constraints [8, 17]. A significant amount of research on submodular optimization is on generalizing the classical computer science problems by replacing simpler objective functions with general submodular functions. Examples of submodular minimizations over combinatorial constraints include load balancing, balanced cut [17], vertex cover [7, 11, 18], shortest path, perfect matching, spanning tree [7] or min-cut [15].

However, it seems that for almost every natural graph problem in P (shortest (s, t)-path, matching, spanning tree, or minimum (s, t)-cut) its submodular generalization becomes hard. Let \( f : 2^{E(G)} \to \mathbb{R}_{\geq 0} \) be a monotone submodular cost function defined by a value-giving oracle on the edges of an undirected graph \( G \) with \( m \) edges and \( n \) vertices. The following computational tasks require exponentially many queries to the value oracle:

- Finding an \( O(n^{2/3-\epsilon}) \)-approximation of the minimum cost of an \((s, t)\)-path (SUBMODULAR SHORTEST \((s, t)\)-PATH) [7];
- Finding an \( O(n^{1-\epsilon}) \)-approximation of the minimum cost of a perfect matching (SUBMODULAR PERFECT MATCHING) [7];
- Finding an \( O(n^{1/2-\epsilon}) \)-approximation of the minimum cost of a spanning tree (SUBMODULAR MINIMUM SPANNING TREE) [7];
- Finding an \( O(n^{1/3-\epsilon}) \)-approximation of the minimum cost of an \((s, t)\)-cut (SUBMODULAR MINIMUM \((s, t)\)-CUT) [15].

We discover an interesting anomaly, a classical problem in P, whose monotone submodular generalization strongly deviates from this common pattern. This is the problem of computing the girth, that is, the length of a shortest cycle, of an undirected graph. In sharp contrast to all these non-approximability results, we show that the problem of finding a cycle in a graph with minimum monotone submodular cost admits a \textbf{polynomial-time approximation scheme} (PTAS) and a quasipolynomial-time algorithm when the values of the submodular function are polynomially-bounded integers. More precisely, for a graph \( G \) and a function \( f : 2^{V(G)} \to \mathbb{R}_{\geq 0} \), we define \( \text{OPT} = \min\{f(C) : C \subseteq V(G) \text{ induces a cycle of } G\} \). Our first main result is the following theorem.

**Theorem 1.** There is an algorithm that given an \( n \)-vertex graph \( G \), parameter \( \epsilon > 0 \), and a monotone submodular function \( f : 2^{V(G)} \to \mathbb{R}_{\geq 0} \) represented by an oracle, finds a cycle \( C \) in \( G \) with \( f(C) \leq (1+\epsilon) \cdot \text{OPT} \) in time \( n^{O(\log 1/\epsilon)} \).

We stated Theorem 1 for a function \( f \) defined on the vertices of a graph. An easy reduction by placing a new vertex on every edge shows that the same result holds for monotone submodular functions defined on the edges of a multigraph, see Corollary 4.

When the function \( f \) is integer-valued, Theorem 1 (by setting \( \epsilon = \frac{1}{w+1} \) with \( \text{OPT} \leq w \leq 2\text{OPT} \), where \( w = f(C) \) for the cycle \( C \) returned by the approximation algorithm for \( \epsilon = 1 \)) implies that a cycle of cost OPT can be found in time \( n^{O(\log \text{OPT})} \). In particular, when \( \text{OPT} = n^{O(1)} \), it gives a quasipolynomial-time algorithm computing a cycle of minimum monotone submodular cost. For example, this holds when \( f \) is a rank function of a matroid.
Corollary 1. There is an algorithm that given an $n$-vertex graph $G$ and an integer monotone submodular function $f : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$ represented by an oracle, finds a cycle $C$ in $G$ with $f(C) = \text{OPT}$ in time $n^{O(\log \text{OPT})}$.

Our second main result is that the running times of the algorithms of Theorem 1 and Corollary 1 are asymptotically tight. Note that it is sufficient to prove Corollary 1 to be tight, as any improvement to Theorem 1 would also improve Corollary 1.

Theorem 2. There is no algorithm computing a cycle of cost at most $\text{OPT}$ on a given $n$-vertex graph and an integer monotone submodular function $f : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$ represented by an oracle, using at most $g(\text{OPT}) \cdot n^{o(\log \text{OPT})}$ queries to the oracle, for any computable function $g$.

Corollary 2. There is no algorithm computing a cycle of cost at most $(1 + \varepsilon) \cdot \text{OPT}$ on a given $n$-vertex graph and an integer monotone submodular function $f : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$ represented by an oracle, using at most $t(1/\varepsilon) \cdot n^{o(\log 1/\varepsilon)}$ queries to the oracle, for any computable function $t$.

In particular, Theorem 2 rules out fixed-parameter tractability (FPT) parameterized by OPT and Corollary 2 rules out efficient polynomial-time approximation schemes (EPTAS).

The same construction as in Theorem 2 also rules out the improvement of the quasipolynomial time in the setting where $\text{OPT} = O(n)$.

Theorem 3. There is no algorithm computing a cycle of cost at most $\text{OPT} = O(n)$ on a given $n$-vertex graph and an integer monotone submodular function $f : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$ represented by an oracle, using at most $n^{o(\log n)}$ queries to the oracle.

We note that on directed graphs the problem is much harder: The same construction as the one by Goel et al. [7] for undirected $(s, t)$-path shows that $O(n^{2/3 - \varepsilon})$-approximation for the minimum cost directed cycle requires an exponential number of queries to the oracle.

Theorem 1 also yields a PTAS for computing the submodular connectivity of a planar multigraph. The connectivity of a connected multigraph is the size of its minimum cut, that is, the minimum number of edges whose removal disconnects it. In Monotone Submodular Connectivity (also known as Monotone Submodular Min-Cut), for a connected multigraph $G$ with monotone submodular cost function $f$ on $E(G)$, the task is to identify the minimum cost $f(C)$ of a cut $C \subseteq E(G)$. In a connected planar multigraph $G$, an edge set of every simple cycle of $G$ is an edge set of an inclusion minimal edge cut in the dual of $G$, and vice versa. Thus by Theorem 1, we have the following corollary.

Corollary 3. There is an algorithm that given a planar $m$-edge multigraph $G$, parameter $\varepsilon > 0$, and a monotone submodular function $f : 2^{\mathcal{E}(G)} \to \mathbb{R}_{\geq 0}$ represented by an oracle, finds a cut $C$ in $G$ with $f(C) \leq (1 + \varepsilon) \cdot \text{OPT}$ in time $m^{O(\log 1/\varepsilon)}$ (where $\text{OPT}$ is the minimum cost of a cut).

The same lower bounds of Theorems 2 and 3 apply also to this setting (with $n$ replaced by $m$), showing that Corollary 3 is optimal, because the graph we use for the lower bound is planar (in particular, it is a dual of a planar multigraph). The best previously known upper bound on submodular connectivity on planar graphs is due to Jegelka and Bilmes [15] who gave an $O(\sqrt{n})$-approximation for this problem.

An interesting variant of submodular connectivity was considered by Ghaffari, Karger, and Panigrahi [6]. In the Hedge Connectivity problem, the edge set of a multigraph $G$ is partitioned into sets called hedges. The graph is $k$-hedge-connected if it is necessary to remove at least $k$ edge sets (hedges) in order to disconnect $G$. Ghaffari, Karger, and Panigrahi [6] gave a PTAS of running time $n^{O(\log 1/\varepsilon)}$ and a quasipolynomial-time exact algorithm for hedge connectivity. Very recently Jaffke et al. [14] (see also [13]) complemented this result by showing that the quasi-polynomial running time bounds also apply to the MONOTONE SUBMODULAR MIN-CUT problem.
Theorem 4. There is a $2^{O(k \log k)} N^{O(1)}$ time algorithm for Wide Family Hitting.

We then show that there is a polynomial-time algorithm if $|F_i|$ is bounded for every $i$. This corresponds to the case when the graph of the construction has bounded degree.
Theorem 5. Let $|\mathcal{F}_i| \leq d$ for every $i$. Then there is a $k^{O(\log d)}N^{O(1)}$ time randomized algorithm for Wide Family Hitting.

The rest of the paper is organized as follows. In Section 2 we give formal definitions and preliminary results. In Section 3 we give the algorithm of Theorem 1. In Section 4 we show the lower bounds Theorems 2 and 3. In Section 5 we prove Theorems 4 and 5. We then conclude in Section 6, in particular discussing open problems related to Hedge Minimum Cycle and Wide Family Hitting.

2 PRELIMINARIES

In this section, we introduce basic notation used throughout the paper.

We use standard graph-theoretic terminology and refer to the textbook of Diestel [3] for missing notions. We consider only finite graphs, and the considered graphs are assumed to be undirected if it is not explicitly said to be otherwise. For a graph $G$, we use $V(G)$ and $E(G)$ to denote its vertex and edge set, respectively. Throughout the paper we use $n = |V(G)| = |G|$ and $m = |E(G)|$. For a graph $G$ and a subset $X \subseteq V(G)$ of vertices, we write $G[X]$ to denote the subgraph of $G$ induced by $X$. For a vertex $v$, we denote by $N_G(v)$ the (open) neighborhood of $v$, i.e., the set of vertices that are adjacent to $v$ in $G$. For $X \subseteq V(G)$, $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$. The degree of a vertex $v$ is $d_G(v) = |N_G(v)|$. We may omit subscripts if the considered graph is clear from a context.

A path $P$ in $G$ is a subgraph of $G$ with $V(P) = \{v_0, \ldots, v_\ell\}$ and $E(P) = \{v_{i-1}v_i \mid 1 \leq i \leq \ell\}$. We write $v_0v_1 \cdots v_\ell$ to denote $P$; the vertices $v_0$ and $v_\ell$ are end-vertices of $P$, the vertices $v_1, \ldots, v_{\ell-1}$ are internal, and $\ell$ is the length of $P$. For a path $P$ with end-vertices $s$ and $t$, we say that $P$ is an $(s,t)$-path. A cycle is a graph $C$ with $V(C) = \{v_1, \ldots, v_\ell\}$ for $\ell \geq 3$ and $E(C) = \{v_{i-1}v_i \mid 1 \leq i \leq \ell\}$, where we assume that $v_0 = v_\ell$. We write $C = v_1 \cdots v_\ell$ to denote a cycle in $G$.

Definition 1. Given a finite set $U$, a function $f : 2^U \to \mathbb{R}$ is submodular if for every $X, Y \subseteq U$,

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y).$$

We also will use an equivalent formulation of submodularity, that is, for any $X \subseteq Y$ and $v \notin Y$,

$$f(X \cup \{v\}) - f(X) \geq f(Y \cup \{v\}) - f(Y).$$

Throughout the paper we assume that the considered submodular functions $f : 2^U \to \mathbb{R}$ are given by value-giving oracles returning the value $f(X)$ for every $X \subseteq U$ in unit time. We also assume the real RAM computational model for operations with the values of considered functions, i.e., we assume that basic arithmetic operations over real numbers are performed in unit time. In this paper, we consider functions $f$ defined on subsets of the vertex or edge set of a graph. Slightly abusing notation, we may write $f(H)$ instead of $f(V(H))$ or $f(E(H))$ for a subgraph $H$ of $G$.

A submodular function is monotone if for every $X \subseteq Y \subseteq U$, $f(X) \leq f(Y)$. We note that it is well-known that a rank function of a matroid is a monotone submodular function with nonnegative integer values.

3 PTAS FOR SHORTEST CYCLES WITH MONOTONE SUBMODULAR COSTS

In this section, we demonstrate a PTAS for finding a shortest cycle with nonnegative monotone submodular costs. If a connected component of a graph $G$ is a tree, it does not contain any cycle. In this case, the problem of finding a cycle in this component is meaningless. From now on, we assume that all connected components of graphs considered throughout the section contain cycles. For a graph $G$ and a function $f : 2^{V(G)} \to \mathbb{R}_{\geq 0}$, we define

$$\text{OPT}(G, f) = \min \{ f(C) \mid C \text{ is a cycle of } G \};$$

we write OPT instead of OPT($G, f$) if $G$ and $f$ are clear from the context.
First, we show that the problem admits a factor-2 approximation. Besides an approximate solution, our algorithm computes a family of induced tree-subgraphs rooted in the vertices of $G$ that will be crucial for PTAS.

We would like to stress that the next result is stated for subadditive functions, i.e., functions $f : 2^U \rightarrow \mathbb{R}$, for some finite set $U$, such that for every $X, Y \subseteq U$, $f(X) + f(Y) \geq f(X \cup Y)$. Notice that every nonnegative submodular function is also a subadditive function.

**Lemma 1.** There is an algorithm $\mathcal{A}$ that, given a graph $G$ and a monotone subadditive function $f : 2^V(G) \rightarrow \mathbb{R}_{\geq 0}$, in time $O(n(m + n \log n))$ finds a cycle $C$ with $f(C) \leq 2\text{OPT}$. Furthermore, the algorithm returns a family of induced tree-subgraphs $T_f = \{T_f(v)\}_{v \in V(G)}$ in $G$ such that for every $v \in V(G)$, (i) $v \in V(T_f(v))$ and (ii) for every $x \in V(T_f(v))$ and $y \in N_G(x) \setminus V(T_f(v))$, $f(P_y) \geq \text{OPT}/2$, where $P$ is the unique $(v, x)$-path in $T_f(v)$.

**Proof.** Our algorithm is based on the classical Dijkstra’s algorithm for finding shortest paths [4]. Let $v \in V(G)$. The algorithm constructs a tree rooted in $v$ by assigning labels $p(x)$ for vertices $x \in V(G)$, where $p(x)$ is the parent of $x$ in the tree; initially $p(v) = v$ and $p(x) = \text{empty}$ for every $x \in V(G)$ distinct from $v$. For $x \in V(G)$ with nonempty $p(x)$, we use $P_x$ to denote the unique $(v, x)$-path defined by these labels. We also assign labels $d(x)$ for $x \in V(G)$, where $d(x) = f(P_x)$ if $p(x)$ is nonempty. Then the following subroutine computes a cycle $C_v$ associated with $v$ and $T_f(v)$ defined by the set of vertices given together with their labels $p(x)$.

To analyze the algorithm, denote by $q(x) = \min\{f(P) : P$ is a $(v, x)$-path in $G\}$ for every $x \in V(G)$. Clearly, $q(x) \leq d(x)$ for $x \in V(G)$. For a real number $h \geq f(v)$, let $G_h$ be the subgraph of $G$ induced by the set of vertices $\{x \in V(G) : q(x) \leq h\}$. Let $h^*$ be the minimum number such that $G_{h^*}$ contains a cycle. Notice that such a number exists, because the connected component of $G$ containing $v$ is not a tree. Note also that for every $h < h^*$, $G_h$ is a tree. The crucial observation is that the algorithm assigns the labels $d(x) = q(x)$ for $x \in V(G_h)$ if $h < h^*$ and the labels $p(x)$
define the induced tree \( G_h \). Furthermore, the algorithm stops in line (8), where \( d(x) = q(x) = h^* \) and \( d(y) = q(y) \leq h^* \). Because \( xy \in E(G) \) and \( y \neq p(x) \), the graph \( G[V(P_x) \cup V(P_y)] \) contains a cycle \( C_y \). Because \( f(P_x) = q(x) \) and \( f(P_y) = q(y) \), we have that \( f(C_y) \leq 2h^* \). Since \( d(x) = h^* \), we have that \( T_f(v) \) constructed in line (10) is an induced tree in \( G \).

Clearly, \( v \in V(T_f(v)) \), and condition (i) for \( T_f(v) \) is fulfilled. By definition, \( f(C_v) \geq \) OPT. Hence, \( h^* \geq \) OPT/2. If there are \( x \in V(T_f(v)) \) and \( y \in N_G(v) \setminus V(T_f(v)) \) such that \( f(p_x y) < \) OPT/2, then \( q(y) < h^* \) and \( y \) should be in \( T_f(v) \). This implies that (ii) holds.

We run \( CYCLE(G, v, f) \) for all \( v \in V(G) \) and construct \( T_f = \{ T_f(v) \}_{v \in V(G)} \). To find \( C \), we consider the cycles \( C_v \) for \( v \in V(G) \) and select a cycle \( C \) of minimum cost. To show that \( f(C) \leq 2OPT \), consider \( v \in V(C) \). Then \( C \) contains a \((v, y)\)-path \( P = P_x y \), where \( x \in V(T_f(v)) \) and \( y \) is adjacent to \( x \). Then \( f(P) \geq h^* \) for \( h^* \) defined for this vertex \( v \). Because \( C \) is chosen as a cycle of minimum cost among the cycles \( C_v, v \in V(G) \), and \( f(C_v) \leq 2h^*, f(C) \leq 2OPT \).

To evaluate the running time, note that Dijkstra’s algorithm can be implemented to run in \( O(m + n \log n) \) time by the results of Fredman and Tarjan [5]. Using exactly the same approach, we conclude that for each \( v \in V(G) \), Cycle\((G, v, f)\) can be implemented to run in \( O(m + n \log n) \) time. Since the algorithm is called for every \( v \in V(G) \), the total running time is \( O(n(m + n \log n)) \). This concludes the proof.

Let \( T = \{ T(v) \}_{v \in V(G)} \) be a family of induced tree-subgraphs in a graph \( G \) such that \( v \in V(T(v)) \) for every \( v \in V(G) \). For \( v \in V(G) \), we define the family of paths

\[
\mathcal{P}(v) = \{ P_y : P \text{ is a } (v, x)\text{-path for } x \in V(T(v)) \text{ and } y \in N_G(x) \setminus V(T(v)) \},
\]

and set \( \mathcal{P}(T) = \bigcup_{v \in V(G)} \mathcal{P}(v) \). We use the following easy property of these paths.

**Lemma 2.** Let \( \mathcal{P}(T) \) be the family of paths constructed for \( T = \{ T(v) \}_{v \in V(G)} \). Then for every cycle \( C \), there is a path \( P \in \mathcal{P}(T) \) such that \( P \) is a segment of \( C \). Furthermore, \( |\mathcal{P}(T)| \leq nm \) and the sets of vertices of the paths of \( \mathcal{P}(T) \) can be listed in \( O(n^2m) \) time.

**Proof.** Let \( \mathcal{P}(T) = \bigcup_{v \in V(G)} \mathcal{P}(v) \), where \( \mathcal{P}(v) \) is defined as in (1). Consider a vertex \( v \in V(C) \). Because \( T(v) \) is an induced tree in \( G \), \( C \) contains a path \( P_y \), where \( P \) is a \((v, x)\)-path in \( T(v) \) and \( y \in N_G(x) \setminus V(T(v)) \). By definition, \( P_y \in \mathcal{P}(v) \). This proves that \( C \) contains as a segment a path from \( \mathcal{P}(T) \). Since every vertex \( y \in V(G) \setminus V(T(v)) \) has at most \( d_G(y) \) neighbors in \( T(v) \), the number of paths in \( \mathcal{P}(v) \) does not exceed \( m \). Hence, \( |\mathcal{P}(T)| \leq nm \). To list the set of vertices of the paths of \( \mathcal{P}(v) \), we consider every vertex \( y \in V(G) \setminus V(T(v)) \) and for each neighbor \( x \in T(v) \), we trace the unique \((x, v)\)-path with at most \( n \) vertices. Therefore, the sets of vertices of the paths of \( \mathcal{P}(T) \) can be listed in \( O(n^2m) \) time.

We are ready to prove Theorem 1, which we restate here.

**Theorem 1.** There is an algorithm that given an \( n \)-vertex graph \( G \), parameter \( \epsilon > 0 \), and a monotone submodular function \( f : 2^V(G) \rightarrow \mathbb{R}_{\geq 0} \) represented by an oracle, finds a cycle \( C \) in \( G \) with \( f(C) \leq (1 + \epsilon) \cdot \text{OPT} \) in time \( n^{O(\log 1/\epsilon)} \).

**Proof.** The rough idea is that we construct a recursive branching algorithm using Lemmas 1 and 2. In particular, the algorithm from Lemma 1 constructs a family of induced trees \( T_f \). Then by Lemma 2, a solution cycle \( C \) should contain some path \( P \in \mathcal{P}(T_f) \) as a segment. We branch on these paths. However, instead of looking for a cycle containing \( P \), we simply redefine the function by setting \( g(X) = f(X \cup V(P)) - f(P) \) for each \( X \subseteq V(G) \) using the property that for any cycle \( C \), \( f(C) \leq f(V(C) \cup V(P)) = g(C) + f(P) \) and \( f(C) = g(C) + f(P) \) if \( V(P) \not\subseteq V(C) \). Then we solve the problem recursively for the new function. Because \( f(P) \geq \text{OPT}/2 \) by Lemma 1, we require a logarithmic in \( 1/\epsilon \) depth of the search tree before we can apply a 2-approximation from Lemma 1 to obtain a factor-\((1 + \epsilon)\) approximation.
To describe the algorithm formally, we construct the subroutine \textsc{Find-Cycle}(G, g, k), which takes as its input G, a monotone submodular function \( g : 2^{V(G)} \to \mathbb{R}_{\geq 0} \), and an integer \( k \geq 0 \). The subroutine returns a cycle \( C \) of \( G \) with \( g(C) \leq (1 + \frac{1}{2^k}) \text{OPT}(G, g) \). Initially, \( g := f \). The parameter \( k \) defines the depth of recursion and is initially set to \( k := \lceil \log 1/\varepsilon \rceil \). To solve the problem for our original instance, we call \textsc{Find-Cycle}(G, f, \lceil \log 1/\varepsilon \rceil ). Recall that we use \( \mathcal{A} \) to denote the algorithm from Lemma 1.

\begin{algorithm}
\caption{\textsc{Find-Cycle}(G, g, k)}
\textbf{Input:} A graph \( G \), function \( g \), and \( k \geq 0 \).
\textbf{Result:} A cycle \( C \) of \( G \).
\begin{algorithmic}[1]
\State call \( \mathcal{A}(G, g) \) to obtain a cycle \( C \) and a family of subtrees \( T_g \);
\If {\( g(C) > 0 \) and \( k > 0 \)}
\State construct \( \mathcal{P} = \mathcal{P}(T_g) \);
\State \textbf{foreach} \( P \in \mathcal{P} \) \textbf{do}
\State \quad set \( g'(X) := g(V(P) \cup X) - g(P) \) for \( X \subseteq V(G) \);
\State \quad call \textsc{Find-Cycle}(G, g', k - 1) to find a cycle \( C' \);
\State \textbf{if} \( g(C') < g(C) \) \textbf{then}
\State \quad set \( C := C' \);
\EndIf
\State return \( C \)
\end{algorithmic}
\end{algorithm}

To show correctness, note that if \( g : 2^{V(G)} \to \mathbb{R}_{\geq 0} \) is a monotone submodular function, then each function \( g' \) introduced in line (6) is also a monotone submodular function with nonnegative values, that is, the input \textsc{Find-Cycle}(G, g', k - 1) in line (7) is feasible. Further, \textsc{Find-Cycle}(G, g, k) is finite, because the depth of the recursion is upper bounded by \( k \). Also the subroutine algorithm always returns some cycle of \( G \) because \( G \) is distinct from a forest by our assumption. Hence, to prove the correctness of \textsc{Find-Cycle}(G, g, k), we have to show that it returns a cycle \( C \) with \( g(C) \leq (1 + \frac{1}{2^k}) \text{OPT}(G, g) \). We show this by induction on \( k \).

If \( k = 0 \), then the algorithm returns the cycle \( C \) produced by \( \mathcal{A}(G, g) \) and, therefore, \( g(C) \leq 2 \text{OPT}(G, g) = (1 + \frac{1}{2}) \text{OPT}(G, g) \). Let \( k > 0 \) and assume that \textsc{Find-Cycle}(G, g', k - 1) called in line (7) outputs \( C' \) with \( g'(C') \leq (1 + \frac{1}{2^{k - 1}}) \text{OPT}(G, g') \).

If \( g(C) = 0 \) for the cycle \( C \) constructed by \( \mathcal{A}(G, g) \) in line (2), then the claim is trivial. Assume that \( g(C) > 0 \). Let \( C' \) be a cycle of \( G \) with \( g(C') = \text{OPT}(G, g) \). By Lemmas 1 and 2, there is \( P \in \mathcal{P}(T_g) \) such that \( P \) is a segment of \( C' \) and \( g(P) \geq \text{OPT}(G, g)/2 \). We consider \( P \) in the loop in lines (5)–(11). Then, for the function \( g' \) considered in line (6),

\[ g'(C') = g(C') - g(P) \leq g(C') - \text{OPT}(G, g)/2 = \text{OPT}(G, g)/2. \]

Therefore,

\[ \text{OPT}(G, g') \leq \text{OPT}(G, g) - g(P) \text{ and } \text{OPT}(G, g') \leq \text{OPT}(G, g)/2. \] (2)

Let \( C' \) be the cycle produced by \textsc{Find-Cycle}(G, g', k - 1) in line (7). By the inductive assumption

\[ g'(C') \leq (1 + \frac{1}{2^{k - 1}}) \text{OPT}(G, g'). \]
Then by the definition of $g$ and (2),
\[
g(C') \leq g(V(C') \cup V(P)) = g'(C') + g(P) \leq \left(1 + \frac{1}{2^k}\right) \text{OPT}(G, g') + g(P)
\]
\[
= (\text{OPT}(G, g') + g(P)) + \frac{1}{2^k} \text{OPT}(G, g')
\]
\[
\leq \text{OPT}(G, g) + \frac{1}{2^k} \text{OPT}(G, g) = \left(1 + \frac{1}{2^k}\right) \text{OPT}(G, g).
\]

By the choice of $C$ in lines (8)–(9), the algorithm outputs a cycle $C$ with $g(C) \leq g(C') \leq (1 + \frac{1}{2^k}) \text{OPT}(G, g)$. This concludes the correctness proof.

We call \textsc{Find-Cycle}(G, f, k), where $k = \lceil \log 1/\epsilon \rceil$, to solve the problem for the original instance. Because the algorithm outputs a cycle $C$ with $f(C) \leq (1 + \frac{1}{2^k}) \text{OPT}(G, f)$ and $k = \lceil \log 1/\epsilon \rceil$, $f(C) \leq (1 + \epsilon) \text{OPT}(G, f)$, that is, we obtain the desired approximation.

To evaluate the running time, note first that we switch to the function $g$ in line (6). We can make the following easy observation about such functions. Suppose that $f_1, f_2, f_3 : 2^V(G) \rightarrow \mathbb{R}_{\geq 0}$ are functions such that for every $X \subseteq V(G)$, $f_2(X) = f_1(X \cup A) - f_1(A)$ and $f_3(X) = f_2(X \cup B) - f_2(B)$ for some $A, B \subseteq V(G)$. Then
\[
f_3(X) = f_2(X \cup B) - f_2(B) = (f_1(X \cup B \cup A) - f_1(A)) - (f_1(A \cup B) - f_1(A))
\]
\[
= f_1(X \cup (B \cup A)) - f_1(A \cup B).
\]
Using this observation iteratively, using only the oracle for the input function $f$, the values of all other functions occurring in the algorithm could be computed in $O(n)$ time for each $X \subseteq V(G)$.

Computing $C$ and $T_v$ in line (2) can be done in $O(n^2(m + n \log n))$ time by Lemma 1 taking into account that each value $g(X)$ can be computed in $O(n)$ time. The construction of $P(T_v)$ can be done in $O(n^m)$ time by Lemma 2. The number of paths $P$ considered in the loop in lines (5)–(11) is at most $nm$ by Lemma 2. Therefore, the number of recursive calls of \textsc{Find-Cycle}(G, g', k - 1) in line (7) is at most $nm \leq n^3$. The depth of the search tree is at most $\lceil \log 1/\epsilon \rceil$. Therefore, the total running time is $n^{O(\log \log \text{OPT})}$. This concludes the proof.

When the function $f$ is integer-valued, Theorem 1 (by setting $\epsilon = \frac{1}{w+1}$ with $\text{OPT} \leq w \leq 2\text{OPT}$, where $w = f(C)$ for the cycle $C$ returned by the approximation algorithm for $\epsilon = 1$) implies that a cycle of cost $\text{OPT}$ can be found in time $n^{O(\log \text{OPT})}$. In particular, when $\text{OPT} = n^{O(1)}$, we obtain a quasi-polynomial algorithm computing the cycle of minimum submodular cost. For example, this holds if $f$ is a rank function of a matroid.

**Corollary 1.** There is an algorithm that given an $n$-vertex graph $G$ and an integer monotone submodular function $f : 2^V(G) \rightarrow \mathbb{Z}_{\geq 0}$ represented by an oracle, finds a cycle $C$ in $G$ with $f(C) = \text{OPT}$ in time $n^{O(\log \text{OPT})}$.

Finally in this section, we observe that our results can be easily translated for the edge version of the problem, even on multigraphs. For monotone submodular function $f : 2^E(G) \rightarrow \mathbb{R}_{\geq 0}$, we define $\text{OPT} = \min \{f(C) : C \subseteq E(G) \text{ is a cycle of } G\}$ in the same way as for the vertex costs.

**Corollary 4.** Let $G$ be an $m$-edge multigraph, $\epsilon > 0$, and $f : 2^E(G) \rightarrow \mathbb{R}_{\geq 0}$ a monotone submodular function represented by an oracle. Then a cycle $C$ in $G$ with $f(C) \leq (1 + \epsilon) \cdot \text{OPT}$ can be found in time $m^{O(\log 1/\epsilon)}$.

**Proof.** We construct a graph $G'$ by subdividing each edge of $G$ once, that is, for each edge $xy \in E(G)$, we introduce a new vertex $v_{xy}$, make $v_{xy}$ adjacent to $x$ and $y$, and delete $xy$. For a subdivision vertex $v_{xy}$, define $e(v_{xy}) = xy$. Let $W$ be the set of subdivision vertices. We define $g : 2^V(G') \rightarrow \mathbb{R}_{\geq 0}$ by setting $g(X) = f((e(v) : v \in W \cap X))$ for each $X \subseteq V(G')$. The definition implies
that \( g : 2^{V(G')} \rightarrow \mathbb{R}_{\geq 0} \) is a monotone submodular function and that an oracle for \( f \) can be translated into an oracle for \( g \). There is one-to-one correspondence between cycles of \( G \) and \( G' \), because each cycle \( C' \) is obtained from a cycle \( C \) of \( G \) by subdividing edges and \( f(C) = g(C') \). Therefore, we can apply Theorem 1 for \( G' \) and \( g \).

\[ \Box \]

4 LOWER BOUND

In this section we prove the lower bounds of Theorems 2 and 3. Both of these lower bounds will follow from the same construction, although with different parameters.

We give the lower bounds for the setting where the function \( f \) is defined on the edges of a multigraph, which then by Corollary 4 translates into a lower bound when the function is defined on vertices of a graph. In our construction the function \( f \) is integer-valued.

Our lower bound is based on the following construction. For positive integers \( k \) and \( p \) we define a multigraph \( G(k, p) \) with \( k + 1 \) vertices and \( pk + 1 \) edges (see Figure 1) and a monotone submodular function \( f : 2^{E(G(k, p))} \rightarrow \mathbb{N} \) so that \( \text{OPT}(G(k, p), f) = 2^{k+1} - 1 \). Then, for each cycle \( C \) of length \( k + 1 \) of \( G(k, p) \) we define a monotone submodular function \( f_C : 2^{E(G(k, p))} \rightarrow \mathbb{N} \) so that \( \text{OPT}(G(k, p), f_C) = 2^{k+1} - 2 \) and \( f_C \) differs from \( f \) only on the cycle \( C \). Deciding whether an oracle represents the function \( f \) or one of the functions \( f_C \) will then require querying each cycle \( C \) of length \( k + 1 \) and there are \( p^k \) such cycles in \( G(k, p) \).

**Construction of \( G(k, p) \).** The multigraph \( G(k, p) \) has vertex set \( \{v_1, v_2, \ldots, v_{k+1}\} \). For every pair of consecutive vertices \( v_i, v_{i+1}, 1 \leq i \leq k \), there are \( p \) parallel edges \( F_i = \{e_{i1}^p, \ldots, e_{ip}^p\} \) with endpoints \( v_i \) and \( v_{i+1} \). One more edge \( e_{k+1} \) connects \( v_1 \) and \( v_{k+1} \), see Figure 1. In total, \( G(k, p) \) has \( k + 1 \) vertices and \( m = pk + 1 \) edges. The multigraph \( G(k, p) \) contains \( p^k \) cycles of length \( k + 1 \). Each such cycle passes through all the vertices of the multigraph in the order \( v_1, v_2, \ldots, v_{k+1}, v_1 \).

**Constructions of \( f \) and \( f_C \).** We define the following function \( f \) on the subsets \( X \) of \( E(G(k, p)) \). First, if \( X \subseteq E(G(k, p)) \) contains a cycle, i.e., \( |X| \geq k + 1 \) or there is \( i \) so that \( |X \cap F_i| \geq 2 \), we define

\[ f(X) = 2^{k+1} - 1. \]

Otherwise, i.e., \( X \subseteq E(G(k, p)) \) does not contain a cycle, we define

\[ f(X) = 2^{k+1} - 2^{k+1-|X|}, \]

and by definition we have that \( \text{OPT}(G(k, p), f) = 2^{k+1} - 1 \).

For a cycle \( C \subseteq E(G(k, p)) \) of length \( |C| = k + 1 \), the function \( f_C \) is defined as \( f_C(X) = f(X) \) for \( X \neq C \), and \( f_C(C) = 2^{k+1} - 2 \). Clearly \( \text{OPT}(G(k, p), f_C) = 2^{k+1} - 2 \), and this optimum is given uniquely by the cycle \( C \).

It is clear from the definitions that the functions \( f \) and \( f_C \) are monotone. Next we establish the submodularities of \( f \) and \( f_C \). Note that it is sufficient to prove that \( f_C \) is submodular, as then the submodularity of \( f \) follows by writing \( f \) as a restriction of \( f_C \) on \( G(k, p + 1) \).

**Lemma 3.** The function \( f_C \) is submodular.

**Proof.** To prove the submodularity of \( f_C \), we show that for every two sets \( X \subseteq Y \subseteq E(G(k, p)) \) and \( e \in E(G(k, p)) \setminus Y \),

\[ f_C(X \cup \{e\}) - f_C(X) \geq f_C(Y \cup \{e\}) - f_C(Y). \]

(3)

Depending on \( X, Y \), and \( e \), we consider different cases.

**Case 1:** \( f_C(X \cup \{e\}) = 2^{k+1} - 1 \). Then also \( f_C(Y \cup \{e\}) = 2^{k+1} - 1 \), and as \( f_C(X) \leq f_C(Y) \) by monotonicity, (3) follows.
Case 2: $f_C(Y) = 2^{k+1} - 1$. In this case $f_C(Y) = f_C(Y \cup \{e\})$, and (3) follows by the monotonicity of $f_C$.

Case 3: $f_C(Y \cup \{e\}) = 2^{k+1} - 1$. If either $f_C(X \cup \{e\}) = 2^{k+1} - 1$ or $f_C(Y) = 2^{k+1} - 1$, then we are done by the previous cases. Otherwise, $|X \cup \{e\}| \leq |Y| \leq k + 1$, and we consider two subcases.

Subcase 3a: $X \cup \{e\} = C$. Then $|X| = k$ and hence $|Y| \geq k + 1$. In this case because $Y \neq X \cup \{e\}$, $f_C(Y) = f_C(Y \cup \{e\}) = 2^{k+1} - 1$, while $f_C(X) \leq f_C(X \cup \{e\})$ by the monotonicity of $f_C$.

Subcase 3b: $X \cup \{e\} \neq C$. In this case, $X \cup \{e\}$ does not contain a cycle, and therefore we have that $f_C(X \cup \{e\}) - f_C(X) = 2^{k+1} - 2^{k+1-|X|-1} - (2^{k+1} - 2^{k+1-|X|}) = 2^k - |X|$ and $|X| \leq k - 1$. Then, if $Y = C$, we have that $f_C(Y \cup \{e\}) - f_C(Y) = 1 \leq 2^{k-|X|}$. If $Y \neq C$, then $Y$ does not contain a cycle and we have that $f_C(Y \cup \{e\}) - f_C(Y) = 2^{k+1} - 1 - (2^{k+1} - 2^{k+1-|Y|}) < 2^{k+1-|Y|} \leq 2^{k-|X|}$. (For the last inequality we use $|X| < |Y|$.)

Case 4: None of the previous cases holds. In this case $X \cup \{e\}$ does not contain a cycle, so we have that $f_C(X \cup \{e\}) - f_C(X) = 2^{k-|X|}$. If $Y \cup \{e\} = C$, then $f_C(Y \cup \{e\}) - f_C(Y) = 2^{k+1} - 2 - (2^{k+1} - 2^{k+1-k}) = 0$. If $Y \cup \{e\} \neq C$, then $f_C(Y \cup \{e\}) - f_C(Y) = 2^{k-|Y|} \leq 2^{k-|X|}$. \qed

Now each of the functions $f_C$ and the function $f$ could be represented by the oracle, and the optimum depends on whether the function represented by the oracle is $f$ or one of the functions $f_C$. Therefore, it remains to argue that we cannot distinguish between $f$ or one of the functions $f_C$ in less than $p^k$ queries.

**Lemma 4.** Let $g : 2^{E(G(k,p))} \to \mathbb{N}$ be a function represented by an oracle, with a promise that either $g = f$ or $g = f_C$ for some cycle $C$ of $G(k, p)$ of length $|C| = k + 1$. It requires at least $p^k$ queries to the oracle to determine if $g = f$.

**Proof.** Suppose the oracle answers the queries always according to the function $f$, and an algorithm terminates after asking less than $p^k$ queries. Because $G(k, p)$ has $p^k$ cycles of length $k + 1$, there exists some cycle $C$ so that the algorithm has not queried $C$, and therefore as $f$ and $f_C$ are equivalent on all inputs except $C$, all the answers are consistent with both $f$ and $f_C$. Therefore the algorithm cannot decide correctly whether $g = f$ or $g = f_C$. \qed

Next we summarize the lower bound that follows from the constructions of the multigraph $G(k, p)$, the functions $f$, and $f_C$, and Lemma 4.

**Lemma 5.** For any positive integers $p, k$, there exists a graph $G$ with $kp + k + 2$ vertices and an integer submodular function $f : 2^{V(G)} \to \mathbb{N}$ represented by an oracle so that deciding whether $\text{OPT}(G, f) = 2^{k+1} - 2$ or $\text{OPT}(G, f) = 2^{k+1} - 1$ requires at least $p^k$ queries to the oracle.

**Proof.** We take the multigraph $G(k, p)$ with $m = kp + 1$ edges and let $f : 2^{E(G(k,p))} \to \mathbb{N}$ be a function represented by an oracle. By Lemma 4, deciding whether $\text{OPT}(G(k,p), f) = 2^{k+1} - 2$ or $\text{OPT}(G(k,p), f) = 2^{k+1} - 1$ requires $p^k$ queries to the oracle in the worst case. This construction is for a multigraph where the function is on the edges, but by the argument of Corollary 4 the lower bound also holds for graphs with $kp + k + 2$ vertices where the function is on the vertices. \qed

By making use of Lemma 5, we establish Theorem 2 with the lower bound matching the algorithmic bound of Theorem 1. We restate the theorem here.

**Theorem 2.** There is no algorithm computing a cycle of cost at most $\text{OPT}$ on a given $n$-vertex graph and an integer monotone submodular function $f : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$ represented by an oracle, using at most $g(\text{OPT}) \cdot n^{o(\log \text{OPT})}$ queries to the oracle, for any computable function $g$.\[ACM Transactions on Algorithms, Vol. 20, No. 1, Article 2. Publication date: November 2023.\]
Proof. We assume without loss of generality that \( g \) is non-decreasing and \( g(x) \geq x \) for every \( x \in \mathbb{R}_{\geq 0} \). Assume that there is an algorithm that makes at most \( t(OPT, n) = g(OPT) \cdot n^{\log OPT} \) queries. Now, there exist some large enough \( N \) and \( k' \) so that \( t(OPT, n) < g(OPT) \cdot n^{\log_2 OPT}/16 \) for all \( OPT \geq k' \) and \( n \geq N \). We apply Lemma 5 with \( p = g(4k') \cdot N \) and \( k = \lceil \log_2 k' \rceil \). Let \( n = kp + k = kN \cdot g(k') + \lceil \log_2 k' \rceil + 2 \). Because \( g \) is non-decreasing and \( g(x) \geq x \) for every \( x \in \mathbb{R}_{\geq 0} \), we have that \( \lceil \log_2 k' \rceil \leq k' \leq g(4k') \) and it holds that \( N \leq n \leq kN \cdot g(k') \leq (g(4k') \cdot N)^2 \). We get a graph with \( n \) vertices, where \( N \leq n \leq (g(4k') \cdot N)^2 \), and optimum OPT with \( k' \leq OPT \leq 4k' \) in which the problem requires at least
\[
(g(4k') \cdot N)^{\lceil \log_2 k' \rceil} \geq g(4k') \cdot (g(4k') \cdot N)^{\lceil \log_2 k' \rceil - 1} \geq g(OPT) \cdot n^{\log_2 OPT}/16
\]
queries to solve. This contradicts the existence of such an algorithm.

We then establish Theorem 3.

**Theorem 3.** There is no algorithm computing a cycle of cost at most \( OPT = O(n) \) on a given \( n \)-vertex graph and an integer monotone submodular function \( f : 2^V(G) \rightarrow \mathbb{Z}_{\geq 0} \) represented by an oracle, using at most \( n^{o(\log n)} \) queries to the oracle.

**Proof.** Assume there is an algorithm that makes at most \( t(n) = n^{o(\log n)} \) queries. Now, there exists a large enough \( N \) so that \( t(n) < n^{\log n}/4 \) for all \( n \geq N \). However, applying Lemma 5 with \( p = N \) and \( k = \lceil \log_2 N \rceil \) gives a graph with \( n \) vertices, where \( N \leq n \leq N^2 \), and optimum OPT \( \leq 4n \), in which the problem requires at least \( N^{\lceil \log_2 N \rceil} \geq n^{\log_2 n}/4 \) queries to solve. This contradicts the existence of such an algorithm. \( \square \)

## 5 The Wide Family Hitting Problem

Motivated by the question whether HEDGE MINIMUM CYCLE admits a polynomial-time algorithm, and the fact that our algorithm for monotone submodular functions is optimal already on a very restricted class of graphs considered in Section 4, we study the complexity of HEDGE MINIMUM CYCLE on the subdivisions of \( G(k, p) \) (see Figure 1). In this class, HEDGE MINIMUM CYCLE is equivalent to a problem which we call WIDE FAMILY HITTING.

For an integer \( k \) and a universe \( U \), we say that a family \( F \) of sets is \( k \)-wide if for any two distinct sets \( A, B \in F \) it holds that \( |A \cup B| > k \). In the WIDE FAMILY HITTING problem, the input consists of an integer \( k \), a universe \( U \), and \( m \) \( k \)-wide families \( F_1, \ldots, F_m \) over the universe \( U \). The task is to decide if it is possible to select one set from each family, i.e., sets \( S_1 \in F_1, S_2 \in F_2, \ldots, S_m \in F_m \) so that \( \bigcup_{i=1}^m S_i \leq k \). We denote the size of the input by \( N = \sum_{i=1}^m \sum_{A \in F_i} |A| \).

To see the relations between HEDGE MINIMUM CYCLE and WIDE FAMILY HITTING, we first show reduction from WIDE FAMILY HITTING to HEDGE MINIMUM CYCLE. Consider \( m \) \( k \)-wide families \( F_1, \ldots, F_m \) over the universe \( U \). We construct the vertex-colored graph \( G \), where the vertices are colored by the elements of \( U \), as follows:

- construct \( m + 1 \) vertices \( v_0, \ldots, v_m \) and color them by a special color \( c \not\in U \);
- for each \( i \in \{1, \ldots, m\} \), construct \( |F_i| \) \((v_{i-1}, v_i)\)-paths such that for every \( S \in F_i \), we have a path with \( |S| \) internal vertices colored by the elements of \( S \subseteq U \);
- make \( v_0 \) and \( v_m \) adjacent.

Let also \( k' = k + 1 \). It can be seen that \( G \) has a cycle \( C \), whose vertices are colored by at most \( k' \) colors, if and only if there are \( S_i \in F_i \) for \( i \in \{1, \ldots, m\} \) such that \( \bigcup_{i=1}^m S_i \leq k \). To prove this, notice that because \( F_1, \ldots, F_m \) are \( k \)-wide, any cycle \( C \) in \( G \) containing vertices of at most \( k' \) colors should contain \((v_{i-1}, v_i)\)-paths for each \( i \in \{1, \ldots, m\} \). For every \( i \in \{1, \ldots, m\} \), let \( S_i \in F_i \) be the set of colors of the internal vertices of the \((v_{i-1}, v_i)\)-path in \( C \). If \( C \) contains vertices of at
most $k' = k + 1$ colors, then $| \bigcup_{i=1}^{m} S_i | \leq k$. For the opposite direction, let $S_i \in \mathcal{F}_i$ for $i \in \{1, \ldots, m\}$ be such that $| \bigcup_{i=1}^{m} S_i | \leq k$. Then we construct the cycle $C$ in $G$ by concatenating the $(v_{i-1}, v_i)$-paths whose internal vertices are colored by the elements of $S_i$ and completing the cycle by the addition of the edge $v_0v_m$. Clearly, the vertices of $C$ are colored by at most $k' = k + 1$ colors.

To reduce HEDGE MINIMUM CYCLE TO WIDE FAMILY HITTING on subdivisions of the graphs $G$ illustrated on Figure 1, assume that $G$ is of the following form:

- $G$ has $m + 1$ vertices $v_0, \ldots, v_m$ and $v_0$ is adjacent to $v_m$,
- for each $i \in \{1, \ldots, m\}$, $G$ has a family of internally vertex-disjoint paths $\mathcal{P}_i$ between $v_{i-1}$ and $v_i$ such that each path has at least one internal vertex.

Suppose also that $c : V(G) \rightarrow U$ is a coloring function that colors the vertices of $G$ by colors from a set $U$. For each $i \in \{1, \ldots, m\}$, we define $\mathcal{F}_i = \{c(P) : P \in \mathcal{P}_i\}$, that is, $\mathcal{F}_i$ is the family of the sets of colors of the paths from $\mathcal{P}_i$. If $C$ is a cycle of $G$, then either (i) $C$ is formed by two paths $P, Q \in \mathcal{P}_i$ for some $i \in \{1, \ldots, m\}$ or (ii) $C$ contains the concatenation of $m$ paths $P_i \in \mathcal{P}_i$ for $i \in \{1, \ldots, m\}$. If we are looking for a cycle $C$ containing at most $k$ colors, we can use brute force to check whether there is such a cycle of type (i), because the number of such cycles is quadratic in the size of $G$. Suppose that this is not the case and we have (ii). Then each family $\mathcal{F}_i$ is $k$-wide, and a cycle containing vertices of at most $k$ colors exists if and only if there are $S_i \in \mathcal{F}_i$ for $i \in \{1, \ldots, m\}$ such that $| \bigcup_{i=1}^{m} S_i | \leq k$.

We first show that in contrast to the lower bound from Theorem 3, the WIDE FAMILY HITTING problem is fixed-parameter-tractable when parameterized by $k$. We use the following lemma for it.

**Lemma 6.** Let $X \subseteq U$ be a set and $\mathcal{F}$ a $k$-wide family of sets over $U$. There are at most $2^{|X|}$ sets $A \in \mathcal{F}$ with $|A \cup X| \leq k$ and $|A| \leq |X|$.

**Proof.** Suppose there are sets $A, B \in \mathcal{F}$ with $A \cap X = B \cap X$, $|A \cup X| \leq k$, $|B \cup X| \leq k$, $|A| \leq |X|$, and $|B| \leq |X|$. Then we have that

$$|A \cup B| = |A| + |B| - |A \cap B| \leq |A| + |B| - |A \cap X| \leq |A| + |X| - |A \cap X| = |A \cup X| \leq k,$$

which would contradict the fact that $\mathcal{F}$ is $k$-wide. Therefore, all sets $A \in \mathcal{F}$ with $|A \cup X| \leq k$ and $|A| \leq |X|$ have a different intersection with $X$, implying that there are at most $2^{|X|}$ of them. □

We will also use the following lemma in both of the algorithms of this section.

**Lemma 7.** Let $X \subseteq U$ be a set with $|X| \leq k$ and $\mathcal{F}$ a $k$-wide family of sets over $U$. For any two sets $A, B \in \mathcal{F}$ it holds that $|X \cup A| - |X| + |X \cup B| - |X| > k - |X|$.

**Proof.** Note that $|X \cup A| + |X \cup B| - |X| \geq |A \cup B| > k$. □

Now we give our FPT algorithm.

**Theorem 4.** There is a $2^{O(k \log k)} N^{O(1)}$ time algorithm for WIDE FAMILY HITTING.

**Proof.** First, we guess the largest set $X = S_i$ selected to the solution. Then we can remove all sets $A$ from the other families with $|A| > |X|$ or $|A \cup X| > k$. Therefore, as $|X| \leq k$, by Lemma 6, we can now assume that $|\mathcal{F}_i| \leq 2^k$ for each $i$.

Then, we process the families $\mathcal{F}_i$ in an order from $i = 1$ to $i = m$, accumulating a partial solution $X$ being the union of the selected sets so far. Suppose that at index $i$, the family $\mathcal{F}_i$ contains a set $S_i$ with $S_i \subseteq X$. Then, we can greedily include $S_i$ to the solution. Otherwise, we branch on which set $S_i \in \mathcal{F}_i$ we include to the solution, which increases the size of our partial solution $X$. As we can increase $X$ at most $k$ times and $|\mathcal{F}_i| \leq 2^k$, this gives a $2^k k^k N^{O(1)} = 2^{O(k^2)} N^{O(1)}$ time algorithm.
To optimize the algorithm to $2^{O(k \log k)} N^{O(1)}$ time, we say that a set $A \in \mathcal{F}_i$ is \emph{light} with respect to the partial solution $X$ if $|X \cup A| - |X| \leq (k - |X|)/2$. In particular, a set $A$ is light if including it to $X$ decreases the remaining budget by at most half, while a set $A$ is \emph{heavy} if including it to $X$ decreases the remaining budget by more than a half. By Lemma 7, $\mathcal{F}_i$ contains at most one light set.

In the branching, we can select a heavy set at most $O(\log k)$ times, and otherwise we select a light set. As there are $2^k$ options only when we select a heavy set and only one option when we select a light set, the time complexity becomes $2^k (2^k)^{O(\log k)} N^{O(1)} = 2^{O(k \log k)} N^{O(1)}$. □

Then, we give a polynomial-time algorithm when $|\mathcal{F}_i|$ is bounded.

**Theorem 5.** Let $|\mathcal{F}_i| \leq d$ for every $i$. Then there is a $k^{O(\log d)} N^{O(1)}$ time randomized algorithm for \textsc{Wide Family Hitting}.

**Proof.** As in the proof of Theorem 4, we again process the families $\mathcal{F}_i$ from $i = 1$ to $i = m$, but this time instead of branching, we decide probabilistically which set to include in the solution.

At step $i$, let $X \subseteq U$ denote the accumulated partial solution so far (the union of the selected sets), and let $b_i = k - |X|$ be the remaining budget. Again, as in Theorem 4, we say that set $A \in \mathcal{F}_i$ is light if $|X \cup A| - |X| \leq b_i/2$, i.e., including $A$ to the solution takes less than half of the remaining budget. Otherwise a set $A \in \mathcal{F}_i$ is heavy. By Lemma 7, there is at most one light set in $\mathcal{F}_i$.

First, if there is a set $A \in \mathcal{F}_i$ with $A \subseteq X$, we can greedily select the set $A$. Otherwise, if $b_i = 0$ we must return that there is no solution, and if $b_i \geq 1$, our algorithm selects a set from $\mathcal{F}_i$ as follows. If there is a light set $L \in \mathcal{F}_i$, let $c_i = |X \cup L| - |X| \geq 1$ be the cost of $L$. Otherwise, we let $c_i = b/2$. Note that in both cases $c_i \leq b_i/2$. By Lemma 7, the cost of any heavy set $H \in \mathcal{F}_i$ is $|X \cup H| - |X| > b_i - c_i$. Our algorithm includes the light set $L$ to the solution with probability $\frac{b_i - c_i}{b_i}$ (if a light set exists), and any heavy set $H$ with probability $\frac{c_i}{b_i}$. Note that as $|\mathcal{F}_i| \leq d$, these probabilities sum up to a number at most 1.

We claim that the probability that our algorithm finds a solution if one exists is at least

$$\frac{1}{2k} \cdot \left( \frac{1}{d} \right)^{1 + \log_2 k}.$$

We prove this by induction on $b$. The base case is that the set in $\mathcal{F}_i$ that belongs to the solution takes up all of the remaining budget, in particular, that a heavy set $H \in \mathcal{F}_i$ with $|X \cup H| - |X| = b_i$ is in the solution. In this case, the algorithm is correct as long as it selects $H$ at this step, as the remaining steps will be deterministic. As $c_i \geq 1/2$ and $k \geq b_i \geq 1$, the probability of correctness is

$$\frac{c_i}{b_i} \geq \frac{1}{2b_i} \cdot \frac{1}{d} \geq \frac{1}{2b_i} \cdot \left( \frac{1}{d} \right)^{1 + \log_2 b_i} \geq \frac{1}{2k} \cdot \left( \frac{1}{d} \right)^{1 + \log_2 k},$$

so the base case is satisfied.

Otherwise, a set from $\mathcal{F}_i$ that does not take all of the remaining budget belongs to the correct solution. Suppose this set is light. Now, by induction, the probability that the algorithm is correct is

$$\frac{b_i - c_i}{b_i} \cdot \frac{1}{2(b_i - c_i)} \cdot \left( \frac{1}{d} \right)^{1 + \log_2 (b_i - c_i)} \geq \frac{1}{2b_i} \cdot \left( \frac{1}{d} \right)^{1 + \log_2 b_i} \geq \frac{1}{2k} \cdot \left( \frac{1}{d} \right)^{1 + \log_2 k},$$

so the induction holds.

Then, suppose that the set from $\mathcal{F}_i$ that belongs to the correct solution is a heavy set $H \in \mathcal{F}_i$. Note that in this case the remaining budget is $b_i - (|X \cup H| - |X|) < b_i - (b_i - c_i) = c_i \leq b_i/2$. By
induction, the algorithm is correct with probability
\[
\frac{c_i}{b_id} \cdot \frac{1}{2(b_i - (|X \cup H| - |X|))} \cdot \left(\frac{1}{d}\right)^{1+\log_2(b_i-(|X \cup H| - |X|))} \geq \frac{c_i}{b_i} \cdot \frac{1}{2c_i} \cdot \left(\frac{1}{d}\right)^{2+\log_2(b_i/2)} \\
\geq \frac{1}{2b_i} \cdot \left(\frac{1}{d}\right)^{1+\log_2 b_i} \\
\geq \frac{1}{2k} \cdot \left(\frac{1}{d}\right)^{1+\log_2 k}.
\]

We have analyzed all cases in the induction, and therefore the algorithm is correct with probability \(\frac{1}{\pi} \cdot (1/d)^{1+\log_2 k}\), and therefore, repeating it \(2k \cdot d^{O(\log k)} = k^{O(\log d)}\) times yields a correct result with constant probability.

6 CONCLUSION

We gave an \(n^{O(\log 1/\epsilon)}\) time PTAS for the shortest monotone submodular cycle problem, and showed unconditional lower bounds establishing that this algorithm is optimal even in a very restricted setting, in particular even when the function is integer-valued, \(\text{OPT} = O(n)\), and the graph is planar and has bounded pathwidth.

We leave several open questions. The main question about minimum cycles is the complexity of MINIMUM CYCLE. From what we know, there is no evidence against the existence of a polynomial-time algorithm. On the other hand, it also could be that our quasipolynomial-time algorithm for integer-valued monotone submodular functions is also optimal for MINIMUM CYCLE. This problem seems difficult, and therefore we believe it is worth exploring even some special cases of it. In particular, we also ask if the WIDE FAMILY HITTING problem admits a polynomial-time algorithm. While Theorem 4 shows that the special case of WIDE FAMILY HITTING is fixed-parameter tractable, it remains a challenging question whether the HEDGE MINIMUM CYCLE is fixed-parameter tractable in the general case. Of course, if the problem is in P this would resolve all these questions.

In the other direction, towards showing the hardness of MINIMUM CYCLE, we ask a purely combinatorial question which is a prerequisite for showing the hardness. We say that a subset \(S\) of the hedges is a minimal partial solution if \(|S| \leq k\), and there is a pair of vertices \(s, t\) so that \(S\) induces a \((s, t)\)-path, but no subset of \(S\) induces an \((s, t)\)-path. We ask if there is a construction of a graph with hedges where the number of minimal partial solutions is superpolynomial. Note that if the number of minimal partial solutions is polynomially bounded, then we can solve HEDGE MINIMUM CYCLE in polynomial time by a simple algorithm enumerating them.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable in-depth comments that led to the improvement of the presentation of the results of this paper.

REFERENCES

[1] Hajo Broersma, Xueliang Li, Gerhard J. Woeginger, and Shenggui Zhang. 2005. Paths and cycles in colored graphs. Australas. J. Comb. 31 (2005), 299–312. http://ajc.maths.uq.edu.au/pdf/31/ajc_v31_p299.pdf
[2] William H. Cunningham. 1985. On submodular function minimization. Combinatorica 5, 3 (1985), 185–192. DOI: https://doi.org/10.1007/BF02579361
[3] Reinhard Diestel. 2017. Graph Theory, 5th Edition. Graduate texts in mathematics, Vol. 173. Springer. DOI: https://doi.org/10.1007/978-3-662-53622-3
[4] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. Numer. Math. 1 (1959), 269–271. DOI: https://doi.org/10.1007/BF01386390
[5] Michael L. Fredman and Robert Endre Tarjan. 1984. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. In Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 338–346. DOI: https://doi.org/10.1109/SFCS.1984.715934

[6] Mohsen Ghaffari, David R. Karger, and Debmalya Panigrahi. 2017. Random contractions and sampling for hypergraph and hedge connectivity. In Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM, 1101–1114. DOI: https://doi.org/10.5555/3039686.3039757

[7] Gagan Goel, Chinmay Karande, Pushkar Tripathi, and Lei Wang. 2009. Approximability of combinatorial problems with multi-agent submodular cost functions. In Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS). IEEE, 755–764. DOI: https://doi.org/10.1109/FOCS.2009.81

[8] Michel X. Goemans, Nicholas J. A. Harvey, Satoru Iwata, and Vahab S. Mirrokni. 2009. Approximating submodular functions everywhere. In Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM, 535–544. DOI: https://doi.org/10.1137/1.9781611973068

[9] Martin Grötschel, László Lovász, and Alexander Schrijver. 1981. The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1, 2 (1981), 169–197. DOI: https://doi.org/10.1007/BF02579273

[10] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. 2001. A combinatorial strongly polynomial algorithm for minimizing submodular functions. Journal of the ACM (JACM) 48, 4 (2001), 761–777. DOI: https://doi.org/10.1145/502090.502096

[11] Satoru Iwata and Kiyohito Nagano. 2009. Submodular function minimization under covering constraints. In Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS). IEEE, 671–680. DOI: https://doi.org/10.1109/FOCS.2009.31

[12] Satoru Iwata and James B. Orlin. 2009. A simple combinatorial algorithm for submodular function minimization. In Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM, 1230–1237. DOI: https://doi.org/10.5555/1496770.1496903

[13] Lars Jaffke, Paloma T. Lima, Tomáš Masarík, Marcin Pilipczuk, and Uéverton S. Souza. 2022. A tight quasi-polynomial bound for Global Label Min-Cut. CoRR abs/2207.07426 (2022). arXiv:2207.07426

[14] Lars Jaffke, Paloma T. Lima, Tomáš Masarík, Marcin Pilipczuk, and Uéverton S. Souza. 2023. A tight quasi-polynomial bound for global label min-cut. In Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM, 290–303. DOI: https://doi.org/10.1137/1.9781611977554.ch12

[15] Stefanie Jegelka and Jeff Bilmes. 2016. Graph cuts with interacting edge weights: Examples, approximations, and algorithms. Mathematical Programming (2016), 1–42. DOI: https://doi.org/10.1007/s10107-016-1038-y

[16] Alexander Schrijver. 2000. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. Journal of Combinatorial Theory, Series B 80, 2 (2000), 346–355. DOI: https://doi.org/10.1006/jctb.2000.1989

[17] Zoya Svitkina and Lisa Fleischer. 2011. Submodular approximation: Sampling-based algorithms and lower bounds. SIAM J. Comput. 40, 6 (2011), 1715–1737. DOI: https://doi.org/10.1137/10078335

[18] Laurence A. Wolsey. 1982. An analysis of the greedy algorithm for the submodular set covering problem. Combinatorica 2, 4 (1982), 385–393. DOI: https://doi.org/10.1007/BF02579435

[19] Peng Zhang, Jin-Yi Cai, Lin-Qing Tang, and Wen-Bo Zhao. 2011. Approximation and hardness results for label cut and related problems. Journal of Combinatorial Optimization 21, 2 (2011), 192–208. DOI: https://doi.org/10.1007/s10878-009-9222-0

Received 12 November 2022; revised 18 September 2023; accepted 24 September 2023