# Lossy Kernelization

Daniel Lokshtanov[1], Fahad Panolan[2], M. S. Ramanujan[3], Saket Saurabh[4]

## ABSTRACT

In this paper we propose a new framework for analyzing the performance of pre-processing 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 $\alpha$-approximate kernel. Loosely speaking, a polynomial size $\alpha$-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 $|I'| + k' \leq k^{O(1)}$. 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 $(I, k)$.

Amongst our main technical contributions are $\alpha$-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 NP $\subseteq$ 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 $\alpha$-approximate kernel of polynomial size, for any $\alpha \geq 1$, unless NP $\subseteq$ coNP/Poly.

## 1 INTRODUCTION

Polynomial time pre-processing 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 pre-processing 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 $\Pi$ is a subset of $\Sigma^* \times \mathbb{N}$, where $\Sigma$ is a finite alphabet. The goal of parameterized algorithms is to determine whether an instance $(I, k)$ given as input belongs to $\Pi$ 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 $\Pi$ 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 [4, 5, 16, 19, 21], or a kernel of a specific size [10, 11, 22]. Over the last decade many new results and several new techniques have been discovered, see the survey articles by Kratsch [27] or Lokshtanov *et al.* [28] for recent developments, or the textbooks [9, 14] 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 $\alpha$-approximate solution to $(I', k')$ there is no guarantee that we will be able to get an $\alpha$-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 pre-processing 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.*

[1]University of Bergen, daniello@ii.uib.no
[2]University of Bergen, fahad@ii.uib.no
[3]University of Warwick, R.Maadapuzhi-Sridharan@warwick.ac.uk
[4]The Institute of Mathematical Sciences, saket@imsc.res.in

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 $\alpha$-approximate kernels. Loosely speaking an ($\alpha$)-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*.

*Robust.* We show that the key notions behave consistently with related notions from parameterized complexity, kernelization, approximation and FPT-approximation algorithms. For instance, we show that a problem admits an $\alpha$-approximate kernel *if and only if* it is FPT-$\alpha$-approximable, mirroring the equivalence between FPT and kernelization [9].

*Versatile.* 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.

*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 [8], and FPT-approximation for Partial Vertex Cover [30] and for Minimal Linear Arrangement parameterized by the vertex cover number [17] 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 $\alpha$-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.* [18] initiated the study of lossy kernelization. They proposed a definition of lossy kernelization called *$\alpha$-fidelity kernels*. Essentially, an $\alpha$-fidelity kernel is a polynomial time pre-processing procedure such that an *optimal solution* to the reduced instance translates to an *$\alpha$-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 $\alpha$-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 $\alpha$-approximate kernels introduced in this work is a better model of lossy kernelization than $\alpha$-fidelity kernels are.

It is important to note that even though the definition of $\alpha$-approximate kernels crucially differs from the definition of $\alpha$-fidelity kernels [18], it seems that most of the pre-processing algorithms that establish the existence of $\alpha$-approximate kernels can be used to establish the existence of $\alpha$-fidelity kernels and vice versa. In particular, all of the $\alpha$-fidelity kernel results of Fellows *et al.* [18] can be translated to $\alpha$-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 [4, 6, 13] precluding them from admitting polynomial kernels under widely believed complexity theoretic assumptions. 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 $\mathsf{NP} \subseteq \mathsf{coNP/Poly}$, 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 [23]. Next we discuss our results for each of the six problems in more detail. Due to space constraints, we include only a subset of our results in the extended abstract and refer the reader to the full version [29] for the complete set of results and proofs.

**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 [2], admits a factor 2 approximation algorithm [2, 34], and is known not to admit a factor $(2 - \epsilon)$ approximation algorithm assuming the Unique Games conjecture [26]. Further, an approximation algorithm with ratio below 1.36 would imply that $\mathsf{P} = \mathsf{NP}$ [12]. From the perspective of kernelization, it is easy to show that Connected Vertex Cover admits a kernel with at most $2^k$ vertices [9], where $k$ is the solution size. On the other hand, Dom *et al.* [13] 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 $k^{O(1/\epsilon)}$. Our results for Connected Vertex Cover show that allowing an arbitrarily small multiplicative loss in precision drastically improves the worst-case behaviour of pre-processing 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 $O(\log n)$ approximation algorithm [33], and is known not to admit an approximation algorithm [20] with factor $O((\log n)^{\frac{1}{2}-\epsilon})$ 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 [6] unless NP $\subseteq$ 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 $k^{O(\frac{1}{\epsilon \log \epsilon})}$. 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 $\Sigma$ and a string $s$ in $\Sigma^*$. 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 $\Sigma$ such that there exists a collection $C$ of pairwise disjoint factors in $s$, such that for every $a \in S$ there is an $a$-factor in $C$. This stringology problem shows up in the proof of the kernelization lower bound of Bodlaender *et al.* [6] 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 [1] has factor $O(\frac{n}{\log n})$, and the problem cannot be approximated [24] within a factor $2^{(\log n)^{1-\epsilon}}$ for any $\epsilon > 0$, unless NP= 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 [4]. 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 $\alpha$-approximate kernel of polynomial size unless NP $\subseteq$ 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 [4, 5, 16, 19, 21] 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 [19] and Bodlaender *et al.* [5] 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 [36], and known not to admit a $(c \ln n)$-approximation algorithm for any $c < 1$ unless P = NP [31]. 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 [32] 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 $n^{O(1)}$ [13] unless NP $\subseteq$ 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 $\alpha$-approximate kernel of size $n^{O(1)}$ for any constant $\alpha$. This can be seen as a significant strengthening of the lower bound of Dom et al. [13]. 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 [32] 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.

## 2 SETTING UP THE FRAMEWORK

We begin by giving a "definition" of $\alpha$-approximate kernelization that ought to be sufficient for reading the rest of the paper and understanding most of the arguments. This will be followed by the formal definitions upon which the framework is built.

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 $\alpha$-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 $\alpha$-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 $\alpha$-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 $\alpha$-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 $k/s$.*

Typically we are interested in $\alpha$-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 $\alpha$-approximate kernels of smallest possible size, with smallest possible $\alpha$. 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 $\epsilon$. 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 $\epsilon$, we say that the PSAKS is *efficient*. All of the positive results achieved in this paper are PSAKSes, but not all are efficient.

## 2.1 Approximate Kernelization

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 2.1.* A parameterized optimization (minimization or maximization) problem $\Pi$ is a computable function

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

The *instances* of a parameterized optimization problem $\Pi$ 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 $\Pi$ 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 2.2.* For a parameterized minimization problem $\Pi$, the *optimum value* of an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ is

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

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

When the problem $\Pi$ 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 $\Pi$ is finite, therefore the minimum or maximum is well defined. We remark that the function $\Pi$ in Definition 2.1 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 Parameterized Complexity a problem is *fixed parameter tractable* if input instances of size $n$ with parameter $k$ can be "solved" in time $f(k)n^{O(1)}$ 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 2.3.* Let $\Pi$ be a parameterized optimization problem. An *algorithm for* $\Pi$ is an algorithm that given as input an instance $(I, k)$, outputs a solution $s$ and halts. The algorithm *solves* $\Pi$ 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 $\Pi$ is *decidable* if there exists an algorithm that solves $\Pi$.

*Definition 2.4.* A parameterized optimization problem $\Pi$ is *fixed parameter tractable* (FPT) if there is an algorithm that solves $\Pi$, such that the running time of the algorithm on instances of size $n$ with parameter $k$ is upper bounded by $f(k)n^{O(1)}$ for a computable function $f$.

We remark that Definition 2.3 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 [30], 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 $\infty$ 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: $VC(G, k, S) = \infty$ if $S$ is not a vertex cover of $G$ and $VC(G, k, S) = \min(|S|, k+1)$ otherwise. 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 [35] 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 $\infty$, 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.

*Structrural Parameterizations.* We now present 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 $\sigma$.

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: $OLA((G, S), k, \sigma) = -\infty$ if $S$ is not a vertex cover of $G$ of size at most $k$, $OLA((G, S), k, \sigma) = \infty$ if $\sigma$ is not a linear layout and $OLA((G, S), k, \sigma) = val(\sigma, G)$ otherwise. 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 $\sigma$ 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 2.5.* A **polynomial time pre-processing algorithm** $\mathcal{A}$ for a parameterized optimization problem $\Pi$ 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 $\Pi$ 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 $\Pi$, 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 pre-processed. 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 $\Pi$ 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 pre-processing algorithms whose output size is finite and bounded by a computable function of the parameter.

*Definition 2.6.* A **kernelization** (or *kernel*) for a parameterized optimization problem $\Pi$ is a polynomial time pre-processing algorithm $\mathcal{A}$ such that $\text{size}_{\mathcal{A}}$ is upper bounded by a computable function $g : \mathbb{N} \to \mathbb{N}$.

If the function $g$ in Definition 2.6 is a polynomial, we say that $\Pi$ admits a *polynomial kernel*. Similarly, if $g$ is a linear, quadratic or cubic function of $k$ we say that $\Pi$ 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 $\Pi$ is *decidable* if there exists an algorithm that solves $\Pi$, where the definition of "solves" is given in Definition 2.3.

Proposition 2.7. *A decidable parameterized optimization problem $\Pi$ is FPT if and only if it admits a kernel.*

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

Note that although Definition 2.8 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 $\alpha$-approximate kernel. We defined kernels by first defining polynomial time pre-processing algorithms (Definition 2.5) and then adding size constraints on the output (Definition 2.6). Similarly, we will first define the notion of $\alpha$-approximate polynomial time pre-processing algorithms, and then define $\alpha$-approximate kernels by adding size constraints on the output of the pre-processing algorithm.

*Definition 2.9.* Let $\alpha \geq 1$ be a real number and $\Pi$ be a parameterized optimization problem. An **$\alpha$-approximate polynomial time pre-processing algorithm** $\mathcal{A}$ for $\Pi$ 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 $\Pi$ 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 $\Pi$, 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 $\Pi$ is a minimization problem then

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

The definition for maximization problems is analogous. Definition 2.9 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 2.5 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 $\alpha$-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 2.5).

*Definition 2.10.* An **$\alpha$-approximate kernelization** (or an *$\alpha$-approximate kernel*) for a parameterized optimization problem $\Pi$, and real $\alpha \geq 1$, is an $\alpha$-approximate polynomial time pre-processing algorithm $\mathcal{A}$ such that $\text{size}_{\mathcal{A}}$ is upper bounded by a computable function $g : \mathbb{N} \to \mathbb{N}$.

Just as for regular kernels, if the function $g$ in Definition 2.10 is a polynomial, we say that $\Pi$ admits an $\alpha$-approximate polynomial kernel. If $g$ is a linear, quadratic or cubic function, then $\Pi$ admits a linear, quadratic or cubic $\alpha$-approximate kernel, respectively.

Proposition 2.7 establishes that a parameterized optimization problem $\Pi$ admits a kernel if and only if it is FPT. Next we establish a similar equivalence between FPT-approximation algorithms and approximate kernelization.

Proposition 2.11. *For every $\alpha \geq 1$ and decidable parameterized optimization problem $\Pi$, $\Pi$ admits a fixed parameter tractable $\alpha$-approximation algorithm if and only if $\Pi$ has an $\alpha$-approximate kernel.*

The proof of Proposition 2.11 is identical to the proof of Proposition 2.7, but with the FPT algorithm replaced by the fixed parameter tractable $\alpha$-approximation algorithm, and the kernel replaced with the $\alpha$-approximate kernel. On an intuitive level, it should be easier to compress an instance than it is to solve it. For $\alpha$-approximate kernelization this intuition can be formalized.

Theorem 2.12. *For every $\alpha \geq 1$ and decidable parameterized optimization problem $\Pi$, $\Pi$ admits a polynomial time $\alpha$-approximation algorithm if and only if $\Pi$ has an $\alpha$-approximate kernel of constant size.*

The proof of Theorem 2.12 is simple; if there is an $\alpha$-approximate kernel of constant size one can brute force the reduced instance and lift the optimal solution of the reduced instance to an $\alpha$-approximate solution to the original. On the other hand, if there is a factor $\alpha$ approximation algorithm, the reduction algorithm can just output any instance of constant size. Then, the solution lifting algorithm can just directly compute an $\alpha$-approximate solution to the original instance using the approximation algorithm.

We remark that Proposition 2.11 and Theorem 2.12 also applies to approximation algorithms and approximate kernels with super-constant approximation ratio. We also remark that with our definition of $\alpha$-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 which is a win from the perspective of optimization problems.

*Reduction Rules and Strict $\alpha$-Approximate Kernels.* Kernelization algorithms in the literature [9, 14] 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 2.5. 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 a kernel (as defined in Definition 2.6) we prove an upper bound on the size of a 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.

Unfortunately, one cannot chain $\alpha$-approximate polynomial time pre-processing algorithms, as defined in Definition 2.9, in this way. In particular, each successive application of an $\alpha$-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 2.13.* Let $\alpha \geq 1$ be a real number, and $\Pi$ be a parameterized optimization problem. An $\alpha$-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 $\Pi$ 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 $\Pi$ 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 2.13 is that an $\alpha$-strict approximate pre-processing algorithm may incur error on near-optimal solutions, but that they have to preserve factor $\alpha$-approximation. If $s'$ is an $\alpha$-approximate solution to $(I', k')$ then $s$ must be an $\alpha$-approximate solution to $(I, k)$ as well. Furthermore, if the ratio of $\Pi(I', k', s')$ to $OPT(I', k')$ is *worse* than $\alpha$, 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 2.13, also automatically satisfy the conditions of Definition 2.9. Therefore, to prove that $\mathcal{R}_{\mathcal{A}}$ and solution lifting algorithm constitute a strict $\alpha$-approximate polynomial time pre-processing algorithm it is not necessary to prove that they constitute a $\alpha$-approximate polynomial time pre-processing algorithm first. The advantage of Definition 2.13 is that strict $\alpha$-approximate polynomial time pre-processing algorithms do chain - the composition of two strict $\alpha$-approximate polynomial time pre-processing algorithms is again a strict $\alpha$-approximate polynomial time pre-processing algorithm.

We can now formally define what a reduction rule is. A reduction rule for a parameterized optimization problem $\Pi$ is simply a polynomial time algorithm computing a map $\mathcal{R}_{\mathcal{A}} : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$. 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 2.14.* A reduction rule is said to be $\alpha$-**safe for** $\Pi$ if there exists a solution lifting algorithm, such that the rule together with the solution lifting algorithm constitute a strict $\alpha$-approximate polynomial time pre-processing algorithm for $\Pi$. A reduction rule is **safe** if it is 1-safe.

*Definition 2.15.* An $\alpha$-approximate kernel $\mathcal{A}$ is called **strict** if $\mathcal{A}$ is a strict $\alpha$-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 $\epsilon$. However, as $\epsilon$ 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 $\epsilon$, but that the size of the kernel grows so fast when $\epsilon$ 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 2.16.* A **polynomial size approximate kernelization scheme** (PSAKS) for a parameterized optimization problem $\Pi$ is a family of $\alpha$-approximate polynomial kernelization algorithms, with one such algorithm for every $\alpha > 1$.

Definition 2.16 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 $\alpha$ outputs the source code of an $\alpha$-approximate polynomial kernelization algorithm for $\Pi$. In other words, one could think of a uniform PSAKS as a single $\alpha$-approximate polynomial kernelization algorithm where $\alpha$ is part of the input, and the size of the output depends on $\alpha$. 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 $\alpha$ can be upper bounded by $f(\alpha) \cdot k^{g(\alpha)}$ for some functions $f$ and $g$ independent of $|I|$ and $k$.

*Definition 2.17.* 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 $\alpha$ can be upper bounded by $f(\alpha) \cdot k^c$ for a function $f$ of $\alpha$ and constant $c$ independent of $I$, $k$ and $\alpha$. On the other hand, 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 $\alpha$ can be upper bounded by $f(\alpha) \cdot |I|^c$ for a function $f$ of $\alpha$ and constant $c$ independent of $I$, $k$, $\alpha$, 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 $\alpha$ can be upper bounded by $f'(\alpha) \cdot |I|^c$ for a function $f'$ of $\alpha$ and constant $c$ independent of $I$, $k$ and $\alpha$. As earlier, we say that a PSAKS is **strict** if it is a strict $\alpha$-approximate kernel for every $\alpha > 1$.

## 3 APPROXIMATE KERNEL FOR CONNECTED VERTEX COVER

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

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

We show that for every $\alpha > 1$, CVC has a polynomial size strict $\alpha$-approximate kernel. 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 3.1. *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 3.1. *Reduction Rule 3.1 is $\alpha$-safe.*

Proof. To show that Rule 3.1 is $\alpha$-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 $\alpha$-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

$$\frac{CVC(G, k, S)}{OPT(G, k)} \leq \frac{CVC(G', k', S') + D}{OPT(G', k') + (D - 1)}$$

By Fact 1, the quantity on the right hand side is upper bounded by $\max\{CVC(G', k', S')/OPT(G', k'), \alpha\}$. □

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

Lemma 3.2. *Reduction Rule 3.2 is 1-safe.*

Lemma 3.3. *Let $(G, k)$ be an instance irreducible by rules 3.1 and 3.2, such that $OPT(G, k) \leq k$. Then $|V(G)| \leq O(k^d + k^2)$.*

Theorem 3.4. Connected Vertex Cover *admits a strict time efficient PSAKS with $O(k^{\lceil \frac{\alpha}{\alpha-1} \rceil} + k^2)$ vertices.*

## 4 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 $\alpha$-approximate kernel, and then running a brute force search or an approximation algorithm on the reduced instance.

### 4.1 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 [7] 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 4.1 ([7]). *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 4.1 can easily be turned into a PSAKS for Steiner Tree parameterized by the number of terminals, defined below.

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

To get a $(1 + \epsilon)$-approximate kernel it is sufficent to pick $k$ based on $\epsilon$, 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 $O(|R|^k)$, 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 $O(\log |R|)$ bits for each weight is more than sufficient for the desired precision.

Theorem 4.2. 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 [36]. 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 $O(3^k|E(G)||V(G)|)$ using the algorithm of Dreyfus and Wagner[15]. Mark all the vertices in $V(T_{R'})$. After this process is completed, some $O(k|R|^k)$ 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 4.1 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 $O(\log |R| + \log \frac{1}{\epsilon})$ bits to store each edge weight. It follows that the bitsize of the instance $(G', R, \hat{w})$ is $|R|^{2^{O(1/\epsilon)}}$. It can be argued that, for every $c \geq 1$, a $c$-approximate Steiner tree $T'$ for the instance $(G', R, \hat{w})$ is a $c(1+\epsilon)$-approximate Steiner tree for the instance $(G, R, w)$, concluding the proof. □

## 5 LOWER BOUNDS FOR APPROXIMATE KERNELIZATION

In this section we set up a framework for proving lower bounds on the size of $\alpha$-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 [5]. In particular, Bodlaender et al. [5] defined cross composition and showed that if an NP-hard language $L$ admits a cross composition into a parameterized (decision) problem $\Pi$ and $\Pi$ admits a polynomial kernel, then $L$ has an *OR-distillation* algorithm. Fortnow and Santhanam [19] 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 $\Pi$, all we have to do is to find an NP-hard langluage $L$ and give a cross composition from $L$ into $\Pi$. Then, if $\Pi$ has a polynomial kernel, then combining the cross composition and the kernel with the results of Bodlaender et al. [5] and Fortnow and Santhanam [19] would prove that NP $\subseteq$ coNP/Poly. In other words a cross composition from $L$ into $\Pi$ proves that $\Pi$ does not have a polynomial kernel unless NP $\subseteq$ coNP/Poly.

In order to prove lower bounds on the size of $\alpha$-approximate kernels, we generalize the notion of cross compositions to $\alpha$-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 $\alpha$-gap cross compositions, we first need to recall the definition of Bodlaender et al. [5] of polynomial equivalence relations on $\Sigma^*$, where $\Sigma$ is a finite alphabet.

*Definition 5.1 (polynomial equivalence relation [5]).* An equivalence relation $R$ on $\Sigma^*$, where $\Sigma$ 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 $(\max_{x \in S} |x|)^{O(1)}$ equivalence classes.

*Definition 5.2 ($\alpha$-gap cross composition for maximization problem).* Let $L \subseteq \Sigma^*$ be a language, where $\Sigma$ is a finite alphabet and let $\Pi$ be a parameterized maximization problem. We say that $L$ *$\alpha$-gap cross composes* into $\Pi$ (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 $\Pi$ 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 $L$ $\alpha$-gap cross composes to $\Pi$.

One can similarly define $\alpha$-gap cross compositions for minimization problems.

*Definition 5.3.* The definition of $\alpha$-gap cross composition for minimization problem $\Pi$ can be obtained by replacing conditions $(i)$ and $(ii)$ of Definition 5.2 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 $\alpha$-approximate kernels, Definition 5.3 can be extended to encompass $\alpha$-gap cross composition where $\alpha$ 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 $\alpha$-approximate kernels where $\alpha$ is super-constant.

One of the main ingredients required 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 $\alpha$-approximate kernels. As we will see this is very similar to the definition of $\alpha$-approximate polynomial time pre-processing algorithm (Definition **??**).

*Definition 5.4.* Let $\alpha \geq 1$ be a real number. Let $\Pi$ and $\Pi'$ be two parameterized optimization problems. An **$\alpha$-approximate polynomial parameter transformation** ($\alpha$-appt for short) $\mathcal{A}$ from $\Pi$ to $\Pi'$ 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 $\Pi$ the reduction algorithm outputs an instance $(I', k')$ of $\Pi'$. The solution lifting algorithm takes as input an instance $(I, k)$ of $\Pi$, the output instance $(I', k') = \mathcal{R}_{\mathcal{A}}(I, k)$ of $\Pi'$, and a solution $s'$ to the instance $I'$ and outputs a solution $s$ to $(I, k)$. If $\Pi$ is a minimization problem then

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

The definition for maximization problems is analogous.

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 $\alpha$-approximate kernel to $\alpha$-approximate compression. The only difference is that in the later case the reduced instance can be an instance of any parameterized optimization problem.

*Definition 5.5.* Let $\alpha \geq 1$ be a real number. Let $\Pi$ and $\Pi'$ be parameterized optimization problems. An **$\alpha$-approximate compression** from $\Pi$ to $\Pi'$ is an $\alpha$-appt $\mathcal{A}$ from $\Pi$ to $\Pi'$ such that $\text{size}_{\mathcal{A}}(k) = \sup\{|I'| + k' : (I', k') = \mathcal{R}_{\mathcal{A}}(I, k), I \in \Sigma^*\}$, is upper bounded by a computable function $g : \mathbb{N} \to \mathbb{N}$, where $\mathcal{R}_{\mathcal{A}}$ is the reduction algorithm.

For the sake of proving approximate kernel lower bounds, it is immaterial that $\Pi'$ in the Definition 5.5 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 $\alpha$-approximate compression and we do not specify the target problem $\Pi'$, we mean the existence of $\alpha$-approximate compression into any optimization problem $\Pi'$. For more detailed exposition about lower bound machinery about polynomial compression for decision problems we refer to the textbook [9].

Towards building a framework for lower bounds we would like to prove a theorem analogous to the one by Bodlaender et al. [5]. In particular, we would like to show that an $\alpha$-gap cross composition from an NP-hard language $L$ into a parameterized optimization problem $\Pi$, together with an $\alpha$-approximate compression of polynomial size yield an OR-distillation for the language $L$. Then the result of Fortnow and Santhanam [19] would immediately imply that any parameterized optimization problem $\Pi$ that has an $\alpha$-gap cross composition from an NP-hard language $L$ cannot have an $\alpha$-approximate compression unless $\text{NP} \subseteq \text{coNP/Poly}$. 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 [19], we make use of the more general result of Dell and van Melkebeek [11] that rules out cheap oracle communication protocols for NP-hard problems. We first give necessary definitions that allow us to formulate our statements.

*Definition 5.6 (Oracle Communication Protocol [11]).* Let $L \subseteq \Sigma^*$ be a language, where $\Sigma$ 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 5.7 (Complementary Witness Lemma [11]). *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 $O(t(s) \log t(s))$, where the first player can be conondeterministic. Then $L \in \text{coNP/Poly}$.*

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

Lemma 5.8. *Let $L$ be a language and $\Pi$ be a nice parameterized optimization problem. If $L$ $\alpha$-gap cross composes to $\Pi$, and $\Pi$ has a polynomial sized $\alpha$-approximate compression, then $L \in \text{coNP/Poly}$.*

Theorem 5.9. *Let $L$ be an NP-hard language and $\Pi$ be a nice parameterized optimization problem. If $L$ $\alpha$-gap cross composes to $\Pi$, and $\Pi$ has a polynomial sized $\alpha$-approximate compression, then $\text{NP} \subseteq \text{coNP/Poly}$.*

We note that Lemma 5.7 applies even if the first player works in co-nondeterministic polynomial time. Thus, a co-nondeterministic $\alpha$-gap cross composition together with an $\alpha$-approximate compression from an NP-hard language would still allow us to conclude that NP $\subseteq$ coNP/Poly.

*Longest Path.* We now show that Longest Path does not admit an $\alpha$-approximate compression of polynomial size for any $\alpha \geq 1$ unless NP $\subseteq$ coNP/Poly. The parameterized optimization version of the Longest Path problem, that we call Path, is defined as follows: $\textsc{Path}(G, k, P) = -\infty$ if $P$ is not a path in $G$ and $\textsc{Path}(G, k, P) = \min\{k+1, |V(P)|-1\}$ otherwise. We show that Path does not have a polynomial sized $\alpha$-approximate compression for any constant $\alpha \geq 1$. We prove this by giving an $\alpha$-gap cross composition from a $\alpha$-Gap Long Path. The problem $\alpha$-Gap Long Path is a promise problem which is defined as follows.

*Definition 5.10.* The $\alpha$-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 $\alpha$-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 $\alpha$-Gap Long Path.

It is known that $\alpha$-Gap Long Path is NP-hard [24]. We show that it in fact also $\alpha$-gap cross composes to Path for any $\alpha \geq 1$.

Lemma 5.11. *$\alpha$-Gap Long Path $\alpha$-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 $\alpha$-Gap Long Path, the $\alpha$-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 $\alpha$-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 $\alpha$-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. □

Theorem 5.12. Path *does not have an $\alpha$-approximate compression of polynomial size for any $\alpha \geq 1$, unless* NP $\subseteq$ coNP/Poly.

## 6 CONCLUSION AND DISCUSSIONS

Our framework for studying lossy kernelization, and the methods for showing lower bounds for approximate kernelization point to 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 and so on. We conclude with a list of concrete problems for future research.

- 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?
- Do Multiway Cut and 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 $(s_1, t_1), (s_2, t_2), \ldots, (s_\ell, t_\ell)$. 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.
- 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 NP $\not\subseteq$ coNP/Poly or another reasonable complexity theoretic assumption?
- One may extend the notion of approximate kernels to approximate Turing kernels [9] in a natural way. Does Independent Set parameterized by treewidth admit a polynomial size approximate Turing kernel with a constant approximation ratio?
- 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? [3, 25]
- The notion of $\alpha$-gap cross compositions can be modified to "AND $\alpha$-gap cross compositions" in the same way that AND-compositions relate to OR-compositions [4]. In order to directly use such "AND $\alpha$-gap cross compositions" to show lower bounds for approximate kernelization, one needs an analogue of Lemma 5.7 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 [4, 16] to oracle communication protocols (see the conclusion section of Drucker [16], open question number 1). Can this strengthening of the AND-distillation conjecture be related to a well known complexity theoretic assumption?

## ACKNOWLEDGEMENTS

The authors thank Dániel Marx and for enlightening discussions on related work in the literature, and Magnus Wahlström for pointing out the remark about randomized pre-processing algorithms following Definition 2.5.

## REFERENCES

[1] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-Coding. *J. ACM* 42, 4 (1995), 844–856.
[2] Esther M. Arkin, Magnús M. Halldórsson, and Refael Hassin. 1993. Approximating the Tree and Tour Covers of a Graph. *Inf. Process. Lett.* 47, 6 (1993), 275–282.
[3] Sanjeev Arora, Boaz Barak, and David Steurer. 2015. Subexponential Algorithms for Unique Games and Related Problems. *J. ACM* 62, 5 (2015), 42.
[4] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. 2009. On problems without polynomial kernels. *J. Comput. Syst. Sci.* 75, 8 (2009), 423–434.
[5] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. 2011. Cross-Composition: A New Technique for Kernelization Lower Bounds. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS)*. 165–176.
[6] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. 2011. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.* 412, 35 (2011), 4570–4578.
[7] Al Borchers and Ding-Zhu Du. 1997. The k-Steiner Ratio in Graphs. *SIAM J. Comput.* 26, 3 (1997), 857–869.
[8] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. 2013. Steiner Tree Approximation via Iterative Randomized Rounding. *J. ACM* 60, 1 (2013), 6.
[9] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. 2015. *Parameterized Algorithms*. Springer.
[10] Holger Dell and Dániel Marx. 2012. 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*. 68–81.
[11] Holger Dell and Dieter van Melkebeek. 2010. 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*. 251–260.
[12] Irit Dinur and Samuel Safra. 2005. On the hardness of approximating minimum vertex cover. *Annals of mathematics* (2005), 439–485.
[13] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. 2014. Kernelization Lower Bounds Through Colors and IDs. *ACM Transactions on Algorithms* 11, 2 (2014), 13:1–13:20.
[14] Rodney G Downey and Michael Ralph Fellows. 2012. *Parameterized complexity*. Springer Science & Business Media.
[15] S. E. Dreyfus and R. A. Wagner. 1971. The steiner problem in graphs. *Networks* 1, 3 (1971), 195–207.
[16] Andrew Drucker. 2015. New Limits to Classical and Quantum Instance Compression. *SIAM J. Comput.* 44, 5 (2015), 1443–1479.
[17] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Hadas Shachnai. 2013. Tractable Parameterizations for the Minimum Linear Arrangement Problem. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*. 457–468.
[18] Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. 2012. 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*. 351–362.
[19] Lance Fortnow and Rahul Santhanam. 2011. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.* 77, 1 (2011), 91–106.
[20] Zachary Friggstad and Mohammad R. Salavatipour. 2011. Approximability of Packing Disjoint Cycles. *Algorithmica* 60, 2 (2011), 395–400.
[21] Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. 2015. A Completeness Theory for Polynomial (Turing) Kernelization. *Algorithmica* 71, 3 (2015), 702–730.
[22] Danny Hermelin and Xi Wu. 2012. 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*. 104–113.
[23] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. 2001. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.* 63, 4 (2001), 512–530.
[24] David R. Karger, Rajeev Motwani, and G. D. S. Ramkumar. 1997. On Approximating the Longest Path in a Graph. *Algorithmica* 18, 1 (1997), 82–98.
[25] Subhash Khot. 2002. 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*. 767–775.
[26] Subhash Khot and Oded Regev. 2008. Vertex cover might be hard to approximate to within 2- $\varepsilon$. *J. Comput. System Sci.* 74, 3 (2008), 335–349.
[27] Stefan Kratsch. 2014. Recent developments in kernelization: A survey. *Bulletin of the EATCS* 113 (2014).
[28] Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. 2012. Kernelization–preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*. Springer, 129–161.
[29] Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. 2016. Lossy Kernelization. *CoRR* abs/1604.04111 (2016). http://arxiv.org/abs/1604.04111
[30] Dniel Marx. 2008. Parameterized Complexity and Approximation Algorithms. *Comput. J.* 51, 1 (2008), 60–78.
[31] Dana Moshkovitz. 2015. The Projection Games Conjecture and the NP-Hardness of ln n-Approximating Set-Cover. *Theory of Computing* 11 (2015), 221–235.
[32] Jelani Nelson. 2007. A Note on Set Cover Inapproximability Independent of Universe Size. *Electronic Colloquium on Computational Complexity (ECCC)* 14, 105 (2007).
[33] Mohammad R. Salavatipour and Jacques Verstraëte. 2005. Disjoint Cycles: Integrality Gap, Hardness, and Approximation. In *Integer Programming and Combinatorial Optimization, 11th International IPCO Conference, Berlin, Germany, June 8-10, 2005, Proceedings*. 51–65.
[34] Carla D. Savage. 1982. Depth-First Search and the Vertex Cover Problem. *Inf. Process. Lett.* 14, 5 (1982), 233–237.
[35] Michael Sipser. 2012. *Introduction to the Theory of Computation*. Cengage Learning.
[36] David P. Williamson and David B. Shmoys. 2011. *The Design of Approximation Algorithms*. Cambridge University Press.