α-approximate Reductions: a Novel Source of Heuristics for Better Approximation Algorithms

Fredrik Manne
University of Bergen, Bergen, Norway
Fredrik.Manne@uib.no

Geevarghese Philip
Chennai Mathematical Institute, India
gphilip@cmi.ac.in

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Bergen, Norway
saket@imsc.res.in

Prafullkumar Tale
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
prafullkumar.tale@cispa.saarland

Abstract

Lokshtanov et al. [STOC 2017] introduced *lossy kernelization* as a mathematical framework for quantifying the effectiveness of preprocessing algorithms in preserving approximation ratios. α-*approximate reduction rules* are a central notion of this framework. We propose that carefully crafted α-approximate reduction rules can yield improved approximation ratios in practice, while being easy to implement as well. This is distinctly different from the (theoretical) purpose for which Lokshtanov et al. designed α-approximate Reduction Rules. As evidence in support of this proposal we present a new 2-approximate reduction rule for the Dominating Set problem. This rule, when combined with an approximation algorithm for Dominating Set, yields significantly better approximation ratios on a variety of benchmark instances as compared to the latter algorithm alone.

The central thesis of this work is that α-approximate reduction rules can be used as a tool for designing approximation algorithms which perform better in practice. To the best of our knowledge, ours is the first exploration of the use of α-approximate reduction rules as a design technique for practical approximation algorithms. We believe that this technique could be useful in coming up with improved approximation algorithms for other optimization problems as well.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases α-approximate Reduction Rules, Lossy Kernel, Dominating Set, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Supplement Material *Code Repository.* URL: `https://github.com/pptale/DomSet-Lossy`

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:11

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The notion of *kernelization* as defined in the field of parameterized algorithms is a mathematical tool for quantifying the effectiveness of polynomial-time preprocessing algorithms in dealing with decision problems [6, 11]. An instance of a parameterized problem is of the form (I, k) where I is an instance of the underlying—usually, but not necessarily, NP-hard—decision problem and k is a—usually, but not necessarily, numerical—*parameter*. The informal idea is that the parameter k captures some aspect which makes the instance I hard to solve. A *kernelization algorithm* for this parameterized problem takes (I, k) as input, runs in time polynomial in $(|I| + k)$, and outputs an instance (I', k') of the same parameterized problem. Further, I' is a YES instance of the decision problem if and only if I is a YES instance, and the size $(|I'| + k')$ of (I', k') is upper-bounded by some function $f(k)$ of the parameter k alone. (I', k') is then a *kernel* of size at most $f(k)$. A kernelization algorithm is thus a polynomial-time preprocessing algorithm with a guaranteed upper bound on output size, where the guarantee comes in terms of a function of the chosen parameter k.

The notion of kernelization has turned out to be very prescient and fruitful; useful and non-trivial upper and lower bounds on kernel sizes have been derived for literally hundreds of decision problems (See, e.g., [11]). Nonetheless, kernelization has one significant drawback when it comes to judging the effectiveness of preprocessing algorithms. In general, a kernelization algorithm gives no guarantees as to how (or whether) the quality of *approximate solutions* for a kernel relate to the quality of approximate solutions for the original input instance. Kernelization mandates only that the reduced instance (i) be of bounded size, and (ii) preserve the YES/NO answer to the original decision question. It gives no guarantees that a "good" solution for the kernel, obtained by running an approximation algorithm or a heuristic on the kernel, can be converted to a "good" solution for the original instance. This is a serious handicap in practice because in many cases the output of preprocessing algorithms are still too large to be solved exactly, and so must be processed further by approximation algorithms or heuristics. Using kernelization to do the preprocessing may then prevent us from getting—with any sort of approximation guarantee—good solutions to the actual input instance, from good solutions to the kernel.

Lokshtanov et al. [14] introduced *lossy kernelization* as a theoretical framework for studying those polynomial-time preprocessing algorithms which give a two-fold guarantee: (i) on the size of the reduced instance, and (ii) on the loss in approximation factor with which one can convert a solution of a reduced instance to a solution of the input instance. Such algorithms are called *α-approximate kernelization algorithms* or *α-approximate kernels*. We now give an informal description of these algorithms; please see Subsection 2.2 for precise definitions. For a real number $\alpha \geq 1$, an *α-approximate reduction rule* for a parameterized optimization problem consists of a *pair* of algorithms, a *reduction algorithm* and a *solution-lifting algorithm*. The reduction algorithm takes an instance (I, k) of the problem as input, runs in time polynomial in $(|I| + k)$, and outputs an instance (I', k') of the problem. The solution-lifting algorithm takes the two instances $(I, k), (I', k')$ *and* an arbitrary solution S' to (I', k') and constructs a solution S to (I, k) with the following guarantee: If $\beta \geq 1$ is the approximation factor of S' with respect to (I', k') then the approximation factor of S with respect to (I, k) is *no worse than* $\alpha \times \beta$. That is, if we manage to get hold of a "good" approximate solution to the reduced instance (I', k') then the solution-lifting algorithm will give us a solution to the original instance which is not much worse in terms of the respective approximation factors. An *α-approximate kernel* is an α-approximate reduction rule for which the size $(|I'| + k')$ of the reduced instance is upper-bounded by some function of the

parameter k.

In their pioneering work Lokshtanov et al. showed, *inter alia*, that various parameterized problems which do not[1] admit "classical" polynomial kernels have α-approximate kernels (also called "lossy kernels") of polynomial size. This line of work has attracted much attention, and finding lossy kernels of polynomial size, especially for problems which don't admit classical kernels of polynomial size, is an important—and challenging—area of current research in parameterized algorithms [13, 8, 7, 9].

Our Work. We posit that the notion of α-approximate reduction rules has important practical utility beyond, and somewhat orthogonal to, their original theoretical *raison d'être* which we have outlined above. Specifically, we claim that this notion can be used as *an aid for designing approximation algorithms* which work well *in practice*. That is, we propose that starting out by designing an α-approximate reduction rule is a good way to arrive at a heuristic which is easy to implement and has good approximation characteristics. The intuition behind this claim is that the various conditions which—by definition—an α-approximate reduction rule should satisfy have the combined effect that such a rule is likely to have these desirable qualities.

Needless to say, this claim is far from being well-defined. So we can only provide some evidence, and not a proof, in support of the claim. As evidence we present a 2-approximate reduction rule for the well-studied Dominating Set problem, and show by empirical means that this rule is a good approximation heuristic for Dominating Set. We show that when combined with a state-of-the-art approximation algorithm for Dominating Set, the new 2-approximate reduction rule gives improved approximation ratios for a variety of benchmark instances drawn from various sources.

Our Results. Our main theoretical contribution is the new 2-approximate reduction rule for Dominating Set, along with the proof that the rule is indeed 2-approximate. Our main practical contribution is a new heuristic approximation algorithm for Dominating Set. At a high level, this algorithm works as follows. It starts by exhaustively applying two (folklore) 1-approximate reduction algorithms. It then applies the new 2-approximate reduction algorithm once to the resulting instance, to get a reduced instance. It then runs a known approximation algorithm (the "drop-in" approximation algorithm) for Dominating Set on the reduced instance to get an approximate solution for the reduced instance. Finally, it applies the various solution-lifting algorithms in the right order to get an approximate solution to the input instance.

We tested this new heuristic algorithm on various well-known benchmark instances. Since all these benchmark instances consisted of *sparse* graphs, we used the approximation algorithm of Jones et al. [12] for Dominating Set on sparse graphs as the drop-in approximation algorithm. As we describe in Section 5, our three-stage algorithm—first apply the various reductions, then apply the drop-in approximation, and finally apply the lifting algorithms—gives significantly better results on a good fraction of the benchmark instances, as compared to applying just the drop-in algorithm alone to these instances.

Organization of the rest of the paper. In Subsection 2.1 we list various graph-theoretic preliminaries, and in Subsection 2.2 we give a concise description of those concepts from the

[1] Under standard complexity-theoretic assumptions.

lossy kernels framework which we need in the rest of the paper. In Section 3 we describe the two folklore reduction rules for Dominating Set and prove that each of them is 1-approximate. In the same section we state our new reduction rule for Dominating Set and prove that it is 2-approximate. In Section 4 we describe our experimental setup, and in Section 5 we describe our algorithm and the results we obtained on the various benchmark instances. We conclude in Section 6.

2 Preliminaries

For positive constants x, y, z, w, we have $\frac{x+y}{z+w} \leq \max\{\frac{x}{z}, \frac{y}{w}\}$.

2.1 Graph Theory

All our graphs are undirected and simple. For a graph G we use $V(G)$ and $E(G)$ to denote the set of vertices and edges of G, respectively. Unless specified otherwise, we use n, m to denote the cardinalities of sets $V(G)$ and $E(G)$, respectively. Two vertices u, v are said to be *adjacent* if there is an edge uv in the graph. The *neighborhood* of a vertex v, denoted by $N_G(v)$, is the set of vertices adjacent to v and its *degree* $deg_G(v)$ is $|N_G(v)|$. The *closed neighborhood* of a vertex v, denoted $N_G[v]$, is $\{v\} \cup N(v)$. The subscript in the notation for neighborhood and degree is omitted if the graph being discussed is clear. For a subset S of $V(G)$ we define $N(S) = (\bigcup_{v \in S} N(S)) \setminus S$ and $N[S] = \bigcup_{v \in S} N[s]$. For set $S \subseteq V(G)$ we use $G - S$ to denote the graph obtained by deleting S from G, and $G[S]$ to denote the subgraph of G *induced* by S. An *isolated* vertex is not adjacent to any other vertex. A *pendant* is a vertex of degree one.

The *average degree* of graph G, denoted by $avg\_deg(G)$, is defined as $\frac{1}{|V(G)|} \cdot \sum_{v \in V(G)} deg(v)$. It is easy to see that $avg\_deg(G) = 2|E(G)|/V(G)$. A graph G is called *d-degenerate* if every subgraph of G contains a vertex of degree at most d. If G is a d-degenerate graph then $|E(G)| \leq d|V(G)|$.

We say that a vertex set S *dominates* another vertex set W if W is contained in $N[S]$. A *dominating set* for a graph is a set of vertices that dominates the entire vertex set of the graph. More formally, a vertex subset S is a *dominating set* of graph G if $V(G) = N[S]$. Equivalently, a dominating set of G is a set of vertices S such that every vertex which is not in S is adjacent with some vertex in S. In the decision version of the Dominating Set problem, the input consists of a graph G and an integer k, and the task is to determine whether G has a dominating set of size at most k. We work with a generalized version of this problem. In this version, we are given a graph G, an integer k and a bi-partition (V_b, V_r) of vertex set $V(G)$. The objective is to find, if it exists, a vertex subset of size at most k which dominates all the vertices in V_b. Formally, the task is to determine whether there exists a vertex set $S \subseteq V_b \cup V_r$ such that $|S| \leq k$ and $V_b \subseteq N[S]$. The sets V_b, V_r can be thought of as vertices with colours *blue* and *red*, respectively. Vertices coloured blue are yet to be dominated while red vertices have already been dominated. Note that the classical Dominating Set problem reduces to this version by setting $V_b = V(G)$ and $V_r = \emptyset$.

2.2 Some Concepts from the Lossy Kernelization Framework

This section contains a brief overview of those concepts from the lossy kernelization framework that we need to describe our work. We encourage the reader to see [14] for a more comprehensive discussion of this framework. We start with the definition of a *parameterized*

optimization (maximization/minimization) problem, which is the parameterized analog of an optimization problem in the theory of approximation algorithms.

▶ **Definition 1.** *A parameterized optimization (maximization / minimization) problem is a computable function* $\Pi : \Sigma^* \times \mathbb{N} \times \Sigma^* \mapsto \mathbb{R} \cup \{\pm\infty\}$.

The instances of Π are pairs $(I, k) \in \Sigma^* \times \mathbb{N}$ and a solution to (I, k) is simply a string $S \in \Sigma^*$ such that $|S| \leq |I| + k$. The *value* of a solution S is $\Pi(I, k, S)$. In this article, we work with a *parameterized minimization problem*; *parameterized maximization problems* can be dealt with in a similar way. The *optimum value* of (I, k) is defined as: $\mathrm{OPT}_\Pi(I, k) = \min_{S \in \Sigma^*, |S| \leq |I|+k} \Pi(I, k, S)$, and an *optimum solution* for (I, k) is a solution S such that $\Pi(I, k, S) = \mathrm{OPT}_\Pi(I, k)$. For a constant $c \geq 1$, S is a *c-factor approximate* solution for (I, k) if $\frac{\Pi(I,k,S)}{OPT_\Pi(I,k)} \leq c$. We omit the subscript Π if the problem being discussed is clear from the context. We now formally define an α-approximate reduction rule.

▶ **Definition 2.** *Let* $\alpha \geq 1$ *be a real number and* Π *be a parameterized minimization problem. An* α-approximate reduction rule *is defined as a pair of polynomial-time algorithms, called the* reduction algorithm *and the* solution lifting algorithm*, that satisfy the following properties.*

- *Given an instance* (I, k) *of* Π*, the reduction algorithm computes an instance* (I', k') *of* Π.
- *Given the instances* (I, k) *and* (I', k') *of* Π*, and a solution* S' *to* (I', k')*, the solution lifting algorithm computes a solution* S *to* (I, k) *such that* $\frac{\Pi(I,k,S)}{\mathrm{OPT}(I,k)} \leq \alpha \cdot \frac{\Pi(I',k',S')}{\mathrm{OPT}(I',k')}$.

▶ **Definition 3.** *An* α*-approximate reduction rule is said to be* strict *if for every instance* (I, k)*, reduced instance* (I', k') *(which is obtained by applying reduction algorithm on* (I, k)*) and solution* S' *to* (I, k)*, the solution lifting algorithm produces a solution* S *to* (I, k) *which satisfies* $\frac{\Pi(I,k,s)}{\mathrm{OPT}(I,k)} \leq \max\{\frac{\Pi(I',k',s')}{\mathrm{OPT}(I',k')}, \alpha\}$.

▶ **Definition 4.** *An* α-approximate kernel *for* Π *is an* α*-approximate reduction rule such that the size* $(|I'| + k')$ *of the output instance of the reduction algorithm is upper bounded by a computable function of the original parameter* k.

For the sake of completeness, we present a definition of (classical) kernelization.

▶ **Definition 5.** *Given an instance* (I, k) *of some problem* Π *as input, a kernelization algorithm returns an instance* (I', k') *of* Π *which satisfies following properties:*

- $|I'| + k'$ *is upper bounded by some computable function* $f(k)$ *which depends only on* k.
- (I, k) *is a* Yes *instance of* Π *if and only if* (I', k') *is a* Yes *instance of* Π.

For more details on parameterized complexity and kernelization we refer the reader to the books of Downey and Fellows [6], Flum and Grohe [10], Niedermeier [16], and the more recent books by Cygan et al. [4] and Fomin et al. [11].

3 α-approximate Reduction Rules for Dominating Set

In the previous section we defined a generalized version of the Dominating Set problem in which the input is of the form $(G(V_b, V_r), k)$ where G is a graph, k is a positive integer and (V_b, V_r) is a partition of $V(G)$. The objective is to determine whether there exists a vertex set $S \subseteq V_b \cup V_r$ of size at most k such that $V_b \subseteq N[S]$ holds. We define an optimization version of this generalized Dominating Set problem in the following way.

$$\mathrm{DS}(G(V_b, V_r), k, S) = \begin{cases} \infty & \text{if } V_b \not\subseteq N[S] \\ \min\{|S|, k+1\} & \text{otherwise} \end{cases}$$

Note that with this definition, one can treat all dominating sets of size greater or equal to $k+1$ as *equally bad*. From the theoretical point of view it is crucial to define the problem in this manner, as explained in the part titled **Capping the objective function at** $k+1$ on Page 15 of the arXiv version [15] of the pioneering paper of Lokshtanov et al. [14]. However, from a practical point of view, it is not necessary to distinguish solutions whose size lie above or below a specific value. In fact, in most of the practical cases, the parameter k is not known in advance. For the sake of clarity, we replace $\mathrm{DS}(G(V_b, V_r), k, S)$ by $\mathrm{DS}(G(V_b, V_r), S)$ and $\min\{|S|, k+1\}$ by $|S|$ in the above definition. Hence the input of the optimization version is simply $G(V_b, V_r)$. Note that no reduction rule mentioned below changes the value of k. We remark that in classical kernelization, similar reduction rules decrease the value of k but in the optimization version, we can prove the safeness without decreasing the value of k. Hence, we can justify the above simplification in the definition by assuming that the given value of k is equal to $|V(G)|$ which does not change.

The minimum value taken over all valid solutions is denoted by $\mathrm{OPT}(G(V_b, V_r))$. We start with the following simple reduction rules and their corresponding solution lifting algorithms. In classical kernelization, these two rules correspond to taking isolated vertices and the neighbours of pendant vertices into a solution.

▶ **Reduction Rule 3.1.** *Let X be the set of vertices in V_b such that every vertex in X is an isolated vertex in G. Then, move all vertices in X from V_b to V_r. Formally, let $V_b' = V_b \setminus X$ and $V_r' = V_r \cup X$. Return $G(V_b', V_r')$.*

▶ **Solution Lifting Algorithm 3.1.** *Let S' be a solution for $G(V_b', V_r')$ and X be the set of vertices moved from V_b to V_r. Return $S' \cup X$ as a solution for $G(V_b, V_r)$.*

▶ **Lemma 6.** *Reduction Rule 3.1 and Solution Lifting Algorithm 3.1 together constitute a strict 1-approximate reduction rule.*

Proof. Since X is a set of vertices in V_b, any dominating set of G must contain some subset of vertices which dominates X. As X is a collection of isolated vertices, the only way to dominate X is to include all of X in the solution. This implies that $\mathrm{OPT}(G(V_b', V_r')) \leq \mathrm{OPT}(G(V_b, V_r)) - |X|$ holds, since X is contained in the set V_r' in the reduced instance.

Let S' be a solution for $G(V_b', V_r')$. Then $V_b' \subseteq N[S']$ holds. Since $V_b = V_b' \cup X$, the set $S' \cup X$ is a solution for $G(V_b, V_r)$. Hence $\mathrm{DS}(G(V_b, V_r), S' \cup X) \leq |S'| + |X| \leq \mathrm{DS}(G(V_b', V_r')), S') + |X|$ holds.

Combining these two inequalities, we obtain the following inequality which concludes the proof.

$$\frac{\mathrm{DS}(G(V_b, V_r), S' \cup X)}{\mathrm{OPT}(G(V_b, V_r))} \leq \frac{\mathrm{DS}(G(V_b', V_r')), S') + |X|}{\mathrm{OPT}(G(V_b', V_r')) + |X|} \leq \max\left\{\frac{\mathrm{DS}(G(V_b', V_r')), S')}{\mathrm{OPT}(G(V_b', V_r'))}, 1\right\}.$$

◀

▶ **Reduction Rule 3.2.** *Let v be a vertex in V_b such that v is a pendant vertex in G, and let u be the unique neighbor of v in G. Move all vertices in $N[u]$ from V_b to V_r. Formally, let $V_b' = V_b \setminus N[u]$ and $V_r' = V_r \cup N[u]$. Return $G(V_b', V_r')$.*

▶ **Solution Lifting Algorithm 3.2.** *Let S' be a solution for $G(V_b', V_r')$ and u be the vertex mentioned in Reduction Rule 3.2. Return $S' \cup \{u\}$ as a solution for $G(V_b, V_r)$.*

▶ **Lemma 7.** *Reduction Rule 3.2 and Solution Lifting Algorithm 3.2 together constitute a strict 1-approximate reduction rule.*

Proof. Since vertex v is in V_b, any dominating set of G must contain a subset of vertices which dominates v. As v is a pendant vertex, the only vertices which can dominate v are the unique neighbor u of v, or v itself. This implies that $\mathrm{OPT}(G(V_b', V_r')) \leq \mathrm{OPT}(G(V_b, V_r)) - |1|$ holds, as $N[u]$ is now part of V_r' in the reduced instance.

Let S' be a solution for $G(V_b', V_r')$. Then $V_b' \subseteq N[S']$ holds. Since $V_b = V_b' \cup N[u]$, the set $S' \cup \{u\}$ is a solution for $G(V_b, V_r)$. Hence $\mathrm{DS}(G(V_b, V_r), S' \cup \{u\}) \leq |S'| + 1 \leq \mathrm{DS}(G(V_b', V_r')), S') + 1$ holds.

Combining these two inequalities, we obtain the following inequality which concludes the proof.

$$\frac{\mathrm{DS}(G(V_b, V_r), S' \cup \{u\})}{\mathrm{OPT}(G(V_b, V_r))} \leq \frac{\mathrm{DS}(G(V_b', V_r')), S') + 1}{\mathrm{OPT}(G(V_b', V_r')) + 1} \leq \max\left\{\frac{\mathrm{DS}(G(V_b', V_r')), S')}{\mathrm{OPT}(G(V_b', V_r'))}, 1\right\}.$$

◀

Note that we apply Reduction Rule 3.1 only once while we keep repeatedly applying Reduction Rule 3.2 as long as there is a pendant vertex in V_b. It is not difficult to see that the following lemma holds.

▶ **Lemma 8.** *There exists an algorithm that takes a graph $G(V_b, V_r)$ as input, runs in $\mathcal{O}(n + m)$ time, and exhaustively applies Reduction Rules 3.1 and 3.2.*

We now present the final reduction rule.

▶ **Reduction Rule 3.3.** *Find a subset X of vertices in V_b such that there is a one-to-one function $\psi : X \to V_b \setminus X$ where*
1. *for every x in X, $\psi(x) \notin N[x]$,*
2. *for any two vertices x_1, x_2 in X, the vertex $\psi(x_2)$ is not in the set $N(x_1)$, and,*
3. *for any two distinct vertices x_1, x_2 in X, the set $N[\psi(x_1)] \cap N[\psi(x_2)]$ is empty.*

Move all vertices in $N[X] \cap V_b$ to V_r. Formally, let $V_b' = V_b \setminus N[X]$ and $V_r' = V_r \cup N[X]$. Return $G(V_b', V_r')$.

▶ **Solution Lifting Algorithm 3.3.** *Let S' be a solution for $G(V_b', V_r')$ and X be the subset of V_b mentioned in Reduction Rule 3.3. Return $S' \cup X$ as a solution for $G(V_b, V_r)$.*

▶ **Lemma 9.** *Reduction Rule 3.3 and Solution Lifting Algorithm 3.3 together constitute a 2-approximate reduction rule.*

Proof. As $V_b' \subseteq V_b$ holds, any solution for $G(V_b, V_r)$ is also a solution for $G(V_b', V_r')$. This implies $\mathrm{OPT}(G(V_b', V_r')) \leq \mathrm{OPT}(G(V_b, V_r))$.

Let the set X be as defined in Reduction Rule 3.3, and let Z be the range of the function ψ. Formally, $Z := \{z \in V(G) \setminus X \mid \exists x \in X \ ; \ \psi(x) = z\}$. By Properties 1 and 2 of Reduction Rule 3.3, $Z \cap N[X]$ is the empty set. Since only vertices in $N[X]$ have been moved from V_b to V_r, we have that $Z \subseteq V_b'$ holds. Hence any solution S' for $G(V_b', V_r')$ contains at least one vertex from $N[z]$ for every vertex z in Z. By Property 3 of Reduction Rule 3.3, for any two vertices z_1, z_2 in Z, we have that $N[z_1] \cap N[z_2] = \emptyset$ holds. This implies that $|Z| \leq |S'|$ holds. Since ψ is a one-to-one function we get that $|X| = |Z| \leq |S'|$ holds, and this in turn implies that $|S' \cup X| \leq 2|S'|$ holds.

Combining the above inequalities we obtain the following inequality which concludes the proof.

$$\frac{\mathrm{DS}(G(V_b, V_r), S' \cup X)}{\mathrm{OPT}(G(V_b, V_r))} \leq \frac{\mathrm{DS}(G(V_b', V_r')), S') + |X|}{\mathrm{OPT}(G(V_b', V_r')) + |X|} \leq 2 \cdot \frac{\mathrm{DS}(G(V_b', V_r')), S')}{\mathrm{OPT}(G(V_b', V_r'))}$$

◀

We remark that we apply this 2-approximate reduction rule only once.

▶ **Lemma 10.** *There exists an algorithm that takes a graph $G(V_b, V_r)$ as input, runs in time $\mathcal{O}(n \log(n) + m)$ time, and applies Reduction Rule 3.3 once.*

Proof. We present a simple algorithm which implicitly finds a set X and a function ψ as described in Reduction Rule 3.3, in a greedy fashion. This algorithm repeatedly finds a vertex x in the current set V_b and an image $z = \psi(x)$ using the heuristic described below, and moves $N[x] \cap V_b$ into the set V_r. The algorithm stops when it can no longer find both a vertex x and an image z for x using the heuristic.

The algorithm picks a vertex x in V_b which is adjacent to the largest number of vertices in V_b. It then selects an image $z = \psi(x)$ using a heuristic intended to block the fewest number of other vertices from being an image of a vertex picked in a later step. Note that if $z = \psi(x)$ for some x then no vertex which is at distance at most two from z can be an image for any other vertex in the set X (Property 3 of Reduction Rule 3.3). With a slight abuse of notation, we define the size of the second neighborhood of vertex v, denoted by $scd\_nbr(v)$ as $\sum_{u \in N(v)} |N(u)|$. Note that $scd\_nbr(v)$ is an upper bound on the number of vertices that are at distance at most two from vertex v. The algorithm chooses a vertex z such that (i) z is *not* a neighbour of vertex x, and (ii) $scd\_nbr(z)$ is the minimum. It then sets $z = \psi(x)$.

The algorithm can be made to run in the stated time bound using heap data structures. ◀

4 Setup for Experiments

4.1 Hardware and Code

We ran our experiments on an Intel(R) Core(TM) $i5-2500$ CPU @3.30GHz machine running Debian GNU/Linux 8 without using parallel processing. We wrote the code to implement the approximation algorithm and the lossy reduction rule in the `C` language. We used GCC (v6.3.0) to compile the code. We used CPLEX (v12.8.0.0) Concert Technology for `C++` to compute nearly-optimal dominating sets, for the sake of comparisons[2]. We wrote the code to generate random graphs of various kinds, in the Python language (v3.5.3) using networkx (v1.11). All the code [1] are available online.

4.2 Test Cases

We tested our hypothesis on randomly generated graphs and on graphs obtained from the SuiteSparse Matrix Collection [5] (See [3]). We chose graphs with edge count at most ten times their number of vertices. That is, our instances were all graphs with ave_deg at most 20. We divided our test cases into the five categories described below, depending on how they were constructed or obtained. Test cases in each of these categories were classified as **small**, **medium**, or **large**, depending on the number of vertices. For small, medium, and large graphs the number of vertices were in the range $[1000 - 10,000]$, $[10,000 - 30,000]$, and $[30,000 - 50,000]$, respectively. Table 1 shows the number of instances in each category.

Random Binomial Graphs or Erdős-Rényi Graphs $G(n, p)$: For a fixed positive integer n and a positive constant $p \leq 1$, this model constructs a graph on n nodes by adding an edge between each unordered pair of vertices with probability exactly p. Note that picking an edge is a Bernoulli's trial: an edge is picked with probability p independently of the

[2] See Section 5 for the details.

	small	medium	large
Erdős-Rényi Graphs	100	100	100
Uniform Random Graphs	100	100	100
Watts-Strogatz Graphs	100	100	100
d-regular Graphs	100	100	100
SparseSuits Matrix Collection	51	84	50

Table 1 First four rows denotes the number of graphs created in each category. The fifth column denotes the number of graphs obtained from SparseSuits Matrix Collection in respective category.

existence of other edges. Hence the expected number of edges is $p\binom{n}{2}$. We chose the value of n uniformly at random from the specified range. Instead of fixing the probability value p, we fixed the expected average degree (avg_deg) of a target graph. We chose the value of avg_deg uniformly at random from the range $[6-20]$. We assigned $p = avg\_deg/n$ and generated the Erdős-Rényi graph.

Uniform Random Graphs $G(n, m)$: For two positive integers n, m, this model starts with an empty graph on n vertices, and inserts m edges in such a way that all possible $\binom{\binom{n}{2}}{m}$ choices are equally likely. Equivalently, it selects two vertices uniformly at random and adds an edge between them if they are different and no edge is already present. It repeats this process until m edges are added to the graph. As in the previous case, instead of fixing m, we fixed the expected average degree (avg_deg) of the graph. Our program selects n uniformly at random from the specified range and avg_deg from $[6-20]$. The program then generates the graph from the model using $(n, n \cdot avg\_deg/2)$ as the parameters.

Watts-Strogatz Graphs $G(n, d, p)$: This model was proposed by Duncan J. Watts and Steven Strogatz [18]. It is used to produce graphs with small-world properties i.e. short average path lengths and high clustering. The model takes three positive constants n, d, p as parameters where n, p are integers and $p \leq 1$. It first creates a ring over the set $\{1, \ldots, n\}$ and constructs a vertex for each element in the ring. Each vertex in the ring is connected with its d nearest neighbors ($d-1$ neighbors if d is odd) i.e. $d/2$ on each side. After this, the model creates some shortcuts by rewiring edges in the following way: for each edge uv in the original graph replace it by a new edge uw with probability p where w is chosen uniformly at random from existing vertices. Our program selects n uniformly at random from the specified range. As the number of expected edges is $ndp/2$, the program selects d from the range $[3-20]$ and p from $\{0.1, 0.2, \ldots, 0.9\}$ uniformly at random.

Random d-Regular Graphs $G(n, d)$: This model takes two positive integers n, d and returns a d-regular graph on n vertices which does not contain a self-loop or parallel edges. Networkx contains an implementation of this model based on [17]. As the total number of edges is $nd/2$, our program requires that nd be even. It selects an even value for n in the specified range and d from $[3-20]$ uniformly at random.

Suite Sparse Matrix Collection: We select matrices from Suite Sparse Matrix Collection and treat them as adjacency matrices. Hence, we can work with only *symmetric* matrices in this data set. The number of rows (and of columns) corresponds to the number of vertices, and the number of non-zero elements is equal to twice the number of edges. As we want the edge density of resultant graph to be at most 10, we select matrices for which the number of non-zero vertices is at most 20 times of the number of rows.

We make a few remarks about the Barabasi-Albert model which is well known for generating random scale-free networks using a preferential attachment mechanism. Although one can create sparse graphs using this model, a dominating set of such graphs tends to be a small fraction of the total number of vertices. The lossy reduction rules fail to obtain any improvement in such cases. In our experience, the lossy reduction rule yields improvement when the degree distribution is binomial or uniform. In the case of the Barabasi-Albert model, the degree distribution follows a power law. As per our expectations, the lossy reduction rule does not yield any improvement. We performed experiments to confirm this conjecture, and only 4% of instances created using this model showed any improvement.

5 Our Experiments and Results

We implemented the approximation algorithm of Jones et al. [12] for Dominating Set on d-degenerate graphs, as our "drop-in" approximation algorithm. Jones et al. show that if an input graph is d-degenerate then their algorithm outputs a dominating set whose size is at most $\mathcal{O}(d^2)$ time that of the optimum. We note that their algorithm can in fact take a graph G and a bipartition (V_b, V_r) of $V(G)$ as input and find a set S which dominates V_b, such that the set returned by the algorithm is $\mathcal{O}(d^2)$ times the size of the smallest subset of G which dominates V_b. For every graph G in the test cases, we executed the code which performs the *two* experiments described below. To ensure that the simple and well-known Reduction Rules 3.1 and 3.2 do not overshadow the improvement by our 2-approximate lossy reduction rule, we applied these reduction rules in both the experiments.

Exp-AA ("Approximation Algorithm"): For an input graph G, the code applies Reduction Rule 3.1 once and Reduction Rule 3.2 exhaustively assuming $V_b = V(G)$ and $V_r = \emptyset$ as the initial partition of $V(G)$. It then finds an approximate solution for the resulting instance using the approximation algorithm of Jones et al. The code then constructs a solution for the original instance using the Solution Lifting Algorithms 3.1 and 3.2, and returns this solution.

Exp-LA ("Lossy Approximation"): For an input graph G, the code applies Reduction Rule 3.1 once and Reduction Rule 3.2 exhaustively assuming $V_b = V(G)$ and $V_r = \emptyset$ as the initial partition of $V(G)$. It then applies the 2-approximate Reduction Rule 3.3 *once* to obtain another instance, say $G(V_b', V_r')$. Let S' be the set returned by the approximation algorithm of Jones et al. when given $G(V_b', V_r')$ as input. Moreover, let S be the solution for $G(V(G), \emptyset)$ constructed by the Solution Lifting Algorithms 3.1, 3.2, and 3.3 starting from S'. The algorithm returns S as a solution for the original instance.

For a graph G, we use $\texttt{AA}(G)$ and $\texttt{LA}(G)$ to denote the cardinalities of the solutions returned by **Exp-AA** and **Exp-LA**, respectively. We use $\texttt{EX}(G)$ to denote the size of the dominating set obtained using CPLEX. We ran the CPLEX program for 30 seconds on each input. In our experience, CPLEX narrows down the gap between upper and lower limits for the optimum solution to less than 0.5% of a lower limit within this time. We ran our experiments on 1385 instances.

We now briefly discuss the running time to compute $\texttt{AA}(G)$ and $\texttt{LA}(G)$. As expected the time required to compute the latter is (slightly) higher, but these two running times are comparable. Consider the 100 large Erdős-Rényi Graphs instances that we created. Our machine took 15.503s and 48.798s to compute $\texttt{AA}(G)$ and $\texttt{LA}(G)$, respectively, for *all* the graphs. We observed differences of similar magnitude in the other sets of instances. Hence we do not present a detailed analysis of the time comparison.

	small	medium	large
Erdős-Rényi Graphs	61.00	77.00	57.00
Uniform Random Graphs	71.00	60.00	63.00
Watts-Strogatz Graphs	79.00	74.00	75.00
d-regular Graphs	60.00	68.00	76.00
SparseSuits Matrix Collection	52.94	58.33	54.00

Table 2 Each entry denotes the percentage of instances for which $\texttt{LA}(G)$ is strictly smaller than $\texttt{AA}(G)$. That is, this is the percentage of instances for which the algorithm which incorporates the 2-approximate reduction rule performs better than the algorithm which does not use this rule.

	small	medium	large
Erdős-Rényi Graphs	15.98	30.52	37.11
Uniform Random Graphs	8.34	15.98	23.49
Watts-Strogatz Graphs	7.97	19.94	32.15
d-regular Graphs	1.79	2.00	2.30
SparseSuits Matrix Collection	12.45	12.92	4.03

Table 3 Each entry denotes the average value $(\texttt{AA}(G) - \texttt{LA}(G))/\texttt{EX}(G)\%$ for the graphs in which $\texttt{AA}(G) > \texttt{LA}(G)$ holds. The larger this value, the more the impact of using the 2-approximate reduction.

Table 2 summarises the percentage of instances across different types of instances in which $\texttt{LA}(G)$ is strictly smaller than $\texttt{AA}(G)$. In other words, in these fraction of total instances, it is better to find a smaller instance using the reduction rules mentioned in Section 3, apply the drop-in approximation algorithm, and then lift the solution to obtain a dominating set for original, than just applying the approximation algorithm on the original graph. For example, for the large Erdős-Rényi Graphs, in 57% of the graphs, $\texttt{LA}(G)$ resulted in better solution than $\texttt{AA}(G)$.

It is natural to ask for some quantitative measures to see the actual improvement. For every instance in which the above process has shown an improvement, we measure the difference between $\texttt{AA}(G)$ and $\texttt{LA}(G)$ and compare it with $\texttt{EX}(G)$. Table 3 shows the average percentage improvement in each category. For example, for the large Erdős-Rényi Graphs, if $\texttt{LA}(G)$ results in better solution than $\texttt{AA}(G)$ then, on average, the improvement in solution is 37.11% to that of exact solution.

Detailed results for all the instances can be found online at [2].

6 Conclusion

In this work we posit that the α-approximate reductions proposed by Lokshtanov et al. [14] as part of their pioneering work on the Lossy Kernelization framework, can in fact be used to derive good heuristics for obtaining better approximation algorithms. To support this thesis we derive a 2-approximate reduction for the Dominating Set problem, and design an algorithm which uses this rule as a heuristic to obtain better approximation ratios for Dominating Set. We implement this algorithm and run it on a wide variety of benchmark instances. We demonstrate that our algorithm obtains smaller dominating sets for a significant fraction of these benchmarks, as compared to a state-of-the-art approximation algorithm for

Dominating Set.

We believe that α-approximate reductions hold great promise as a way of designing good approximation heuristics for other optimization problems as well.

References

1. *Code Repository*. URL: `https://github.com/pptale/DomSet-Lossy`.
2. *Detailed Results*. URL: `https://tinyurl.com/wdganoo`.
3. *The SuiteSparse Matrix Collection*. URL: `https://sparse.tamu.edu/`.
4. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
5. Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), 2011.
6. Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized complexity*. Springer-Verlag, 2013.
7. Pavel Dvořák, Andreas Emil Feldmann, Dušan Knop, Tomáš Masařík, Tomáš Toufar, and Pavel Veselỳ. Parameterized approximation schemes for steiner trees with small number of steiner vertices. In *35th Symposium on Theoretical Aspects of Computer Science*, 2018.
8. Eduard Eiben, Danny Hermelin, and MS Ramanujan. Lossy kernels for hitting subgraphs. In *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
9. Eduard Eiben, Mithilesh Kumar, Amer E Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy kernels for connected dominating set on sparse graphs. *SIAM Journal on Discrete Mathematics*, 33(3):1743–1771, 2019.
10. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
11. Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
12. Mark Jones, Daniel Lokshtanov, MS Ramanujan, Saket Saurabh, and Ondrej Suchy. Parameterized complexity of directed steiner tree on sparse graphs. *SIAM Journal on Discrete Mathematics*, 31(2):1294–1327, 2017.
13. R Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale. Lossy kernels for graph contraction problems. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
14. Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 224–237, 2017.
15. Daniel Lokshtanov, Fahad Panolan, MS Ramanujan, and Saket Saurabh. Lossy kernelization. corr abs/1604.04111 (2016). *arXiv preprint arXiv:1604.04111*, 2016.
16. Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.
17. A. Steger and N. C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8:377–396, 1999.
18. Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393(6684):440, 1998.