Fixed parameter inapproximability for Clique and Set-Cover

Mohammad T. Hajiaghayi∗ Rohit Khandekar† Guy Kortsarz‡

Abstract

A minimization (resp., maximization) problem is called fixed parameter \((r,t)\)-approximable for two functions \(r,t\) if there exists an algorithm that given an integer \(k\) and a problem instance \(I\) with optimum value \(\text{opt}\), finds either a feasible solution of value at most \(r(k) \cdot k\) (resp., at least \(k/r(k)\)) or a certificate that \(k < \text{opt}\) (resp., \(k > \text{opt}\)), in time \(t(k) \cdot |I|^{O(1)}\). A problem is called fixed parameter \((r,t)\)-hard (or \((r,t)\)-FPT-hard) if it is not \((r,t)\)-approximable. Fellows [7] conjectured that clique and setcover are \((r,t)\)-FPT-hard for all functions \(r\) and \(t\). We prove the first fixed parameter hardness for clique and setcover, for super exponential functions \(t\). Our results are as follows.

1. Assuming ETH and Projection Game Conjecture (PGC), setcover is \((r,t)\)-FPT-hard for \(r(k) = (\log k)^\gamma\) and \(t(k) = \exp(\exp((\log k)^\gamma)) = \exp(k^{(\log k)^f})\) for some constant \(\gamma > 1\) and \(f = \gamma - 1\).

2. Under ETH and a stronger for of PGC, setcover is \((r,t)\)-FPT-hard for \(r(k) = \text{opt}^{d_1}\) and \(t(k) = \exp(\exp(k^{d_2}))\) for some constants \(d_1, d_2 > 0\).

3. Under ETH alone, setcover is \((r,t)\)-FPT-hard for \(r(k) = c\sqrt{\log k}\) and \(t(k) = \exp(k^{(\log k)^f})\) for some constants \(c, f > 0\).

4. Under ETH, for any constant \(c\), clique is \((c,t)\)-FPT-hard for \(t(k) = \exp(\exp(k^d))\) for some constant \(d\) that depends on \(c\). It is also \((r,t)\)-FPT-hard for some super constant function \(r(k)\) and \(t(k) = \exp(\exp(k/q(k)))\) for an arbitrarily slowly increasing function \(q(k)\).

We show that the crux of FP-hardness is reducing the optimum and suggest simple but effective ways to do so. Feige and Kilian [9] proved that the \((\log n)\)-clique problem, i.e. the problem of finding a clique of size \(\log n\) in a graph of size \(n\), can not be solved exactly in time much better than \(n^{\log n}\). Our results are slightly better as they imply a constant inapproximability for any constant, for the \(k\)-clique problem with \(k = \Omega((\log \log n)^{1/O(1)})\) in time less than doubly exponential in \(k\). Marx [1] asked if clique is \((2,t)\)-approximable for any constant \(c\), achieving a \(c\)-approximation for clique requires time roughly double exponential in the parameter.
1 Introduction

In Fixed Parameter Tractability (FPT) theory, we are given a decision problem $P$ with a parameter $k$, that relates to the problem instance. An FPT algorithm for a problem is an exact algorithm that runs in time $t(k) \cdot n^{O(1)}$ where $n$ is the size of the instance and $t$ is an arbitrary function.\(^1\) An FPT approximation algorithm, given a parameter $k$, approximates the desired solution value within a ratio $r(k)$ and runs in time $t(k) \cdot n^{O(1)}$, for a function $r$. More precisely, an FPT-approximation algorithm is defined as follows. For a minimization (resp., maximization) problem $P$, an algorithm is called an $(r, t)$-FPT-approximation algorithm for $P$ with input parameter $k$, if the algorithm takes as input an instance $I$ with (possibly unknown) optimum value $\text{opt}$ and an integer parameter $k$ and either computes a feasible solution to $I$ with value at most $k \cdot r(k)$ (resp., at least $k/r(k)$) or computes a certificate that $k < \text{opt}$ (resp., $k > \text{opt}$) in time $t(k) \cdot |I|^{O(1)}$. Here the goal is to design algorithms with as slow growing functions $r$ and $t$ as possible. A problem is called $(r, t)$-FPT-inapproximable (or, $(r, t)$-FPT-hard) if it does not admit an $(r, t)$-FPT-approximation algorithm. Here the goal is to show inapproximability with as fast growing functions $r$ and $t$ as possible.

In this paper, we use reductions from 3-sat to prove FPT-hardness. When reducing 3-sat to an optimization problem $P$, a “yes” instance of $P$ is an instance obtained from a satisfiable formula, and a “no” instance of $P$ is an instance obtained from a non-satisfiable formula. Whenever we use reductions from 3-sat, we denote the number of variables by $q$, the number of clauses by $m$ and $N = m + q$. We will assume without loss of generality that $m \geq q$ (the other case is symmetric).

Our work is motivated by a conjecture, by Mike Fellows, concerning parameterized approximation for setcover and clique.

Conjecture 1.1 (FPT-hardness of setcover and clique (Fellows [7]))

- **setcover** is $(r, t)$-FPT-hard for any non-decreasing functions $r$ and $t$.
- **clique** is $(r, t)$-FPT-hard for any non-decreasing functions $r(k) = o(k)$ and $t$.

We show that the crux of FP-hardness is making the value of the optimum small. We suggest simple but effective ways of doing so.

2 Previous work

The following relation is known among the parameterized complexity classes: $\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2]$. It is widely believed that $\text{FPT} \neq \text{W}[1]$. In fact $\text{FPT} = \text{W}[1]$ implies that ETH fails.

**Inapproximability that is sub-exponential in $n$:** A widely explored line of research shows, for clique and setcover, a relation between an approximation, and the running time it requires. Such results are discussed in [20]. Recently, [2] improved [20] to get the following

\(^1\)Unless otherwise stated, all mentioned functions are total computable functions from non-negative integers to themselves.
result. For any $r$ larger than some constant and any constant $\epsilon > 0$, any $r$-approximation algorithm for the maximum independent set problem must run in time at least $2^{n^{1-\epsilon}/r^{1+\epsilon}}$. This nearly matches the upper bound of $2^{n/r}$. In this case super exponential running times are out of the question, because the time depends on $n$. This again shows the power of parameterizing algorithms. In the instance we start with, the optimum is very close to $n$. By reducing $\text{OPT}$ we can get inapproximability for CLIQUE and SETCOVER in time super-exponential in $k$, giving a more refined classification of the problems.

To the best of our knowledge, the effort of showing FPT-hardness for CLIQUE and SETCOVER started with [3].

**Theorem 2.1** [3] Under ETH and Projection Game Conjecture (PGC), there exist constants $1 > F_1, F_2 > 0$ such that the SETCOVER problem does not admit an FPT approximation algorithm with ratio $k^{F_1}$ in time $2^{k^{F_2}}$.

The above theorem uses $F_2 < 1$ hence uses time sub-exponential time in $k$ and is not suited for this paper.

**Theorem 2.2** [3] Unless $\text{NP} \subseteq \text{SUBEXP}$, for every $0 < \delta < 1$ there exists a constant $F = F(\delta) > 0$ such that CLIQUE admits no FPT approximation within $k^{1-\delta}$ in time $2^{k^F}$.

As $F < 1$ in the above construction the running time here too is sub-exponential in $\text{OPT}$ and this theorem is not suited for this paper.

These theorems seem unrelated to the results of [20] and [2] because of the difference between $\text{OPT}$ and $n$. For an example of a very strong inapproximability for the minimum maximal independent set problem see [5].

### 3 Nearly linear PCP and their relation to setcover

Before we provide our results, we need to define things like the projection game conjecture and the relation between PCP and SETCOVER.

#### 3.1 Two ways to describe the PCP theorem

**Definition 3.1** In a reduction from the decision version of 3-SAT (or SAT) to an optimization problem $P$, if the instance of $P$ is derived from a satisfiable formula, it is called the yes instance of $P$ and we denote its optimum value by $\text{OPT}_y$. If the reduction is from a non satisfiable 3-SAT formula, then the instance is called a no instance and its optimum is denoted by $\text{OPT}_n$. $\text{opt}_n/\text{opt}_y$ is called the gap of the reduction.

The PCP theorem is a gap reduction from the decision version of SAT to the optimization version of 3-SAT so that in a yes instance there is an assignment that satisfies all clauses and in a no instance only $1 - \epsilon$ clauses can be simultaneously satisfied.

The PCP is at times represented as the following game. There is a prover and a verifier. Consider an alphabet $\Sigma$. The prover presents a string derived from the 3-SAT instance. Each entry in the string belongs to $\Sigma$. The verifier chooses at random some entries of the string, and
according to their value it outputs accepts (namely says that the 3-SAT formula is satisfiable) or rejects.

**Definition 3.2** The size of the PCP is called almost linear if the string has size $m^{1+o(1)}$ with $m$ the size of the SAT instance.

Two terms are used here. The first is completeness which is the fraction of satisfiable clauses in a yes instance, or the probability that given a yes instance the prover will accept. Since there is an assignment that satisfies all clauses the completeness in the PCP theorem is 1.

The soundness is the probability that given a no instance the prover will reject. As there are always at least $\epsilon$ fraction of non satisfiable clauses the soundness is at least $\epsilon$.

The following theorem is due to [4].

**Theorem 3.3** There exists a a constant $c$ such that for any $\epsilon \geq 1/m^c$ there exists an almost linear PCP of size $m \cdot 2^{\log^c m} \cdot P_1(\log m)$ with completeness 1 and soundness $\epsilon$.

For reasons that will become apparent later, we choose $\epsilon = \log^2 n$. Thus we get the following corollary.

**Theorem 3.4** Given a SAT instance of size $m$, there exists a polynomial $P_1$ and a reduction from SAT to a PCP of size $m \cdot 2^{\log^c m} \cdot P_1(\log m)$ with completeness 1 and soundness $\epsilon = 1/\log^2 n$, and $\alpha < 1$.

In [4], the alphabet is exponential in $1/\epsilon$. However, the following conjecture was raised by Moshkovitz [21].

**Conjecture 3.5** The Projection game conjecture: Given a SAT instance of size $m$ it can be reduced to a PCP of size $m \cdot 2^{\log^c m} \cdot P_1(\log m)$ with completeness 1 and soundness $\epsilon = 1/\log^2 m$ and $\alpha < 1$. Furthermore, the alphabet has size $\text{poly}(1/\epsilon)$ namely size $P_2(\log m)$ for some polynomial $P_2$.

### 3.2 The Label-Cover and Min-Rep problems and their relation to the PCP theorem

Arora and Lund [AL96] introduced the Label-Cover problem as a graph-theoretic description of the PCP. The input to the Label-Cover problem consists of a bipartite graph $G(A, B, E)$, with an explicit partitioning of each of $A$ and $B$ into equal-sized subsets, namely $A = \bigcup_{i=1}^q A_i$ and $B = \bigcup_{j=1}^q B_j$. The bipartite graph $G$ induces a super-graph $H$ as follows. The super-vertices (i.e., the vertices of $H$) are the $2q$ sets $A_i$ and $B_j$. A super-edge (an edge in $H$) connects two super-vertices $A_i$ and $B_j$ if there exist some $a \in A_i$ and $b \in B_j$ which are adjacent in $G$.

A pair $(a, b)$ covers a super-edge $(A_i, B_j)$ if $a \in A_i$ and $b \in B_j$ are adjacent in $G$. Let $S \subseteq A_i \cup B_j$. We say that $S$ covers the super-edge $(A_i, B_j)$ if there exist two vertices $a, b \in S$ such that the pair $(a, b)$ covers the super-edge $(A_i, B_j)$. The goal in the Label-Cover problem is to select a single vertex from every $A_i, B_j$ and cover the maximum amount of super edges.

Say that the verifier inspects two PCP entries (this is the case in [4] and in fact in many other PCP). In this case there is a simple mapping of the PCP to a Label-Cover instance.

We can look at an entry in the string as a question, and at the value of this entry as the answer. Thus we create two provers. Every entry in the PCP is either a question $A_i$ for the first
prover or a question $B_j$ for the second prover. In other words, every $A_i$ and $B_j$ corresponds to a single entry in the PCP. Thus $2q$ equals the length of the PCP. Checking an entry that corresponds to some $A_i$ the verifier can see what is the answer for the question $A_i$ or $B_j$. Namely, the sets $A_i, B_j$ will contain the entire alphabet, and one value should be chosen for every $A_i, B_j$ namely for every entry of the PCP. Note that since each supervertex contains all the alphabet, $|A_i| = |B_j| = P_2(\log(m))$ for every $i, j$. A superedges is correspond to two entries in the PCP that the prover can asked the verifier simultaneously.

Now, say that $(A_i, B_j)$ is a super edge and the answer chosen for $A_i$ is $x$, namely, the content of the index in the PCP that corresponds to $A_i$ is $x$ and for $B_i$ it is $y$. We join $x$ and $y$ by an edge iff after learning the value of the two entries, the verifier accepts. The number of vertices in the Label-Cover instance that corresponds to the PCP is $m \cdot 2^{\log^\alpha m} \cdot P_1(\log m) \cdot P_2(\log m)$. The completeness 1 implies that there is a choice of a unique vertex (answer, or alphabet letter) out of every $A_i$ and $B_j$ that covers all superedges. The soundness of $\epsilon = 1/\log^2 n$ means that any selection of one vertex per super vertex covers at most $1/\log^2 n$ superedges.

The Min-Rep problem is to select $S \subseteq A \cup B$ that covers all superedges.

**Claim 3.6** [10] If Label-Cover is $q$-inapproximable then then Min-Rep is $\sqrt{q}$ inapproximable

**Corollary 3.7** The Min-Rep instance derived from the PGc is $\sqrt{\alpha} = \log m$-inapproximable.

The projection property: The name projection game conjecture follows because the Label-Cover instance satisfies the projection property. This means that for ay superedge (query) $(A_i, B_j)$, each vertex in $B_j$ is adjacent to at most one vertex (its projection) in $A_i$. Namely, $A_i \cup B_j$ is a collection of disjoint stars with heads in $A_i$. We sometimes call this property the star property.

### 3.3 Reducing Min-Rep to setcover [13]

Every one of the $m \cdot 2^{\log^\alpha m} \cdot P_1(\log m)P_2(m)$ vertices of the Min-Rep instance become sets in the SETCOVER instance. For every superedge $A_i, B_j$ we add a ground set of elements $M_{ij}$ of size $m$. For every star in $A_i \cup B_j$ join the head of the star to a random half $U$ of $M_{ij}$ and join the leaves of the star to $M_{ij} - U$. If we join a set and an element, it means that the element belongs to the set. In a yes instance all superedges can be covered by unique representatives from every $A, B_j$. Let $S$ be the unique Min-Rep cover. Note that since $(A_i, B_j)$ is covered, we a set $\{a, \ell\} \subseteq S$ with $a$ a head of a star and $\ell$ one of its leaves. Note that $\{a, \ell\}$ cover $M_{ij}$, Namely, $M_{ij}$ is covered by 2 sets (as by definition $a$ and $\ell$ cover two disjoint halves of $M_{ij}$). And so the size of the minimum SETCOVER for a yes instance is just the number of $A_i$ plus the number of $B_j$, namely, $2q$. But recalling that $2q$ is the length of the PCP we set $k = \text{opt}_y = m \cdot 2^{\log^\alpha m} \cdot P_1(\log m)$. On the other hand, in order to cover all edges in a no instance the size of the Min-Rep has to be $k \cdot \log n$ which means a SETCOVER is $\log m$-hard to approximate. Otherwise, taking just few vertices from every $A_i, B_j$ means that most superedges are not covered. This implies that any $a_i, b_j$, vertices are joined to a random half of $M_{ij}$ (because we do not have in $S$ a head of a star and one of its leaves) and so we still need $\log m$ sets from $A_i \cup B_j$ to cover $M_{ij}$. This shows that the SETCOVER instance has gap $\log m$ in any case. The number of supervertices is at most $q^2$ so there are at most $q^2 \cdot m$ elements. Since $2q = m \cdot 2^{\log^\alpha m} \cdot P_1(\log m) < m^2$, there are less
Corollary 3.8 Corollary 3.9 There exists a reduction from the decision version of SAT to a set cover of with $m \cdot 2^{\log^a m} \cdot P_1(\log m)P_2(m)$ sets and with less than $O(m^5)$ elements so that in a yes instance the optimum is $k = m \cdot 2^{\log^a m} \cdot P_1(\log m)$ and in the no instance $\text{opt}_n \geq k \cdot \log m$.

4 Our results

The term $\exp(t)$ means in this paper $c^t$ for some constant $c > 1$.

Theorem 4.1 Under the ETH and PGC conjectures, SetCover is $(r, t)$-FPT-hard for $r(k) = (\log k)^\gamma$ and $t(k) = \exp(\exp((\log k)^\gamma)) \cdot \text{poly}(n) = \exp\left(k^{(\log k)^\gamma}\right) \cdot \text{poly}(n)$ for some constant $\gamma > 1$ and $f = \gamma - 1$.

Theorem 4.2 Under ETH and the above stronger version of PGC there exists constants $d_1, d_2 > 0$ so that SetCover is $(r, t)$-FPT-hard for $r(k) = \text{opt}^{d_1}$ and $t(k) = \exp(\exp(k^{d_2}))$.

We can also prove an inapproximability with super-exponential time in opt that only assumes ETH.

Theorem 4.3 Under ETH alone, SetCover cannot be approximated within $c\sqrt{\log k}$ for some constant $c$, in time $\exp\left(k^{(\log k)^\gamma}\right) \cdot \text{poly}(n)$ for $f$ the same constant from Theorem 4.1.

Theorem 4.4 Under the ETH, for any constant $c$, Clique is $(c, t)$-FPT-hard for $t(k) = \exp(\exp(k^d))$ for some constant $d$ that depends on $c$. It is also $(r, t)$-FPT-hard for some super constant function $r(k)$ and $t(k) = \exp(\exp(k/q(k)))$ for an arbitrarily slowly increasing function $q(k)$.

An open question raised by Marx: In the survey [1] Marx raised the question if its possible to get a ratio 2 in time $f(\text{opt})n^{O(1)}$ for some $f$. We show that any constant ratio approximation $c$ requires time $t(k) = \exp(\exp(k^d))$ for some constant $d'$. While we are not solving the open problem raised by Marx, we show that a constant ratio requires running time that is almost doubly exponential in $k$.

Recall that the log $n$-Clique problem, is the Clique problem when the optimum is $\log n$. Feige et. al. identified this problem as very important, because any algorithm that runs in time much better than the exhaustive search algorithm, shows that NP problems can be solved in subexponential time.

Our result is not directly related to [9]. Still we show that if the optimum is of value $\Omega(\log \log n)$, a $c$ approximation requires doubly exponential in $k$. Indeed, the new optimum $k'$ equals $\Theta(\log \log n)$. Thus, we give inapproximability with an exponentially smaller value of OPT.

Remark: Our results do not give any inapproximability if $\text{opt} = \log \log \log n$. 

Definition 4.5 In all our hardness proof we are basing the proof on reduction from 3-SAT with a constant gap between a yes and a no instance, and we set the parameter to $k = \text{opt}_y$.

This means that unlike some FPT-hardness in the literature, our $k$ equals the optimum of some instance.

5 Inapproximability for Set Cover with super-exponential time in $k$

In this section we prove Theorem 4.1.

We start with the setcover instance of Theorem 3.9 We denote the size of a yes instance by $k$. The idea is to make the optimum much smaller. We introduce a set $s \in S'$ as $s = \cup_{i=1}^p s_i$ for each subcollection $\{s_1, s_2, \ldots, s_p\} \subseteq S$ of size $p = \lfloor m / \log m \rfloor$.

Claim 5.1 The number of sets in the new instance $S' = (U, S')$ is $2^{o(m)}$. The new instance can be constructed in time $2^{o(m)}$.

Proof: Recall that the number of sets is $\sigma = m \cdot 2^{\log^\alpha m} \cdot P_1((\log(m)))$. Thus since $p \leq \sigma/2$, the number of sets in the new instance is

$$\sum_{p=1}^{\lfloor m / \log m \rfloor} \binom{\sigma}{p} \leq \lfloor m / \log m \rfloor \cdot \left( \frac{e \cdot 2^{\log^\alpha m} \cdot P_1(\log(m))}{\lfloor m / \log m \rfloor} \right)^{m / \log m}.$$

We use the inequality $\binom{n}{p} \leq (ne/p)^p$ to upper-bound this by

$$\lfloor m / \log m \rfloor \cdot \left( e \cdot 2^{\log^\alpha m} \cdot P_1(\log(m)) \cdot 2 \log m \right)^{m / \log m} = 2^{O(m/\log m)} = 2^{o(m)}.$$

The last equality holds since $0 < \alpha < 1$. It is easy to see that the new instance can be created in $2^{o(m)}$ time. □

Clearly, any optimum will use as few sets of size (roughly) $m / \log m$ and so the gap between a “Yes” instance and a “No” hardly changed. Namely, the new optimum for a yes instance, $k' = \text{opt}'_y$ and of a no instance $\text{opt}'_n$ are related as $\text{opt}'_y / \lfloor m / \log m \rfloor \leq \text{opt}'_n \leq \lfloor \text{opt}_1 / \lfloor m / \log m \rfloor \rfloor$. Therefore the gap between the new optima of a yes instance and a no instance continues to be $\log m$ and the new optimum of the yes instance is at least $k' = \text{opt}'_y = \lfloor \text{opt}_y / (m / \log m) \rfloor = O(2^{\log^\alpha m} \log m P_1(\log(m)))$. And the optimum for a no instance $\text{opt}'_y$ is at least $k' \cdot \log m$.

Now define two functions $r(k) = (\log k)^\gamma$ and $t(k) = \exp(\exp((\log k)^\gamma))$ for any $1 < \gamma < 1/\alpha$, as given in Theorem 4.1. Note that $r(k') = O((\log^\alpha m)^\gamma) = o((\log^\alpha m)^{1/\alpha}) = o(\log m)$ and $t(k') = 2^{o(m)}$. Thus setcover is $(r, t)$-FPT-hard for these functions, proving Theorem 4.1.

Remark: The above calculations explain why we use an almost linear PCP. Indeed, otherwise the running time would be much larger than $2^m$.

6 FPT hardness for clique

We use the following theorem by Dinur.
Theorem 6.1 There exist a reduction from SAT of size $m$ to a 3-SAT of size $m \cdot \text{polylog } m$ so that for a yes instance all clauses of the 3-SAT can be satisfied, and for a no instance at most $1/2$ of the clauses can be simultaneously satisfied.

Thus, if $m', q'$ be the new number of clauses and variables in the optimization version of 3-SAT instance, $m' + q' = O(m \cdot \text{poly log}(m))$.

There is a linear reduction from 3-SAT to CLIQUE so that for a yes instance $k = \text{opt}_y = m'$ and for a no instance $\text{OPT}_n \leq (1 - \epsilon)m'$ (see [8]). The term $O(m \cdot \text{poly log}(m))$ is bounded by $m \cdot (\log m)^b$ for some constant $b$.

Corollary 6.2 There exists a reduction from the decision version of 3-SAT to the optimization version of CLIQUE, so that the number of vertices in the CLIQUE instance is $n \leq m \cdot (\log m)^b$ for some constant $b$, and $\text{OPT}_y/\text{opt}_n \geq 1/(1 - \epsilon)$ for some constant $\epsilon$. The optimum $k = \text{opt}_y = m'$ for a yes instance is known.

Proof of the theorem 4.4 We create a new CLIQUE instance. We parameterize by the parameter $k = \text{opt}_y$. Let $f(m)$ be any super constant slowly growing function. Introduce a vertex into the new graph for each subset of size $m/(\lceil b \log \log m \cdot f(m) \rceil)$ vertices in the old CLIQUE instance for the above constant $b$. Every such set is called a ”supervertex”. Two supervertices $A, B$, are connected by an edge, if $A \cup B$ is a clique, and $A \cap B = \emptyset$. The last condition, namely, the fact that two sets that are connected must be disjoint is not needed in the SETCOVER reduction, but it is crucial here.

We ignore floors and ceilings from now on as we saw in the previous proof them hardly have an affect.

Claim 6.3 The new instance of the CLIQUE problem has size $2^{o(m)}$.

Proof: The size of the original clique is bounded by $m \cdot (\log m)^b$. We take all subsets of size $m/(\lceil b \log \log m \cdot f(m) \rceil)$. Using $\binom{n}{p} \leq (n/e)^p$, we get that the number of supervertices is at most $O((\log^{b+1} m)^{m/(\lceil b \log \log m \cdot f(m) \rceil)}) = 2^O(m/f(m)) = 2^{o(m)}$. The last inequality is because $f(m) = \omega(1)$. The number of edges in the new CLIQUE instance, being at most the square of the number of vertices, is also $2^{o(m)}$.

Claim 6.4 The gap between a yes and a no instance remains $1/(1 - \epsilon)$

Proof: Let $\rho = m/(\lceil b \log \log m \cdot f(m) \rceil)$. The new maximum of a yes instance is $k/\rho$ and of a no instance $(1 - \epsilon)k/\rho$ giving the claim.

Graph products: Graph products with parameter $i$, take a graph of size $n$ and produce a graph of size $n^i$ so that the gap grows to $(1/(1 - \epsilon))^i$ and $k = \text{OPT}_y$ grows to $k = \text{opt}_y^i$. Taking a constant graph product, we can make $r(k) \geq c$ for any constant $c$. The running time does not change, except that $d$ depends on $c$. Thus the problem admits a $(c, \exp(\exp(k^d)))$-FPT-hardness with $d$ a constant that depends on $c$.

The running time: We got a constant gap in which the optimum (of a yes instance) is $k' = \text{opt}_y' \leq O(\log m)^{b+1})$. Thus $k'^d = o(\log m)$ for some constant $d$. Thus $t(k') = \exp(\exp(k^d)) = 2^{o(m)}$. Hence the problem admits a $(c, \exp(\exp(k^d)))$-FPT-hardness with $d$ a constant that depends on $c$.

To get super constant $r(k)$ we use graph products with $i = \sqrt{(f(m))}$ and the size of the graph remains $2^{o(m)}$ by Claim 6.3. Thus gives a gap of $r(k) = (1/(1 - \epsilon))\sqrt{f(m)}$. Clearly $r(k)$ is a super
constant function. The increase in the size of the optimum mean that \( t(k) = \exp(\exp(k/q(k))) \) for an arbitrarily slowly increasing function \( q(k) \).

**Proof of Theorem 4.2** Let \( P_1 \) and \( P_2 \) be polynomial as in Corollary 3.9. We assume

**Conjecture 6.5** There exists a constant \( c > 0 \) and a PCP of size \( m \cdot \text{poly log}(m) P_1(1/\epsilon) \), for any \( \epsilon \) so that \( \epsilon \geq 1/m^c \).

This following was conjectured to hold by Moshkovitz in a private communication. We now use the above conjecture and show a much stronger FPT inapproximability for SETCOVER. By Corollary 3.9, and the above conjecture we get the following corollary, using \( \epsilon = c'/\log^2 m \) for a large enough constant \( c' \):

**Corollary 6.6** There exists a a reduction from 3-SAT to SETCOVER so that:

1. The number of sets is \( \sigma = m \cdot P_1(\log(m)) \)
2. The number of elements is \( \text{poly}(m) \).
3. The value of the optimum in yes instance is exactly \( k = m \cdot P_2(\log(m)) \) and that in the “no” instance is larger by a factor of at least \( \log m \cdot k \).

A proof of along the lines of the proof of Theorems 4.1 and 4.4 gives that SETCOVER is \((k^{d'}, \exp(\exp(k/r(k))))\)-FPT-hard.

### 7 Proof of Theorem 4.3

The following is proved in [4]. Let \( P_1(m) \) and \( P_2(m) \) be two polynomials as in Corollary 3.9.

**Theorem 7.1** There exists a constant \( c \) and a PCP of size \( m \cdot 2^{\log^\alpha m} \cdot \text{poly}(1/\epsilon) \), such that the size of the alphabet is at most \( \exp(1/\epsilon) \) and the gap that can be chosen to be \( 1/\epsilon \) for any \( \epsilon > 1/m^c \).

**Proof:** We choose \( \epsilon = \ln 2 \cdot \log^\alpha m \). Note that \( \exp(\epsilon) = 2^{\log^\alpha m} \). Therefore when using 3-SAT to SETCOVER described in Corollary 3.9 we get:

**Corollary 7.2** There exists a constant \( d > 0 \), and a constant \( 0 < \alpha < 1 \) and a reduction from 3-SAT to SETCOVER so that:

1. The number of sets is \( m \cdot 2^{2\log^\alpha m} \)
2. The number of elements is \( \text{poly}(m) \).
3. The gap is \( d \cdot \sqrt{\log^\alpha m} \).
4. The optimum of a yes instance does not change, namely, is \( \text{OPT} = 2^{\log^\alpha m} \cdot \text{poly log}(m) \).

The optimum for a yes instance, namely \( k \) does not change because it does not depend on the size of the alphabet.

The gap is \( d \sqrt{\log^\alpha k} \) for some constant \( d \). \( \text{OPT} = 2^{\log^\alpha k} \). Thus for some constant \( c \), the problem is \((c \cdot \sqrt{\log k}, \exp(\exp(k/\log(\log k))))\)-FPT hard for the same constant \( f > 0 \), that appears in Theorem 4.1. **□**
References


