Lossy Kernelization

Daniel Lokshtanov[*] Fahad Panolan[*] M. S. Ramanujan[†] Saket Saurabh[*‡]

January 25, 2017

Abstract

In this paper we propose a new framework for analyzing the performance of preprocessing algorithms. Our framework builds on the notion of kernelization from parameterized complexity. However, as opposed to the original notion of kernelization, our definitions combine well with approximation algorithms and heuristics. The key new definition is that of a polynomial size α-approximate kernel. Loosely speaking, a polynomial size α-approximate kernel is a polynomial time pre-processing algorithm that takes as input an instance (I, k) to a parameterized problem, and outputs another instance (I', k') to the same problem, such that $|I'| + k' \leq k^{\mathcal{O}(1)}$. Additionally, for every $c \geq 1$, a c-approximate solution s' to the pre-processed instance (I', k') can be turned in polynomial time into a $(c \cdot \alpha)$-approximate solution s to the original instance (I, k).

Our main technical contribution are α-approximate kernels of polynomial size for three problems, namely Connected Vertex Cover, Disjoint Cycle Packing and Disjoint Factors. These problems are known not to admit any polynomial size kernels unless $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$. Our approximate kernels simultaneously beat both the lower bounds on the (normal) kernel size, and the hardness of approximation lower bounds for all three problems. On the negative side we prove that Longest Path parameterized by the length of the path and Set Cover parameterized by the universe size do not admit even an α-approximate kernel of polynomial size, for any $\alpha \geq 1$, unless $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$. In order to prove this lower bound we need to combine in a non-trivial way the techniques used for showing kernelization lower bounds with the methods for showing hardness of approximation.

[*]University of Bergen, Norway. {daniello,fahad.panolan}@ii.uib.no
[†]Technische Universität Wien, Vienna, Austria. ramanujan@ac.tuwien.ac.at
[‡]The Institute of Mathematical Sciences, HBNI, Chennai, India. saket@imsc.res.in

1 Introduction

Polynomial time preprocessing is one of the widely used methods to tackle NP-hardness in practice. However, for decades there was no mathematical framework to analyze the performance of preprocessing heuristics. The advent of parameterized complexity made such an analysis possible. In parameterized complexity every instance I comes with an integer *parameter* k, and the goal is to efficiently solve the instances whose parameter k is small. Formally a parameterized decision problem Π is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The goal of parameterized algorithms is to determine whether an instance (I, k) given as input belongs to Π or not.

On an intuitive level, a low value of the parameter k should reflect that the instance (I, k) has some additional structure that can be exploited algorithmically. Consider an instance (I, k) such that k is very small and I is very large. Since k is small, the instance is supposed to be easy. If I is large and easy, this means that large parts of I do not contribute to the computational hardness of the instance (I, k). The hope is that these parts can be identified and reduced in polynomial time. This intuition is formalized as the notion of *kernelization*. Let $g : \mathbb{N} \to \mathbb{N}$ be a function. A *kernel of size* $g(k)$ for a parameterized problem Π is a polynomial time algorithm that takes as input an instance (I, k) and outputs another instance (I', k') such that $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi$ and $|I'| + k' \leq g(k)$. If $g(k)$ is a linear, quadratic or polynomial function of k, we say that this is a linear, quadratic or polynomial kernel, respectively.

The study of kernelization has turned into an active and vibrant subfield of parameterized complexity, especially since the development of complexity-theoretic tools to show that a problem does not admit a polynomial kernel [6, 7, 25, 32, 36], or a kernel of a specific size [16, 17, 37]. Over the last decade many new results and several new techniques have been discovered, see the survey articles by Kratsch [42] or Lokshtanov *et al.* [44] for recent developments, or the textbooks [15, 23] for an introduction to the field.

Despite the success of kernelization, the basic definition has an important drawback: *it does not combine well with approximation algorithms or with heuristics.* This is a serious problem since after all the ultimate goal of parameterized algorithms, or for that matter of any algorithmic paradigm, is to eventually solve the given input instance. Thus, the application of a pre-processing algorithm is always followed by an algorithm that finds a solution to the reduced instance. In practice, even after applying a pre-processing procedure, the reduced instance may not be small enough to be solved to optimality within a reasonable time bound. In these cases one gives up on optimality and resorts to approximation algorithms or heuristics instead. Thus it is *crucial* that the solution obtained by an approximation algorithm or heuristic when run on the reduced instance provides a good solution to the original instance, or at least *some* meaningful information about the original instance. The current definition of kernels allows for kernelization algorithms with the unsavory property that running an approximation algorithm or heuristic on the reduced instance provides *no insight whatsoever* about the original instance. In particular, the *only* thing guaranteed by the definition of a kernel is that the reduced instance (I', k') is a yes instance if and only if the original instance (I, k) is. If we have an α-approximate solution to (I', k') there is no guarantee that we will be able to get an α-approximate solution to (I, k), or even able to get any feasible solution to (I, k).

There is a lack of, and a real need for, a mathematical framework for analysing the performance of preprocessing algorithms, such that the framework not only combines well with parameterized and exact exponential time algorithms, but also with approximation algorithms and heuristics. *Our main conceptual contribution is an attempt at building such a framework.*

The main reason that the existing notion of kernelization does not combine well with approximation algorithms is that the definition of a kernel is deeply rooted in decision problems. The starting point of our new framework is an extension of kernelization to optimization problems.

This allows us to define α-approximate kernels. Loosely speaking an (α)-approximate kernel of size $g(k)$ is a polynomial time algorithm that given an instance (I, k) outputs an instance (I', k') such that $|I'| + k' \leq g(k)$ and any c-approximate solution s' to the instance (I', k') can be turned in polynomial time into a $(c \cdot \alpha)$-approximate solution s to the original instance (I, k). In addition to setting up the core definitions of the framework we demonstrate that our formalization of lossy pre-processing is *robust*, *versatile* and *natural*.

To demonstrate *robustness* we show that the key notions behave consistently with related notions from parameterized complexity, kernelization, approximation algorithms and FPT-approximation algorithms. More concretely we show that a problem admits an α-approximate kernel *if and only if* it is FPT-α-approximable, mirroring the equivalence between FPT and kernelization [15]. Further, we show that the existence of a polynomial time α-approximation algorithm is equivalent to the existence of an α-approximate kernel of constant size.

To demonstrate *versatility* we show that our framework can be deployed to measure the efficiency of pre-processing heuristics both in terms of the value of the optimum solution, and in terms of structural properties of the input instance that do not necessarily have any relation to the value of the optimum. In the language of parameterized complexity, we show that framework captures approximate kernels both for problems parameterized by the value of the optimum, and for structural parameterizations.

In order to show that the notion of α-approximate kernelization is *natural*, we point to several examples in the literature where approximate kernelization has already been used implicitly to design approximation algorithms and FPT-approximation algorithms. In particular, we show that the best known approximation algorithm for Steiner Tree [11], and FPT-approximation for Partial Vertex Cover [45] and for Minimal Linear Arrangement parameterized by the vertex cover number [29] can be re-interpreted as running an approximate kernelization first and then running an FPT-approximation algorithm on the preprocessed instance.

A common feature of the above examples of α-approximate kernels is that they beat both the known lower bounds on kernel size of traditional kernels and the lower bounds on approximation ratios of approximation algorithms. Thus, it is quite possible that many of the problems for which we have strong inapproximability results and lower bounds on kernel size admit small approximate kernels with approximation factors as low as 1.1 or 1.001. If this is the case, it would offer up at least a partial explanation of why pre-processing heuristics combined with brute force search perform so much better than what is predicted by hardness of approximation results and kernelization lower bounds. This gives another compelling reason for a systematic investigation of lossy kernelization of parameterized optimization problems.

The observation that a lossy pre-processing can simultaneously achieve a better size bound than normal kernelization algorithms as well as a better approximation factor than the ratio of the best approximation algorithms is not new. In particular, motivated by this observation Fellows *et al.* [30] initiated the study of lossy kernelization. Fellows *et al.* [30] proposed a definition of lossy kernelization called *α-fidelity kernels*. Essentially, an α-fidelity kernel is a polynomial time pre-processing procedure such that an *optimal solution* to the reduced instance translates to an *α-approximate solution* to the original. Unfortunately this definition suffers from the same serious drawback as the original definition of kernels - it does not combine well with approximation algorithms or with heuristics. Indeed, in the context of lossy pre-processing this drawback is even more damning, as there is no reason why one should allow a loss of precision in the pre-processing step, but demand that the reduced instance has to be solved to optimality. Furthermore the definition of α-fidelity kernels is usable only for problems parameterized by the value of the optimum, and falls short for structural parameterizations. For these reasons we strongly believe that the notion of α-approximate kernels introduced in this work is a better model of lossy kernelization than α-fidelity kernels are.

It is important to note that even though the definition of α-approximate kernels crucially

Problem Name	Apx.	Apx. Hardness	Kernel	Apx. Ker. Fact.	Appx. Ker. Size
Connected V.C.	2 [4,52]	$(2-\epsilon)$ [41]	no $k^{\mathcal{O}(1)}$ [22]	$1 < \alpha$	$k^{f(\alpha)}$
Cycle Packing	$\mathcal{O}(\log n)$ [51]	$(\log n)^{\frac{1}{2}-\epsilon}$ [33]	no $k^{\mathcal{O}(1)}$ [8]	$1 < \alpha$	$k^{f(\alpha)}$
Disjoint Factors	2	no PTAS	no $\lvert\Sigma\rvert^{\mathcal{O}(1)}$ [8]	$1 < \alpha$	$\lvert\Sigma\rvert^{f(\alpha)}$
Longest Path	$\mathcal{O}(\frac{n}{\log n})$ [2]	$2^{(\log n)^{1-\epsilon}}$ [39]	no $k^{\mathcal{O}(1)}$ [6]	any α	no $k^{\mathcal{O}(1)}$
Set Cover/n	$\ln n$ [55]	$(1-\epsilon)\ln n$ [47]	no $n^{\mathcal{O}(1)}$ [22]	any α	no $n^{\mathcal{O}(1)}$
Hitting Set/n	$\mathcal{O}(\sqrt{n})$ [48]	$2^{(\log n)^{1-\epsilon}}$ [48]	no $n^{\mathcal{O}(1)}$ [22]	any α	no $n^{\mathcal{O}(1)}$
Vertex Cover	2 [55]	$(2-\epsilon)$ [21,41]	$2k$ [15]	$1 < \alpha < 2$	$2(2-\alpha)k$ [30]
d-Hitting Set	d [55]	$d-\epsilon$ [20,41]	$\mathcal{O}(k^{d-1})$ [1]	$1 < \alpha < d$	$\mathcal{O}((k \cdot \frac{d-\alpha}{\alpha-1})^{d-1})$ [30]
Steiner Tree	1.39 [11]	no PTAS [12]	no $k^{\mathcal{O}(1)}$ [22]	$1 < \alpha$	$k^{f(\alpha)}$
OLA/v.c.	$\mathcal{O}(\sqrt{\log n}\log\log n)$ [28]	no PTAS [3]	$f(k)$ [43]	$1 < \alpha < 2$	$f(\alpha)2^k k^4$
Partial V.C.	$(\frac{4}{3}-\epsilon)$ [27]	no PTAS [49]	no $f(k)$ [35]	$1 < \alpha$	$f(\alpha)k^5$

Figure 1: *Summary of known and new results for the problems considered in this paper. The columns show respectively: the best factor of a known approximation algorithm, the best known lower bound on the approximation ratio of polynomial time approximation algorithms, the best known kernel (or kernel lower bound), the approximation factor of the relevant approximate kernel, and the size of that approximate kernel. In the problem name column, V.C. abbreviates vertex cover. For* Set Cover *and* Hitting Set*, n denotes the size of the universe. The approximate kernelization results for the top block of problems constitute our main technical contribution. The middle block re-states the results of Fellows et al. [30] in our terminology. For the bottom block, the stated approximate kernelization results follow easily by re-interpreting in our terminology a pre-processing step within known approximation algorithms (see Section 6).*

differs from the definition of α-fidelity kernels [30], it seems that most of the pre-processing algorithms that establish the existence of α-approximate kernels can be used to establish the existence of α-fidelity kernels and vice versa. In particular, all of the α-fidelity kernel results of Fellows *et al.* [30] can be translated to α-approximate kernels.

Our Results. Our main technical contribution is an investigation of the lossy kernelization complexity of several parameterized optimization problems, namely Connected Vertex Cover, Disjoint Cycle Packing, Disjoint Factors, Longest Path, Set Cover and Hitting Set. For all of these problems there are known lower bounds [6,8,22] precluding them from admitting polynomial kernels under widely believed complexity theoretic assumtions. Indeed, all of these six problems have played a central role in the development of the tools and techniques for showing kernelization lower bounds.

For Connected Vertex Cover, Disjoint Cycle Packing and Disjoint Factors we give approximate kernels that beat both the known lower bounds on kernel size and the lower bounds on approximation ratios of approximation algorithms. On the other hand, for Longest Path and Set Cover we show that even a constant factor approximate kernel of polynomial size would imply NP $\subseteq$ coNP/Poly, collapsing the polynomial hierarchy. For Hitting Set we show that a constant factor approximate kernel of polynomial size would violate the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi and Zane [38]. Next we discuss our results for each of the six problems in more detail. An overview of the state of the art, as well as the results of this paper can be found in Table 1.

Approximate Kernels. In the Connected Vertex Cover problem we are given as input a graph G, and the task is to find a smallest possible *connected vertex cover* $S \subseteq V(G)$. A vertex set S is a connected vertex cover if $G[S]$ is connected and every edge has at least one endpoint in S. This problem is NP-complete [4], admits a factor 2 approximation algorithm [4,52], and is known not to admit a factor $(2-\epsilon)$ approximation algorithm assuming the Unique Games conjecture [41]. Further, an approximation algorithm with ratio below 1.36 would imply that P = NP [21]. From the perspective of kernelization, it is easy to show that Connected Vertex Cover admits a kernel with at most 2^k vertices [15], where k is the solution size. On the other hand, Dom *et al.* [22] showed that Connected Vertex Cover does not admit a kernel of polynomial size, unless NP $\subseteq$ coNP/Poly. In this work we show that Connected Vertex Cover admits a *Polynomial Size Approximate Kernelization Scheme*, or PSAKS, the approximate kernelization analogue of a polynomial time approximation scheme (PTAS). In particular, for every $\epsilon > 0$, Connected Vertex Cover admits a simple $(1+\epsilon)$-approximate kernel of polynomial size. The size of the kernel is upper bounded by $k^{\mathcal{O}(1/\epsilon)}$. Our results for Connected Vertex Cover show that allowing an arbitrarily small multiplicative loss in precision drastically improves the worst-case behaviour of preprocessing algorithms for this problem.

In the Disjoint Cycle Packing problem we are given as input a graph G, and the task is to find a largest possible collection $\mathcal{C}$ of pairwise disjoint vertex sets of G, such that every set $C \in \mathcal{C}$ induces a cycle in G. This problem admits a factor $\mathcal{O}(\log n)$ approximation algorithm [51], and is known not to admit an approximation algorithm [33] with factor $\mathcal{O}((\log n)^{\frac{1}{2}-\epsilon})$ for any $\epsilon > 0$, unless all problems in NP can be solved in randomized quasi-polynomial time. With respect to kernelization, Disjoint Cycle Packing is known not to admit a polynomial kernel [8] unless $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$. We prove that Disjoint Cycle Packing admits a PSAKS. More concretely we show that for every $\epsilon > 0$, Disjoint Cycle Packing admits a $(1+\epsilon)$-approximate kernel of size $k^{\mathcal{O}(\frac{1}{\epsilon \log \epsilon})}$. Again, relaxing the requirements of a kernel to allow an arbitrarily small multiplicative loss in precision yields a qualitative leap in the upper bound on kernel size from exponential to polynomial. Contrasting the simple approximate kernel for Connected Vertex Cover, the approximate kernel for Disjoint Cycle Packing is quite complex.

On the way to obtaining a PSAKS for Disjoint Cycle Packing we consider the Disjoint Factors problem. In Disjoint Factors, input is an alphabet Σ and a string s in Σ^*. For a letter $a \in \Sigma$, an *a-factor* in s is a substring of s that starts and ends with the letter a, and a *factor* in s is an a-factor for some $a \in \Sigma$. Two factors x and y are *disjoint* if they do not overlap in s. In Disjoint Factors the goal is to find a largest possible subset S of Σ such that there exists a collection $\mathcal{C}$ of pairwise disjoint factors in s, such that for every $a \in S$ there is an a-factor in $\mathcal{C}$. This stringology problem shows up in the proof of the kernelization lower bound of Bodlaender *et al.* [8] for Disjoint Cycle Packing. Indeed, Bodlaenderr *et al.* first show that Disjoint Factors parameterized by alphabet size $|\Sigma|$ does not admit a polynomial kernel, and then reduce Disjoint Factors to Disjoint Cycle Packing in the sense that a polynomial kernel for Disjoint Cycle Packing would yield a polynomial kernel for Disjoint Factors. Here we go in the other direction - first we obtain a PSAKS for Disjoint Factors parameterized by $|\Sigma|$, and then lift this result to Disjoint Cycle Packing parameterized by solution size.

Lower Bounds for Approximate Kernels. A *path* P in a graph G is a sequence $v_1 v_2, \ldots v_t$ of distinct vertices, such that each pair of consecutive vertices in P are adjacent in G. The *length* of the path P is $t-1$, the number of vertices in P minus one. In Longest Path, the input is a graph G and the objective is to find a path of maximum length. The best approximation algorithm for Longest Path [2] has factor $\mathcal{O}(\frac{n}{\log n})$, and the problem cannot be approximated [39] within a factor $2^{(\log n)^{1-\epsilon}}$ for any $\epsilon > 0$, unless $\mathsf{NP} = \mathsf{DTIME}(2^{\log n^{O(1)}})$. Further, Longest Path is not expected to admit a polynomial kernel. In fact it was one of the first FPT problems for which the existence of a polynomial kernel was ruled out [6]. We show that even within the realm of approximate kernelization, Longest Path remains hard. In particular we show that for any $\alpha \geq 1$, Longest Path does not admit an α-approximate kernel of polynomial size unless $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$.

In order to show the approximate kernelization lower bound for Longest Path, we extend the complexity-theoretic machinery for showing kernelization lower bounds [6,7,25,32,36] to our framework of parameterized optimization problems. In particular we amalgamate the notion of *cross-compositions*, used to show kernelization lower bounds, with *gap-creating reductions*, used to show hardness of approximation bounds, and define *gap creating cross-compositions*. Then, adapting the proofs of Fortnow and Santhanam [32] and Bodlaender *et al.* [7] to our setting, we show that this notion can be used to prove lower bounds on the size of approximate kernels. Once the framework of gap creating cross-compositions is set up, it trivially applies to Longest Path.

After setting up the framework for showing lower bounds for approximate kernelization, we consider the approximate kernelization complexity of two more problems, namely Set Cover and Hitting Set, both parameterized by universe size. In both problems input is a family $\mathcal{S}$ of subsets of a universe U. We use n for the size of the universe U and m for the number of sets in $\mathcal{S}$. A *set cover* is a subfamily $\mathcal{F}$ of $\mathcal{S}$ such that $\bigcup_{S \in \mathcal{F}} S = U$. In the Set Cover problem the objective is to find a set cover $\mathcal{F}$ of minimum size. A *hitting set* is a subset X of U such that every $S \in \mathcal{S}$ has non-empty intersection with X, and in the Hitting Set problem the goal is to find a hitting set of minimum size.

The two problems are dual to each other in the following sense: given $(\mathcal{S}, U)$ we can define the *dual family* $(\mathcal{S}^*, U^*)$ as follows. U^* has one element u_X for every set $X \in \mathcal{S}$, and $\mathcal{S}^*$ has one set S_v for every element $v \in U$. For every $X \in \mathcal{S}$ and $v \in U$ the set $S_v \in \mathcal{S}^*$ contains the element u_X in U^* if and only if $v \in X$. It is well known and easy to see that the dual of the dual of $(\mathcal{S}, U)$ is $(\mathcal{S}, U)$ itself, and that hitting sets of $(\mathcal{S}, U)$ correspond to set covers in $(\mathcal{S}^*, U^*)$ and vice versa. This duality allows us to translate algorithms and lower bounds between Set Cover to Hitting Set. However, this translation *switches the roles of n (the universe size) and m (the number of sets)*. For example, Set Cover is known to admit a factor $(\ln n)$-approximation algorithm [55], and known not to admit a $(c \ln n)$-approximation algorithm for any $c < 1$ unless $\mathsf{P} = \mathsf{NP}$ [47]. The duality translates these results to a $(\ln m)$-approximation algorithm, and a lower bound ruling out $(c \ln m)$-approximation algorithms for any $c < 1$ for Hitting Set. Nelson [48] gave a $O(\sqrt{m})$-approximation algorithm, as well as a lower bound ruling out a polynomial time $O(2^{(\log m)^c})$-approximation for any $c < 1$ for Set Cover, assuming the ETH. The duality translates these results to a $O(\sqrt{n})$-approximation algorithm, as well as a lower bound under ETH ruling out a polynomial time $O(2^{(\log n)^c})$-approximation for any $c < 1$ for Hitting Set. Observe that even though Set Cover and Hitting Set are dual to each other they behave very differently with respect to approximation algorithms that measure the quality of the approximation in terms of the universe size n.

For *kernelization* parameterized by universe size n, the two problems behave in a more similar fashion. Both problems admit kernels of size $O(2^n)$, and both problems have been shown not to admit kernels of size $n^{O(1)}$ [22] unless $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$. However, the two lower bound proofs are quite different, and the two lower bounds do not follow from one another using the duality.

For Set Cover parameterized by n, we deploy the framework of gap creating cross-compositions to show that the problem does not admit an α-approximate kernel of size $n^{O(1)}$ for any constant α. This can be seen as a significant strengthening of the lower bound of Dom et al. [22]. While the gap creating cross-composition for Longest Path is very simple, the gap creating cross-composition for Set Cover is quite delicate, and relies both on a probabilistic construction and a de-randomization of this construction using co-non-determinism.

Our lower bound for Set Cover parameterized by universe size n translates to a lower bound for Hitting Set parameterized by the number m of sets, but says nothing about Hitting Set parameterized by n. We prove that for every $c < 1$, even a $O(2^{(\log n)^c})$-approximate kernel of size $n^{O(1)}$ for Hitting Set would imply a $O(2^{(\log n)^{c'}})$-approximation algorithm for Hitting Set for some $c' < 1$. By the result of Nelson [48] this would in turn imply that the ETH is false. Hence, Hitting Set does not admit a $O(2^{(\log n)^c})$-approximate kernel of size $n^{O(1)}$ assuming the ETH.

We remark that the lower bounds proved using the framework of gap creating cross compositions, and in particular the lower bounds for Longest Path and Set Cover, also rule out approximate *compressions* to any other parameterized optimization problems. On the other hand, our lower bound for Hitting Set only rules out approximate *kernels*. As a consequence the lower bounds for Longest Path and Set Cover have more potential as starting points for reductions showing that even further problems do not admit approximate kernels.

Summary. In this paper we set up a new framework for the study of lossy pre-processing algorithms, and demonstrate that the framework is natural, versatile and robust. For several well studied problems, including Steiner Tree, Connected Vertex Cover and Cycle Packing we show that a "barely lossy" kernelization can get dramatically better upper bounds on the kernel size than what is achievable with normal kernelization. We extend the machinery for showing kernelization lower bounds to the setting of approximate kernels, and use these new methods to prove lower bounds on the size of approximate kernels for Longest Path parameterized by the objective function value, and Set Cover and Hitting Set parameterized by universe size. Especially Set Cover parameterized by universe size has been a useful starting point for reductions showing lower bounds for traditional kernelization [10, 18, 22, 31, 34, 46, 56]. We are therefore confident that our work lays a solid foundation for future work on approximate kernelization. Nevertheless, *this paper raises many more questions than it answers. It is our hope that our work will open the door for a systematic investigation of lossy pre-processing.*

Organization of the Paper. In Section 2 we set up notations. In section 3 we set up the necessary definitions to formally define and discuss approximate kernels, and relate the new notions to well known definitions from approximation algorithms and parameterized complexity. In section 4 we give a PSAKS for Connected Vertex Cover. In section 5 we give PSAKSes for Disjoint Factors and Disjoint Cycle Packing. In section 6 we show how (parts of) existing approximation algorithms for Partial Vertex Cover, Steiner Tree and Optimal Linear Arrangement can be re-interpreted as approximate kernels for these problems. In section 7 we set up a framework for proving lower bounds on the size of α-approximate kernels for a parameterized optimization problem. In sections 8 and 9 we deploy the new framework to prove lower bounds for approximate kernelization of Longest Path and Set Cover. In section 10 we give a lower bound for the approximate kernelization of Hitting Set by showing that a "too good" approximate kernel would lead to a "too good" approximation algorithm. We conclude with an extensive list of open problems in section 11.

A Guide to the Paper. In order to read any of the sections on concrete problems, as well as our lower bound machinery (sections 4-10) one needs more formal definitions of approximate kernelization and related concepts than what is given in the introduction. These definitions are given in section 3.

We have provided informal versions of the most important definitions in subsection 3.1. It should be possible to read subsection 3.1 and then proceed directly to the technical sections (4-10), only using the rest of section 3 occasionally as a reference. Especially the positive results of sections 4-6 should be accessible in this way. However, a reader interested in how approximate kernelization fits within a bigger picture containing approximation algorithms and kernelization should delve deeper into Section 3.

All of the approximate kernelization results in sections 4-6 may be read independently of each other, except that the kernels for Disjoint Factors and Disjoint Cycle Packing in section 5 are related. The approximate kernel for Connected Vertex Cover given in section 4 gives a simple first example of an approximate kernel, in fact a PSAKS. The approximate kernels for Disjoint Factors and Disjoint Cycle Packing given in section 5 are the most technically interesting positive results in the paper.

Section 7 sets up the methodology for proving lower bounds on approximate kernelization, this methodology is encapsulated in Theorem 8. The statement of Theorem 8 together with the definitions of all objects in the statement are necessary to read the two lower bound sections (8 and 9) that apply this theorem. The lower bound for Longest Path in section 8 is a direct application of Theorem 8. The lower bound for Set Cover in section 9 is the most technically interesting lower bound in the paper. The lower bound for Hitting Set in Section 10 does not rely on Theorem 8, and may be read immediately after subsection 3.1

2 Preliminaries

We use $\mathbb{N}$ to denote the set of natural numbers. For a graph G we use $V(G)$ and $E(G)$, to denote the vertex and edge sets of the graph G respectively. We use standard terminology from the book of Diestel [19] for those graph-related terms which are not explicitly defined here. For a vertex v in $V(G)$, we use $d_G(v)$ to denote the degree of v, i.e the number of edges incident on v, in the (multi) graph G. For a vertex subset $S \subseteq V(G)$, we use $G[S]$ and $G-S$ to be the graphs induced on S and $V(G)\setminus S$ respectively. For a graph G and an induced subgraph G' of G, we use $G-G'$ to denote the graph $G-V(G')$. For a vertex subset $S \subseteq V(G)$, we use $N_G(S)$ and $N_G[S]$ to denote the open neighbourhood and closed neighbourhood of S in G. That is, $N_G(S) = \{v \mid (u,v) \in E(G), u \in S\} \setminus S$ and $N_G[S] = N_G(S) \cup S$. For a graph G and an edge $e \in E(G)$, we use G/e to denote the graph obtained by contracting e in G. If P is a path from a vertex u to a vertex v in graph G then we say that u, v are the *end* vertices of the path P and P is a (u,v)-path. For a path P, we use $V(P)$ to denote the set of vertices in the path P and the length of P is denoted by $|P|$ (i.e, $|P| = |V(P)| - 1$). For a cycle C, we use $V(C)$ to denote the set of vertices in the cycle C and length of C, denoted by $|C|$, is $|V(C)|$. Let $P_1 = x_1x_2 \ldots x_r$ and $P_2 = y_1y_2 \ldots y_s$ be two paths in a graph G, $V(P_1) \cap V(P_2) = \emptyset$ and $x_ry_1 \in E(G)$, then we use P_1P_2 to denote the path $x_1x_2 \ldots x_ry_1 \ldots y_s$. We say that $P = x_1x_2 \ldots x_r$ is an induced path in a multigraph G, if $G[V(P)]$ is same as the simple graph P. We say that a path P is a non trivial path if $|V(P)| \geq 2$. For a path/cycle Q we use $N_G(Q)$ and $N_G[Q]$ to denote the set $N_G(V(Q))$ and $N_G[V(Q)]$ respectively. For a set of paths/cycles $\mathcal{Q}$, we use $|\mathcal{Q}|$ and $V(\mathcal{Q})$ to denote the number of paths/cycles in $\mathcal{Q}$ and the set $\bigcup_{Q \in \mathcal{Q}} V(Q)$ respectively. The chromatic number of a graph G is denoted by $\chi(G)$. An undirected graph G is called an interval graph, if it is formed from set $\mathcal{I}$ of intervals by creating one vertex v_I for each interval $I \in \mathcal{I}$ and adding edge between two vertices v_I and $v_{I'}$, if $I \cap I' \neq \emptyset$. An interval representation of an interval graph G is a set of intervals from which G can be formed as described above. The following facts are useful in later sections.

Fact 1. *For any positive reals x, y, p and q,* $\min\left(\frac{x}{p}, \frac{y}{q}\right) \leq \frac{x+y}{p+q} \leq \max\left(\frac{x}{p}, \frac{y}{q}\right)$

Fact 2. *For any $y \leq \frac{1}{2}$, $(1-y) \geq \left(\frac{1}{4}\right)^y$.*

3 Setting up the Framework

For the precise definition of approximate kernels, all its nuances, and how this new notion relates to approximation algorithms, FPT algorithms, FPT-approximation algorithms and kernelization, one should read Subsection 3.2. For the benefit of readers eager to skip ahead to the concrete results of the paper, we include in Subsection 3.1 a "definition" of α-approximate kernelization that should be sufficient for reading the rest of the paper and understanding most of the arguments.

3.1 Quick and Dirty "Definition" of Approximate Kernelization

Recall that we work with *parameterized problems.* That is, every instance comes with a parameter k. Often k is "the quality of the solution we are looking for". For example, does G have a connected vertex cover of size at most k? Does G have at least k pairwise vertex disjoint cycles? When we move to optimization problems, we change the above two questions to: Can you find a connected vertex cover of size at most k in G? If yes, what is the smallest one you can find? Or, can you find at least k pairwise vertex disjoint cycles? If no, what is the largest collection of pairwise vertex disjoint cycles you can find? Note here the difference in how minimization and maximization problems are handled. For minimization problems, a bigger objective function value is undesirable, and k is an "upper bound on the 'badness' of the solution". That is, solutions worse than k are so bad we do not care precisely how bad they are. For maximization problems, a bigger objective function value is desirable, and k is an "upper bound on how good the solution has to be before one is fully satisfied". That is, solutions better than k are so good that we do not care precisely how good they are.

In many cases the parameter k does not directly relate to the quality of the solution we are looking for. Consider for example, the following problem. Given a graph G and a set Q of k terminals, find a smallest possible Steiner tree T in G that contains all the terminals. In such cases, k is called a *structural parameter*, because k being small restricts the structure of the input instance. In this example, the structure happens to be the fact that the number of terminals is 'small'.

Let $\alpha \geq 1$ be a real number. We now give an informal definition of α-approximate kernels. The kernelization algorithm should take an instance I with parameter k, run in polynomial time, and produce a new instance I' with parameter k'. Both k' and the size of I' should be bounded in terms of just the parameter k. That is, there should exist a function $g(k)$ such that $|I'| \leq g(k)$ and $k' \leq g(k)$. This function $g(k)$ is the *size* of the kernel. Now, a solution s' to the instance I' should be useful for finding a good solution s to the instance I. What precisely this means depends on whether k is a structural parameter or the "quality of the solution we are looking for", and whether we are working with a maximization problem or a minimization problem.

- If we are working with a structural parameter k then we require the following from α-approximate kernels: For every $c \geq 1$, a c-approximate solution s' to I' can be transformed in polynomial time into a $(c \cdot \alpha)$-approximate solution to I.

- If we are working with a minimization problem, and k is the quality of the solution we are looking for, then k is an "upper bound on the badness of the solution". In this case we require the following from α-approximate kernels: For every $c \geq 1$, a c-approximate solution s' to I' can be transformed in polynomial time into a $(c \cdot \alpha)$-approximate solution s to I. However, if the quality of s' is "worse than" k', or $(c \cdot \alpha) \cdot OPT(I) > k$, the algorithm that transforms S' into S is allowed to fail. Here $OPT(I)$ is the value of the optimum solution of the instance I.

  The solution lifting algorithm is allowed to fail precisely if the solution S' given to it is "too bad" for the instance I', or if the approximation guarantee of being a factor of $c \cdot \alpha$ away from the optimum for I allows it to output a solution that is "too bad" for I anyway.

- If we are working with a maximization problem, and k is the quality of the solution we are looking for, then k is an "upper bound on how good the solution has to be before one is fully satisfied". In this case we require the following from α-approximate kernels: For every $c \geq 1$, if s' is a c-approximate solution s' to I' *or* the quality of s' is at least k'/c, then s' can be transformed in polynomial time into a $(c \cdot \alpha)$-approximate solution s to I. However, if $OPT(I) > k$ then instead of being a $(c \cdot \alpha)$-approximate solution s to I, the output solution s can be any solution of quality at least $k/(c \cdot \alpha)$.

  In particular, if $OPT(I') > k'$ then the optimal solution to I' is considered "good enough", and the approximation ratio c of the solution s' to I' is computed as "distance from being good enough", i.e as k'/s'. Further, if $OPT(I) > k$ then we think of the optimal solution to I as "good enough", and measure the approximation ratio of s in terms of "distance from being good enough", i.e as k/s.

We encourage the reader to instantiate the above definitions with $c \in \{1, 2\}$ and $\alpha \in \{1, 2\}$. That is, what happens to optimal and 2-approximate solutions to the reduced instance when the approximate kernel incurs no loss ($\alpha = 1$)? What happens to optimal and 2-approximate solutions to the reduced instance when the approximate kernel incurs a factor 2 loss (i.e $\alpha = 2$)?

Typically we are interested in α-approximate kernels of *polynomial size*, that is kernels where the size function $g(k)$ is upper bounded by $k^{O(1)}$. Of course the goal is to design α-approximate kernels of smallest possible size, with smallest possible α. Sometimes we are able to obtain a $(1+\epsilon)$-approximate kernel of polynomial size for every $\epsilon > 0$. Here the exponent and the constants of the polynomial may depend on ϵ. We call such a kernel a Polynomial Size Approximate Kernelization Scheme, and abbreviate it as PSAKS. If only the constants of the polynomial $g(k)$ and not the exponent depend on ϵ, we say that the PSAKS is *efficient.* All of the positive results achieved in this paper are PSAKSes, but not all are efficient.

3.2 Approximate Kernelization, The Real Deal.

We will be dealing with approximation algorithms and solutions that are not necessarily optimal, but at the same time relatively "close" to being optimal. To properly discuss these concepts they have to be formally defined. Our starting point is a parameterized analogue of the notion of an *optimization problem* from the theory of approximation algorithms.

Definition 3.1. *A parameterized optimization (minimization or maximization) problem* Π *is a computable function*

$$\Pi \ : \ \Sigma^* \times \mathbb{N} \times \Sigma^* \rightarrow \mathbb{R} \cup \{\pm\infty\}.$$

The *instances* of a parameterized optimization problem Π are pairs $(I, k) \in \Sigma^* \times \mathbb{N}$, and a *solution* to (I, k) is simply a string $s \in \Sigma^*$, such that $|s| \leq |I| + k$. The *value* of the solution s is $\Pi(I, k, s)$. Just as for "classical" optimization problems the instances of Π are given as input, and the algorithmic task is to find a solution with the best possible value, where best means minimum for minimization problems and maximum for maximization problems.

Definition 3.2. *For a parameterized minimization problem* Π*, the* optimum value *of an instance* $(I, k) \in \Sigma^* \times \mathbb{N}$ *is*

$$OPT_\Pi(I,k) = \min_{\substack{s \in \Sigma^* \\ |s| \leq |I|+k}} \Pi(I,k,s).$$

For a parameterized maximization problem Π*, the optimum value of* (I, k) *is*

$$OPT_\Pi(I,k) = \max_{\substack{s \in \Sigma^* \\ |s| \leq |I|+k}} \Pi(I,k,s).$$

For an instance (I, k) *of a parameterized optimization problem* Π*, an* optimal solution *is a solution* s *such that* $\Pi(I, k, s) = OPT_\Pi(I, k)$*.*

When the problem Π is clear from context we will often drop the subscript and refer to $OPT_\Pi(I, k)$ as $OPT(I, k)$. Observe that in the definition of $OPT_\Pi(I, k)$ the set of solutions over which we are minimizing/maximizing Π is finite, therefore the minimum or maximum is well defined. We remark that the function Π in Definition 3.1 depends *both* on I and on k. Thus it is possible to define parameterized problems such that an optimal solution s for (I, k) is not necessarily optimal for (I, k').

For an instance (I, k) the *size* of the instance is $|I| + k$ while the integer k is referred to as the *parameter* of the instance. Parameterized Complexity deals with measuring the running time of algorithms in terms of both the input size and the parameter. In Parameter Complexity a problem is *fixed parameter tractable* if input instances of size n with parameter k can be "solved" in time $f(k)n^{\mathcal{O}(1)}$ for a computable function f. For decision problems "solving" an instance means to determine whether the input instance is a "yes" or a "no" instance to the problem. Next we define what it means to "solve" an instance of a parameterized optimization problem, and define fixed parameter tractability for parameterized optimization problems.

Definition 3.3. *Let Π be a parameterized optimization problem. An* algorithm for Π *is an algorithm that given as input an instance (I, k), outputs a solution s and halts. The algorithm* solves Π *if, for every instance (I, k) the solution s output by the algorithm is optimal for (I, k). We say that a parameterized optimization problem Π is* decidable *if there exists an algorithm that solves Π.*

Definition 3.4. *A parameterized optimization problem Π is* fixed parameter tractable *(FPT) if there is an algorithm that solves Π, such that the running time of the algorithm on instances of size n with parameter k is upper bounded by $f(k)n^{\mathcal{O}(1)}$ for a computable function f.*

We remark that Definition 3.3 differs from the usual formalization of what it means to "solve" a decision problem. Solving a decision problem amounts to always returning "yes" on "yes"-instances and "no" on "no"-instances. For parameterized optimization problems the algorithm has to produce an optimal solution. This is analogous to the definition of optimization problems most commonly used in approximation algorithms.

We remark that we could have built the framework of approximate kernelization on the existing definitions of parameterized optimization problems used in parameterized approximation algorithms [45], indeed the difference between our definitions of parameterized optimization problems and those currently used in parameterized approximation algorithms are mostly notational.

Parameterizations by the Value of the Solution. At this point it is useful to consider a few concrete examples, and to discuss the relationship between parameterized optimization problems and decision variants of the same problem. For a concrete example, consider the Vertex Cover problem. Here the input is a graph G, and the task is to find a smallest possible *vertex cover* of G: a subset $S \subseteq V(G)$ is a *vertex cover* if every edge of G has at least one endpoint in S. This is quite clearly an optimization problem, the feasible solutions are the vertex covers of G and the objective function is the size of S.

In the most common formalization of the Vertex Cover problem as a *decision problem* parameterized by the solution size, the input instance G comes with a parameter k and the instance (G, k) is a "yes" instance if G has a vertex cover of size at most k. Thus, the parameterized decision problem "does not care" whether G has a vertex cover of size even smaller than k, the only thing that matters is whether a solution of size at most k is present.

To formalize Vertex Cover as a parameterized optimization problem, we need to determine for every instance (G, k) which value to assign to potential solutions $S \subseteq V(G)$. We can encode the set of feasible solutions by giving finite values for vertex covers of G and ∞ for all other sets. We want to distinguish between graphs that do have vertex covers of size at most k and the ones that do not. At the same time, we want the computational problem of solving the instance (G, k) to become easier as k decreases. A way to achieve this is to assign $|S|$ to all vertex covers S of G of size at most k, and $k + 1$ for all other vertex covers. Thus, one can formalize the Vertex Cover problem as a parameterized optimization problem as follows.

$$VC(G, k, S) = \begin{cases} \infty & \text{if S is not a vertex cover of G,} \\ \min(|S|, k + 1) & \text{otherwise.} \end{cases}$$

Note that this formulation of Vertex Cover "cares" about solutions of size less than k. One can think of k as a threshold: for solutions of size at most k we care about what their size is, while all solutions of size larger than k are equally bad in our eyes, and are assigned value $k+1$.

Clearly any FPT algorithm that solves the parameterized optimization version of Vertex Cover also solves the (parameterized) decision variant. Using standard self-reducibility techniques [53] one can make an FPT algorithm for the decision variant solve the optimization variant as well.

We have seen how a minimization problem can be formalized as a parameterized optimization problem parameterized by the value of the optimum. Next we give an example for how to do this for maximization problems. In the Cycle Packing problem we are given as input a graph G, and the task is to find a largest possible collection $\mathcal{C}$ of pairwise vertex disjoint cycles. Here a *collection of vertex disjoint cycles* is a collection $\mathcal{C}$ of vertex subsets of G such that for every $C \in \mathcal{C}$, $G[C]$ contains a cycle and for every $C, C' \in \mathcal{C}$ we have $V(C) \cap V(C') = \emptyset$. We will often refer to a collection of vertex disjoint cycles as a *cycle packing.*

We can formalize the Cycle Packing problem as a parameterized optimization problem parameterized by the value of the optimum in a manner similar to what we did for Vertex Cover. In particular, if $\mathcal{C}$ is a cycle packing, then we assign it value $|\mathcal{C}|$ if $|\mathcal{C}| \leq k$ and value $k+1$ otherwise. If $|\mathcal{C}|$ is not a cycle packing, we give it value $-\infty$.

$$CP(G, k, \mathcal{C}) = \begin{cases} -\infty & \text{if } \mathcal{C} \text{ is not a cycle packing,} \\ \min(|\mathcal{C}|, k+1) & \text{otherwise.} \end{cases}$$

Thus, the only (formal) difference between the formalization of parameterized minimization and maximization problems parameterized by the value of the optimum is how infeasible solutions are treated. For minimization problems infeasible solutions get value ∞, while for maximization problems they get value $-\infty$. However, there is also a "philosophical" difference between the formalization of minimization and maximization problems. For minimization problems we do not distinguish between feasible solutions that are "too bad"; solutions of size more than k are all given the same value. On the other hand, for maximization problems all solutions that are "good enough", i.e. of size at least $k+1$, are considered equal.

Observe that the "capping" of the objective function at $k+1$ *does not make sense for approximation algorithms* if one insists on k being the (un-parameterized) optimum of the instance I. The parameterization discussed above is *by the value of the solution that we want our algorithms to output*, not by the unknown optimum. We will discuss this topic in more detail in the paragraph titled "**Capping the objective function at $k+1$**", after the notion of approximate kernelization has been formally defined.

Structrural Parameterizations. We now give an example that demonstrates that the notion of parameterized optimization problems is robust enough to capture not only parameterizations by the value of the optimum, but also parameterizations by structural properties of the instance that may or may not be connected to the value of the best solution. In the Optimal Linear Arrangement problem we are given as input a graph G, and the task is to find a bijection $\sigma : V(G) \to \{1, \ldots, n\}$ such that $\sum_{uv \in E(G)} |\sigma(u) - \sigma(v)|$ is minimized. A bijection $\sigma : V(G) \to \{1, \ldots, n\}$ is called a *linear layout*, and $\sum_{uv \in E(G)} |\sigma(u) - \sigma(v)|$ is denoted by $val(\sigma, G)$ and is called the *value* of the layout σ.

We will consider the Optimal Linear Arrangement problem for graphs that have a relatively small vertex cover. This can be formalized as a parameterized optimization problem as follows:

$$OLA((G, S), k, \sigma) = \begin{cases} -\infty & \text{if S is not vertex cover of } G \text{ of size at most } k, \\ \infty & \text{if } \sigma \text{ is not a linear layout,} \\ val(\sigma, G) & \text{otherwise.} \end{cases}$$

In the definition above the first case takes precendence over the second: if S is not vertex cover of G of size at most k and σ is not a linear layout, $OLA((G, S), k, \sigma)$ returns $-\infty$. This ensures that malformed input instances do not need to be handled.

Note that the input instances to the parameterized optimization problem described above are pairs $((G, S), k)$ where G is a graph, S is a vertex cover of G of size at most k and k is the parameter. This definition allows algorithms for Optimal Linear Arrangement parameterized by vertex cover to assume that the vertex cover S is given as input.

Kernelization of Parameterized Optimization Problems. The notion of a kernel (or kernelization algorithm) is a mathematical model for polynomial time pre-processing for decision problems. We will now define the corresponding notion for parameterized optimization problems. To that end we first need to define a polynomial time pre-processing algorithm.

Definition 3.5. *A* **polynomial time pre-processing algorithm** $\mathcal{A}$ *for a parameterized optimization problem* Π *is a pair of polynomial time algorithms. The first one is called the* **reduction algorithm***, and computes a map* $\mathcal{R}_{\mathcal{A}} : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$*. Given as input an instance* (I, k) *of* Π *the reduction algorithm outputs another instance* $(I', k') = \mathcal{R}_{\mathcal{A}}(I, k)$*.*

The second algorithm is called the **solution lifting algorithm***. This algorithm takes as input an instance* $(I, k) \in \Sigma^* \times \mathbb{N}$ *of* Π*, the output instance* (I', k') *of the reduction algorithm, and a solution* s' *to the instance* (I', k')*. The solution lifting algorithm works in time polynomial in* $|I|$*,*k*,*$|I'|$*,*k' *and* s'*, and outputs a solution* s *to* (I, k)*. Finally, if* s' *is an optimal solution to* (I', k') *then* s *is an optimal solution to* (I, k)*.*

Observe that the solution lifting algorithm could contain the reduction algorithm as a subroutine. Thus, on input (I, k, I', k', s') the solution lifting algorithm could start by running the reduction algorithm (I, k) and produce a transcript of *how* the reduction algorithm obtains (I', k') from (I, k). Hence, when designing the solution lifting algorithm we may assume without loss of generality that such a transcript is given as input. For the same reason, it is not really necessary to include (I', k') as input to the solution lifting algorithm. However, to avoid starting every description of a solution lifting algorithm with "we compute the instance (I', k') from (I, k)", we include (I', k') as input. The notion of polynomial time pre-processing algorithms could be extended to *randomized* polynomial time pre-processing algorithms, by allowing both the reduction algorithm and the solution lifting algorithm to draw random bits, and *fail* with a small probability. With such an extension it matters whether the solution lifting algorithm has access to the random bits drawn by the reduction algorithm, because these bits might be required to re-construct the transcript of how the reduction algorithm obtained (I', k') from (I, k). If the random bits of the reduction algorithm are provided to the solution lifting algorithm, the discussion above applies.

A kernelization algorithm is a polynomial time pre-processing algorithm for which we can prove an upper bound on the size of the output instances in terms of the parameter of the instance to be preprocessed. Thus, the *size* of a polynomial time pre-processing algorithm $\mathcal{A}$ is a function $\text{size}_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ defined as follows.

$$\text{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' : (I', k') = \mathcal{R}_{\mathcal{A}}(I, k), I \in \Sigma^*\}.$$

In other words, we look at all possible instances of Π with a fixed parameter k, and measure the supremum of the sizes of the output of $\mathcal{R}_{\mathcal{A}}$ on these instances. At this point, recall that the *size* of an instance (I, k) is defined as $|I| + k$. Note that this supremum may be infinite; this happens when we do not have any bound on the size of $\mathcal{R}_{\mathcal{A}}(I, k)$ in terms of the input parameter k only. Kernelization algorithms are exactly these polynomial time preprocessing algorithms whose output size is finite and bounded by a computable function of the parameter.

Definition 3.6. *A* **kernelization** *(or* kernel*) for a parameterized optimization problem* Π *is a polynomial time pre-processing algorithm* $\mathcal{A}$ *such that size*$_{\mathcal{A}}$ *is upper bounded by a computable function* $g : \mathbb{N} \to \mathbb{N}$*.*

If the function g in Definition 3.6 is a polynomial, we say that Π admits a *polynomial kernel*. Similarly, if g is a linear, quadratic or cubic function of k we say that Π admits a linear, quadratic, or cubic kernel, respectively.

One of the basic theorems in Parameterized Complexity is that a decidable parameterized decision problem admits a kernel if and only if it is fixed parameter tractable. We now show that this result also holds for parameterized optimization problems. We say that a parameterized optimization problem Π is *decidable* if there exists an algorithm that solves Π, where the definition of "solves" is given in Definition 3.3.

Proposition 3.1. *A decidable parameterized optimization problem Π is FPT if and only if it admits a kernel.*

Proof. The backwards direction is trivial; on any instance (I, k) one may first run the reduction algorithm to obtain a new instance (I', k') of size bounded by a function $g(k)$. Since the instance (I', k') has bounded size and Π is decidable one can find an optimal solution s' to (I', k') in time upper bounded by a function $g'(k)$. Finally one can use the solution lifting algorithm to obtain an optimal solution s to (I, k).

For the forward direction we need to show that if a parameterized optimization problem Π is FPT then it admits a kernel. Suppose there is an algorithm that solves instances Π of size n with parameter k in time $f(k)n^c$. On input (I, k) the reduction algorithm runs the FPT algorithm for n^{c+1} steps. If the FPT algorithm terminates after at most n^{c+1} steps, the reduction algorithm outputs an instance (I', k') of constant size. The instance (I', k') is hard-coded in the reduction algorithm and does not depend on the input instance (I, k). Thus $|I'| + k'$ is upper bounded by a constant. If the FPT algorithm does not terminate after n^{c+1} steps the reduction algorithm halts and outputs the instance (I, k). Note that in this case $f(k)n^c > n^{c+1}$, which implies that $f(k) > |I|$. Hence the size of the output instance is upper bounded by a function of k.

We now describe the solution lifting algorithm. If the reduction algorithm output (I, k) then the solution lifting algorithm just returns the same solution that it gets as input. If the reduction algorithm output (I', k') this means that the FPT algorithm terminated in polynomial time, which means that the solution lifting algorithm can use the FPT algorithm to output an optimal solution to (I, k) in polynomial time, regardless of the solution to (I', k') it gets as input. This concludes the proof. $\square$

Parameterized Approximation and Approximate Kernelization. For some parameterized optimization problems we are unable to obtain FPT algorithms, and we are also unable to find satisfactory polynomial time approximation algorithms. In this case one might aim for FPT-approximation algorithms, algorithms that run in time $f(k)n^c$ and provide good approximate solutions to the instance.

Definition 3.7. *Let $\alpha \geq 1$ be constant. A fixed parameter tractable α-approximation algorithm for a parameterized optimization problem Π is an algorithm that takes as input an instance (I, k), runs in time $f(k)|I|^{\mathcal{O}(1)}$, and outputs a solution s such that $\Pi(I, k, s) \leq \alpha \cdot OPT(I, k)$ if Π is a minimization problem, and $\alpha \cdot \Pi(I, k, s) \geq OPT(I, k)$ if Π is a maximization problem.*

Note that Definition 3.7 only defines constant factor FPT-approximation algorithms. The definition can in a natural way be extended to approximation algorithms whose approximation ratio depends on the parameter k, on the instance I, or on both.

We are now ready to define one of the key new concepts of the paper - the concept of an α-approximate kernel. We defined kernels by first defining polynomial time pre-processing algorithms (Definition 3.5) and then adding size constraints on the output (Definition 3.6). In a similar manner we will first define α-approximate polynomial time pre-processing algorithms, and then define α-approximate kernels by adding size constraints on the output of the pre-processing algorithm.

Definition 3.8. *Let $\alpha \geq 1$ be a real number and Π be a parameterized optimization problem. An* **α-approximate polynomial time pre-processing algorithm** *$\mathcal{A}$ for Π is a pair of polynomial time algorithms. The first one is called the* **reduction algorithm***, and computes a map $\mathcal{R}_{\mathcal{A}} : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$. Given as input an instance (I, k) of Π the reduction algorithm outputs another instance $(I', k') = \mathcal{R}_{\mathcal{A}}(I, k)$.*

The second algorithm is called the **solution lifting algorithm***. This algorithm takes as input an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ of Π, the output instance (I', k') of the reduction algorithm, and a solution s' to the instance (I', k'). The solution lifting algorithm works in time polynomial in $|I|$,k,$|I'|$,k' and s', and outputs a solution s to (I, k). If Π is a minimization problem then*

$$\frac{\Pi(I, k, s)}{OPT(I, k)} \leq \alpha \cdot \frac{\Pi(I', k', s')}{OPT(I', k')}.$$

If Π is a maximization problem then

$$\frac{\Pi(I, k, s)}{OPT(I, k)} \cdot \alpha \geq \frac{\Pi(I', k', s')}{OPT(I', k')}.$$

Definition 3.8 only defines constant factor approximate polynomial time pre-processing algorithms. The definition can in a natural way be extended approximation ratios that depend on the parameter k, on the instance I, or on both. Additionally, the discussion following Definition 3.5 also applies here. In particular we may assume that the solution lifting algorithm also gets as input a transcript of how the reduction algorithm obtains (I', k') from (I, k). The size of an α-approximate polynomial time pre-processing algorithm is defined in exactly the same way as the size of a polynomial time pre-processing algorithm (from Definition 3.5).

Definition 3.9. *An* **α-approximate kernelization** *(or α-*approximate kernel*) for a parameterized optimization problem Π, and real $\alpha \geq 1$, is an α-approximate polynomial time pre-processing algorithm $\mathcal{A}$ such that $size_{\mathcal{A}}$ is upper bounded by a computable function $g : \mathbb{N} \to \mathbb{N}$.*

Just as for regular kernels, if the function g in Definition 3.9 is a polynomial, we say that Π admits an α-approximate polynomial kernel. If g is a linear, quadratic or cubic function, then Π admits a linear, quadratic or cubic α-approximate kernel, respectively.

Proposition 3.1 establishes that a parameterized optimization problem Π admits a kernel if and only if it is FPT. Next we establish a similar equivalence between FPT-approximation algorithms and approximate kernelization.

Proposition 3.2. *For every $\alpha \geq 1$ and decidable parameterized optimization problem Π, Π admits a fixed parameter tractable α-approximation algorithm if and only if Π has an α-approximate kernel.*

The proof of Proposition 3.2 is identical to the proof of Proposition 3.1, but with the FPT algorithm replaced by the fixed parameter tractable α-approximation algorithm, and the kernel replaced with the α-approximate kernel. On an intuitive level, it should be easier to compress an instance than it is to solve it. For α-approximate kernelization this intuition can be formalized.

Theorem 1. *For every $\alpha \geq 1$ and decidable parameterized optimization problem Π, Π admits a polynomial time α-approximation algorithm if and only if Π has an α-approximate kernel of constant size.*

The proof of Theorem 1 is simple; if there is an α-approximate kernel of constant size one can brute force the reduced instance and lift the optimal solution of the reduced instance to an α-approximate solution to the original. On the other hand, if there is a factor α approximation algorithm, the reduction algorithm can just output any instance of constant size. Then, the solution lifting algorithm can just directly compute an α-approximate solution to the original instance using the approximation algorithm.

We remark that Proposition 3.2 and Theorem 1 also applies to approximation algorithms and approximate kernels with super-constant approximation ratio. We also remark that with our definition of α-approximate kernelization, by setting $\alpha = 1$ we get essentially get back the notion of kernel for the same problem. The difference arises naturally from the different goals of decision and optimization problems. In decision problems we aim to correctly classify the instance as a "yes" or a "no" instance. In an optimization problem we just want as good a solution as possible for the instance at hand. In traditional kernelization, a yes/no answer to the reduced instance translates without change to the original instance. With our definition of approximate kernels, a sufficiently good solution (that is, a witness of a yes answer) will always yield a witness of a yes answer to the original instance. However, the *failure* to produce a sufficiently good solution to the reduced instance does not stop us from *succeeding* at producing a sufficiently good solution for the original one. From the perspective of optimization problems, such an outcome is a win.

Capping the objective function at $k + 1$. We now return to the topic of parameterizing optimization problems by the value of the solution, and discuss the relationship between (approximate) kernels for such parameterized optimization problems and (traditional) kernels for the parameterized decision version of the optimization problem.

Consider a traditional optimization problem, say Vertex Cover. Here, the input is a graph G, and the goal is to find a vertex cover S of G of minimum possible size. When *parameterizing* Vertex Cover by the objective function value we need to provide a parameter k such that solving the problem on the same graph G becomes progressively easier as k decreases. In parameterized complexity this is achieved by considering the corresponding *parameterized decision* problem where we are given G and k and asked whether there exists a vertex cover of size at most k. Here k is the parameter. If we also required an algorithm for Vertex Cover to produce a solution, then the above parameterization can be interpreted as follows. Given G and k, output a vertex cover of size at most k or fail (that is, return that the algorithm could not find a vertex cover of size at most k.) If there exists a vertex cover of size at most k then the algorithm is not allowed to fail.

A c-approximation algorithm for the Vertex Cover problem is an algorithm that given G, outputs a solution S of size no more than c times the size of the smallest vertex cover of G. So, how do approximation and parameterization mix? For $c \geq 1$, there are *two* natural ways to define a parameterized c-approximation algorithm for Vertex Cover.

(a) Given G and k, output a vertex cover of size at most k or fail (that is, return that the algorithm could not find a vertex cover of size at most k.) If there exists a vertex cover of size at most k/c then the algorithm is not allowed to fail.

(b) Given G and k, output a vertex cover of size at most ck or fail (that is, return that the algorithm could not find a vertex cover of size at most ck.) If there exists a vertex cover of size at most k then the algorithm is not allowed to fail.

Note that if we required the approximation algorithm to run in *polynomial time*, then both definitions above would yield exactly the definition of polynomial time c-approximation algorithms, by a linear search or binary search for the appropriate value of k. In the parameterized setting the running time depends on k, and the two formalizations are different, but nevertheless equivalent up to a factor c in the value of k. That is $f(k) \cdot n^{\mathcal{O}(1)}$ time algorithms and $g(k)$ size kernels for parameterization (b) translate to $f(ck) \cdot n^{\mathcal{O}(1)}$ time algorithms and $g(ck)$ kernels for parameterization (a) and vice versa.

By defining the parameterized optimization problem for Vertex Cover in such a way that the objective function depends on the parameter k, one can achieve either one of the two discussed formulations. By defining $VC(G, k, S) = \min\{|S|, k+1\}$ for vertex covers S we obtain formulation (a). By defining $VC(G, k, S) = \min\{|S|, \lceil ck \rceil + 1\}$ for vertex covers S we obtain formulation (b). It is more meaningful to define the computational problem *independently of the (approximation factor of) algorithms for the problem.* For this reason we stick to formulation (a) in this paper.

Reduction Rules and Strict α-Approximate Kernels. Kernelization algorithms in the literature [15, 23] are commonly described as a set of *reduction rules*. Here we discuss reduction rules in the context of parameterized optimization problems. A reduction rule is simply a polynomial time pre-processing algorithm, see Definition 3.5. The reduction rule *applies* if the output instance of the reduction algorithm is not the same as the input instance. Most kernelization algorithms consist of a set of reduction rules. In every step the algorithm checks whether any of the reduction rules apply. If a reduction rule applies, the kernelization algorithm runs the reduction algorithm on the instance and proceeds by working with the new instance. This process is repeated until the instance is *reduced*, i.e. none of the reduction rules apply. To prove that this is indeed a kernel (as defined in Definition 3.6) one proves an upper bound on the size of any reduced instance.

In order to be able to make kernelization algorithms as described above, it is important that reduction rules can be *chained*. That is, suppose that we have an instance (I, k) and run a pre-processing algorithm on it to produce another instance (I', k'). Then we run another pre-processing algorithm on (I', k') to get a third instance $(I^\star, k^\star)$. Given an optimal solution $s^\star$ to the last instance, we can use the solution lifting algorithm of the second pre-processing algorithm to get an optimal solution s' to the instance (I', k'). Then we can use the solution lifting algorithm of the first pre-processing algorithm to get an optimal solution s to the original instance (I, k).

Unfortunately, one can not chain α-approximate polynomial time pre-processing algorithms, as defined in Definition 3.8, in this way. In particular, each successive application of an α-approximate pre-processing algorithm increases the gap between the approximation ratio of the solution to the reduced instance and the approximation ratio of the solution to the original instance output by the solution lifting algorithm. For this reason we need to define *strict* approximate polynomial time pre-processing algorithms.

Definition 3.10. *Let $\alpha \geq 1$ be a real number, and Π be a parameterized optimization problem. An α-approximate polynomial time pre-processing algorithm is said to be* **strict** *if, for every instance (I, k), reduced instance $(I', k') = \mathcal{R}_\mathcal{A}(I, k)$ and solution s' to (I', k'), the solution s to (I, k) output by the solution lifting algorithm when given s' as input satisfies the following.*

- *If Π is a minimization problem then $\frac{\Pi(I,k,s)}{OPT(I,k)} \leq \max\left\{\frac{\Pi(I',k',s')}{OPT(I',k')}, \alpha\right\}$.*
- *If Π is a maximization problem then $\frac{\Pi(I,k,s)}{OPT(I,k)} \geq \min\left\{\frac{\Pi(I',k',s')}{OPT(I',k')}, \frac{1}{\alpha}\right\}$.*

The intuition behind Definition 3.10 is that an α-strict approximate pre-processing algorithm may incur error on near-optimal solutions, but that they have to preserve factor α-approximation. If s' is an α-approximate solution to (I', k') then s must be a α-approximate solution to (I, k) as well. Furthermore, if the ratio of $\Pi(I', k', s')$ to $OPT(I', k')$ is *worse* than α, then the ratio of $\Pi(I, k, s)$ to $OPT(I, k)$ should not be worse than the ratio of $\Pi(I', k', s')$ to $OPT(I', k')$.

We remark that a reduction algorithm $\mathcal{R}_\mathcal{A}$ and a solution lifting algorithm that together satisfy the conditions of Definition 3.10, also automatically satisfy the conditions of Definition 3.8. Therefore, to prove that $\mathcal{R}_\mathcal{A}$ and solution lifting algorithm constitute a strict α-approximate polynomial time pre-processing algorithm it is not necessary to prove that they constitute a α-approximate polynomial time pre-processing algorithm first. The advantage of Definition 3.10

is that strict α-approximate polynomial time pre-processing algorithms do chain - the composition of two strict α-approximate polynomial time pre-processing algorithms is again a strict α-approximate polynomial time pre-processing algorithm.

We can now formally define what a reduction rule is. A reduction rule for a parameterized optimization problem Π is simply a polynomial time algorithm computing a map $\mathcal{R}_{\mathcal{A}} : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$. In other words, a reduction rule is "half" of a polynomial time pre-processing algorithm. A reduction rule is only useful if the other half is there to complete the pre-processing algorithm.

Definition 3.11. *A reduction rule is said to be α-**safe for** Π if there exists a solution lifting algorithm, such that the rule together with the solution lifting algorithm constitute a strict α-approximate polynomial time pre-processing algorithm for Π. A reduction rule is **safe** if it is 1-safe.*

In some cases even the final kernelization algorithm is a strict α-approximate polynomial time pre-processing algorithm. This happens if, for example, the kernel is obtained only by applying α-safe reduction rules. Strictness yields a tigher connection between the quality of solutions to the reduced instance and the quality of the solutions to the original instance output by the solution lifting algorithms. Thus we would like to point out which kernels have this additional property. For this reason we define strict α-approximate kernels.

Definition 3.12. *An α-approximate kernel $\mathcal{A}$ is called **strict** if $\mathcal{A}$ is a strict α-approximate polynomial time pre-processing algorithm.*

Polynomial Size Approximate Kernelization Schemes. In approximation algorithms, the best one can hope for is usually an *approximation scheme*, that is an approximation algorithm that can produce a $(1+\epsilon)$-approximate solution for every $\epsilon > 0$. The algorithm runs in polynomial time for every fixed value of ϵ. However, as ϵ tends to 0 the algorithm becomes progressively slower in such a way that the algorithm cannot be used to obtain optimal solutions in polynomial time.

In the setting of approximate kernelization, we could end up in a situation where it is possible to produce a polynomial $(1+\epsilon)$-approximate kernel for every fixed value of ϵ, but that the size of the kernel grows so fast when ϵ tends to 0 that this algorithm cannot be used to give a polynomial size kernel (without any loss in solution quality). This can be formalized as a polynomial size approximate kernelization scheme.

Definition 3.13. *A **polynomial size approximate kernelization scheme** (PSAKS) for a parameterized optimization problem Π is a family of α-approximate polynomial kernelization algorithms, with one such algorithm for every $\alpha > 1$.*

Definition 3.13 states that a PSAKS is a *family* of algorithms, one for every $\alpha > 1$. However, many PSAKSes are *uniform*, in the sense that there exists an algorithm that given α outputs the source code of an α-approximate polynomial kernelization algorithm for Π. In other words, one could think of a uniform PSAKS as a single α-approximate polynomial kernelization algorithm where α is part of the input, and the size of the output depends on α. From the definition of a PSAKS it follows that the size of the output instances of a PSAKS when run on an instance (I, k) with approximation parameter α can be upper bounded by $f(\alpha) \cdot k^{g(\alpha)}$ for some functions f and g independent of $|I|$ and k.

Definition 3.14. *A **size efficient** PSAKS, or simply an **efficient** PSAKS (EPSAKS) is a PSAKS such that the size of the instances output when the reduction algorithm is run on an instance (I, k) with approximation parameter α can be upper bounded by $f(\alpha) \cdot k^c$ for a function f of α and constant c independent of I, k and α.*

Notice here the analogy to efficient polynomial time approximation schemes, which are nothing but α-approximation algorithms with running time $f(\alpha) \cdot n^c$. A PSAKS is required to run in polynomial time for every fixed value of α, but the running time is allowed to become worse and worse as α tends to 1. We can define *time-efficient* PSAKSes analagously to how we defined EPSAKSes.

Definition 3.15. *A PSAKS is said to be* **time efficient** *if (a) the running time of the reduction algorithm when run on an instance (I, k) with approximation parameter α can be upper bounded by $f(\alpha) \cdot |I|^c$ for a function f of α and constant c independent of I, k, α, and (b) the running time of the solution lifting algorithm when run on an instance (I, k), reduced instance (I', k') and solution s' with approximation parameter α can be upper bounded by $f'(\alpha) \cdot |I|^c$ for a function f' of α and constant c independent of I, k and α.*

Just as we distinguished between normal and strict α-approximate kernels, we say that a PSAKS is **strict** if it is a strict α-approximate kernel for every $\alpha > 1$.

A *quasi-polynomial time* algorithm is an algorithm with running time $\mathcal{O}(2^{(\log n)^c})$ for some constant c. In approximation algorithms, one is sometimes unable to obtain a PTAS, but still can make a $(1+\epsilon)$-approximation algorithm that runs in quasi-polynomial time for every $\epsilon > 1$. This is called a quasi-polynomial time approximation scheme. Similarly, one might be unable to give a PSAKS, but still be able to give a α-approximate kernel of quasi-polynomial size for every $\alpha > 1$.

Definition 3.16. *A* **quasi-polynomial size approximate kernelization scheme** *(QP-SAKS) for a parameterized optimization problem Π is a family of α-approximate kernelization algorithms, with one such algorithm for every $\alpha > 1$. The size of the kernel of the α-approximate kernelization algorithm should be upper bounded by $\mathcal{O}(f(\alpha)2^{(\log k)^{g(\alpha)}})$ for functions f and g independent of k.*

4 Approximate Kernel for Connected Vertex Cover

In this section we design a PSAKS for Connected Vertex Cover. The parameterized optimization problem Connected Vertex Cover(CVC) is defined as follows.

$$CVC(G, k, S) = \begin{cases} \infty & \text{if } S \text{ is not a connected vertex cover of the graph } G \\ \min\{|S|, k+1\} & \text{otherwise} \end{cases}$$

We show that CVC has a polynomial size strict α-approximate kernel for every $\alpha > 1$. Let (G, k) be the input instance. Without loss of generality assume that the input graph G is connected. Let d be the least positive integer such that $\frac{d}{d-1} \leq \alpha$. In particular, $d = \lceil \frac{\alpha}{\alpha-1} \rceil$. For a graph G and an integer k, define H to be the set of vertices of degree at least $k+1$. We define I to be the set of vertices which are not in H and whose neighborhood is a subset of H. That is $I = \{v \in V(G) \setminus H \mid N_G(v) \subseteq H\}$. The kernelization algorithm works by applying two reduction rules exhaustively. The first of the two rules is the following.

Reduction Rule 4.1. *Let $v \in I$ be a vertex of degree $D \geq d$. Delete $N_G[v]$ from G and add a vertex w such that the neighborhood of w is $N_G(N_G(v)) \setminus \{v\}$. Then add k degree 1 vertices $v_1, \ldots, v_k$ whose neighbor is w. Output this graph G', together with the new parameter $k' = k - (D-1)$.*

Lemma 4.1. *Reduction Rule 4.1 is α-safe.*

Proof. To show that Rule 4.1 is α-safe we need to give a solution lifting algorithm to go with the reduction. Given a solution S' to the instance (G', k'), if S' is a connected vertex cover of G' of size at most k' the algorithm returns the set $S = (S' \setminus \{w, v_1, \ldots, v_k\}) \cup N_G[v]$. Otherwise the solution lifting algorithm returns $V(G)$. We now need to show that the reduction rule together with the above solution lifting algorithm constitutes a strict α-approximate polynomial time pre-processing algorithm.

First we show that $OPT(G', k') \leq OPT(G, k) - (D - 1)$. Consider an optimal solution S^* to (G, k). We have two cases based on the size of S^*. If $|S^*| > k$ then $CVC(G, k, S) = k + 1$; in fact $OPT(G, k) = k + 1$. Furthermore, any connected vertex cover of G' has value at most $k' + 1 = k - (D - 1) + 1 \leq OPT(G, k) - (D - 1)$. Now we consider the case when $|S^*| \leq k$. If $|S^*| \leq k$ then $N_G(v) \subseteq S^*$, since the degree of all the vertices in $N_G(v)$ is at least $k + 1$ and S^* is a vertex cover of size at most k. Then $(S^* \setminus N_G[v]) \cup \{w\}$ is a connected vertex cover of G' of size at most $|S^*| - (D - 1) = OPT(G, k) - (D - 1)$.

Now we show that $CVC(G, k, S) \leq CVC(G', k', S') + D$. If S' is a connected vertex cover of G' of size strictly more than k' then $CVC(G, k, S) \leq k + 1 = k' + D < k' + 1 + D = CVC(G', k', S') + D$. Suppose now that S' is a connected vertex cover of G' of size at most k'. Then $w \in S'$ since w has degree at least k in G'. Thus $|S| \leq |S'| - 1 + D + 1 \leq |S'| + D$. Finally, $G[S]$ is connected because $G[N_G[v]]$ is connected and $N_G(N_G[v]) = N_{G'}(w) \setminus \{v_1, \ldots, v_k\}$. Hence S is a connected vertex cover of G. Thus $CVC(G, k, S) \leq CVC(G', k', S') + D$. Therefore, we have that

$$\frac{CVC(G,k,S)}{OPT(G,k)} \leq \frac{CVC(G',k',S') + D}{OPT(G',k') + (D-1)} \leq \max\left(\frac{CVC(G',k',S')}{OPT(G',k')}, \alpha\right).$$

The last transition follows from Fact 1. This concludes the proof. ☐

The second rule is easier than the first, if any vertex v has at least $k + 1$ false twins, then remove v. A *false twin* of a vertex v is a vertex u such that $uv \notin E(G)$ and $N(u) = N(v)$.

Reduction Rule 4.2. *If a vertex v has at least $k + 1$ false twins, then remove v, i.e output $G' = G - v$ and $k' = k$.*

Lemma 4.2. *Reduction Rule 4.2 is 1-safe.*

Proof. The solution lifting algorithm takes as input a set S' to the reduced instance and returns the same set $S' = S$ as a solution to the original instance. To see that $OPT(G', k) \leq OPT(G, k)$, consider a smallest connected vertex cover S^* of G. Again, we will distinguish between two cases either $|S^*| > k$ or $|S^*| \leq k$. If $|S^*| > k$ then $OPT(G', k) \leq k + 1 = OPT(G, k)$. Thus, assume $|S^*| \leq k$. Then there is a false twin u of v that is not in S^*. Then $S^* \setminus \{v\} \cup \{u\}$ is a connected vertex cover of $G - v$ of size at most k.

Next we show that $CVC(G, k, S) \leq CVC(G', k', S')$. If $|S'| > k' = k$ then clearly, $CVC(G, k, S) \leq k + 1 = k' + 1 = CVC(G', k', S')$. So let us assume that $|S'| \leq k$. Observe that, as v has $k + 1$ false twins, all vertices in $N(v)$ have degree at least $k + 1$ in $G - v$. Thus, $N(v) \subseteq S' = S$ and S is a connected vertex cover of G, and hence $CVC(G, k, S) \leq CVC(G', k', S')$. As a result,

$$\frac{CVC(G,k,S)}{OPT(G,k)} \leq \frac{CVC(G',k',S')}{OPT(G',k')}$$

This concludes the proof. ☐

Lemma 4.3. *Let (G, k) be an instance irreducible by rules 4.1 and 4.2, such that $OPT(G, k) \leq k$. Then $|V(G)| \leq \mathcal{O}(k^d + k^2)$.*

Proof. Since $OPT(G,k) \leq k$, G has a connected vertex cover S of size at most k. We analyze separately the size of the three sets H, I and $V(G) \setminus (H \cup I)$. First $H \subseteq S$ so $|H| \leq k$. Furthermore, every vertex in I has degree at most $d-1$, otherwise Rule 4.1 applies. Thus, there are at most $\binom{k}{d-1}$ different subsets X of $V(G)$ such that there is a vertex v in I such that $N(v) = I$. Since each vertex v has at most k false twins it follows that $|I| \leq \binom{k}{d-1} \cdot (k+1) = \mathcal{O}(k^d)$.

Finally, every edge that has no endpoints in H has at least one endpoint in $S \setminus H$. Since each vertex in $S \setminus H$ has degree at most k it follows that there are at most $k|S| \leq k^2$ such edges. Each vertex that is neither in H nor in I must be incident to at least one edge with no endpoint in H. Thus there are at most $2k^2$ vertices in $V(G) \setminus (I \cup H)$ concluding the proof. ☐

Theorem 2. Connected Vertex Cover *admits a strict time efficient PSAKS with $\mathcal{O}(k^{\lceil \frac{\alpha}{\alpha-1} \rceil} + k^2)$ vertices.*

Proof. The kernelization algorithm applies the rules 4.1 and 4.2 exhaustively. If the reduced graph G has more than $\mathcal{O}(k^d + k^2)$ vertices then, by Lemma 4.3, $OPT(G,k) = k+1$ and the algorithm may return any conneccted vertex cover of G as an optimal solution. Thus the reduced graph has at most $\mathcal{O}(k^d + k^2)$ vertices, since $d = \lceil \frac{\alpha}{\alpha-1} \rceil$ the size bound follows. The entire reduction procedure runs in polynomial time (independent of α), hence the PSAKS is time efficient. ☐

5 Disjoint Factors and Disjoint Cycle Packing

In this section we give PSAKes for Disjoint Factors and Disjoint Cycle Packing. The main ingredient of our lossy kernels is a combinatorial object that "preserves" labels of all the independent sets of a labelled graph. We will make this precise in the next section and then use this crucially to design PSAKes for both Disjoint Factors and Disjoint Cycle Packing.

5.1 Universal independent set covering

We start the subsection by defining a combinatorial object, which we call, *ϵ-universal labelled independent set covering (ϵ-ulisc)*. After formally defining it, we give an efficient construction for finding this objects when the input graph enjoys some special properties. Informally, ϵ-ulisc of a labelled graph G is an induced subgraph of G which preserves approximately *all the labelled independent sets*. The formal definition is given below. Here, $\epsilon > 0$ is a fixed constant.

ϵ-Universal Labelled Independent Set Covering (ϵ-ULISC)
Input: A graph G, an integer $q \in \mathbb{N}$ and a labelling function $\Gamma : V(G) \to [q]$
Output: A subset $X \subseteq V(G)$ such that for any independent set S in G, there is an independent set S' in $G[X]$ with $\Gamma(S') \subseteq \Gamma(S)$ and $|\Gamma(S')| \geq (1-\epsilon)|\Gamma(S)|$. The set X is called ϵ-ulisc.

Obviously, for any $\epsilon > 0$ and a labelled graph G, the whole graph G itself is an ϵ-ulisc. Our objective here is to give ϵ-ulisc with size as small as possible. Here, we design a polynomial time algorithm which gives an ϵ-ulisc for an interval graph G of size at most $(q \cdot \chi(G))^{\mathcal{O}(\frac{1}{\epsilon} \log \frac{1}{\epsilon})}$. Here, $\chi(G)$ denotes the chromatic number of the graph G.

ϵ-ulisc for Interval Graphs. From now onwards, in this subsection whenever we will use graph we mean *an interval graph*. We also assume that we have an interval representation of the graph we are considering. We use the terms *vertex* as well as *interval*, interchangeably, to denote the vertex of an interval graph. Let (G, q, Γ) be an input instance of ϵ-ULISC. We first compute a proper coloring $\kappa : V(G) \to [\chi(G)]$ of G. It is well known that a proper coloring of an interval graph with the minimum number of colors can be computed in polynomial time [14]. Now using the proper coloing function κ, we refine the labelling Γ to another labelling Λ of G as follows: for any $u \in V(G)$, $\Lambda(u) = (\Gamma(u), \kappa(u))$. An important property of the labelling Λ is the following: for any $i \in \{(a,b) \mid a \in [q], b \in [\chi(G)]\}$, $\Lambda^{-1}(i)$ is an *independent set* in G. From now onwards, we will assume that we are working with the labelling function $\Lambda : V(G) \to [k]$, where $\boxed{k = q \cdot \chi(G)}$ and $\Lambda^{-1}(i)$ is an independent set in G for any $i \in [k]$. We say that a subset of labels $Z \subseteq [k]$ is *realizable* in a graph G, if there exists an independent set $S \subseteq V(G)$ such that $\Lambda(S) = Z$. We first show that an ϵ-ulisc for G with respect to the labelling Λ refining Γ is also an ϵ-ulisc for G with respect to the labelling Γ.

Lemma 5.1. *Let X be a vertex subset of G. If X is an ϵ-ulisc for (G, Λ) then it is also an ϵ-ulisc for (G, Γ).*

Proof. Let I be an independent set of G and let $\Gamma(I)$ denote the set of labels on the vertices of I. We first compute a subset I' of I by selecting exactly one vertex from I for each label in $\Gamma(I)$. Clearly, $\Gamma(I) = \Gamma(I')$. Observe that since any label is used at most once on any vertex in I' we have that $|I'| = |\Gamma(I')| = |\Lambda(I')|$. In particular, for any pair of vertices $u, v \in I'$, $\Gamma(u) \neq \Gamma(v)$. By the property of the set X, we have an independent set S' in $G[X]$ with $\Lambda(S') \subseteq \Lambda(I')$ and $|\Lambda(S')| \geq (1-\epsilon)|\Lambda(I')|$. Since, for any pair of vertices $u, v \in I'$, $\Gamma(u) \neq \Gamma(v)$, we have that $\Gamma(S') \subseteq \Gamma(I') = \Gamma(I)$ and $|\Gamma(S')| \geq (1-\epsilon)|\Gamma(I')| = (1-\epsilon)|\Gamma(I)|$. This concludes the proof. $\square$

Lemma 5.1 implies that we can assume that the input labelling is also a proper coloring of G by increasing the number of labels by a multiplicative factor of $\chi(G)$. We first define notions of *rich* and *poor* labels; which will be crucially used in our algorithm.

Definition 5.1. *For any induced subgraph H of G we say that a label $\ell \in [k]$ is* rich *in H, if there are at least k vertices in H that are labelled ℓ. Otherwise, the label ℓ is called* poor *in H.*

We start with a simple lemma that shows that in an interval graph all the rich labels are realizable by an independent set of G. In particular we show the following lemma.

Lemma 5.2. *Let H be an induced subgraph of G, Λ be a labelling function as defined above and R be the set of rich labels in H. Then R is realizable in H. Moreover, an independent set S such that $\Lambda(S) = R$ can be computed in polynomial time.*

Proof. For our proof we will design an algorithm which constructs an independent set S such that $\Lambda(S) = R$. We assume that we have an interval representation of H and for any vertex $v \in V(H)$, let I_v be the interval corresponding to the vertex v. Our algorithm is recursive and as an input takes the tuple (H, Λ, R). In the base case it checks whether there is a vertex $w \in V(H)$ such that $\Lambda(w) \in R$. If there is no w such that $\Lambda(w) \in R$ then the algorithm outputs an $\emptyset$. Otherwise, pick a vertex I_u in H' such that $\Lambda(u) \in R$ and the value of the right endpoint of the interval I_u is minimum among all the vertices that are labelled with labels from R in H.Having found I_u (or u), we recursively solve the problem on the input $(H' = H - N[u], \Lambda|_{V(H')}, R \setminus \{\Lambda(u)\})$. Here, $\Lambda|_{V(H')}$ is the labelling Λ restricted to the vertices in $V(H')$. Let S' be the output of the recursive call on the input $(H', \Lambda|_{V(H')}, R \setminus \{\Lambda(u)\})$. Our algorithm will output $S' \cup \{u\}$.

Now we prove the correctness of the algorithm. Towards this we prove the following statement using induction on $|R|$: for any induced subgraph H of G and $R \subseteq [k]$ such that for any $j \in R$, the number of vertices in H labelled with j is at least $|R|$, then the above algorithm on input (H, Λ, R) will output an independent set S such that $\Lambda(S) = R$. The base case is when $|R| = 0$, and statement holds trivially. Now consider the induction step. Let u be the vertex picked by the algorithm such that $\Lambda(u) \in R$ and the value of the right endpoint of the interval I_u, corresponding to u, is the minimum among all such intervals. Since for any $j \in R$, $\Lambda^{-1}(j)$

is independent, and I_u is an interval (vertex) with minimum right endpoint value, we have that for any $i \in R$, the number of intervals labelled with i that intersects with I_u is at most 1. This implies that for any $i \in R \setminus \{\Lambda(u)\}$, the number of vertices labelled with i in $H - N[u]$ is at least $|R| - 1$ (because the number of vertices labelled with i in H is at least $|R|$). Hence, by the induction hypothesis, the recursive call on the input $(H' = H - N[u], \Lambda|_{V(H')}, R \setminus \{\Lambda(u)\})$ will output an independent set S' such that $\Lambda(S') = R \setminus \{\Lambda(u)\}$. Since $S' \cap N[u] = \emptyset$, we have that $S' \cup \{u\}$ is the required independent set for the input (H, Λ, R). This completes the correctness proof. $\square$

Before we give the formal construction for the desired ϵ-ulisc for G, we first give an intuitive explanation of our strategy. If we are seeking for an upper bound on ϵ-ulisc in terms of k, then Lemma 5.2 suggests the following natural strategy: for rich labels, we find an independent set, say I_{rich}, of size at most k (as the number of labels itself is upper bounded by k) that realizes it and add all the vertices in this set to ϵ-ulisc we are constructing. Let us denote the the ϵ-ulisc we are constructing by X. For the poor labels, we know that by definition each label appears on at most k vertices and thus in total the number of vertices that have poor labels is upper bounded by k^2. We include all the vertices that have poor labels to ϵ-ulisc (the set X) we are constructing. So at this stage if G has an independent set S such that all the labels used on the vertices in S ($\Lambda(S)$) are rich then we can find an appropriate independent subset of I_{rich} that realizes all the labels of $\Lambda(S)$. On the other hand, if we have an independent set S such that all the labels used on the vertices in S are poor then it self realizes itself. That is, since we have kept all the vertices that are poor, the set S itself is a contained inside the ϵ-ulisc we are constructing and thus it self realizes itself. The problem arises when we have an independent set S that has vertices having both rich labels as well as poor labels. We deal with this case essentially by the following case distinctions. Let $\Lambda(S)$ denote the set of labels used on the vertices in S and $\Lambda(S)_{\mathsf{rich}}$ and $\Lambda(S)_{\mathsf{poor}}$ denote the set of rich and poor labels in $\Lambda(S)$, respectively.

1. If $|\Lambda(S)_{\mathsf{rich}}| \geq (1-\epsilon)|\Lambda(S)|$, then we are again done as we can find an appropriate independent subset of I_{rich} that realizes all the labels of $\Lambda(S)_{\mathsf{rich}}$.
2. Since the first case does not arise we have that the number of rich labels in $\Lambda(S)$, that is, $|\Lambda(S)_{\mathsf{rich}}|$ is upper bounded by $(1-\epsilon)|\Lambda(S)|$ and that $|\Lambda(S)_{\mathsf{poor}}| \geq \epsilon|\Lambda(S)|$. Thus, in this case it is possible that $|\Lambda(S)_{\mathsf{poor}}| = |\Lambda(S)_{\mathsf{rich}}| = \frac{1}{2}|\Lambda(S)|$ and hence it is *possible* that there is no independent set S' in $G[X]$ (the set X constructed so far) with $\Lambda(S') \subseteq \Lambda(S)$ and $|\Lambda(S')| \geq (1-\epsilon)|\Lambda(S)|$. Thus, we need to enrich the set X further. Towards this we use the following strategy. Let Q be the set of endpoints of the intervals labelled with poor labels. Furthermore, assume that all the intervals of G are between $(0, a)$. Now for every $p, q \in Q \cup \{0, a\}$, let $Y_{p,q}$ denote the set of intervals of G which is fully contained in the open interval (p, q). For every, $p, q \in Q \cup \{0, a\}$, we recursively find the desired ϵ-ulisc in $G[Y_{p,q}]$ and then take the union. Clearly, this is a branching algorithm with every node in the recursion tree having $\mathcal{O}(k^4)$ children. See Figure 2 for an illustration of the process. The *idea* of this branching procedure is that given an independent set S we would like to pack all the vertices in S with poor labels and then having made this choice we get disjoint induced subgraph of G (by removing all the vertices in S with poor labels and their neighorhiood) where we "would like to pack" the vertices in S that have rich labels. By our construction it is evident that the set S' we will obtain by packing labels in the disjoint induced subgraphs of G is compatible with the choice of packing all the vertices with poor labels in S. To get an upper bound on the size of the set X we are constructing we show that the recursion tree can be truncated at the depth of $\lceil \mathcal{O}(\frac{1}{\epsilon} \log \frac{1}{\epsilon}) \rceil$

Now we give our main lemma that gives an algorithm for finding the desired ϵ-ulisc for G with respect to the labelling function Λ.

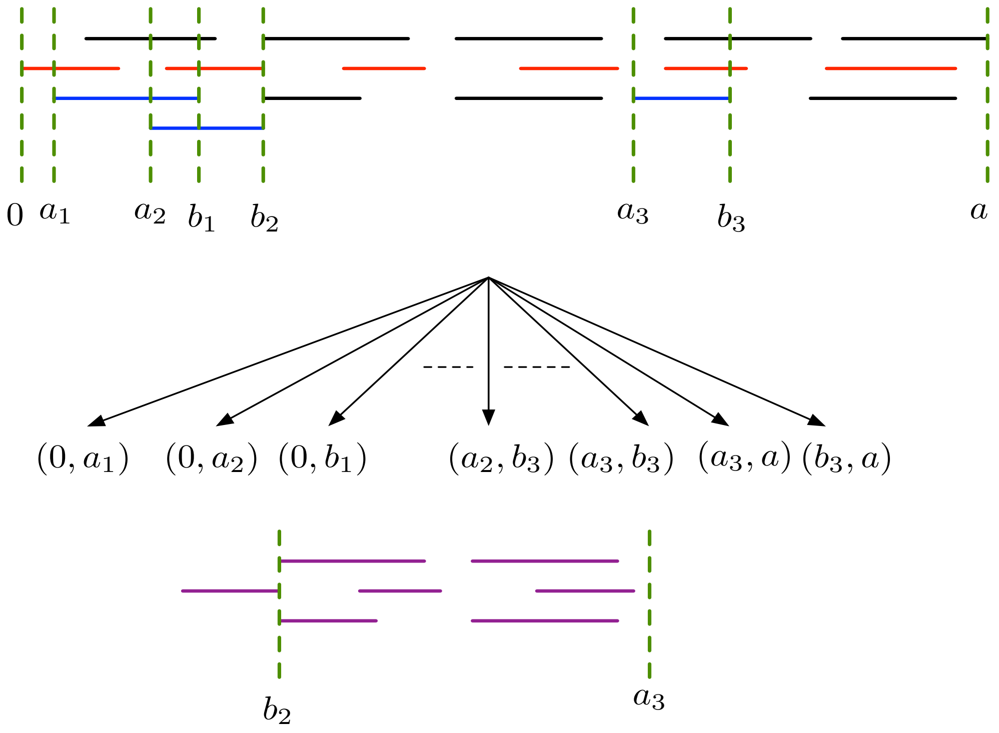


Figure 2: *An illustration for the process of obtaining ϵ-ulisc. Intervals colored with red denote I_{rich} and intervals colored with blue denote vertices labelled with poor label. The instance below corresponds to branching on (a_2, b_3).*

Lemma 5.3. *Let G be an interval graph, $k \in \mathbb{N}$ and $\epsilon' > 0$. Let $\Lambda : V(G) \to [k]$ be a labelling function such that for any $i \in [k]$, $\Lambda^{-1}(i)$ is an independent set in G. Then, there is a polynomial time algorithm that finds a set $X \subseteq V(G)$ of cardinality $k^{\mathcal{O}(\frac{1}{\epsilon'} \log \frac{1}{\epsilon'})}$ such that for any realizable set $Z \subseteq [k]$ in G, there is a realizable subset $Z' \subseteq Z$ of cardinality at least $(1 - 2\epsilon')|Z|$ in $G[X]$.*

Proof. Our polynomial time algorithm is a bounded depth recursive procedure. Towards that we define a recursive marking procedure Mark-Interval which takes as input an induced subgraph of G and a positive integer and marks intervals of G. Vertices corresponding to the marked intervals will correspond to the desired set X. Our algorithm is called Mark-Interval. See Algorithm 1 for a detailed formal description of the algorithm. We call the procedure Mark-Interval on input $(G, d = \lceil \frac{1}{\epsilon'} \log \frac{1}{\epsilon'} \rceil)$ to get the required set X, which is the set of vertices marked by the procedure. Without loss of generality we assume that all the intervals in G are contained in $(0, a)$ for some $a \in \mathbb{N}$.

We first show that the procedure Mark-Interval on input (G, d) marks at most $k^{\mathcal{O}(d)} = k^{\mathcal{O}(\frac{1}{\epsilon'} \log \frac{1}{\epsilon'})}$ intervals. Let X be the set of marked intervals. In Step 5, Mark-Interval marks at most k intervals, one for each rich label in G. In Step 7, Mark-Interval mark all intervals which are labelled with poor labels in G and the number of such intervals is at most k^2. This implies that number of points in Q is at most $2k^2 + 2$. Hence the procedure makes at most $\binom{2k^2+2}{2}$ recursive calls. Thus, the total number of marked intervals is bounded by the recurrence relation, $T(d) \leq (k^2 + k) + \binom{2k^2+2}{2} T(d-1)$ and $T(1) = 0$. This recurrence relation solves to $k^{\mathcal{O}(d)}$. This implies that the cardinality of the set of marked vertices by Mark-Interval$(G, \lceil \frac{1}{\epsilon'} \log \frac{1}{\epsilon'} \rceil)$ is at most $k^{\mathcal{O}(\frac{1}{\epsilon'} \log \frac{1}{\epsilon'})}$.

Now we show the correctness of the algorithm. Towards that we first prove the following claim.

Algorithm 1: Mark-Interval (H, d'), where H is an induced subgraph of G and $d' \in \mathbb{N}$

```
1  if $d' = 1$ then
2      return
3  Let $R$ be the set of rich labels in $H$.
4  Apply the algorithm mentioned
   in Lemma 5.2 and let $S$ be its output (Note that $S$ is an independent set and $\Lambda(S) = R$).
5  Mark all the intervals in $S$.
6  Let $P$ be the set of intervals which are labelled with poor labels in $H$.
7  Mark all the intervals in $P$.
8  Let $Q$ be the set of endpoints of the intervals in $P$.
9  forall the $p, q \in Q \cup \{0, a\}$ do
10     Mark-Interval($H[Y_{p,q}], d' - 1$), where $Y_{p,q}$ is the set of intervals of $H$ which is fully
       contained in the open interval $(p, q)$.
```

Claim 5.1. *Let H be an induced subgraph of G, $d' \leq d$ be a positive integer and X' be the set of marked vertices by the procedure* Mark-Interval *on input (H, d'). If $W \subseteq [k]$ is realizable in H, then there is a subset $W' \subseteq W$ such that W' is realizable in $H[X']$ and $|W'| \geq (1 - \epsilon' - (1 - \epsilon')^{d'})|W|$.*

Proof. We prove the claim using induction on d'. The base case is when $d' = 1$. When $d' = 1$, $(1 - \epsilon' - (1 - \epsilon')^{d'})|W| = 0$ and empty set is the required set W' of labels. Now consider the induction step. We assume that the claim is true for any $1 \leq d'' < d'$. If at least $(1 - \epsilon')|W|$ labels in W are rich then in Step 4, the procedure Mark-Interval computes an independent set S such that $\Lambda(S)$ is the set of all rich labels in H and vertices in S is marked in Step 5. This implies that at leasts $(1 - \epsilon')|W| \geq (1 - \epsilon' - (1 - \epsilon')^{d'})|W|$ labels in W are realizable in $H[X']$. Now we are in the case where strictly less than $(1 - \epsilon')|W|$ labels in W are rich. That is, the number of poor labels contained in W appearing on the vertices of H is at least $\epsilon'|W|$. Let U be an independent set in H such that $\Lambda(U) = W$ and let U_p be the subset of U which are labeled with poor labels from H. Notice that $|U_p| \geq \epsilon'|W|$. In Step 7, procedure Mark-Interval marks all the intervals in U_p. Let $[a_1, b_1], \ldots, [a_\ell, b_\ell]$ be the set of intervals in U_p such that $a_1 < b_1 < a_2 < b_2 < \ldots < b_\ell$. All the intervals in $U \setminus U_p$ are disjoint from U_p. That is, there exists a family of intervals $\{V_0, V_1, \ldots, V_\ell\}$ such that $\bigcup_{i=0}^{\ell} V_i = U \setminus U_p$ and for any $i \in \{0, \ldots, \ell\}$, the intervals in V_i are contained in (b_i, a_{i+1}), where $b_0 = 0$ and $a_{\ell+1} = a$. The recursive procedure Mark-Interval on input (H, d') calls recursively with inputs $(H[Y_{b_i,a_{i+1}}], d' - 1)$, $i \in \{0, \ldots, \ell\}$. Here $V_i \subseteq Y_{b_i,a_{i+1}}$. Let $W_i = \Lambda(V_i)$. Notice that $W_i \cap W_j = \emptyset$ for $i \neq j$ and $\Lambda(U_p) \cup \bigcup_{i=0}^{\ell} W_i = W$. By induction hypothesis, for any $i \in \{0, \ldots, \ell\}$, there exists $W'_i \subseteq W_i \subseteq W$ such that $|W'_i| \geq (1 - \epsilon' - (1 - \epsilon')^{d'-1})|W_i|$ and W'_i is realizable in $H[X_i]$ where X_i is the set of vertices marked by Mark-Interval($H[Y_{b_i,a_{i+1}}], d' - 1$). This implies that $\Lambda(U_p) \cup \bigcup_{i=0}^{\ell} W_i'$ is realizable in $H[X]$. Now we lower bound the size of $\Lambda(U_p) \cup \bigcup_{i=0}^{\ell} W_i'$.

$$\begin{aligned}
|\Lambda(U_p) \cup \bigcup_{i=0}^{\ell} W_i'| &\geq |\Lambda(U_p)| + \sum_{i=0}^{\ell}(1-\epsilon' - (1-\epsilon')^{d'-1})|W_i| \\
&= |\Lambda(U_p)| + (1-\epsilon' - (1-\epsilon')^{d'-1}) \sum_{i=0}^{\ell} |W_i| \\
&= |\Lambda(U_p)| + (1-\epsilon' - (1-\epsilon')^{d'-1})|W \setminus \Lambda(U_p)| + |W \setminus \Lambda(U_p)| - |W \setminus \Lambda(U_p)| \\
&\geq |W| - (\epsilon' + (1-\epsilon')^{d'-1})|W \setminus \Lambda(U_p)| \\
&\geq |W| - \epsilon'|W| - (1-\epsilon')^{d'-1}|W \setminus \Lambda(U_p)| \\
&\geq |W| - \epsilon'|W| - (1-\epsilon')^{d'}|W| \qquad \text{(Because } |W \setminus \Lambda(U_p)| < (1-\epsilon')|W|\text{)} \\
&\geq (1-\epsilon' - (1-\epsilon')^{d'})|W|
\end{aligned}$$

This completes the proof of the claim. □

Let $Z \subseteq [k]$ be a set of labels which is realizable in G. Now, by Claim 5.1, we have that there exists $Z' \subseteq Z$ such that Z' is realizable in $G[X]$ and $|Z'| \geq (1-\epsilon'-(1-\epsilon')^{\frac{1}{\epsilon'}\log\frac{1}{\epsilon'}})|Z| \geq (1-2\epsilon')|Z|$. This completes the proof. □

Now we are ready to prove the main result of this section.

Lemma 5.4. *Let G be an interval graph, $q \in \mathbb{N}$, $\epsilon > 0$, and $\Gamma : V(G) \to [q]$ be a labelling function. Then there is a polynomial time algorithm which finds a set $X \subseteq V(G)$ of cardinality $(q \cdot \chi(G))^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$ such that X ϵ-ulisc of G.*

Proof. We start by refining the labelling Γ to Λ such that Λ is a proper coloring of G. As explained before, we first compute a proper coloring $\kappa : V(G) \to [\chi(G)]$ of G in polynomial time [14]. Now using the proper coloing function κ and the labelling Γ, we define labelling Λ of G as follows: for any $u \in V(G)$, $\Lambda(u) = (\Gamma(u), \kappa(u))$. Now we set $\epsilon' = \frac{\epsilon}{2}$ and apply Lemma 5.3 on G, Λ, $k = q \cdot \chi(G)$ and ϵ' to get a set $X \subseteq V(G)$ of cardinality $k^{\mathcal{O}(\frac{1}{\epsilon'}\log\frac{1}{\epsilon'})}$ such that for any realizable set $Z \subseteq [k]$ in G, there is a realizable subset $Z' \subseteq Z$ of cardinality at least $(1-2\epsilon')|Z|$ in $G[X]$. That is, $X \subseteq V(G)$ is of cardinality $k^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$ such that for any realizable set $Z \subseteq [k]$ in G, there is a realizable subset $Z' \subseteq Z$ of cardinality at least $(1-\epsilon)|Z|$ in $G[X]$. Note that a set X is ϵ-ulisc for G if and only if for any realizable set $Z \subseteq [k]$ in G, there is a realizable subset $Z' \subseteq Z$ of cardinality at least $(1-\epsilon)|Z|$ in $G[X]$. This implies that X is ϵ-ulisc for (G, Λ). However, by Lemma 5.1, we know that if X is an ϵ-ulisc for (G, Λ) then it is also an ϵ-ulisc for (G, Γ). This concludes the proof. □

5.2 Disjoint Factors

In this section, we give a PSAKS for the parameterized optimization problem Disjoint Factors (DF). To define this problem we first need to set up some definitions. For a string $L = a_1 a_2 \ldots a_n$ over an alphabet Σ, we use $L[i, j]$, where $1 \leq i \leq j \leq n$, to denote the substring $a_i \ldots a_j$. In this section we would like to distinguish between two substrings $L[i, j]$ and $L[i', j']$, where $i \neq i'$ or $j \neq j'$, even if the string $L[i, j]$ is exactly same as the string $L[i', j']$. Thus we call L' is a "position substring" of L, to emphasize L' is substring of L associated with two indices. We say two position substrings L_1 and L_2 are disjoint if they do not overlap (even at the starting or at the ending of the substrings). For example $L[i, j]$ and $L[j, j']$ are overlapping and not disjoint. We say that a string L' is a *string minor* of L, if L' can be obtained from L by deleting some position substrings of L. A *factor* of a string L is a position substring of length at least 2 which starts and ends with the same letter (symbol). A factor is called *x-factor* if the factor starts and end at a letter $x \in \Sigma$. Two factors are called distinct if they start at different letters. A set $\mathcal{S}$ of factors in L is called a set of *disjoint factors* if each pair of factors in $\mathcal{S}$ are *disjoint and distinct*. That is, no two factors in $\mathcal{S}$ start at the same letter and pairwise they do not overlap. This immediately implies that for any string L over Σ, the cardinality of any set of *disjoint factors* is at most $|\Sigma|$. For a set of disjoint factors $\mathcal{S}$ of L and $\Sigma' \subseteq \Sigma$, we say that $\mathcal{S}$ is a Σ'-factor if for each element x in Σ', there is a factor in $\mathcal{S}$, starting and ending at x.

In the Disjoint Factors problem, introduced in [8], input is an alphabet Σ and a string L in Σ^*, the task is to find a maximum cardinality set of disjoint factors in L. Bodlaender *et al.* [8] proved that Disjoint Factors is NP-complete by reduction from 3-SAT, and also that Disjoint Factors parameterized by $|\Sigma|$ does not admit a polynomial kernel unless $\mathsf{coNP} \subseteq \mathsf{NP}/\mathsf{Poly}$. The reduction of Bodlaender *et al.* started from a *gap* variant of 3-SAT where every variable appears in at most a constant number of clauses [54] shows that Disjoint Factors is in fact APX-hard, which means that it does not admit a PTAS unless $\mathsf{P} = \mathsf{NP}$. We will consider Disjoint Factors when parameterized by the alphabet size $|\Sigma|$. Formally, the parameterized optimization problem that we consider is defined as follows.

$$DF(L, |\Sigma|, \mathcal{S}) = \begin{cases} -\infty & \text{if } \mathcal{S} \text{ is not a set of disjoint factors of } L \\ |\mathcal{S}| & \text{otherwise} \end{cases}$$

We remark that in the original definition of Disjoint Factors of Bodlaender *et al.* [8], the objective is simply to decide whether it is possible to find $|\Sigma|$ disjoint factors in the input string L. The variant of the problem discussed here is the natural maximization variant of the original problem. Next we give a PSAKS for this problem, in other words a polynomial size α-approximate kernel for any $\alpha > 1$.

Definition 5.2. *Let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be a set of mutually disjoint position substrings of a string L. Then we use $L/\mathcal{S}$ to denote the string obtained from L after deleting all position substrings in $\mathcal{S}$. For example if $L = a_1 \cdots a_{11}$ and $\mathcal{S} = \{L[2,4], L[7,9]\}$, then $L/\mathcal{S} = a_1a_5a_6a_{10}a_{11}$.*

The following lemma states that we can pull back a solution of a string from a solution of its string minor.

Lemma 5.5. *Let L be a string over an alphabet Σ and $\mathcal{S}$ be a set containing distinct position substrings (non-overlapping strings). Let L' be a string minor of L obtained by deleting position substrings in $\mathcal{S}$. Then, there is a polynomial time algorithm, given $L, L', \mathcal{S}$ and a solution $\mathcal{F}'$ of $(L', |\Sigma|)$, computes a solution $\mathcal{F}$ of $(L, |\Sigma|)$ of cardinality $|\mathcal{F}'|$.*

Proof. The proof follows from the fact that for each string w in $\mathcal{F}'$ we can associate indices i and j in L such that $L[i,j]$ is an x-factor if and only if w is an x-factor in L'. Clearly, the algorithms runs in polynomial time. □

Theorem 3. Disjoint Factors *parameterized by $|\Sigma|$ admits a PSAKS.*

Proof. We need to show that for any $\epsilon > 0$, there is a polynomial sized $(1-\epsilon)$-approximate kernel for Disjoint Factors. Towards that given an instance of Disjoint Factors, we will construct a labelled interval graph G and use ϵ-ulisc of G to reduce the length of the input string. Let $(L, |\Sigma|)$ be an input instance of Disjoint Factors and $k = |\Sigma|$. Now we construct an instance $(G, |\Sigma|, \Gamma)$ of ϵ-ULISC. We define the graph and the labelling function $\Gamma : V(G) \to \Sigma$ as follows. Let $L = a_1a_2 \ldots a_n$ where $a_i \in \Sigma$. For any $i \neq j$ such that $a_i = a_j$ and $a_r \neq a_i$ for all $i < r < j$, we construct an interval $I_{i,j} = [i, j]$ on real line and label it with a_i. Observe that since $a_i = a_j$, we have that it is an a_i-factor. The set of intervals constructed form the interval representation of G. Each interval in G corresponds to a factor in L. By construction, we have that any point belongs to at most two intervals of the same label. This implies that the cardinality of largest clique in G, and hence $\chi(G)$, is upper bounded by $2|\Sigma|$ (because interval graphs are perfect graphs). Now we apply Lemma 5.4 on input $(G, |\Sigma|, \Gamma)$ and ϵ. Let $X \subseteq V(G)$ be the output of the algorithm. By Lemma 5.4, we have that $|X| = |\Sigma|^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$. Let $P = \{q : q \text{ is an endpoint of an interval in X}\}$ and $\mathcal{S} = \{L[j,j] \ : \ j \in [n] \setminus P\}$. The reduction algorithm will output $(L' = L/\mathcal{S}, |\Sigma|)$ as the reduced instance. Since $|P| = |\Sigma|^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$, we have that the length of $L/\mathcal{S}$ is at most $|\Sigma|^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$.

The solution lifting algorithm is same as the one mentioned in Lemma 5.5. Let $\mathcal{F}'$ be a set of disjoint factors of the reduced instance $(L', |\Sigma|)$ and let $\mathcal{F}$ be the output of solution lifting algorithm. By Lemma 5.5, we have that $|\mathcal{F}| = |\mathcal{F}'|$. To prove the correctness we need to prove the approximation guarantee of $\mathcal{F}$. Towards that we first show that $OPT(L/\mathcal{S}, |\Sigma|) \geq (1-\epsilon) OPT(L, |\Sigma|)$. Let $\mathcal{P}$ be a set of maximum sized disjoint factors in L. Without loss of generality we can assume that for each factor $L[i,j]$ in $\mathcal{P}$, $L[i'] \neq L[i]$ for all $i < i' < j$. This implies that each factor in $\mathcal{P}$ corresponds to an interval in G. Moreover, these set of intervals U (intervals corresponding to $\mathcal{P}$) form an independent set in G with distinct labels. By Lemma 5.4, there is an independent set Y in $G[X]$ such that $\Gamma(Y) \subseteq \Gamma(U)$ and $|\Gamma(Y)| \geq (1-\epsilon)|\Gamma(U)|$. Each interval in Y corresponds to a factor in $L/\mathcal{S}$ and its label corresponds to the starting symbol of the factor. This implies that $L/\mathcal{S}$ has a set of disjoint factors of cardinality at least $(1-\epsilon)|\mathcal{P}| = (1-\epsilon)OPT(L, |\Sigma|)$. Hence, we have

$$\frac{|\mathcal{F}|}{OPT(L, |\Sigma|)} \geq (1-\epsilon)\frac{|\mathcal{F}'|}{OPT(L/\mathcal{S}, |\Sigma|)}.$$

This concludes the proof. □

5.3 Disjoint Cycle Packing

In this subsection we design a PSAKS for the Disjoint Cycle Packing (CP) problem. The parameterized optimization problem Disjoint Cycle Packing (CP) is formally defined as,

$$CP(G,k,P) = \begin{cases} -\infty & \text{if } P \text{ is not a set of vertex disjoint cycles in } G \\ \min\{|P|, k+1\} & \text{otherwise} \end{cases}$$

We start by defining feedback vertex sets of a graph. Given a graph G and a vertex subset $F \subseteq V(G)$, F is called a *feedback veretx set* of G if $G - F$ is a forest. We will make use of the following well-known Erdős-Pósa Theorem relating feedback vertex set and the number of vertex disjoint cycles in a graph.

Lemma 5.6 ([26]). *There exists a constant c such that for each positive integer k, every (multi) graph either contains k vertex disjoint cycles or it has a feedback vertex set of size at most $ck\log k$. Moreover, there is a polynomial time algorithm that takes a graph G and an integer k as input, and outputs either k vertex disjoint cycles or a feedback vertex set of size at most $ck\log k$.*

The following lemma allows us to reduce the size of the input graph G if it has a small feedback vertex set.

Lemma 5.7. *Let (G,k) be an instance of* Disjoint Cycle Packing *and F be a feedback vertex set of G. Suppose there are strictly more than $|F|^2(2|F|+1)$ vertices in $G-F$ whose degree in $G-F$ is at most 1. Then there is a polynomial time algorithm $\mathcal{A}$ that, given an instance (G,k) and a feedback vertex set satisfying the above properties, returns a graph G' (which is a minor of G) such that $OPT(G,k) = OPT(G',k)$, $|V(G')| = |V(G)| - 1$ and $F \subseteq V(G')$ is still a feedback vertex set of G'. Further, given a cycle packing $\mathcal{S}'$ in G', there is a polynomial time algorithm $\mathcal{B}$ which outputs a cycle packing $\mathcal{S}$ in G such that $|\mathcal{S}| = |\mathcal{S}'|$.*

Proof. The algorithm $\mathcal{A}$ works as follows. Let $|F| = \ell$ and for $(u, v) \in F \times F$, let $L(u, v)$ be the set of vertices of degree at most 1 in $G - F$ such that each $x \in L(u, v)$ is adjacent to both u and v (if $u = v$, then $L(u, u)$ is the set of vertices which have degree at most 1 in $G - F$ and at least two edges to u). Suppose that the number of vertices of degree at most 1 in $G - F$ is strictly more than $\ell^2(2\ell + 1)$. For each pair $(u, v) \in F \times F$, if $L(u, v) > 2\ell + 1$ then we mark an arbitrary set of $2\ell + 1$ vertices from $L(u, v)$, else we mark all the vertices in $L(u, v)$. Since there are at most $\ell^2(2\ell + 1)$ marked vertices, there exists an unmarked vertex w in $G - F$ such that $d_{G-F}(w) \leq 1$. If $d_{G-F}(w) = 0$, then algorithm $\mathcal{A}$ returns $(G - w, k)$. Suppose $d_{G-F}(w) = 1$. Let e be the unique edge in $G - F$ which is incident to w. Algorithm $\mathcal{A}$ returns $(G/e, k)$. Clearly $F \subseteq V(G')$ and F is a feedback vertex set of G'.

Let (G', k) be the instance returned by algorithm $\mathcal{A}$. Since G' is a minor of G, $OPT(G, k) \geq OPT(G', k)$. (A graph H is called a minor of an undirected graph $G^\star$, if we can obtain H from $G^\star$ by a sequence of edge deletions, vertex deletions and edge contractions.) Now we show that $OPT(G, k) \leq OPT(G', k)$. Let $G' = G/e$, $e = (w, z)$, $d_{G-F}(w) = 1$ and w is an unmarked vertex. Let $\mathcal{C}$ be a maximum set of vertex disjoint cycles in G. Observe that if $\mathcal{C}$ does not contain a pair of cycles each intersecting a different endpoint of e, then contracting e will keep the resulting cycles vertex disjoint in G/e. Therefore, we may assume that $\mathcal{C}$ contains 2 cycles C_w and C_z where C_w contains w and C_z contains z. Now, the neighbor(s) of w in C_w must lie in F. Let these neighbors be x and y (again, x and y are not necessarily distinct). Since $w \in L(x, y)$ and it is unmarked, there are $2\ell + 1$ vertices in $L(x, y)$ which are already marked by the marking procedure. Further, since for each vertex $u \in V(\mathcal{C})$, with $d_{G-F}(u) \leq 1$, at least one neighbour of u in the cycle packing $\mathcal{C}$ is from F and each vertex $v \in V(\mathcal{C}) \cap F$ can be adjacent to at most 2 vertices from $L(x, y)$, we have that at most 2ℓ vertices from $L(x, y)$ are in $V(\mathcal{C})$. This implies that at least one vertex (call it w'), marked for $L(x, y)$ is not in $V(\mathcal{C})$. Therefore we can route the cycle C_w through w' instead of w, which gives us a set of $|\mathcal{C}|$ vertex disjoint cycles in G/e. Suppose $G' = G - w$ and $d_{G-F}(w) = 0$. Then by similar arguments we can show that $OPT(G, k) = OPT(G - w, k)$.

Algorithm $\mathcal{B}$ takes a solution $\mathcal{S}'$ of the instance (G', k) and outputs a solution $\mathcal{S}$ of (G, k) as follows. If G' is a subgraph of G (i.e, G' is obtained by deleting a vertex), then $\mathcal{S} = \mathcal{S}'$. Otherwise, let $G' = G/e$, $e = (u, v)$ and let w be the vertex in G' created by contracting (u, v). If $w \notin V(\mathcal{S}')$, then $\mathcal{S} = \mathcal{S}'$. Otherwise let $C = wv_1 \ldots v_\ell$ be the cycle in $\mathcal{S}'$ containing w. We know that $v_1, v_\ell \in N_G(\{u, v\})$. If $v_1, v_\ell \in N_G(u)$, then $C' = uv_1 \ldots v_\ell$ is a cycle in G which is vertex disjoint from $\mathcal{S}' \setminus \{C\}$. If $v_1, v_\ell \in N_G(v)$, then $C' = vv_1 \ldots v_\ell$ is a cycle in G which is vertex disjoint from $\mathcal{S}' \setminus \{C\}$. In either case $\mathcal{S} = (\mathcal{S}' \setminus \{C\}) \cup \{C'\}$. If $v_1 \in N_G(u)$ and $v_\ell \in N_G(v)$, then $C'' = uv_1 \ldots v_\ell vu$ is a cycle in G which is vertex disjoint from $\mathcal{S}' \setminus \{C\}$. In this case $\mathcal{S} = (\mathcal{S}' \setminus \{C\}) \cup \{C''\}$. This completes the proof of the lemma. $\square$

Lemma 5.7 leads to the following reduction rule which is 1-safe (follows from Lemma 5.7).

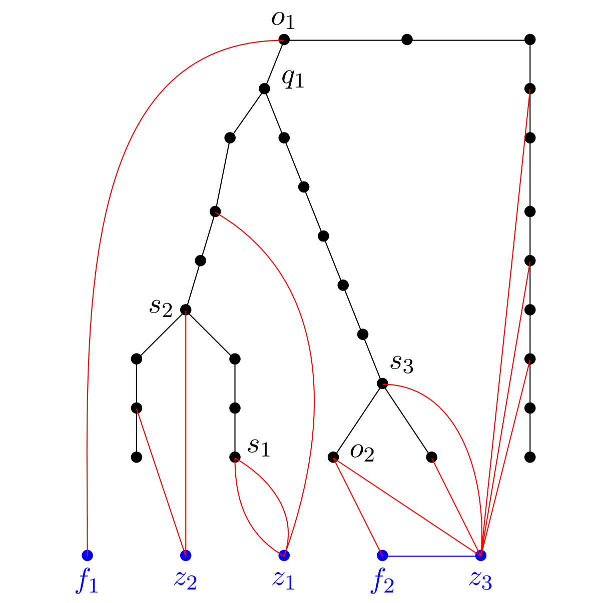
Reduction Rule 5.1. *Let (G, k) be an instance of* Disjoint Cycle Packing *and let F be a feedback vertex set of G such that the forest $G - F$ contains strictly more than $|F|^2(2|F| + 1)$ vertices of degree at most 1. Then run the algorithm $\mathcal{A}$ mentioned in Lemma 5.7 on (G, k) and F, and return (G', k), where G', a minor of G, is the output of the algorithm $\mathcal{A}$.*

The following observation follows from Lemma 5.7.

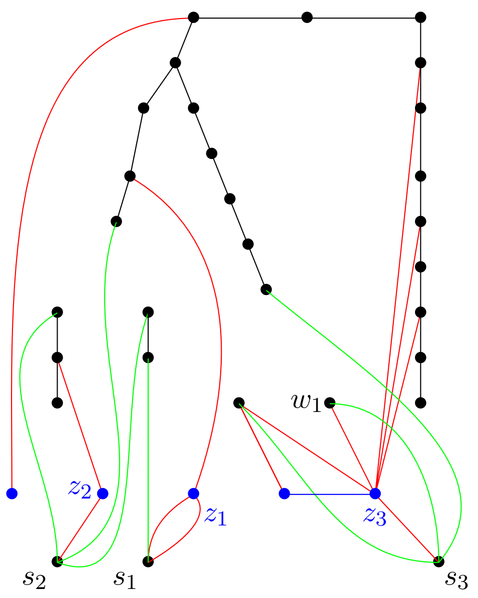
Observation 5.1. *Let (G, k) be an instance of* Disjoint Cycle Packing *and (G', k) be the instance obtained after applying Reduction Rule 5.1. Then $OPT(G, k) = OPT(G', k)$.*

The Reduction Rule 5.1, may create multiple edges in the reduced instance. To bound the number of multi-edges between a pair of vertices, we use the following simple reduction rule.

Reduction Rule 5.2. *Let (G, k) be an instance of* Disjoint Cycle Packing *and there exist two vertices $u, v \in G$ such that there are at least 3 edges between u and v. Then delete all but two edges between u and v.*



(a) *$F = \{f_1, f_2, z_1, z_2, z_3\}$ is a feedback vertex set of G and the tree $G - F$ is rooted at o_1. Algorithm $\mathcal{B}$ will output $F' = \{z_1, z_2, z_3\}$ and $S = \{s_1, s_2, s_3\}$ when $k > 3$.*



(b) *The vertices of S is drawn separately with edges between S and $G - F$ colored green*

Figure 3: *An example of Lemma 5.8*

Since any set of vertex disjoint cycles in G can use at most two edges between u and v, it is safe to delete remaining edges between them and hence Reduction Rule 5.2 is 1-safe. Hence, in the rest of the section we always assume that the number of edges between any pair of vertices is at most 2. The following lemma allows us to find a subset F', of a feedback vertex set F, of cardinality at most $OPT(G, k)$ such that the large portion of the graph $G - F$ is connected to F' and not to $F \setminus F'$.

Lemma 5.8. *Let (G, k) be an instance of* Disjoint Cycle Packing *and let F be a feedback vertex set of G. Then there is a polynomial time algorithm $\mathcal{B}$ that given (G, k) and F, either outputs k vertex disjoint cycles in G or two sets $F' \subseteq F$ and $S \subseteq V(G - F)$ such that (i) $|F'|, |S| \leq OPT(G, k)$ and (ii) for any $w \in F \setminus F'$ and any connected component C of $G - (F \cup S)$, $|N(w) \cap V(C)| \leq 1$.*

Proof. We know that $G - F$ is a forest. We consider each tree in $G - F$ as a rooted tree, where the root is chosen arbitrarily. Now we create a *dummy root* r and connect to all the roots in $G - F$. The resulting graph T with vertex set $(V(G) \cup \{r\}) \setminus F$ is a tree rooted at r. The level of a vertex $v \in V(T)$ is the distance between r and v, denoted by $d_T(r, v)$. Let T' be a rooted tree, then for a vertex $v \in V(T')$ we use T'_v to denote the subtree of T' rooted at v.

Now we are ready to give a procedure to find the desired sets F' and S. Initially we set $T' := T$, $F' := \emptyset$ and $S := \emptyset$. Let $u \in V(T')$ such that $d_{T'}(r, u)$ is maximized and there is a vertex $w \in F \setminus F'$ with the property that $G[V(T'_u) \cup \{w\}]$ has a cycle. Then, we set $T' := T' - T'_u$, $F' := F' \cup \{w\}$ and $S := S \cup \{u\}$. We continue this procedure until $|F'| = |S| = k$ or the above step is not applicable. Let $F' = \{w_1, \ldots, w_{k'}\}$. Notice that by the above process there are vertex disjoint subtrees $T_1, \ldots, T_{k'}$ of $T - r$ such that for each $i \in [k']$, $G[V(T_i) \cup \{w_i\}]$ has a cycle. Thus when $k' = k$, our algorithm $\mathcal{B}$ will output one cycle from each $G[V(T_i) \cup \{w_i\}]$, $i \in [k]$ as the output. Otherwise, since in each step the algorithm picks a vertex with highest level, each connected component C of $T - S$ and $w \in F \setminus F'$, $|N(w) \cap V(C)| \leq 1$. Algorithm $\mathcal{B}$ will output F' and S are the required sets. Notice that, in this case $|F'| = |S| = k' < k$. We have seen that there are $|F'|$ vertex disjoint cycles in G. This implies that $|F'| = |S| \leq OPT(G, k)$. An illustration is given in Figure 3. Figure 3a depicts a graph G with a feedback vertex set F and the sets F' and S chosen by the algorithm. In Figure 3b, the graph $G - (F' \cup S)$ is drawn separately to see the properties mentioned in the lemma.

□

Using Lemma 5.8, we will prove the following decomposition lemma and after this the structure of the reduced graph becomes "nice" and our algorithm boils down to applications of ϵ-ULISC on multiple auxiliary interval graphs.

Lemma 5.9. *Let (G, k) be an instance of* Disjoint Cycle Packing. *Then there is a polynomial time algorithm $\mathcal{A}$ which either outputs k vertex disjoint cycles or a minor G' of G, and $Z, R \subseteq V(G')$ with the following properties.*

(i) $OPT(G, k) = OPT(G', k)$,

(ii) $|Z| \leq OPT(G, k)$, $|R| = \mathcal{O}(k^4 \log^4 k)$,

(iii) $G' - (Z \cup R)$ is a collection $\mathcal{P}$ of $\mathcal{O}(k^4 \log^4 k)$ non trivial paths, and

(iv) for each path $P = u_1 \cdots u_r$ in $\mathcal{P}$, no internal vertex is adjacent to a vertex in R, $d_{G'[R \cup \{u_1\}]}(u_1) \leq 1$, and $d_{G'[R \cup \{u_r\}]}(u_r) \leq 1$.

Furthermore, given a cycle packing $\mathcal{S}'$ in G', there is a polynomial time algorithm $\mathcal{D}$ which outputs a cycle packing $\mathcal{S}$ in G such that $|\mathcal{S}| = |\mathcal{S}'|$.

Proof. We first give a description of the polynomial time algorithm $\mathcal{A}$ mentioned in the statement of the lemma. It starts by running the algorithm mentioned in Lemma 5.6 on input (G, k) and if it returns k vertex disjoint cycles, then $\mathcal{A}$ returns k vertex disjoint cycles in G and stops. Otherwise, let F be a feedback vertex set of G. Now $\mathcal{A}$ applies Reduction Rule 5.1 repeatedly using the feedback vertex set F until Reduction Rule 5.1 is no longer applicable. Let (G', k) be the reduced instance after the exhaustive application of Reduction Rule 5.1. By Lemma 5.7, we have that $F \subseteq V(G')$ and $G' - F$ is a forest. Now, $\mathcal{A}$ runs the algorithm $\mathcal{B}$ mentioned in Lemma 5.8 on input (G', k) and F. If $\mathcal{B}$ returns k vertex disjoint cycles in G', then $\mathcal{A}$ also returns k vertex disjoint cycles in G. The last assertion follows from the fact that Reduction Rule 5.1 (applied to get G') is 1-safe. Otherwise, let $F' \subseteq F$ and $S \subseteq V(G' - F)$ be the output of $\mathcal{B}$. Next we define a few sets that will be used by $\mathcal{A}$ to construct its output.

1. Let Q be the set of vertices of $G' - F$ whose degree in $G' - F$ is at least 3.
2. Let $O = \bigcup_{w \in F \setminus F'} N(w) \cap V(G' - F)$; and
3. let W be the vertices of degree 0 in $G' - (F \cup Q \cup O \cup S)$.

Algorithm $\mathcal{A}$ returns G', $Z = F'$ and $R = Q \cup O \cup S \cup W \cup (F \setminus F')$ as output. In the example given in Figure 3, $Z = \{z_1, z_2, z_3\}, F \setminus F' = \{f_1, f_2\}, S = \{s_1, s_2, s_3\}, Q = \{q_1, s_2, s_3\}, O = \{o_1, o_2\}$ and $W = \{w_1\}$.

Now we prove the correctness of the algorithm. If $\mathcal{A}$ outputs k vertex disjoint cycles in G, then we are done. Otherwise, let $Z = F'$ and $R = Q \cup O \cup S \cup W \cup (F \setminus F')$ be the output of $\mathcal{A}$. Now we prove G', Z and R indeed satisfy the properties mentioned in the statement of lemma. Since G' is obtained after repeated applications of Reduction Rule 5.1, by Observation 5.1, we get that $OPT(G, k) = OPT(G', k)$ and hence proving property (i).

By Lemma 5.8, we have that $|F'| = |S| \leq OPT(G, k)$. Hence the size of $Z (= F')$ is as desired. Next we bound the size of R. By Lemma 5.6, we have that $|F| \leq ck \log k$, where c is a fixed constant. By Lemma 5.7, we have that $F \subseteq V(G')$, $G' - F$ is a forest, and the number of vertices of degree at most 1 in $G' - F$ is upper bounded by $|F|^2(2|F| + 1) = \mathcal{O}(k^3 \log^3 k)$. Since the number of vertices of degree at least 3 in a forest is at most the number of leaves in the forest, we can conclude that cardinality of Q, the set of vertices of degree at least 3 in $G' - F$ is upper bounded by $\mathcal{O}(k^3 \log^3 k)$. It is well-known that the number of maximal degree 2 paths in a forest is upper bounded by the sum of the number of leaves and the vertices of degree at least 3 (for example see [50] for a proof). This immediately implies the following claim.

Claim 5.2. *$G' - (F \cup Q)$ is a collection of $\mathcal{O}(k^3 \log^3 k)$ paths.*

The following claim proves properties (ii) and (iii) stated in the lemma.

Claim 5.3. *$|R| = \mathcal{O}(k^4 \log^4 k)$ and the number of paths in $\mathcal{P}$ is at most $\mathcal{O}(k^4 \log^4 k)$.*

Proof. Observe that $G' - (F \cup Q)$ is a collection of $\mathcal{O}(k^3 \log^3 k)$ paths and thus it has at most $\mathcal{O}(k^3 \log^3 k)$ connected components. This implies that, $G' - (F \cup Q \cup S)$ has at most $\mathcal{O}(k^3 \log^3 k)$ connected components and in particular $G' - (F \cup S)$ has at most $\mathcal{O}(k^3 \log^3 k)$ connected components. Let γ_s denote the number of connected components of $G' - (F \cup S)$. By Lemma 5.8, we have that for any $w \in F \setminus F'$ and any connected component C of $G' - (F \cup S)$, $|N_{G'}(w) \cap V(C)| \leq 1$. Thus, for every vertex $w \in F \setminus F'$ we have that $|N(w) \cap V(G' - F)| \leq |S| + \gamma_s = \mathcal{O}(k^3 \log^3 k)$. This implies that the cardinality of O, the set $\bigcup_{w \in F \setminus F'} N(w) \cap V(G' - F)$, is upper bounded by $\mathcal{O}(|F \setminus F'| \cdot k^3 \log^3 k) = \mathcal{O}(k^4 \log^4 k)$. By Lemma 5.8, we have that $|S| \leq OPT(G, k) \leq k + 1$. Since $|O \cup S| = \mathcal{O}(k^4 \log^4 k)$ and by Claim 5.2, we can conclude that the number of paths in $G' - (F \cup Q \cup O \cup S)$ is at most $\mathcal{O}(k^4 \log^4 k)$. Notice that W is the family of paths on single vertices in the collection of paths of $G' - (F \cup Q \cup O \cup S)$. Since the number of maximal paths in $G' - (F \cup Q \cup O \cup S)$ is at most $\mathcal{O}(k^4 \log^4 k)$, we have that $|W| = \mathcal{O}(k^4 \log^4 k)$ and the number of maximal paths in $G' - (F \cup Q \cup O \cup S \cup W) = G' - (Z \cup R)$ (i.e, the number of paths in $\mathcal{P}$) is at most $\mathcal{O}(k^4 \log^4 k)$.

Since $|S| \leq OPT(G, k) \leq k + 1$, $|F| \leq ck \log k$, $|Q| = \mathcal{O}(k^3 \log^3 k)$, $|O \cup S| = \mathcal{O}(k^4 \log^4 k)$ and $|W| = \mathcal{O}(k^4 \log^4 k)$, we can conclude that the cardinality of $R = Q \cup O \cup S \cup W \cup (F \setminus F')$, is upper bounded by $\mathcal{O}(k^4 \log^4 k)$. This concludes the proof. ☐

Finally, we will show the last property stated in the lemma. Since $G' - F$ is a forest and Q is the set of vertices of degree at least 3 in the forest $G' - F$, we have that any internal vertex of any path in $G' - (Q \cup F)$ is not adjacent to Q. Also, since any vertex w, which is an internal vertex of a path in $G' - (Q \cup F)$ and adjacent to a vertex in $F \setminus F'$, belongs to O, we can conclude that no internal vertex of any path in $G' - (Q \cup O \cup F)$ is adjacent to $Q \cup O \cup (F \setminus F')$. This implies that no internal vertex of any path in $G' - (Q \cup O \cup S \cup W \cup F) = G' - (Z \cup R)$ is adjacent to $Q \cup O \cup S \cup W \cup (F \setminus F') = R$. Now we claim that an endpoint u of a path P in $\mathcal{P}$ has at most one edge between u and R. Let u be an endpoint of P. Since $O = \bigcup_{w \in F \setminus F'} N(w) \cap V(G' - F)$ and $u \notin O$, we can conclude that u is not adjacent to any vertex in $F \setminus F'$. Since $u \in V(G' - (F \cup Q))$, the degree of u in $G' - F$ is at most 2. Since P is a non trivial path $|N(u) \cap (V(G' - F) \setminus V(P))| \leq 1$. Since $G' - F$ is a forest, $|N(u) \cap (V(G' - F) \setminus V(P))| \leq 1$, and u is not adjacent to any vertex in $F \setminus F'$, we conclude that $d_{G'[R \cup \{u\}]}(u) \leq 1$.

The solution lifting algorithm, $\mathcal{D}$, is basically obtained by solution lifting algorithm used in the Reduction Rule 5.1. That is, given a cycle packing $\mathcal{S}'$ in G', $\mathcal{D}$ repeatedly applies the solution lifting algorithm of Reduction Rule 5.1 to obtain a cycle packing $\mathcal{S}$ in G such that $|\mathcal{S}| = |\mathcal{S}'|$. The correctness of the algorithm $\mathcal{D}$ follows from the fact that Reduction Rule 5.1 is 1-safe, and G' is obtained from G by repeated application of Reduction Rule 5.1. An illustration of a path $P \in \mathcal{P}$, Z and R can be found in Figure 4. This completes the proof of the lemma. ☐

Observe that Lemma 5.9 decomposes the graph into $k^{\mathcal{O}(1)}$ simple structures, namely, paths in $\mathcal{P}$ combined together with a set of size $k^{\mathcal{O}(1)}$. Note that the only *unbounded* objects in G' are the paths in $\mathcal{P}$. The reason we can not reduce the size of P is that a vertex in Z can have unbounded neighbors on it. See Figure 4 for an illustration. However, Lemma 5.9 still

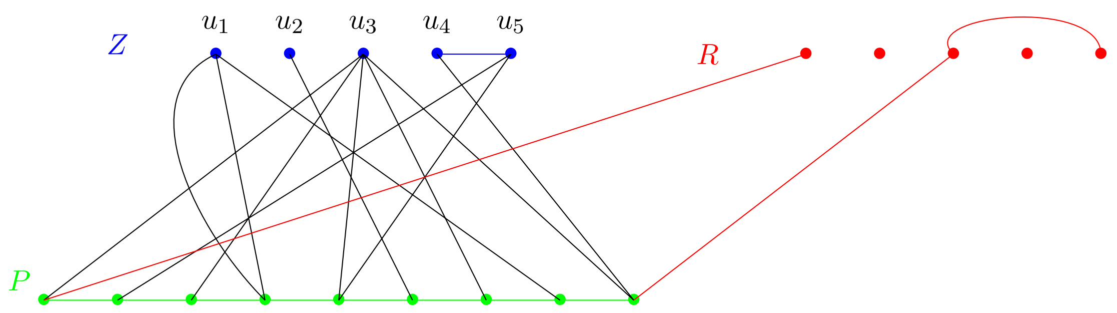


Figure 4: *An example of a path P in $\mathcal{P}$, Z and R*

provides us the required decomposition which will be used to cast several instances of ϵ-ULISC. In particular for every path $P \in \mathcal{P}$, we will have one instance of ϵ-ULISC. We will compute ϵ-ulisc for each of these instances and reduce the path size to get the desired kernel.

Theorem 4. *For any $\epsilon > 0$, there is polynomial sized $(1-\epsilon)$-approximate kernel for* Disjoint Cycle Packing. *That is,* Disjoint Cycle Packing *admits a PSAKS.*

Proof. Let (G, k) be an input instance of Disjoint Cycle Packing. The reduction algorithm $\mathcal{R}$ works as follows. It first runs the algorithm $\mathcal{A}$ mentioned in Lemma 5.9. If the algorithm returns k vertex disjoint cycles, then $\mathcal{R}$ return these cycles. Otherwise, let G', Z and R be the output of $\mathcal{A}$, satisfying four properties mentioned in Lemma 5.9. Important properties that will be most useful in our context are:

- $|Z| \leq OPT(G,k)$, $|R| = \mathcal{O}(k^4 \log^4 k)$; and
- $G' - (Z \cup R)$ is a collection $\mathcal{P}$ of non trivial paths such that for any path $P \in \mathcal{P}$ we have that no internal vertex of P is adjacent to any vertex of R.

Now $\mathcal{R}$ will solve several instances of ϵ-ULISC to bound the length of each path in $\mathcal{P}$. Towards this we fix a path

$$P = v_1 v_2 \dots v_\ell \text{ in } \mathcal{P}.$$

Our objective is to apply Lemma 5.4 to reduce the length of P. Our algorithm finds a set of small number of relevant vertices on P and reduces P in *a single step* even though we use Lemma 5.4 several times to identify relevant vertices. Next we give the construction for applying Lemma 5.4 in order to find the relevant vertices. To find relevant vertices of P, we create $(|Z|+1)^2$ labelled interval graphs, one for every $(x,y) \in Z \cup \{\clubsuit\} \times Z \cup \{\clubsuit\}$ with $Z \times Z$ being the set of labels. That is, for the path P and $(x,y) \in Z \cup \{\clubsuit\} \times Z \cup \{\clubsuit\}$ we create a labelled interval graph $H_P^{(x,y)}$ as follows. Our labelling function will be denoted by $\Gamma_P^{(x,y)}$.

1. The set of labels is $\Sigma = Z \times Z$.
2. Let $P^{(x,y)} = v_r \dots v_{r'}$ be the subpath of P such that v_{r-1} is the first vertex in P adjacent to x and $v_{r'+1}$ is the last vertex on P adjacent to y. If $x = \clubsuit$, then $v_r = v_1$ and if $y = \clubsuit$, then $v_{r'} = v_\ell$. Indeed, if $x = \clubsuit$ and $y = \clubsuit$ then $v_r = v_1$ and $v_{r'} = v_\ell$.
3. We say that a subpath Q' of $P^{(x,y)}$ is a *potential* (u_1, u_2)*-subpath*, where $(u_1, u_2) \in Z \times Z$, if either $u_1 Q' u_2$ or $u_2 Q' u_1$ is an induced path (induced cycle when $u_1 = u_2$) in G'. Essentially, the potential subpath is trying to capture the way a cycle can interact with a subpath in P with its neighbors on the cycle being u_1 and u_2.
4. For each $(u_1, u_2) \in Z \times Z$ and a potential (u_1, u_2)-subpath $Q' = v_i \dots v_j$ we create an interval $I_{Q'}^{(u_1,u_2)} = [i, j]$ and label it with (u_1, u_2). That is, $\Gamma_P^{(x,y)}(I_{Q'}^{(u_1,u_2)}) = (u_1, u_2)$. We would

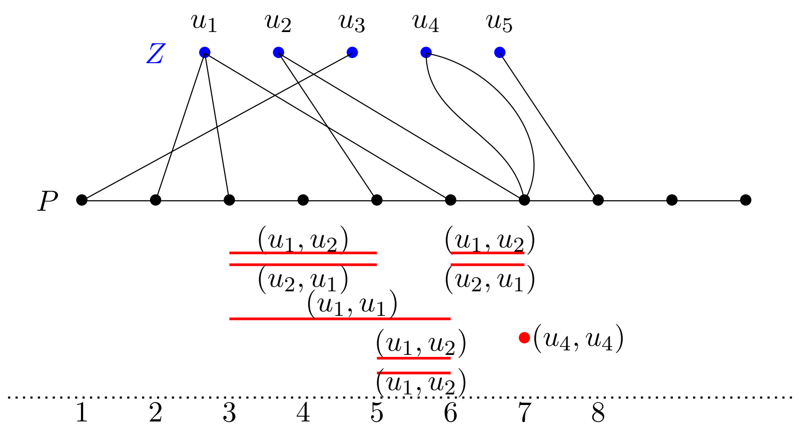


Figure 5: *An example of $H_P^{(u_1,u_5)}$. The interval representation of $H_P^{(u_1,u_5)}$ along with labels is drawn below the path P. The real line is represented using a dotted line.*

like to emphasize that when $u_1 = u_2$ and $v_i = v_j$, we create an interval $I_{Q'}^{(u_1,u_2)} = [i,j]$ only if there are two edges between u_1 and v_i. Also notice that if we have created an interval $I_{Q'}^{(u_1,u_2)} = [i,j]$ with label (u_1,u_2), then we have created an interval $I_{Q'}^{(u_2,u_1)} = [i,j]$ with label (u_2,u_1) as well.

This completes the construction of $H_P^{(x,y)}$ and the labelling function $\Gamma_P^{(x,y)}$. See Figure 5 for an illustration. The fact that $H_P^{(x,y)}$ is an interval graph follows from the fact that in fact to construct $H_P^{(x,y)}$, we have given an interval representation for it. Having, created the interval graph and a labelling function $\mathcal{R}$ runs the following steps.

1. Now using Lemma 5.9, $\mathcal{R}$ computes a set $X_P^{(x,y)}$ such that $X_P^{(x,y)}$ is a $\frac{\epsilon}{2}$-ulisc of $H_P^{(x,y)}$. Now we define a few sets.

$$\begin{aligned}
S_P^{(x,y)} &= \{v_i : i \text{ is an endpoint of an interval in } H_P^{(x,y)}\}\\
K_P &= \{v_1, v_\ell\} \cup \bigcup_{u\in Z} \{v : v \text{ is the first or last vertex on } P \text{ such that } uv \in E(G')\}\\
S_P &= \bigcup_{(x,y)\in Z\cup\{\clubsuit\}\times Z\cup\{\clubsuit\}} S_P^{(x,y)}\\
D_P &= V(P) \setminus (S_P \cup K_P).
\end{aligned}$$

2. Now, $\mathcal{R}$ will do the following modification to shorten P: delete all the edges between D_P and Z, and then contract all the remaining edges incident with vertices in D_P. In other words, let $\{v_{i_1}, \ldots v_{i_{\ell'}}\} = S_P \cup K_P$, where $1 = i_1 < i_2 < \ldots < i_{\ell'} = \ell$. Then delete D_P and add edges $v_{i_j}v_{i_{j+1}}, j \in [\ell'-1]$. Let P' be the path obtained from P, by the above process. We use the same vertex names in P' as well to represent a vertex. That is, if a vertex u in $V(P)$ is not deleted to obtain P', we use u to represent the same vertex.
3. Let G'' be the graph obtained after this modification has been done for all paths $P \in \mathcal{P}$. Finally, $\mathcal{R}$ returns (G'', k) as the reduced instance.

Solution Lifting Algorithm. Notice that G'' is a minor of G' and hence a minor of G. Given a set S' of vertex disjoint cycles in G'', the solution lifting algorithm computes a set S of vertex disjoint cycles in G of cardinality $|S'|$ by doing reverse of the minor operations used to obtain G'' from G. All this can be done in polynomial time because the solution lifting algorithm knows the minor operations done to get G'' from G.

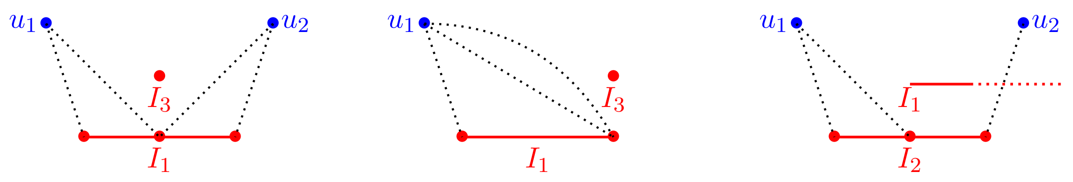


Figure 6: *Illustration of proof of Claim 5.4. The case when $I_3 = [p, p]$ is drawn in the left and middle figures. The figure in the middle represents the case when $j_1 = p$ and $u_1 = u_2$. The case when I_1 and I_2 intersects at strictly more than one point can be seen in the right most figure. The black dotted curves represent the edges in the graph.*

Next we need to prove the correctness of the algorithm. Towards that we first bound the size of G''.

Bounding the size of G''. As a first step to bound the size of G'', we bound the chromatic number of $H_P^{(x,y)}$, where $P \in \mathcal{P}$ and $(x, y) \in Z \cup \{\clubsuit\} \times Z \cup \{\clubsuit\}$. In fact what we will bound is the size of the maximum clique of $H_P^{(x,y)}$.

Claim 5.4. *For any $P \in \mathcal{P}$ and $(x, y) \in Z \cup \{\clubsuit\} \times Z \cup \{\clubsuit\}$, $\chi(H_P^{(x,y)}) = \mathcal{O}(k^2)$.*

Proof. To prove the claim, it is enough to show that the size of a maximum clique in $H_P^{(x,y)}$ is at most $\mathcal{O}(k^2)$. Let $P^{(x,y)} = v_r \dots v_{r'}$. We know that in the interval representation of $H_P^{(x,y)}$, all the intervals are contained in $[r, r']$. We claim that for any point $p \in [r, r']$ and $(u_1, u_2) \in Z \times Z$, the number of intervals labelled (u_1, u_2) and containing the point p is at most 2. Towards a contradiction assume that there are three intervals $I_1 = [i_1, j_1], I_2 = [i_2, j_2], I_3 = [i_3, j_3]$ such that $\Gamma_P^{(x,y)}(I_1) = \Gamma_P^{(x,y)}(I_2) = \Gamma_P^{(x,y)}(I_3) = (u_1, u_2)$ and all the intervals I_1, I_2 and I_3 contain the point p. Since for each $r \le i, j \le r'$ and $(u_1, u_2) \in Z \times Z$ we have created at most one interval $[i, j]$ with label (u_1, u_2), all the intervals I_1, I_2 and I_3 are distinct intervals in the real line.

We first claim that no interval in $\{I_1, I_2, I_3\}$ is same as $[p, p]$. Suppose $I_3 = [p, p]$. Since all the interval in $\{I_1, I_2, I_3\}$, are different and $I_3 = [p, p]$ we have that $I_1 \neq [p, p]$, but contains p. This implies that either $i_1 \neq p$ or $j_1 \neq p$. We consider the case $i_1 \neq p$. The case that $j_1 \neq p$ is symmetric. Let $Q_1 = v_{i_1} v_{i_1+1} \dots v_{j_1}$. We know that $u_1 v_p u_2$ is an induced path (induced cycle when $u_1 = u_2$ and two edges between u_1 and p). This implies that neither $u_1 Q_1 u_2$ nor $u_2 Q_1 u_1$ is an induced path, because $v_p \in \{v_{i_1+1} \dots v_{j_1}\}$. We would like to clarify that when $j_1 = p$ and $u_1 = u_2$, $u_1 Q_1 u_1$ is cycle and there are two edges between v_{j_1} and u_1. This implies that $u_1 Q_1 u_1$ is a not an induced cycle. See Figure 6 for illustration.

Since p is a common point in I_1, I_2 and I_3 and none of these intervals is equal to $[p, p]$, there are two intervals in $\{I_1, I_2, I_3\}$ such that they intersect at strictly more than one point. Without loss of generality we assume that the intersection of I_1 and I_2 contains at least 2 points. Also, since I_1 and I_2 are different intervals on the real line, one endpoint of an interval is fully inside another interval (not as the endpoint of the other interval). Let $Q_2 = v_{i_2} v_{i_2+1} \dots v_{j_2}$. Assume that $i_1 \in (i_2, j_2)$. All other cases are symmetric to this case. We know that $u_1 v_{i_1} \in E(G')$ or $u_2 v_{i_1} \in E(G')$. This implies that neither $u_1 Q_2 u_2$ nor $u_2 Q_2 u_1$ is an induced path. This contradicts the fact that we created an interval $[i_2, j_2]$ with label (u_1, u_2). See Figure 6 for illustration.

We have proved that for any point $p \in [r, r']$, the number of intervals containing p with the same label is upper bounded by 2. This implies that the cardinality of a largest clique in $H_P^{(x,y)}$ is at most twice the number of labels. Thus, the size of the largest cliques is upper bounded by $\mathcal{O}(|Z|^2)$. By Lemma 5.9, we know that $|Z| \le OPT(G, k) \le k + 1$ and thus $\mathcal{O}(|Z|^2)$ is bounded by $\mathcal{O}(k^2)$. Since the chromatic number of an interval graph is upper bounded by the size of a maximum clique, the proof of the claim follows. $\square$

By Lemma 5.9, we know that $|\mathcal{P}| = \mathcal{O}(k^4 \log^4 k)$. For each $P \in \mathcal{P}$ and $(x, y) \in Z \cup \{\clubsuit\} \times$

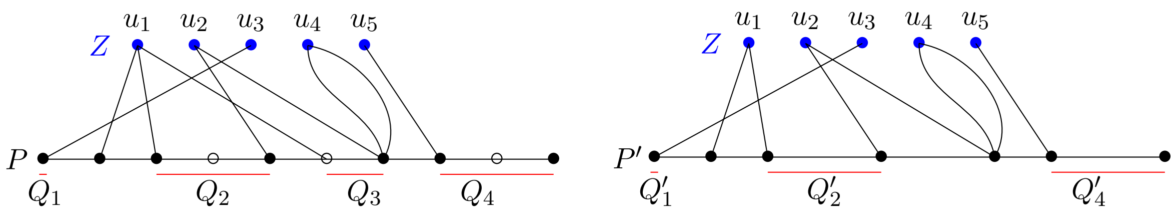


Figure 7: *An example of intersection of $\mathcal{C}$ and a path $P \in \mathcal{P}$. The vertices in P colored black belong to $V(P')$. The intersection of $\mathcal{C}$ and P are the set of paths $\mathcal{Q} = \{Q_1, Q_2, Q_3, Q_4\}$. Here $Q_2, Q_3 \in \mathcal{Q}_1$ and $Q_1, Q_4 \in \mathcal{Q}_2$. The substitute paths for $\mathcal{Q}$ is in the figure at the right hand side*

$Z \cup \{\clubsuit\}$, we created a subset $S_P^{(x,y)}$ of $V(P)$ of cardinality $(|Z|^2 \cdot \chi(H_P^{(x,y)})^{\mathcal{O}(\frac{2}{\epsilon}\log\frac{2}{\epsilon})} = k^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$. Hence, the cardinality of S_P is also upper bounded by $k^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$. The cardinality of K_P is at most $2|Z|+2 = \mathcal{O}(k)$. This implies that the reduced path P' has at most $k^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$ vertices. Also, we know that $|\mathcal{P}| = \mathcal{O}(k^4 \log^4 k)$, hence the total number of vertices across all the paths of $\mathcal{P}$ after the reduction is upper bounded by $k^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$. This together with the fact that $|Z| \leq OPT(G,k)$ and $|R| = \mathcal{O}(k^4 \log^4 k)$ imply that $|V(G'')|$ is upper bounded by $k^{\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})}$. This completes the proof of upper bound on the size of G''.

Correctness of lossy reduction. Finally, we show that indeed (G'', k) is a $(1-\epsilon)$-approximate kernel for Disjoint Cycle Packing. Towards this we show the following claim.

Claim 5.5. *$OPT(G'', k) \geq (1-\epsilon)OPT(G', k)$.*

Proof. Let $\mathcal{C}$ be an optimum solution to (G', k). Without loss of generality we can assume that each cycle in $\mathcal{C}$ is a *chordless cycle*. Let $\mathcal{Q}$ be the non-empty subpaths of cycles in $\mathcal{C}$ induced in the graph $G' - (Z \cup R)$. That is, $\mathcal{Q}$ is the collection of supaths in the intersection of $\mathcal{C}$ and $\mathcal{P}$. For any $Q \in \mathcal{Q}$, there exists two vertices $u, v \in R \cup Z$ such that uQv is a subpath in $\mathcal{C}$. Because of property (iv) of Lemma 5.9, for any $Q \in \mathcal{Q}$ with $|V(Q)| = 1$, at least one of the endpoint of Q is connected to a vertex from Z in the cycle packing $\mathcal{C}$. We say a path Q' is a *substitute* for $Q \in \mathcal{Q}$ if $uQ'v$ is a subpath in G'' where $u, v \in R \cup Z$ and uQv is a subpath in $\mathcal{C}$. In what follows, for at least $(1-\epsilon)|\mathcal{Q}|$ paths in $\mathcal{Q}$, we identify substitutes in the reduced graph G'' which are pairwise vertex disjoint.

We partition the paths in $\mathcal{Q}$ into $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Notice that $Q_i \in \mathcal{Q}$ is a subpath of a cycle $C \in \mathcal{C}$ and the neighbors (could be the same) of both the endpoints of Q_i on C are in $R \cup Z$. If the neighbors of both endpoints of Q_i on C are in Z, then we include Q_i in $\mathcal{Q}_1$. Otherwise Q_i is in $\mathcal{Q}_2$. See Figure 7 for an illustration. For each $Q \in \mathcal{Q}_2$, we give a substitute path as follows. We know that there is a path $P \in \mathcal{P}$ such that either $P = QQ'$ or $P = Q'Q$ for some Q' where $V(Q')$ can be an empty set too. If $Q = P$, then we replace Q with P' (Note that P' is the path obtained from P in the reduction process). Also, notice that end vertices of P and P' are same (because endvertices of P belong to K_P) and hence P' is a substitute for Q. Suppose $P = QQ'$ where $V(Q') \neq \emptyset$. Let C_Q be the cycle in $\mathcal{C}$ such that Q is a subpath of C_Q. Let $Q = v_1 \ldots v_d$. Let z be the neighbour of v_d in C_Q which is from Z (recall that no internal vertex of P is adjacent to any vertex of R). Since C_Q is a chordless cycle, none of $v_1, \ldots, v_{d-1}$ is adjacent to z. This implies that $v_1, v_d \in K_P$ and hence P' contains a subpath P'_Q from v_1 to v_{d-1} with internal vertices from $\{v_2, \ldots, v_{d-1}\}$. In this case P'_Q is a substitute for Q. In a similar way, we can construct a substitute for Q when $P = Q'Q$ where $V(Q') \neq \emptyset$. Let $\mathcal{Q}'_2$ be the set of substitute paths constructed for paths in $\mathcal{Q}_2$. Notice that $\mathcal{Q}'_2$ is a collection of vertex disjoint paths in $G'' - (Z \cup R)$ and it has one substitute path for each $Q \in \mathcal{Q}_2$. See Figure 7 for an illustration.

Now we construct substitute paths for $\mathcal{Q}_1$. Here, we construct substitute paths for at least $(1-\epsilon)|\mathcal{Q}_1|$ paths and these paths will be vertex disjoint. Moreover, these paths will be vertex disjoint from the paths in $\mathcal{Q}'_2$ as well. Let P be a path in $\mathcal{P}$ such that at least one path in $\mathcal{Q}_1$ is a subpath of P. Let $\mathcal{Q}_1(P) \subseteq \mathcal{Q}_1$ be a subset of $\mathcal{Q}_1$ containing all the paths in $\mathcal{Q}_1$ that is a subpath of P. There are at most two paths in $\mathcal{Q}_2$ which are subpaths of P. Let F and L be these paths, where F and L could be empty too. Let the neighbours of F and L in Z in the cycle packing $\mathcal{C}$ be x and y, respectively (here, $x = \clubsuit$ if $F = \emptyset$ and $y = \clubsuit$ if $L = \emptyset$). Then, consider the following decomposition of path $P = FP^{\star}L$. We claim that $P^{\star} = P^{(x,y)}$. That is, $P^{\star}$ is a path for which we would have created the interval graph $H_P^{(x,y)}$. Observe that, if F is non-empty then x does not have any neighbor on F as cycles in $\mathcal{C}$ are *chordless*. Similarly, if L is non-empty then y does not have any neighbor on L. Thus, if F and L are both non-empty then indeed the *last* vertex of F is the *first* vertex on P that is a neighbor of x and the *first* vertex of L is the *last* vertex on P that is a neighbor of y. This implies that indeed we would have created the interval graph $H_P^{(x,y)}$. We can argue similarly if either F is empty or L is empty. Now consider the interval graph $H_P^{(x,y)}$. This graph is constructed from $P^{(x,y)}$. Since each cycle in $\mathcal{C}$ is a chordless cycle, we have that each subpath $Q \in \mathcal{Q}_1(P)$ is a potential (u_1, u_2)-subpath of $P^{(x,y)}$ where either $u_1 Q u_2$ or $u_2 Q u_1$ is a subpath of $\mathcal{C}$ and $u_1, u_2 \in Z$. Since each vertex in $V(\mathcal{C})$ has degree two in $\mathcal{C}$, for a pair (u_1, u_2) we have *at most two potential* (u_1, u_2) *paths* Q_1 and Q_2 in $\mathcal{Q}_1(P)$. Also note that these supaths Q_1 and Q_2 are potential (u_2, u_1)-subpaths as well. So when there are two paths $Q_1, Q_2 \in \mathcal{Q}_1(P)$ such that $u_1 Q_1 u_2$ and $u_1 Q_2 u_2$ are subpaths of $\mathcal{C}$, then we consider Q_1 as a potential (u_1, u_2)-subpath and Q_2 as a potential (u_2, u_1)-subpath. Now we can consider $\mathcal{Q}_1(P)$ as a set of potential subpaths of $P^{(x,y)}$. That is, for each $Q \in \mathcal{Q}_1(P)$, there is an interval $I_Q^{(u_1,u_2)}$ with label (u_1, u_2) and (u_1, u_2) is not a label of any other intervals corresponding to a subpath in $\mathcal{Q}_1(P) \setminus \{Q\}$. Let $\mathcal{I}_1(P)$ be the set of interval created for the potential subpaths in $\mathcal{Q}_1(P)$. We have explained that for any $(u_1, u_2) \in Z \times Z$, there is at most one potential (u_1, u_2)-subpath in $\mathcal{Q}_1(P)$. Also notice that, since $\mathcal{Q}_1(P)$ is a collection of vertex disjoint paths, the interval constructed for corresponding potential subpaths are disjoint. This implies that $\mathcal{I}_1(P)$ is an independent set in $H_P^{(x,y)}$ and $|\Gamma_P^{(x,y)}(\mathcal{I}_1(P))| = |\mathcal{I}_1(P)|$. By Lemma 5.4, we have that there is a subset $\Sigma' \subseteq \Gamma_P^{(x,y)}(\mathcal{I}_1(P))$ such that there is an independent set S of cardinality $(1-\frac{\epsilon}{2})|\mathcal{I}_1(P)|$ in $X_P^{(x,y)}$ and $\Gamma_P^{(x,y)}(S) = \Sigma'$. This implies that there are at least $(1-\frac{\epsilon}{2})|\mathcal{I}_1(P)| = (1-\frac{\epsilon}{2})|\mathcal{Q}_1(P)|$ of paths in $\mathcal{Q}_1(P)$ has *substitute paths* in P' which are vertex disjoint from F and L, where P' is the path obtained from P in the reduction process using Lemma 5.4. This implies that for each $P \in \mathcal{P}$, at least $(1-\frac{\epsilon}{2})|\mathcal{Q}_1(P)|$ paths has substitute paths in G'' and they are vertex disjoint subpaths of P' and does not intersect with F and L. We denote the set of substitute paths in P' by $\mathcal{Q}_1'(P')$. This implies that the substitute paths for $\mathcal{Q}_1$ are vertex disjoint and they are vertex disjoint from the substitute paths for $\mathcal{Q}_2$. Let these substitute paths form a set $\mathcal{Q}_1' = \cup_{P\in\mathcal{P}} \mathcal{Q}_1'(P')$. Also notice that since each vertex $u \in Z$, has degree at most 2 in $\mathcal{C}$ and $|Z| \leq OPT(G', k)$, the total number of paths in $\mathcal{Q}_1$ is at most $2OPT(G', k)$. From each $\mathcal{Q}_1(P)$, at least $(1-\frac{\epsilon}{2})|\mathcal{Q}_1(P)|$ paths have substitute paths in G''. Recall that,

$$\mathcal{Q}_1 = \biguplus_{P\in\mathcal{P}} \mathcal{Q}_1(P) \text{ and } \mathcal{Q}_1' = \biguplus_{P\in\mathcal{P}} \mathcal{Q}_1'(P').$$

That is, $\mathcal{Q}_1$ ($\mathcal{Q}_1'$) is the disjoint union of $\mathcal{Q}_1(P)$ ($\mathcal{Q}_1'(P')$) for $P \in \mathcal{P}$. Thus,

$$\begin{aligned} |\mathcal{Q}_1| - |\mathcal{Q}_1'| &= \sum_{P\in\mathcal{P}} |\mathcal{Q}_1(P)| - |\mathcal{Q}_1'(P')| \\ &\leq \sum_{P\in\mathcal{P}} |\mathcal{Q}_1(P)| - \left(1 - \frac{\epsilon}{2}\right)|\mathcal{Q}_1(P)| \\ &= \left(\sum_{P\in\mathcal{P}} \frac{\epsilon}{2}|\mathcal{Q}_1(P)|\right) = \frac{\epsilon}{2}|\mathcal{Q}_1|. \end{aligned}$$

This implies that $|\mathcal{Q}_1| - |\mathcal{Q}_1'| \leq \frac{\epsilon}{2}|\mathcal{Q}_1| \leq \epsilon OPT(G', k)$. This implies that $\mathcal{Q}_1' \cup \mathcal{Q}_2'$ contains at least $(1-\epsilon)OPT(G', k)$ substitute paths. Each path in $\mathcal{Q}$ for which we do not have a substitute path can destroy at most one cycle in $\mathcal{C}$. Recall that, $\mathcal{C}$ is an optimum solution to (G', k). This implies that G'' contains at least $(1-\epsilon)OPT(G', k)$ vertex disjoin cycles. This completes the proof of the claim. $\square$

By Lemma 5.9, we know that $OPT(G', k) = OPT(G, k)$ and hence by Claim 5.5 we get that $OPT(G'', k) \geq (1-\epsilon)OPT(G, k)$. We know that given a solution $\mathcal{S}'$ of (G'', k) the solution lifting algorithm will output a solution $\mathcal{S}$ of same cardinality for (G, k). Therefore, we have

$$\frac{|\mathcal{S}|}{OPT(G,k)} \geq (1-\epsilon)\frac{|\mathcal{S}'|}{OPT(G'',k)}.$$

This gives the desired PSAKS for Disjoint Cycle Packing and completes the proof. $\square$

6 Approximate Kernelization in Previous Work

In this section we show how some of the existing approximation algorithms and FPT approximation algorithms can be re-interpreted as first computing an α-approximate kernel, and then running a brute force search or an approximation algorithm on the reduced instance.

6.1 Partial Vertex Cover

In the Partial Vertex Cover problem the input is a graph G on n vertices, and an integer k. The task is to find a vertex set $S \subseteq V(G)$ of size k, maximizing the number of edges with at least one end-point in S. We will consider the problem parameterized by the solution *size* k. Note that the solution size is *not* the objective function value. We define Partial Vertex Cover as a parameterized optimization problem as follows.

$$PVC(G,k,S) = \begin{cases} -\infty & |S| > k \\ \text{Number of edges incident on } S & \text{Otherwise} \end{cases}$$

Partial Vertex Cover is W[1]-hard [35], thus we do not expect an FPT algorithm or a kernel of *any* size to exist for this problem. On the other hand, Marx [45] gave a $(1+\epsilon)$-approximation algorithm for the problem with running time $f(k,\epsilon)n^{\mathcal{O}(1)}$. We show here that the approximation algorithm of Marx [45] can be re-interpreted as a PSAKS.

Theorem 5. Partial Vertex Cover *admits a strict time and size efficient PSAKS.*

Proof. We give an α-approximate kernelization algorithm for the problem for every $\alpha > 1$. Let $\epsilon = 1 - \frac{1}{\alpha}$ and $\beta = \frac{1}{\epsilon}$. Let (G,k) be the input instance. Let $v_1, v_2, \ldots, v_n$ be the vertices of G in the non-increasing order of degree, i.e $d_G(v_i) \geq d_G(v_j)$ for all $1 \geq i > j \geq n$. The kernelization algorithm has two cases based on degree of v_1.
Case 1: $d_G(v_1) \geq \beta\binom{k}{2}$. In this case $S = \{v_1, \ldots, v_k\}$ is a α-approximate solution. The number of edges incident to S is at least $(\sum_{i=1}^{k} d_G(v_i)) - \binom{k}{2}$, because at most $\binom{k}{2}$ edges have both end points in S and they are counted twice in the sum $(\sum_{i=1}^{k} d_G(v_i))$. The value of the optimum solution is at most $\sum_{i=1}^{k} d_G(v_i)$. Now consider the value, $PVC(G,k,S)/OPT(G,k)$.

$$\frac{PVC(G,k,S)}{OPT(G,k)} \geq \frac{(\sum_{i=1}^{k} d_G(v_i)) - \binom{k}{2}}{\sum_{i=1}^{k} d_G(v_i)} \geq 1 - \frac{\binom{k}{2}}{d_G(v_1)} \geq 1 - \frac{1}{\beta} = \frac{1}{\alpha}$$

The above inequality implies that S is an α-approximate solution. So the kernelization algorithm outputs a trivial instance $(\emptyset, 0)$ in this case.

Case 2: $d_G(v_1) < \beta\binom{k}{2}$**.** Let $V' = \{v_1, v_2, \ldots, v_{k\lceil\beta\binom{k}{2}\rceil+1}\}$. In this case the algorithm outputs (G', k), where $G' = G[N_G[V']]$. We first clam that $OPT(G', k) = OPT(G, k)$. Since G' is a subgraph of G, $OPT(G', k) \leq OPT(G, k)$. Now it is enough to show that $OPT(G', k) \geq OPT(G, k)$. Towards that, we prove that there is an optimum solution that contains only vertices from the set V'. Suppose not, then consider the solution S which is lexicographically smallest in the ordered list $v_1, \ldots v_n$. The set S contains at most $k-1$ vertices from V' and at least one from $V \setminus V'$. Since degree of each vertex in G is at most $\lceil\beta\binom{k}{2}\rceil - 1$ and $|S| \leq k$, we have that $|N_G[S]| \leq k\lceil\beta\binom{k}{2}\rceil$. This implies that there exists a vertex $v \in V'$ such that $v \notin N_G[S]$. Hence by including the vertex v and removing a vertex from $S \setminus V'$, we can cover at least as many edges as S can cover. This contradicts our assumption that S is lexicographically smallest. Since G' is a subgraph of G any solution of G' is also a solution of G. Thus we have shown that $OPT(G', k) = OPT(G, k)$. So the algorithm returns the instance (G', k) as the reduced instance. Since G' is a subgraph of G, in this case, the solution lifting algorithm takes a solution S' of (G', k) as input and outputs S' as a solution of (G, k). Since $OPT(G', k) = OPT(G, k)$, it follows that $\frac{PVC(G,k,S')}{OPT(G,k)} = \frac{PVC(G',k,S')}{OPT(G',k)}$.

The number of vertices in the reduced instance is $\mathcal{O}(k \cdot \lceil\frac{1}{\epsilon}\binom{k}{2}\rceil^2) = \mathcal{O}(k^5)$. The running time of the algorithm is polynomial in the size of G. Since the algorithm either finds an α-approximate solution (Case 1) or reduces the instance by a 1-safe reduction rule (Case 2), this kernelization scheme is strict. □

6.2 Steiner Tree

In the Steiner Tree problem we are given as input a graph G, a subset R of $V(G)$ called the *terminals* and a weight function $w : E(G) \to \mathbb{N}$. A *Steiner tree* is a subtree T of G such that $R \subseteq V(T)$, and the *cost* of a tree T is defined as $w(T) = \sum_{e \in E(T)} w(e)$. The task is to find a Steiner tree of minimum cost. We may assume without loss of generality that the input graph G is complete and that w satisfies the triangle inequality: for all $u, v, w \in V(G)$ we have $w(uw) \leq w(uv) + w(vw)$. This assumption can be justified by adding for every pair of vertices u,v the edge uv to G and making the weight of uv equal the shortest path distance between u and v. If multiple edges are created between the same pair of vertices, only the lightest edge is kept.

Most approximation algorithms for the Steiner Tree problem rely on the notion of a k-restricted Steiner tree, defined as follows. A *component* is a tree whose leaves coincide with a subset of terminals, and a k-component is a component with at most k leaves. A k-restricted Steiner tree $\mathcal{S}$ is a collection of k-components, such that the union of these components is a Steiner tree T. The cost of $\mathcal{S}$ is the sum of the costs of all the k-components in $\mathcal{S}$. Thus an edge that appears in several different k-components of $\mathcal{S}$ will contribute several times to the cost of $\mathcal{S}$, but only once to the cost of T. The following result by Borchers and Du [9] shows that for every $\epsilon > 0$ there exists a k such that the cost of the best k-restricted Steiner tree $\mathcal{S}$ is not more than $(1+\epsilon)$ times the cost of the best Steiner tree. Thus approximation algorithms for Steiner Tree only need to focus on the best possible way to "piece together" k-components to connect all the terminals.

Proposition 6.1 ([9])**.** *For every $k \geq 1$, graph G, terminal set R, weight function $w : E(G) \to \mathbb{N}$ and Steiner tree T, there is a k-restricted Steiner Tree $\mathcal{S}$ in G of cost at most $(1 + \frac{1}{\lfloor \log_2 k \rfloor}) \cdot w(T)$.*

Proposition 6.1 can easily be turned into a PSAKS for Steiner Tree parameterized by the number of terminals, defined below.

$$ST((G,R),k',T) = \begin{cases} -\infty & \text{if } |R| > k' \\ \infty & \text{if } T \text{ is not a Steiner tree for } R \\ w(T) & \text{otherwise} \end{cases}$$

To get a $(1+\epsilon)$-approximate kernel it is sufficent to pick k based on ϵ, compute for each k-sized subset $R' \subseteq R$ of terminals an optimal Steiner tree for R', and only keep vertices in G that appear in these Steiner trees. This reduces the number of vertices of G to $\mathcal{O}(|R|^k)$, but the edge weights can still be large making the bitsize of the kernel super-polynomial in $|R|$. However, it is quite easy to show that keeping only $\mathcal{O}(\log |R|)$ bits for each weight is more than sufficient for the desired precision.

Theorem 6. Steiner Tree *parameterized by the number of terminals admits a PSAKS.*

Proof. Start by computing a 2-approximate Steiner tree T_2 using the classic factor 2 approximation algorithm [55]. For every vertex $v \notin R$ such that $\min_{x\in R} w(vx) \geq w(T_2)$ delete v from G as v may never participate in any optimal solution. By the triangle inequality we may now assume without loss of generality that for every edge $uv \in E(G)$ we have $w(uv) \leq 6OPT(G, R, w)$.

Working towards a $(1+\epsilon)$-approximate kernel of polynomial size, set k to be the smallest integer such that $\frac{1}{\lfloor \log_2 k \rfloor} \leq \epsilon/2$. For each subset R' of R of size at most k, compute an optimal steiner tree $T_{R'}$ for the instance (G, R', w) in time $\mathcal{O}(3^k|E(G)||V(G)|)$ using the algorithm of Dreyfus and Wagner [24]. Mark all the vertices in $V(T_{R'})$. After this process is done, some $\mathcal{O}(k|R|^k)$ vertices in G are marked. Obtain G' from G by deleting all the unmarked vertices in $V(G) \setminus R$. Clearly every Steiner tree in G' is also a Steiner tree in G, we argue that $OPT(G', R, w) \leq (1+\frac{\epsilon}{2})OPT(G, R, w)$.

Consider an optimal Steiner tree T for the instance (G, R, w). By Proposition 6.1 there is a k-restricted Steiner Tree $\mathcal{S}$ in G of cost at most $(1+\frac{1}{\lfloor \log_2 k \rfloor}) \cdot w(T) \leq (1+\frac{\epsilon}{2})OPT(G, R, w)$. Consider a k-component $C \in \mathcal{S}$, and let R' be the set of leaves of C - note that these are exactly the terminals appearing in C. C is a Steiner tree for R', and so $T_{R'}$ is a Steiner tree for R' with $w(T_{R'}) \leq w(C)$. Then $\mathcal{S}' = (\mathcal{S} \setminus \{C\}) \cup \{T_{R'}\}$ is a k-restricted Steiner Tree of cost no more than $(1+\frac{\epsilon}{2})OPT(G, R, w)$. Repeating this argument for all k-components of $\mathcal{S}$ we conclude that there exists a k-restricted Steiner Tree $\mathcal{S}$ in G of cost at most $(1+\frac{\epsilon}{2})OPT(G, R, w)$, such that all k-components in $\mathcal{S}$ only use marked vertices. The union of all of the k-components in $\mathcal{S}$ is then a Steiner tree in G' of cost at most $(1+\frac{\epsilon}{2})OPT(G, R, w)$.

We now define a new weight function $\hat{w} : E(G') \to \mathbb{N}$, by setting

$$\hat{w}(e) = \left\lfloor w(e) \cdot \frac{4|R|}{\epsilon \cdot OPT(G, R, w)} \right\rfloor$$

Note that since $w(e) \leq 6 \cdot OPT(G, R, w)$ it follows that $\hat{w}(e) \leq \frac{24|R|}{\epsilon}$. Thus it takes only $\mathcal{O}(\log |R| + \log \frac{1}{\epsilon})$ bits to store each edge weight. It follows that the bitsize of the instance $(G', R, \hat{w})$ is $|R|^{2^{\mathcal{O}(1/\epsilon)}}$. We now argue that, for every $c \geq 1$, a c-approximate Steiner tree T' for the instance $(G', R, \hat{w})$ is also a $c(1+\epsilon)$-approximate Steiner tree for the instance (G, R, w).

First, observe that the definition of $\hat{w}$ implies that for every edge e we have the inequality

$$w(e) \leq \hat{w}(e) \cdot \frac{\epsilon \cdot OPT(G, R, w)}{4|R|} + \frac{\epsilon \cdot OPT(G, R, w)}{4|R|}.$$

In a complete graph that satisfies the triangle inequality, a Steiner tree on $|R|$ terminals has at most $|R| - 1$ non-terminal vertices. Thus it follows that T' has at most $2|R|$ edges. Therefore,

$$w(T') \leq \hat{w}(T') \cdot \frac{\epsilon \cdot OPT(G, R, w)}{4|R|} + \frac{\epsilon}{2} OPT(G, R, w).$$

Consider now an optimal Steiner tree Q for the instance (G', R, w). We have that

$$w(Q) \cdot \frac{4|R|}{\epsilon \cdot OPT(G, R, w)} \geq \hat{w}(Q),$$

which in turn implies that

$$OPT(G', R, w) \geq OPT(G', R, \hat{w}) \cdot \frac{\epsilon \cdot OPT(G, R, w)}{4|R|}.$$

We can now wrap up the analysis by comparing $w(T')$ with $OPT(G, R, w)$.

$$\begin{aligned} w(T') &\leq \hat{w}(T') \cdot \frac{\epsilon \cdot OPT(G, R, w)}{4|R|} + \frac{\epsilon}{2} OPT(G, R, w) \\ &\leq c \cdot OPT(G', R, \hat{w}) \cdot \frac{\epsilon \cdot OPT(G, R, w)}{4|R|} + \frac{\epsilon}{2} OPT(G, R, w) \\ &\leq c \cdot OPT(G', R, w) + \frac{\epsilon}{2} OPT(G, R, w) \\ &\leq c \cdot (1 + \epsilon/2) \cdot OPT(G, R, w) + \frac{\epsilon}{2} OPT(G, R, w) \\ &\leq c \cdot (1 + \epsilon) \cdot OPT(G, R, w) \end{aligned}$$

This implies that a T' is a $c(1+\epsilon)$-approximate Steiner tree for the instance (G, R, w), concluding the proof. □

6.3 Optimal Linear Arrangement

In the Optimal Linear Arrangement problem we are given as input an undirected graph G on n vertices. The task is to find a permutation $\sigma : V(G) \to \{1, \ldots, n\}$ minimizing the *cost* of σ. Here the cost of a permutation σ is $val(\sigma, G) = \sum_{uv \in E(G)} |\sigma(u) - \sigma(v)|$. Recall the problem Optimal Linear Arrangement parameterized by vertex cover:

$$OLA((G, C), k, \sigma) = \begin{cases} -\infty & \text{if } C \text{ is not vertex cover of } G \text{ of size at most } k, \\ \infty & \text{if } \sigma \text{ is not a linear layout,} \\ val(\sigma, G) & \text{otherwise.} \end{cases}$$

Fellows et al. [29] gave an FPT approximation scheme for Optimal Linear Arrangement parameterized by vertex cover. Their algorithm can be interpreted as a $2^k \cdot (\frac{1}{\epsilon})$ size $(1 + \epsilon)$-approximate kernel combined with a brute force algorithm on the reduced instance. Next we explain how to turn the approximation algorithm of Fellows et al. into a $(1+\epsilon)$-approximate kernel.

Let $((G, C), k)$ be the given instance of Optimal Linear Arrangement and C be a vertex cover of G. The remaining set of vertices $I = V(G) \setminus C$ forms an independent set. Furthermore, I can be partitioned into at most 2^k sets: for each subset S of C we define $I_S = \{v \in I \ : \ N(v) = S\}$. Let $m = |E(G)|$.

Based on ϵ we pick an integer $x = \lfloor \frac{\epsilon n}{4k^2 \cdot k^2 \cdot 2^{k+4}} \rfloor$. From G we make a new graph G_1 by deleting for each $S \subseteq C$ at most x vertices from I_S, such that the size of I_S becomes divisible by x. Clearly $OPT(G_1) \leq OPT(G)$ since G_1 is an induced subgraph of G. Furthermore, for any ordering σ_1 of G_1 one can make an ordering σ of G by appending all the vertices in $V(G) \setminus V(G_1)$ at the end of the ordering. Since there are 2^k choices for $S \subseteq C$, each vertex in $V(G) \setminus V(G_1)$ has degree at most k it follows that

$$val(\sigma, G) \leq val(\sigma_1, G_1) + 2^k \cdot x \cdot k \cdot n \tag{1}$$

One might think that an additive error of $2^k \cdot x \cdot k \cdot n$ is quite a bit, however Fellows et al. show that the optimum value is so large that this is quite insignificant.

Lemma 6.1 ([29]). $OPT(G) \geq \frac{m^2}{4k^2}$

In the following discussion let I^1 be the $V(G_1) \setminus C$ and let $I^1_S = \{v \in I^1 \ : \ N(v) = S\}$ for every $S \subseteq C$. Proceed as follows for each $S \subseteq C$. Since $|I^1_S|$ is divisible by x, we can group I^1_S into $\frac{|I^1_S|}{x}$ groups, each of size x. Define $\widehat{OPT}(G_1)$ to be the value of the best ordering of G_1 among the orderings where, for every group, the vertices in that group appear consecutively. Next we prove the following lemma, which states that there is a near-optimal solution for G_1 , where vertices in the same group appear consecutively.

Lemma 6.2. $\widehat{OPT}(G_1) \leq OPT(G_1) + (kn + m) \cdot 2^k \cdot (k + 1) \cdot x$

To prove Lemma 6.2 we first need an intermediate result. We say that an ordering σ is *homogenous* if, for every u, $v \in I$ such that $N(u) = N(v)$, $\sigma(u) < \sigma(v)$ and there is no $c \in C$ such that $\sigma(u) < \sigma(c) < \sigma(v)$, we have that for every w such that $\sigma(u) < \sigma(w) < \sigma(v)$, $N(w) = N(u)$. Informally this means that between two consecutive vertices of C, the vertices from different sets I_S and $I_{S'}$ "don't mix".

Lemma 6.3 ([29]). *There exists a homogenous optimal linear arrangement of G_1.*

Proof of Lemma 6.2. Consider an optimal linear arrangement of G_1 that is homogenous, as guaranteed by Lemma 6.3. From such an arrangement one can move around at most $2^k \cdot (k+1) \cdot x$ vertices (at most $(k+1) \cdot x$ vertices for each set I^1_S) and make a new one where for every group, the vertices in that group appear consecutively. Since moving a single vertex of degree at most k in an ordering σ can only increase the cost of σ by at most $(kn+m)$, this concludes the proof. $\square$

In the graph G_1, each set I^1_S is partitioned into $\frac{|I^1_S|}{x}$ groups, each of size x. From G_1 we can make a new graph G_2 by keeping C and exactly one vertex from each group, and deleting all other vertices. Thus G_2 is a graph on $|C| + \frac{n-|C|}{x}$ vertices. Each ordering σ_2 of $V(G_2)$ corresponds to an ordering σ_1 of $V(G_1)$ where for every group, the vertices in that group appear consecutively. We will say that the ordering σ_1 is the ordering of $V(G_1)$ *corresponding* to σ_2. Note that for every ordering of σ_1 of $V(G_1)$ where for every group, the vertices in that group appear consecutively there is an ordering σ_2 of G_2 such that σ_1 corresponds to σ_2. The next lemma summarizes the relationship between the cost of σ_1 (in G_1) and the cost of σ_2 (in G_2).

Lemma 6.4. *Let σ_2 be an ordering of G_2 and σ_1 be the ordering of G_1 corresponding to σ_2. Then the following two inequalities hold.*

$$val(\sigma_1, G_1) \leq x^2 \cdot val(\sigma_2, G_2) \qquad (2)$$

$$val(\sigma_2, G_2) \leq \frac{val(\sigma_1, G_1)}{x^2} + (k+1)\frac{m}{x} + k^2\frac{n}{x} \qquad (3)$$

Proof. For the first inequality observe that there is a natural correspondence between edges in G_1 and edges in G_2. Each edge in G_2 corresponds to either 1 or x edges, depending on whether it goes between two vertices of C or between a vertex in C and a vertex in $V(G_2) \setminus C$. Furthermore, for any edge uv in G_1 corresponding to an edge $u'v'$ in G_2 we have that $|\sigma_1(u) - \sigma_1(v)| \leq x \cdot |\sigma_2(u') - \sigma_2(v')|$. This concludes the proof of the first inequality.

For the second inequality, observe that for every edge uv in G_1 corresponding to an edge $u'v'$ in G_2 we have that

$$|\sigma_2(u') - \sigma_2(v')| \leq \frac{|\sigma_1(u) - \sigma_1(v)|}{x} + k + 1.$$

For each edge $u'v'$ in G_2 between a vertex in C and a vertex in $V(G_2) \setminus C$, there are exactly x edges in G_1 corresponding to it. These x edges contribute at least $x(|\sigma_2(u')-\sigma_2(v')|-k-1)$ each to $val(\sigma_1, G_1)$, thus contributing at least $|\sigma_2(u')-\sigma_2(v')|-k-1$ to $\frac{val(\sigma_1,G_1)}{x^2}$. Since there are at most $\frac{m}{x}$ edges in G_2, and at most k^2 edges between vertices in C (and thus un-accounted for in the argument above), each contributing at most $\frac{n}{x}$ to $val(\sigma_2, G_2)$, the second inequality follows. $\square$

Observe that the second inequality of Lemma 6.4 immediately implies that

$$OPT(G_2) \leq \frac{\widehat{OPT}(G_1)}{x^2} + (k+1)\frac{m}{x} + k^2\frac{n}{x} \tag{4}$$

We are now ready to state our main result.

Theorem 7. Optimal Linear Arrangement *parameterized by vertex cover has a* $(1+\epsilon)$-*approximate kernel of size* $\mathcal{O}(\frac{1}{\epsilon}2^k k^4)$.

Proof. The kernelization algorithm outputs the graph G_2 as described above. G_2 has at most $k + n/x \leq \mathcal{O}(\frac{1}{\epsilon}2^k k^4)$ vertices, so it remains to show how a c-approximate solution σ_2 to G_2 can be turned into a $c(1+\epsilon)$-approximate solution σ of G.

Given a c-approximate solution σ_2 to G_2, this solution corresponds to a solution σ_1 of G_1. The ordering σ_1 of $V(G_1)$ corresponds to an ordering σ of $V(G)$, as described in the paragraph right before Lemma 6.1. We claim that this ordering σ is in fact a $c(1+\epsilon)$-approximate solution σ of G.

$$\begin{aligned}
val(\sigma, G) &\leq val(\sigma_1, G_1) + 2^k \cdot x \cdot k \cdot n && \text{(By Equation (1))}\\
&\leq x^2 \cdot val(\sigma_2, G_2) + 2^k \cdot x \cdot k \cdot n && \text{(By Equation (2))}\\
&\leq x^2 \cdot c \cdot OPT(G_2) + 2^k \cdot x \cdot k \cdot n &&\\
&\leq x^2 \cdot c \left(\frac{\widehat{OPT}(G_1)}{x^2} + (k+1)\frac{m}{x} + k^2\frac{n}{x}\right) + 2^k \cdot x \cdot k \cdot n && \text{(By Equation (4))}\\
&\leq c \cdot \widehat{OPT}(G_1) + c \cdot 3k^2 \cdot n \cdot x + 2^k \cdot x \cdot k \cdot n && \text{(Because } m \leq k \cdot n)\\
&\leq c \cdot \Big(OPT(G_1) + (kn+m)(k+1)2^k \cdot x\Big) + c \cdot 2^{k+2} \cdot x \cdot k \cdot n && \text{(By Lemma 6.2)}\\
&\leq c \cdot OPT(G_1) + c \cdot k^2 2^{k+4} \cdot x \cdot n && \text{(Because } m \leq k \cdot n)\\
&\leq c \cdot OPT(G) + c \cdot k^2 2^{k+4} \cdot x \cdot n &&\\
&\leq c \cdot OPT(G) + c \cdot \epsilon \frac{n^2}{4k^2} &&\\
&\leq c \cdot (1+\epsilon) \cdot OPT(G) && \text{(By Lemma (6.1))}
\end{aligned}$$

This concludes the proof. □

7 Lower Bounds for Approximate Kernelization

In this section we set up a framework for proving lower bounds on the size of α-approximate kernels for a parameterized optimization problem. For normal kernelization, the most commonly used tool for establishing kernel lower bounds is by using cross compositions [7]. In particular, Bodlaender et al. [7] defined cross composition and showed that if an NP-hard language L admits a cross composition into a parameterized (decision) problem Π and Π admits a polynomial kernel, then L has an *OR-distillation* algorithm. Fortnow and Santhanam [32] proved that if an NP-hard language L has an OR-distillation, then NP $\subseteq$ coNP/Poly.

In order to prove a kernelization lower bound for a parameterized decision problem Π, all we have to do is to find an NP-hard langluage L and give a cross composition from L into Π. Then, if Π has a polynomial kernel, then combining the cross composition and the kernel with the results of Bodlaender et al. [7] and Fortnow and Santhanam [32] would prove that NP $\subseteq$ coNP/Poly. In other words a cross composition from L into Π proves that Π does not have a polynomial kernel unless NP $\subseteq$ coNP/Poly.

In order to prove lower bounds on the size of α-approximate kernels, we generalize the notion of cross compositions to α-gap cross compositions, which are hybrid of cross compositions and gap creating reductions found in hardness of approximation proofs. To give the formal definition of α-gap cross compositions, we first need to recall the definition of Bodlaender et al. [7] of polynomial equivalence relations on Σ^*, where Σ is a finite alphabet.

Definition 7.1 (polynomial equivalence relation [7])**.** *An equivalence relation R on Σ^*, where Σ is a finite alphabet, is called a* polynomial equivalence relation *if (i) equivalence of any $x, y \in \Sigma^*$ can be checked in time polynomial in $|x| + |y|$, and (ii) any finite set $S \subseteq \Sigma^*$ has at most $(\max_{x \in S} |x|)^{\mathcal{O}(1)}$ equivalence classes.*

Now we define the notion of α-gap cross composition.

Definition 7.2 (α-gap cross composition for maximization problem)**.** *Let $L \subseteq \Sigma^*$ be a language, where Σ is a finite alphabet and let Π be a parameterized maximization problem. We say that L α-*gap cross composes *into Π (where $\alpha \geq 1$), if there is a polynomial equivalence relation R and an algorithm which, given t strings $x_1, \ldots, x_t$ belonging to the same equivalence class of R, computes an instance (y, k) of Π and $r \in \mathbb{R}$, in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that the following holds:*

(i) $OPT(y, k) \geq r$ if and only if $x_i \in L$ for some $1 \leq i \leq t$;

(ii) $OPT(y, k) < \frac{r}{\alpha}$ if and only if $x_i \notin L$ for all $1 \leq i \leq t$; and

(iii) k is bounded by a polynomial in $\log t + \max_{1 \leq i \leq t} |x_i|$.

If such an algorithm exists, then we say that L α-gap cross composes to Π.

One can similarly define α-gap cross compositions for minimization problems.

Definition 7.3. *The definition of α-gap cross composition for minimization problem Π can be obtained by replacing conditions (i) and (ii) of Definition 7.2 with the following conditions (a) and (b) respectively: (a) $OPT(y, k) \leq r$ if and only if $x_i \in L$ for some $1 \leq i \leq t$, and (b) $OPT(y, k) > r \cdot \alpha$ if and only if $x_i \notin L$ for all $1 \leq i \leq t$.*

Similarly to the definition of α-approximate kernels, Definition 7.3 can be extended to encompass α-gap cross composition where α is not a constant, but rather a function of the (output) instance (y, k). Such compositions can be used to prove lower bounds on the size of α-approximate kernels where α is super-constant.

One of the main ingredient to prove *hardness* about computations in different algorithmic models is an appropriate notion of a *reduction* from a problem to another. Next, we define a notion of a polynomial time reduction appropriate for obtaining lower bounds for α-approximate kernels. As we will see this is very similar to the definition of α-approximate polynomial time pre-processing algorithm (Definition 3.8).

Definition 7.4. *Let $\alpha \geq 1$ be a real number. Let Π and Π' be two parameterized optimization problems. An* **α-approximate polynomial parameter transformation** *(α-appt for short) $\mathcal{A}$ from Π to Π' is a pair of polynomial time algorithms, called reduction algorithm $\mathcal{R}_{\mathcal{A}}$ and solution lifting algorithm. Given as input an instance (I, k) of Π the reduction algorithm outputs an instance (I', k') of Π'. The solution lifting algorithm takes as input an instance (I, k) of Π, the output instance $(I', k') = \mathcal{R}_{\mathcal{A}}(I, k)$ of Π', and a solution s' to the instance I' and outputs a solution s to (I, k). If Π is a minimization problem then*

$$\frac{\Pi(I, k, s)}{OPT_{\Pi}(I, k)} \leq \alpha \cdot \frac{\Pi'((I', k'), s')}{OPT_{\Pi'}(I', k')}.$$

If Π is a maximization problem then

$$\frac{\Pi(I,k,s)}{OPT_{\Pi}(I,k)} \cdot \alpha \geq \frac{\Pi'((I',k'),s')}{OPT_{\Pi'}(I',k')}.$$

*If there is a an α-*appt *from Π to Π' then in short we denote it by $\Pi \prec_{\alpha-\mathsf{appt}} \Pi'$.*

In the standard kernelization setting lower bounds machinery also rules out existence of compression algorithms. Similar to this our lower bound machinery also rules out existence of compression algorithms. Towards that we need to generalize the definition of α-approximate kernel to α-approximate compression. The only difference is that in the later case the reduced instance can be an instance of any parameterized optimization problem.

Definition 7.5. *Let $\alpha \geq 1$ be a real number. Let Π and Π' be two parameterized optimization problems. An α-**approximate compression** from Π to Π' is an α-*appt *$\mathcal{A}$ from Π to Π' such that $size_{\mathcal{A}}(k) = \sup\{|I'| + k' : (I',k') = \mathcal{R}_{\mathcal{A}}(I,k), I \in \Sigma^*\}$, is upper bounded by a computable function $g : \mathbb{N} \to \mathbb{N}$, where $\mathcal{R}_{\mathcal{A}}$ is the reduction algorithm in $\mathcal{A}$.*

For the sake of proving approximate kernel lower bounds, it is immaterial that Π' in the Definition 7.5 is a parameterized optimization problem and in fact it can *also be a unparameterized optimization problem*. However, for clarity of presentation we will stick to parameterized optimization problem in this paper. Whenever we talk about an existence of an α-approximate compression and we do not specify the target problem Π', we mean the existence of α-approximate compression into any optimization problem Π'. For more detailed exposition about lower bound machinery about polynomial compression for decision problems we refer to the textbook [15].

Towards building a framework for lower bounds we would like to prove a theorem analogous to the one by Bodlaender et al. [7]. In particular, we would like to show that an α-gap cross composition from an NP-hard language L into a parameterized optimization problem Π, together with an α-approximate compression of polynomial size yield an OR-distillation for the language L. Then the result of Fortnow and Santhanam [32] would immediately imply that any parameterized optimization problem Π that has an α-gap cross composition from an NP-hard language L can not have an α-approximate compression unless NP $\subseteq$ coNP/Poly. Unfortunately, for technical reasons, it seems difficult to make such an argument. Luckily, we *can* complete a very similar argument yielding essentially the same conclusion, but instead of relying on "OR-distillations" and the result of Fortnow and Santhanam [32], we make use of the more general result of Dell and van Melkebeek [17] that rules out cheap oracle communication protocols for NP-hard problems. We first give necessary definitions that allow us to formulate our statements.

Definition 7.6 (Oracle Communication Protocol [17])**.** *Let $L \subseteq \Sigma^*$ be a language, where Σ is a finite alphabet. An oracle communication protocol for the language L is a communication protocol between two players. The first player is given the input x and has to run in time polynomial in the length of x; the second player is computationally unbounded but is not given any part of x. At the end of the protocol the first player should be able to decide whether $x \in L$. The cost of the protocol is the number of bits of communication from the first player to the second player.*

Lemma 7.1 (Complementary Witness Lemma [17])**.** *Let L be a language and $t : \mathbb{N} \to \mathbb{N}$ be polynomial function such that the problem of deciding whether at least one out of $t(s)$ inputs of length at most s belongs to L has an oracle communication protocol of cost $\mathcal{O}(t(s)\log t(s))$, where the first player can be conondeterministic. Then $L \in$ coNP/Poly.*

Our lower bound technique for α-approximate compression for a parameterized optimization problem Π requires the problem Π to be *polynomial time verifiable*. By this we mean that the function Π is computable in polynomial time. We call such problems *nice parameterized optimization problems*. We are now in position to prove the main lemma of this section.

Lemma 7.2. *Let L be a language and Π be a nice parameterized optimization problem. If L α-gap cross composes to Π, and Π has a polynomial sized α-approximate compression, then $L \in$ $\mathsf{coNP/Poly}$.*

Proof. We prove the theorem for the case when Π is a maximization problem. The proof when Π is a minimization problem is analogous and thus it is omitted. By our assumption L α-gap cross composes to Π. That is, there exists a polynomial time algorithm $\mathcal{A}$ that given $t(s)$ strings $x_1, \ldots, x_{t(s)}$, each of length at most s, outputs an instance (y, k) of Π and a number $r \in \mathbb{R}$ such that the following holds.

(i) $OPT(y, k) \geq r$ if and only if $x_i \in L$ for some $1 \leq i \leq t(s)$

(ii) $OPT(y, k) < \frac{r}{\alpha}$ if and only if $x_i \notin L$ for all $1 \leq i \leq t(s)$

(iii) k is upper bounded by a polynomial P_1 of $s + \log(t(s))$. That is, $k \leq \mathsf{P}_1(s + \log(t(s)))$.

By our assumption Π has a polynomial sized α-approximate compression. That is, there is a pair of polynomial time algorithms $\mathcal{B}$ and $\mathcal{C}$, and an optimization problem Π' with the following properties: (a) $\mathcal{B}$ is a reduction algorithm, which takes input (y, k) of Π and outputs an instance (y', k') of Π' such that $|y'| + k' \leq \mathsf{P}_2(k)$ for a polynomial P_2 and (b) $\mathcal{C}$ is a solution lifting algorithm, which given an instance (y, k) of Π, an instance $(y', k')$of Π' and a solution S' to (y', k'), outputs a solution S of (y, k) such that

$$\frac{\Pi(y, k, S)}{OPT_{\Pi}(y, k)} \cdot \alpha \geq \frac{\Pi'(y', k', S')}{OPT_{\Pi'}(y', k')}.$$

Let $t = \mathsf{P}_1 \circ \mathsf{P}_2$, that is, $t(s) = \mathsf{P}_2(\mathsf{P}_1(s))$. We design an oracle communication protocol for the language L using algorithms $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$. The oracle communication protocol for L works as follows.

Step 1: The first player runs the algorithm $\mathcal{A}$ on the $t(s)$ input strings $x_1, \ldots, x_{t(s)}$, each of length at most s, and produces in polynomial time, an instance (y, k) of Π and a number $r \in \mathbb{R}$. Here the value k is upper bounded by $\mathsf{P}_1(s + \log(t(s)))$ (by condition (iii)).

Step 2: The first player runs the reduction algorithm $\mathcal{B}$ on (y, k), producing an instance (y', k') of Π'. Then the first player sends the instance (y', k') to the second player. By the property of algorithm $\mathcal{B}$, the size of the instance (y', k') is upper bounded by $\mathsf{P}_2(k) = \mathsf{P}_2(\mathsf{P}_1(s + \log(t(s))))$, which in turn is equal to $t(s + \log(t(s)))$.

Step 3: The (computationally unbounded) second player sends an optimum solution S' of (y', k') back to the first player.

Step 4: The first player runs the solution lifting algorithm $\mathcal{C}$ on input $(y, k), (y', k')$ and S', and it outputs a solution S of (y, k). Then, if $\Pi(y, k, S) \geq \frac{r}{\alpha}$ the first player declares that there exists an i such that $x_i \in L$. Otherwise the first player declares that $x_i \notin L$ for all i.

All the actions of the first player are performed in polynomial time. The cost of communication is $t(s + \log(t(s))) = \mathcal{O}(t(s))$, since t is a polynomial. We now show that the protocol is correct. Let $x_i \in L$ for some $1 \leq i \leq t(s)$. Since $\mathcal{A}$ is an α-gap cross composition we have that $OPT(y, k) \geq r$ (by condition (i)). Since S' is an optimum solution, by the property of solution lifting algorithm $\mathcal{C}$, S is a solution of (y, k) such that $\Pi(y, k, S) \geq \frac{OPT(y,k)}{\alpha} \geq \frac{r}{\alpha}$. This implies that in Step 4, the first player declares that $x_i \in L$ for some i. Suppose now that $x_i \notin L$ for all i. Then, by the definition of α-gap cross composition algorithms $\mathcal{A}$, we have that $OPT(y, k) < \frac{r}{\alpha}$. This implies that for any S, $\Pi(y, k, S) < \frac{r}{\alpha}$. Thus in Step 4, the first player declares that $x_i \notin L$ for all i. We have just verified that the described oracle communication protocol satisfies all the conditions of Lemma 7.1. Thus, by Lemma 7.1, we have that $L \in \mathsf{coNP/Poly}$. This completes the proof. □

The main theorem of the section follows from Lemma 7.2.

Theorem 8. *Let L be an NP-hard language and Π be a nice parameterized optimization problem. If L α-gap cross composes to Π, and Π has a polynomial sized α-approximate compression, then $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$.*

We note that Lemma 7.1 applies even if the first player works in co-nondeterministic polynomial time. Thus, a co-nondeterministic α-gap cross composition together with an α-approximate compression from an NP-hard language would still yield that $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$. For clarity we formally define co-nondeterministic α-gap cross composition for minimization problem which we later use in this section to derive a lower bound on Set Cover.

Definition 7.7 (co-nondeterministic α-gap cross composition for minimization problem)**.** *Let $L \subseteq \Sigma^*$ be a language, where Σ is a finite alphabet and let Π be a parameterized minimization problem. We say that L* co-nondeterministically α-gap cross composes *into Π (where $\alpha \geq 1$), if there is a polynomial equivalence relation R and a nondeterministic algorithm $\mathcal{A}$ which, given t strings $x_1, x_2, \ldots, x_t$ belonging to the same equivalence class of R, computes an instance (y, k) of Π and $r \in \mathbb{R}$, in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that the following holds.*

(i) if $x_i \in L$ for some $i \in [t]$, then in all the computation paths of $\mathcal{A}$, $OPT(y,k) \leq r$,

(ii) if $x_i \notin L$ for all $i \in [t]$, then there is a computation path in $\mathcal{A}$ with $OPT(y,k) > r \cdot \alpha$, and

(iii) k is bounded by a polynomial in $\log t + \max_{1 \leq i \leq t} |x_i|$.

If such an algorithm exists, then we say L co-nondeterministically α-gap cross composes to Π.

8 Longest Path

In this Section we show that Longest Path does not admit an α-approximate compression of polynomial size for any $\alpha \geq 1$ unless $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$. The parameterized optimization version of the Longest Path problem, that we call Path, is defined as follows.

$$\textsc{Path}(G,k,P) = \begin{cases} -\infty & \text{if } P \text{ is not a path in } G \\ \min\{k+1, |V(P)|-1\} & \text{otherwise} \end{cases}$$

We show that Path does not have a polynomial sized α-approximate compression for any constant $\alpha \geq 1$. We prove this by giving an α-gap cross composition from a α-Gap Long Path. The problem α-Gap Long Path is a promise problem which is defined as follows.

Definition 8.1. *The α-Gap Long Path problem is to determine, given a graph G and an integer k whether:*

- *G has a path of length at least k, in which case we say that (G, k) is a Yes instance of α-Gap Long Path.*
- *the longest path in G has length strictly less than $\frac{k}{\alpha}$, in which case we say that (G, k) is a No instance of α-Gap Long Path.*

It is known that α-Gap Long Path is NP-hard [39].

Lemma 8.1. *α-Gap Long Path α-gap cross composes to Path for any $\alpha \geq 1$.*

Proof. First we make the following polynomial equivalence relation: two instances (G_1, k_1) and (G_2, k_2) are in the same equivalence class if $k_1 = k_2$. Now given t instances $(G_1, k), \ldots, (G_t, k)$ of α-Gap Long Path, the α-gap cross composition algorithm $\mathcal{A}$ just outputs an instance (G, k) of Path, where G is the disjoint union of $G_1, \ldots, G_t$.

Clearly, G contains a path of length k if and only if there exists an i such that G_i contains a path of length k. Thus, $OPT(G,k) \geq r$ if and only if there is an i such that (G_i,k) is a yes instance of α-GAP LONG PATH. For the same reason $OPT(G,k) < \frac{r}{\alpha}$ if and only if (G_i,k) is a NO instance of α-GAP LONG PATH for every i. Finally the parameter k of the output instance is upper bounded by the size of the graphs G_i. This concludes the proof. $\square$

Theorem 8 and Lemma 8.1 yields the following theorem.

Theorem 9. PATH *does not have a polynomial size α-approximate compression for any $\alpha \geq 1$, unless* NP $\subseteq$ coNP/Poly.

9 Set Cover

In this section we show that parameterized optimization version of SET COVER parameterized by universe size does not admit an α-approximate compression of polynomial size for any $\alpha \geq 1$ unless NP $\subseteq$ coNP/Poly. The input of SET COVER is a family $\mathcal{S}$ of subsets of a universe U and the objective is to choose a minimum sized subfamily $\mathcal{F}$ of $\mathcal{S}$ such that $\bigcup_{S\in\mathcal{F}} S = U$. Such a set $\mathcal{F}$ is called a *set cover* of $(\mathcal{S},U)$. Since the parameter used here is a structural parameter, both SET COVER (SC) and its parameterized version SET COVER/n (SC/n) can be defined as follows.

$$\text{SC}/n((\mathcal{S},U),|U|,\mathcal{F}) = \text{SC}((\mathcal{S},U),\mathcal{F}) = \begin{cases} |\mathcal{F}| & \text{if } \mathcal{F} \text{ is a set cover of } (\mathcal{S},U) \\ \infty & \text{otherwise} \end{cases}$$

We show SET COVER/n does not have a polynomial sized α-approximate compression for any constant $\alpha \geq 1$. Towards this we first define the d-SET COVER problem, $d \in \mathbb{N}$. The d-SET COVER problem is a restriction of SET COVER, where each set in the family $\mathcal{S}$ is bounded by d. We show the desired lower bound on SET COVER/n by giving a co-nondeterministic α-gap cross composition from a gap version of d-SET COVER. The problem α-GAP d-SET COVER is a promise problem defined as follows.

Definition 9.1. *The α-GAP d-SET COVER problem is to determine, given a set family $\mathcal{S}$ over a universe U, where the size of each set in $\mathcal{S}$ is upper bounded by d, and an integer r whether:*

- *$OPT_{SC}(\mathcal{S},U) \leq r$, in which case we say that $((\mathcal{S},U),r)$ is a* YES *instance of* α-GAP d-SET COVER.
- *$OPT_{\text{SC}}(\mathcal{S},U) > r\alpha$, in which case we say that $((\mathcal{S},U),r)$ is a* NO *instance of* α-GAP d-SET COVER.

We will use the following known result regarding α-GAP d-SET COVER for our purpose.

Theorem 10 ([12,54]). *For any $\alpha \geq 1$, there is a constant d such that α-GAP d-SET COVER is* NP*-hard.*

To show a lower bound of α-approximate compression for SET COVER/n, our aim here is to give co-nondeterministic α-gap cross composition from α-GAP d-SET COVER. To give a co-nondeterministic cross composition algorithm it is enough to give a randomized cross composition algorithm which is always correct when it returns a NO instance. We give a formal proof about this after the following lemma.

Lemma 9.1. *Given t instances of α-GAP d-SET COVER, $((\mathcal{S}_1,U_1),r),\dots,((\mathcal{S}_t,U_t),r)$ of size s each, $|U_1| = \dots = |U_t| = n$, and $|\mathcal{S}_1| = \dots = |\mathcal{S}_t| = m$, there is a randomized polynomial time algorithm (i.e, polynomial in $t \cdot s$) with one sided error, which outputs an instance $(\mathcal{S},U)$ of* SET COVER *with following guarantees.*

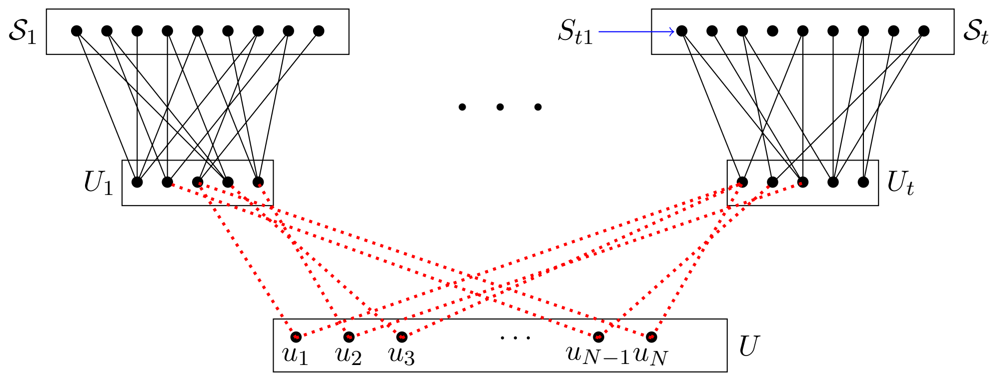


Figure 8: *An illustration of proof of Lemma 9.1. Every set and every element is represented using bullets. An element in a set is represented using an edge between the element and the set. The random assignment to each element in U is represented using dotted lines. The set S'_{t1} created from S_{t1} is $\{u_1, u_2, u_3, u_N\}$.*

(a) *if* $((\mathcal{S}_i, U_i), r)$ *is an* Yes *instance of* α-Gap d-Set Cover *for some* $i \in [t]$, *then*

$$\Pr[OPT_{\text{SC}}(\mathcal{S}, U) \leq r] = 1,$$

(b) *if* $((\mathcal{S}_i, U_i), r)$ *is a* No *instance of* α-Gap d-Set Cover *for all* $i \in [t]$, *then*

$$\Pr[OPT_{\text{SC}}(\mathcal{S}, U) > r\alpha] > 0, \textit{ and}$$

(c) $|U| = n^{2d} \cdot 4^d \cdot 2 \log \binom{m \cdot t}{r \cdot \alpha}$.

Proof. We design an algorithm $\mathcal{A}$ with properties mentioned in the statement of the lemma. We know that $n = |U_1| = \ldots = |U_t|$. For any $i \in [t]$, let $\mathcal{S}_i = \{\mathcal{S}_{i1}, \ldots, \mathcal{S}_{im}\}$. Algorithm $\mathcal{A}$ creates a universe U with $N = n^{2d} \cdot 4^d \cdot 2 \log \binom{m \cdot t}{r \cdot \alpha}$ elements. Now we describe the random process by which we construct the set family $\mathcal{S}$.

For each $u \in U$ and $i \in [t]$, *uniformly at random* assign an element from U_i to u.

That is, in this random process t elements are assigned to each element $u \in U$, one from each U_i. We use $\Gamma_i : U \to U_i$ to represent the random assignment. That is, for each $u \in U$, $i \in [t]$, $\Gamma_i(u)$ denotes the element in U_i that is assigned to u. Observe that an element $w \in U_i$ can be assigned to several elements of U. In other words, the set $\Gamma_i^{-1}(w)$ can have arbitrary size. For each $S_{ij}, i \in [t], j \in [m]$, algorithm $\mathcal{A}$, creates a set

$$S'_{ij} = \bigcup_{w \in S_{ij}} \Gamma_i^{-1}(w).$$

Notice that $|S'_{ij}|$ need not be bounded by any function of d. Let $\mathcal{S} = \{S'_{ij} : i \in [t], j \in [m]\}$. Algorithm $\mathcal{A}$ outputs $(\mathcal{S}, U)$. An illustration is given in Figure 8.

Now we prove the correctness of the algorithm. Suppose there exists $i \in [t]$ such that $((\mathcal{S}_i, U_i), r)$ is a Yes instance of α-Gap d-Set Cover. That is, there exist $S_{ij_1}, \ldots, S_{ij_r} \in \mathcal{S}_i$ such that $S_{ij_1} \cup \cdots \cup S_{ij_r} = U_i$. We know that for each $u \in U$, there is a $w \in U_i$ such that $\Gamma_i(u) = w$. Since $S_{ij_1} \cup \cdots \cup S_{ij_r} = U_i$ and for each $u \in U$, there is a $w \in U_i$ with $\Gamma_i(u) = w$, we can conclude that

$$S'_{ij_1} \cup \cdots \cup S'_{ij_r} = \bigcup_{\ell \in [r], w \in S_{ij_\ell}} \Gamma_i^{-1}(w) = \bigcup_{w \in U_i} \Gamma_i^{-1}(w) = U.$$

This implies that $\{S'_{ij_1}, \ldots, S'_{ij_r}\}$ is a set cover of $(\mathcal{S}, U)$. This proves condition (a) of the lemma.

Now we prove condition (b). In this case, for all $i \in [t]$, $((\mathcal{S}_i, U_i), r)$ is a No instance of α-Gap d-Set Cover. That is, for any $i \in [t]$, $(\mathcal{S}_i, U_i)$ does not have a set cover of cardinality at most $r\alpha$. Let $\mathcal{S}_{\mathsf{input}} = \bigcup_{i\in[t]} \mathcal{S}_i$. Each set in $S'_{ij} \in \mathcal{S}$, $i \in [t], j \in [m]$ is a random variable defined by $\bigcup_{w\in S_{ij}} \Gamma_i^{-1}(w)$. We call S'_{ij} a *set-random variable*. Thus, $\mathcal{S}$ is a set of mt-set random variables where the domain of each set-random variable is the power set of U (that is, 2^U). Thus, to show that $\Pr[OPT_{\mathrm{SC}}(\mathcal{S}, U) > r\alpha] > 0$, we need to show that probability of union of any αr set-random variables covering U is strictly less than $\frac{1}{\binom{mt}{r\alpha}}$. For an ease of presentation, for any $S_{ij} \in \mathcal{S}_{\mathsf{input}}$, we call $\bigcup_{w\in S_{ij}} \Gamma_i^{-1}(w)$, as $\mathsf{Image}(S_{ij})$. Observe that $\mathsf{Image}(S_{ij})$ is also a random variable and it is same as S'_{ij}. For a subset $\mathcal{F} \subseteq \mathcal{S}_{\mathsf{input}}$, by $\mathsf{Image}(\mathcal{F})$ we mean $\{\mathsf{Image}(S) \mid S \in \mathcal{F}\}$. Now we are ready to state and prove our main claim.

Claim 9.1. *For any $\mathcal{F} \subseteq \mathcal{S}_{\mathsf{input}}$ of cardinality $r\alpha$,* $\Pr[\mathsf{Image}(\mathcal{F})$ *is a set cover of* $(\mathcal{S}, U)] < \frac{1}{\binom{mt}{r\alpha}}$.

Proof. We can partition $\mathcal{F} = \mathcal{F}_1 \uplus \ldots \uplus \mathcal{F}_t$ such that $\mathcal{F}_i = \mathcal{F} \cap \{S_{i1}, \ldots, S_{im}\}$. Similarly, we can partition $\mathsf{Image}(\mathcal{F})$ into $\mathsf{Image}(\mathcal{F}_1) \uplus \ldots \uplus \mathsf{Image}(\mathcal{F}_t)$ such that $\mathsf{Image}(\mathcal{F}_i) = \mathsf{Image}(\mathcal{F}) \cap \{S'_{i1}, \ldots, S'_{im}\}$. Le U'_i be the subset of elements of U_i covered by the sets in $\mathcal{F}_i$. Let $X_i = |U'_i|$. Because of our assumption that $((\mathcal{S}_i, U_i), r)$ is a No instance of α-Gap d-Set Cover, we have that the subset U'_i covered by $\mathcal{F}_i$ is a strict subset of U_i and hence

$$\text{for all } i \in [t], X_i < n \tag{5}$$

Since $|\bigcup_{i\in[t]} \mathcal{F}_i| = |\mathcal{F}| = r\alpha$ and the cardinality of each set in $\mathcal{F}$ is at most d, we have

$$\sum_{i\in[t]} X_i < r\alpha d < nd \tag{6}$$

Since $((\mathcal{S}_i, U_i), r)$ is a No instance of α-Gap d-Set Cover for any $i \in [t]$, we have $r\alpha < n$ and hence the last inequality of Equation 6 follows. Towards bounding the probability mentioned in the claim, we first lower bound the following probability, $\Pr[u$ is not covered by $\mathsf{Image}(\mathcal{F})]$, for any fixed $u \in U$.

$$\begin{aligned}
\Pr[u \text{ is not covered by } \mathsf{Image}(\mathcal{F})] &= \bigwedge_{i\in[t]} \Pr[u \text{ is not covered by } \mathsf{Image}(\mathcal{F}_i)] \\
&= \bigwedge_{i\in[t]} (1 - \Pr[u \text{ is covered by } \mathsf{Image}(\mathcal{F}_i)]) \\
&= \bigwedge_{i\in[t]} (1 - \Pr[\Gamma_i(u) \in U'_i]) \\
&= \prod_{i\in[t]} \left(1 - \frac{|U'_i|}{n}\right) \quad (\text{Because, } \forall w \in U_i, \Pr[\Gamma_i(u) = w] = \frac{1}{n}) \\
&= \prod_{i\in[t]} \left(1 - \frac{X_i}{n}\right) \\
&= \prod_{\substack{i\in[t] \text{ such that} \\ X_i \le \frac{n}{2}}} \left(1 - \frac{X_i}{n}\right) \cdot \prod_{\substack{i\in[t] \text{ such that} \\ X_i > \frac{n}{2}}} \left(1 - \frac{X_i}{n}\right) \qquad (7)
\end{aligned}$$

We know, by Equation 6, that $\sum_{i\in[t]} X_i < nd$. This implies that the number of X_i's such that $X_i > \frac{n}{2}$ is at most $2d$. By Equation 5, we have that for any $i \in [t]$, $\left(1 - \frac{X_i}{n}\right) \ge \left(1 - \frac{n-1}{n}\right) = \frac{1}{n}$.

Since the number of X_i's with $X_i > \frac{n}{2}$ is at most $2d$ and $\left(1 - \frac{X_i}{n}\right) \geq \frac{1}{n}$, we can rewrite Equation 7, as follows.

$$\begin{aligned}
\Pr[u \text{ is not covered by } \mathsf{Image}(\mathcal{F})] &\geq \left(\prod_{\substack{i\in[t] \text{ such that} \\ X_i \leq \frac{n}{2}}} \left(1 - \frac{X_i}{n}\right) \right) \cdot \frac{1}{n^{2d}} \\
&\geq \frac{1}{n^{2d}} \prod_{\substack{i\in[t] \text{ such that} \\ X_i \leq \frac{n}{2}}} \left(\frac{1}{4}\right)^{\frac{X_i}{n}} && \text{(By Fact 2)} \\
&\geq \frac{1}{n^{2d}} \cdot \left(\frac{1}{4}\right)^{\frac{\sum X_i}{n}} \\
&\geq \frac{1}{n^{2d}} \cdot \left(\frac{1}{4}\right)^{d} && \text{(By Equation 6)} \qquad (8)
\end{aligned}$$

Since the set of events "u is covered by $\mathsf{Image}(\mathcal{F})$", where $u \in U$, are independent events, we have

$$\begin{aligned}
\Pr[\mathsf{Image}(\mathcal{F}) \text{ is a set cover of } (\mathcal{S}, U)] &\leq \prod_{u\in U} \Pr[u \text{ is covered by } \mathsf{Image}(\mathcal{F})] \\
&\leq \prod_{u\in U} \left(1 - \frac{1}{n^{2d}\cdot 4^d}\right) && \text{(By Equation 8)} \\
&\leq \prod_{u\in U} e^{\frac{-1}{n^{2d}\cdot 4^d}} \\
&< \frac{1}{\binom{mt}{r\alpha}} && \text{(Because } |U| = n^{2d}\cdot 4^d \cdot 2\log\binom{mt}{r\alpha}\text{)}
\end{aligned}$$

This completes the proof of the claim. $\square$

Since the number of subsets of cardinality $r\alpha$ of $\mathcal{S}_{\mathsf{input}}$, is at most $\binom{mt}{r\alpha}$, by Claim 9.1 and union bound we get that

$$\Pr[\exists \text{ a set } \mathcal{F} \subseteq \mathcal{S}_{\mathsf{input}} \text{ of cardinality } r\alpha \text{ such that } \mathsf{Image}(\mathcal{F}) \text{ is a set cover of } (\mathcal{S}, U)] < 1.$$

This completes the proof of condition (b). Condition (c) trivially follows from the construction of U. $\square$

Now we will use the construction given in Lemma 9.1 to prove the main lemma of this section.

Lemma 9.2. *α-Gap d-Set Cover co-nondeterministically α-gap cross composes to* Set Cover$/n$ *for any $\alpha \geq 1$.*

Proof. First we make the following polynomial equivalence relation: two instances $((\mathcal{S}, U), r)$ and $((\mathcal{S}', U'), r')$ of α-Gap d-Set Cover are in the same equivalence class if $|\mathcal{S}| = |\mathcal{S}'|$, $|U| = |U'|$ and $r = r'$. Towards proving the lemma we need to design an algorithm $\mathcal{B}$ with the properties of Definition 7.7. Let $((\mathcal{S}_1, U_1), r), \ldots, ((\mathcal{S}_t, U_t), r)$ be instances of α-Gap d-Set Cover of size s each, $|U_1| = \cdots = |U_t| = n$, and $|\mathcal{S}_1| = \cdots = |\mathcal{S}_t| = m$. Here, t is polynomially bounded in s. Since $((\mathcal{S}_i, U_i), r)$, $i \in [t]$, are instances of α-Gap d-Set Cover, $m \leq \binom{n}{d}$ and hence $t \leq n^c$ for some constant c. Now $\mathcal{B}$ runs the algorithm $\mathcal{A}$ mentioned in the Lemma 9.1, but instead of using the random bits it nondeterministically guesses these bits while running $\mathcal{A}$. Algorithm $\mathcal{B}$ returns $(\mathcal{S}, U)$ and r as output, where $(\mathcal{S}, U)$ is the output of $\mathcal{A}$.

If there exists $i \in [t]$ such that $((\mathcal{S}_i, U_i), r)$ is a Yes instance of α-Gap d-Set Cover, by condition (a) of Lemma 9.1, we can conclude that $OPT_{\mathrm{SC}/n}((\mathcal{S}, U), |U|) \leq r$ and hence satisfies property (i) of Definition 7.7. Suppose $((\mathcal{S}_i, U_i), r)$ is a No instance for all $i \in [t]$. Because of condition (b) of Lemma 9.1, there is a choice of random bits B such that if $\mathcal{A}$ uses B as the random bits then $OPT_{\mathrm{SC}}(\mathcal{S}, U) > r\alpha$. Hence, for the nondeterministic guess B of the algorithm $\mathcal{B}$, we get that $OPT_{\mathrm{SC}/n}((\mathcal{S}, U), |U|) = OPT_{\mathrm{SC}}(\mathcal{S}, U) > r\alpha$. This proves property (ii) of Definition 7.7. By condition (c) of Lemma 9.1, and the facts that $m, t \in n^{\mathcal{O}(1)}$, we get that $|U| = n^{\mathcal{O}(1)}$. This implies the property (iii) of Definition 7.7. This completes the proof of the lemma. $\square$

Theorems 8 and 10, and Lemma 9.2 yields the following theorem.

Theorem 11. Set Cover$/n$ *does not have a polynomial size α-approximate compression for any $\alpha \geq 1$, unless* NP$\subseteq$ coNP/Poly.

10 Hitting Set

In this section we show that a parameterized optimization version of Hitting Set does not admit an $\mathcal{O}(2^{\log^c n})$-approximate kernel of polynomial size for any $c < 1$, unless CNF-SAT can be solved in slightly subexponential time, where universe size n of the input instance is the parameter. Compare to Set Cover our result in this section are much more stronger, but unlike Set Cover here we can only rule out an existence of an approximate kernel and not an approximate compression. The input of Hitting Set is a family $\mathcal{S}$ of subsets of a universe U and the objective is to choose a minimum cardinality subset $X \subseteq U$ such that for all $S \in \mathcal{S}$, $S \cap X \neq \emptyset$. Such a subset X is called a *hitting set* of $(\mathcal{S}, U)$. Since the parameter used here is a structural parameter, both Hitting Set (HS) and its parameterized version Hitting Set$/n$ (HS$/n$) can be defined as follows.

$$\mathrm{HS}/n((\mathcal{S}, U), |U|, X) = \mathrm{HS}((\mathcal{S}, U), X) = \begin{cases} |X| & \text{if } X \text{ is a hitting set of } (\mathcal{S}, U) \\ \infty & \text{otherwise} \end{cases}$$

The following lemma shows that in fact Hitting Set is same as Set Cover but with a different parameter.

Lemma 10.1. *Let $(\mathcal{S}, U)$ be an instance of* Hitting Set. *Let $F_u = \{S \in \mathcal{S} : u \in S\}$ for all $u \in U$ and let $\mathcal{F} = \{F_u : u \in U\}$. Then $OPT_{\mathrm{HS}}(\mathcal{S}, U) = OPT_{\mathrm{SC}}(\mathcal{F}, \mathcal{S})$*

Proof. Let $X \subseteq U$ be a hitting set of $(\mathcal{S}, U)$. Consider the set $\mathcal{F}_X = \{F_u \in \mathcal{F} : u \in X\}$. Since X is a hitting set of $\mathcal{S}$, for any $S \in \mathcal{S}$, there is an element $u \in X$ such that $S \in F_u$. This implies that $\mathcal{F}_X$ is a set cover of $(\mathcal{F}, \mathcal{S})$.

Let $\mathcal{F}' \subseteq \mathcal{F}$ be a set cover of $(\mathcal{F}, \mathcal{S})$. Let $X = \{u \in U : F_u \in \mathcal{F}'\}$. Since $\mathcal{F}'$ is a set cover of $(\mathcal{F}, \mathcal{S})$, for any $S \in \mathcal{S}$, there is a set $F_u \in \mathcal{F}'$ such that $S \in F_u$. This implies that X is a hitting set of $(\mathcal{S}, U)$. This completes the proof of the lemma. $\square$

The following Lemma follows from the $\mathcal{O}(\log n)$-approximation algorithm of Set Cover [13] and Lemma 10.1

Lemma 10.2 ([13]). *There is a polynomial time algorithm which given an instance $(\mathcal{S}, U)$ of* Hitting Set, *outputs a hitting set of cardinality bounded by $\mathcal{O}(OPT_{\mathrm{HS}}(\mathcal{S}, U) \cdot \log |\mathcal{S}|)$.*

The following theorem is a slight weakening of a result by Nelson [48], which we use to prove our theorem.

Theorem 12 ([48]). *For any $c < 1$,* Hitting Set *has no polynomial time $\mathcal{O}(2^{\log^c n})$-approximation unless CNF-SAT with n-variables can be solved in time $2^{\mathcal{O}(2^{\log^{1-1/(\log\log n)^{1/3}} n})}$.*

The assumption used in Theorem 12, implies the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi and Zane [38] and hence it is weaker than ETH.

Theorem 13. *For any $c < 1$,* Hitting Set/n *does not admits a $\mathcal{O}(2^{\log^c n})$-approximate kernel, unless CNF-SAT with n-variables can be solved in time $2^{\mathcal{O}(2^{\log^{1-1/(\log\log n)^{1/3}} n})}$.*

Proof. Suppose there is a $\mathcal{O}(2^{\log^c n})$-approximate kernel $\mathcal{A}$ for Hitting Set/n for some $c < 1$. Then, we argue that we can solve CNF-SAT on n variables in time $2^{\mathcal{O}(2^{\log^{1-1/(\log\log n)^{1/3}} n})}$. Towards that, by Theorem 12, it is enough to give a $\mathcal{O}(2^{\log^{c'} n})$ -approximation algorithm for Hitting Set for some $c' < 1$, where n is the cardinality of the universe in the input instance.

Fix a constant c' such that $c < c' < 1$. We design a $\mathcal{O}(2^{\log^{c'} n})$-approximation algorithm for Hitting Set using $\mathcal{A}$. Let $(\mathcal{S}, U)$ be an instance of HS and let $|U| = n$. Let $\mathcal{R}_{\mathcal{A}}$ and $\mathcal{L}_{\mathcal{A}}$ be the reduction algorithm and solution lifting algorithm of $\mathcal{A}$ respectively. We run the algorithm $\mathcal{R}_{\mathcal{A}}$ on $((\mathcal{S}, U), n)$ and let $((\mathcal{S}', U'), |U'|)$ be the output of $\mathcal{R}_{\mathcal{A}}$. We know that $|\mathcal{S}'| + |U'| = n^{\mathcal{O}(1)}$. Then, by Lemma 10.2, we compute a hitting set W of $(\mathcal{S}', U')$, of cardinality bounded by $\mathcal{O}(OPT_{\text{HS}}(\mathcal{S}', U') \cdot \log n)$. Then, by using solution lifting algorithm $\mathcal{L}_{\mathcal{A}}$, we compute a hitting set X of $((\mathcal{S}, U), n)$. By the property of $\mathcal{O}(2^{\log^c n})$ -approximate kernel $\mathcal{A}$, we can conclude that the cardinality of X is bounded by $\mathcal{O}(2^{\log^c n} \cdot \log n \cdot OPT_{\text{HS}/n}((\mathcal{S}, U), n)) = \mathcal{O}(2^{\log^{c'} n} \cdot OPT_{\text{HS}/n}((\mathcal{S}, U), n))$. This implies that X is a $\mathcal{O}(2^{\log^{c'} n})$ -approximate solution of $(\mathcal{S}, U)$. This completes the proof of the theorem. □

11 Conclusion and Discussions

In this paper we have set up a framework for studying lossy kernelization, and showed that for several problems it is possible to obtain approximate kernels with better approximation ratio than that of the best possible approximation algorithms, and better size bound than what is achievable by regular kernels. We have also developed methods for showing lower bounds for approximate kernelization. There are plenty of problems that are waiting to be attacked within this new framework. Indeed, one can systematically go through the list of all parameterized problems, and investigate their approximate kernelization complexity. For problems that provably do not admit polynomial size kernels but do admit constant factor approximation algorithms, one should search for PSAKSes. For problems with PSAKSes one should search for efficient PSAKSes. For problems with no polynomial kernel and no constant factor approximation, one may look for a constant factor approximate kernel of polynomial size. For problems that do have polynomial kernels, one can search for approximate kernels that are even smaller. We conclude with a list of concrete interesting problems.

- Does Connected Vertex Cover, Disjoint Factors or Disjoint Cycle Packing admit an EPSAKS?
- Does Edge Clique Cover admit a constant factor approximate kernel of polynomial size?
- Does Directed Feedback Vertex Set admit a constant factor approximate kernel of polynomial size?
- Does Multiway Cut or Subset Feedback Vertex Set have a PSAKS?

- Does Disjoint Hole Packing admit a PSAKS? Here a *hole* in a graph G is an induced cycle of length 4 or more.
- Does Optimal Linear Arrangement parameterized by vertex cover admit a constant factor approximate kernel of polynomial size, or even a PSAKS?
- Does Maximum Disjoint Paths admit a constant factor approximate kernel, or even a PSAKS? Here the input is a graph G together with a set of vertex pairs $(s_1, t_1), (s_2, t_2)$, $\ldots$, (s_ℓ, t_ℓ). The goal is to find a maximum size subset $R \subseteq \{1, \ldots, \ell\}$ and, for every $i \in R$ a path P_i from s_i to t_i, such that for every $i, j \in R$ with $i \neq j$ the paths P_i and P_j are vertex disjoint. What happens to this problem when input is restricted to be a planar graph? Or a graph excluding a fixed graph H as a minor? What about chordal graphs, or interval graphs?
- It is known that d-Hitting Set admits a kernel if size $\mathcal{O}(k^d)$, this kernel is also a strict 1-approximate kernel. d-Hitting Set also admits a factor d-approximation in polynomial time, this is a d-approximate kernel of constant size. Can one interpolate between these two extremes by giving an α-approximate kernel of size $\mathcal{O}(k^{f(\alpha)})$ with $f(1) = d$, $f(d) = \mathcal{O}(1)$, and f being a continuous function?
- Our lower bound for approximate kernelization of Hitting Set parameterized by universe size n does not apply to compressions. Can one rule out polynomial size constant factor approximate compressions of Hitting Set parameterized by universe size n assuming $\mathsf{NP} \not\subseteq \mathsf{coNP}/\mathsf{Poly}$ or another reasonable complexity theoretic assumption?
- One may extend the notion of approximate kernels to approximate Turing kernels [15] in a natural way. Does Independent Set parameterized by treewidth admit a polynomial size approximate Turing kernel with a constant approximation ratio? What about a Turing PSAKS?
- Does Treewidth admit an constant factor approximate kernel of polynomial size? Here even a Turing kernel (with a constant factor approximation) would be very interesting.
- What is the complexity of approximate kernelization of Unique Label Cover? [5,40]
- The notion of α-gap cross compositions can be modified to “AND α-gap cross compositions” in the same way that AND-compositions relate to OR-compositions [6]. In order to directly use such “AND α-gap cross compositions” to show lower bounds for approximate kernelization, one needs an analogue of Lemma 7.1 for the problem of deciding whether *all* of the $t(s)$ inputs belong to L. This is essentially a strengthening of the AND-distillation conjecture [6,25] to oracle communication protocols (see the conclusion section of Drucker [25], open question number 1). Can this strengthening of the AND-distillation conjecture be related to a well known complexity theoretic assumption?

Acknowledgement. The authors thank Dániel Marx for enlightening discussions on related work in the literature, and Magnus Wahlström for pointing out the remark about randomized pre-processing algorithms following Definition 3.5.

References

[1] F. N. Abu-Khzam, *A kernelization algorithm for d-hitting set*, J. Comput. Syst. Sci., 76 (2010), pp. 524–531.

[2] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.

[3] C. Ambühl, M. Mastrolilli, and O. Svensson, *Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut*, SIAM J. Comput., 40 (2011), pp. 567–596.

[4] E. M. Arkin, M. M. Halldórsson, and R. Hassin, *Approximating the tree and tour covers of a graph*, Inf. Process. Lett., 47 (1993), pp. 275–282.

[5] S. Arora, B. Barak, and D. Steurer, *Subexponential algorithms for unique games and related problems*, J. ACM, 62 (2015), p. 42.

[6] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin, *On problems without polynomial kernels*, J. Comput. Syst. Sci., 75 (2009), pp. 423–434.

[7] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch, *Cross-composition: A new technique for kernelization lower bounds*, in 28th International Symposium on Theoretical Aspects of Computer Science (STACS), 2011, pp. 165–176.

[8] H. L. Bodlaender, S. Thomassé, and A. Yeo, *Kernel bounds for disjoint cycles and disjoint paths*, Theor. Comput. Sci., 412 (2011), pp. 4570–4578.

[9] A. Borchers and D. Du, *The k-steiner ratio in graphs*, SIAM J. Comput., 26 (1997), pp. 857–869.

[10] R. Bredereck, J. Chen, S. Hartung, S. Kratsch, R. Niedermeier, O. Suchý, and G. J. Woeginger, *A multivariate complexity analysis of lobbying in multiple referenda*, J. Artif. Intell. Res. (JAIR), 50 (2014), pp. 409–446.

[11] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità, *Steiner tree approximation via iterative randomized rounding*, J. ACM, 60 (2013), p. 6.

[12] M. Chlebík and J. Chlebíková, *Approximation hardness of dominating set problems in bounded degree graphs*, Inf. Comput., 206 (2008), pp. 1264–1275.

[13] V. Chvatal, *A greedy heuristic for the set-covering problem*, Mathematics of Operations Research, 4 (1979), pp. 233–235.

[14] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education, 2nd ed., 2001.

[15] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Springer, 2015.

[16] H. Dell and D. Marx, *Kernelization of packing problems*, in Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, 2012, pp. 68–81.

[17] H. Dell and D. van Melkebeek, *Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses*, in Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010, 2010, pp. 251–260.

[18] P. Dey, N. Misra, and Y. Narahari, *Kernelization complexity of possible winner and coalitional manipulation problems in voting*, Theor. Comput. Sci., 616 (2016), pp. 111–125.

[19] R. Diestel, *Graph theory*, vol. 173 of Graduate Texts in Mathematics, Springer-Verlag, Berlin, 3rd ed., 2005.

[20] I. Dinur, V. Guruswami, S. Khot, and O. Regev, *A new multilayered PCP and the hardness of hypergraph vertex cover*, SIAM J. Comput., 34 (2005), pp. 1129–1146.

[21] I. Dinur and S. Safra, *On the hardness of approximating minimum vertex cover*, Annals of mathematics, (2005), pp. 439–485.

[22] M. Dom, D. Lokshtanov, and S. Saurabh, *Kernelization lower bounds through colors and ids*, ACM Transactions on Algorithms, 11 (2014), pp. 13:1–13:20.

[23] R. G. Downey and M. R. Fellows, *Parameterized complexity*, Springer Science & Business Media, 2012.

[24] S. E. Dreyfus and R. A. Wagner, *The steiner problem in graphs*, Networks, 1 (1971), pp. 195–207.

[25] A. Drucker, *New limits to classical and quantum instance compression*, SIAM J. Comput., 44 (2015), pp. 1443–1479.

[26] P. Erdős and L. Pósa, *On independent circuits contained in a graph*, Canad. Journ. Math, 17 (1965), pp. 347–352.

[27] U. Feige and M. Langberg, *Approximation algorithms for maximization problems arising in graph partitioning*, J. Algorithms, 41 (2001), pp. 174–211.

[28] U. Feige and J. R. Lee, *An improved approximation ratio for the minimum linear arrangement problem*, Inf. Process. Lett., 101 (2007), pp. 26–29.

[29] M. R. Fellows, D. Hermelin, F. A. Rosamond, and H. Shachnai, *Tractable parameterizations for the minimum linear arrangement problem*, in Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings, 2013, pp. 457–468.

[30] M. R. Fellows, A. Kulik, F. A. Rosamond, and H. Shachnai, *Parameterized approximation via fidelity preserving transformations*, in Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I, 2012, pp. 351–362.

[31] H. Fernau, F. V. Fomin, G. Philip, and S. Saurabh, *The curse of connectivity:* t*-total vertex (edge) cover*, in Computing and Combinatorics, 16th Annual International Conference, COCOON 2010, Nha Trang, Vietnam, July 19-21, 2010. Proceedings, 2010, pp. 34–43.

[32] L. Fortnow and R. Santhanam, *Infeasibility of instance compression and succinct pcps for NP*, J. Comput. Syst. Sci., 77 (2011), pp. 91–106.

[33] Z. Friggstad and M. R. Salavatipour, *Approximability of packing disjoint cycles*, Algorithmica, 60 (2011), pp. 395–400.

[34] C. Guo and L. Cai, *Obtaining split graphs by edge contraction*, Theoretical Computer Science, 607, Part 1 (2015), pp. 60 – 67.

[35] J. Guo, R. Niedermeier, and S. Wernicke, *Parameterized complexity of vertex cover variants*, Theory Comput. Syst., 41 (2007), pp. 501–520.

[36] D. Hermelin, S. Kratsch, K. Soltys, M. Wahlström, and X. Wu, *A completeness theory for polynomial (turing) kernelization*, Algorithmica, 71 (2015), pp. 702–730.

[37] D. Hermelin and X. Wu, *Weak compositions and their applications to polynomial lower bounds for kernelization*, in Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, 2012, pp. 104–113.

[38] R. Impagliazzo, R. Paturi, and F. Zane, *Which problems have strongly exponential complexity?*, J. Comput. Syst. Sci., 63 (2001), pp. 512–530.

[39] D. R. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, Algorithmica, 18 (1997), pp. 82–98.

[40] S. Khot, *On the power of unique 2-prover 1-round games*, in Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada, 2002, pp. 767–775.

[41] S. Khot and O. Regev, *Vertex cover might be hard to approximate to within 2- ε*, Journal of Computer and System Sciences, 74 (2008), pp. 335–349.

[42] S. Kratsch, *Recent developments in kernelization: A survey*, Bulletin of the EATCS, 113 (2014).

[43] D. Lokshtanov, *Parameterized integer quadratic programming: Variables and coefficients*, Tech. Rep. abs/1511.00310, arXiv CoRR, 2015.

[44] D. Lokshtanov, N. Misra, and S. Saurabh, *Kernelization–preprocessing with a guarantee*, in The Multivariate Algorithmic Revolution and Beyond, Springer, 2012, pp. 129–161.

[45] D. Marx, *Parameterized complexity and approximation algorithms*, The Computer Journal, 51 (2008), pp. 60–78.

[46] N. Misra, F. Panolan, A. Rai, V. Raman, and S. Saurabh, *Parameterized algorithms for max colorable induced subgraph problem on perfect graphs*, in Graph-Theoretic Concepts in Computer Science - 39th International Workshop, WG 2013, Lübeck, Germany, June 19-21, 2013, Revised Papers, 2013, pp. 370–381.

[47] D. Moshkovitz, *The projection games conjecture and the np-hardness of ln n-approximating set-cover*, Theory of Computing, 11 (2015), pp. 221–235.

[48] J. Nelson, *A note on set cover inapproximability independent of universe size*, Electronic Colloquium on Computational Complexity (ECCC), 14 (2007).

[49] E. Petrank, *The hardness of approximation: Gap location*, Computational Complexity, 4 (1994), pp. 133–157.

[50] V. Raman, S. Saurabh, and C. R. Subramanian, *Faster fixed parameter tractable algorithms for finding feedback vertex sets*, ACM Transactions on Algorithms, 2 (2006), pp. 403–415.

[51] M. R. Salavatipour and J. Verstraëte, *Disjoint cycles: Integrality gap, hardness, and approximation*, in Integer Programming and Combinatorial Optimization, 11th International IPCO Conference, Berlin, Germany, June 8-10, 2005, Proceedings, 2005, pp. 51–65.

[52] C. D. Savage, *Depth-first search and the vertex cover problem*, Inf. Process. Lett., 14 (1982), pp. 233–237.

[53] M. Sipser, *Introduction to the Theory of Computation*, Cengage Learning, 2012.

[54] L. Trevisan, *Non-approximability results for optimization problems on bounded degree instances*, in Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece, 2001, pp. 453–461.

[55] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

[56] Y. Yang and J. Guo, *Possible winner problems on partial tournaments: A parameterized study*, in Algorithmic Decision Theory - Third International Conference, ADT 2013, Bruxelles, Belgium, November 12-14, 2013, Proceedings, 2013, pp. 425–439.