Finding Effective k-Vertex Cover Reduction Rule Sequences For Graphs of Various Structural Properties

Mustafa Barodawala [*†]

June 2021

Abstract

We aim to find effective $k-$Vertex Cover Reduction Rule sequences for graphs of various structural properties, by showing that sequences of Reduction Rules for a $k-$Vertex Cover problem instance form equivalence communities, and that, for a given problem instance, the community containing the most effective rule sequence depends on the structural properties of the problem instance. In particular we investigate Erdős-Rényi graphs and planar graphs across their size $|V|$ and respective densities, and Watts-Strogatz graphs across their size $|V|$, mean degree κ, and rewiring probability β. We find a strong relationship between the properties investigated among Erdős-Rényi and planar graphs with the effectiveness of certain rule communities, however similar relationships of Watts-Strogatz graphs could not be found.

Contents

1 Background **4**
- 1.1 Vertex Cover . . . 4
  - 1.1.1 Preliminaries . . . 4
  - 1.1.2 Applications . . . 5
- 1.2 Classical and Parameterized Complexity Theory . . . 7
  - 1.2.1 Classical Complexity . . . 7
  - 1.2.2 Parameterized Complexity . . . 10
- 1.3 Kernelization of the Vertex Cover Problem . . . 11
  - 1.3.1 Kernelization . . . 11
  - 1.3.2 Reduction Rules and Kernelization Algorithms for $k-$Vertex Cover . . . 12
  - 1.3.3 Experimentation on Vertex Cover Kernelizations . . . 17

2 Aims and Objectives **18**

3 Methodology **19**
- 3.1 Graph Datasets . . . 19
- 3.2 Rule Ordering Sequence Search Space . . . 20
- 3.3 Determining The Best Rule Sequences across Structural Parameters . . . 22

[*]ID: 12903718
[†]Email: mustafa.barodawala@student.uts.edu.au

4 Results **24**
4.1 Erdős-Rényi Graph Results 24
4.2 Planar Graph Results 27
4.3 κ-Watts-Strogatz Graph Results 28

5 Discussion **29**
5.1 Erdős-Rényi Discussion 29
5.2 Planar Discussion 29
5.3 Watts-Strogatz Discussion 30

6 Conclusion **31**

Acknowledgements

Thank heck its done,

and by heck i mean dr. luke mathieson, sophie ryan, catriona calantzis, and christopher corby

and also in no particular order other than aesthetic:

alexander cicchitelli
benjamin krajancic
zac papachatgis
maisie cooper
jeriah russo
and adam ozder

and you, the reader [1].

- mazza

[1] Unless you were already mentioned in the above list, there shall be no double-dipping of thank-you's on this paper.

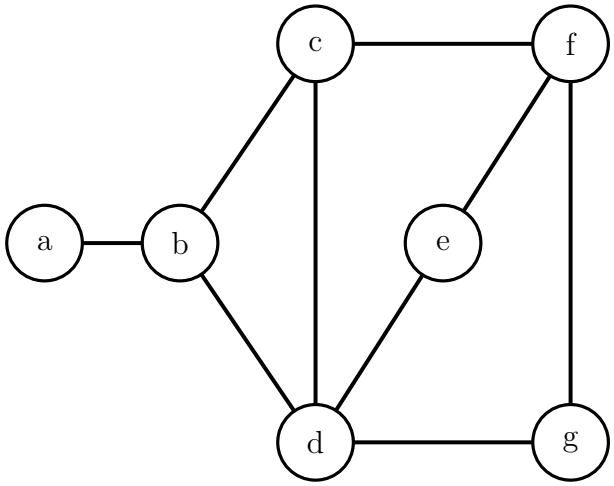


Figure 1: An example of the BAR FIGHT scenario from [7]

1 Background

1.1 Vertex Cover

1.1.1 Preliminaries

A majority of the *interesting* problems in graph theory involve finding some sort of inner structure, or arrangement of the graph that satisfies a particular property. In this paper, we denote a graph as G or $G(V, E)$, where V is the set of vertices of G, and E is the set of edges. We consider the vertex cover of a graph.

Definition 1.1 (Vertex Cover)**.**
Given a graph $G(V, E)$,
A *vertex cover* of G is a set C such that $\forall\ uv \in E,\ u \in C$ or $v \in C$

In other words, a vertex cover of a graph $G(V, E)$ is a subset $C \subseteq V$ such that for every edge $e \in E$, there exists a vertex v in the cover C that is incident on e. This is a classic example of an $\mathcal{NP}$-Complete problem (which we shall cover in Section 1.2).

Consider the graph in Figure 1. We can find a trivial solution where $C = V$, however we can find a non trivial solution, where $C = \{\mathrm{b}, \mathrm{d}, \mathrm{f}\}$. This set satisfies the property that, for each edge in the graph, at least one of its endpoints lies within this cover C.

A natural question we might ask is, what is the smallest cover we are able to achieve?

Problem 1.2 (The MINVC Problem)**.**
Input: A graph $G(V, E)$
Question: What is the size of the minimum vertex cover of G?

The minimum vertex cover can also be thought as the *optimal* vertex cover.

For the example in 1, we have a cover $C = \{\mathrm{b}, \mathrm{d}, \mathrm{f}\}$. One might ask the reasonable question: *'Is $k = 3$ the smallest vertex cover size of G?'*. We can also phrase this question as the negation of *'Does there exist a vertex cover of G with size at most $k = 2$?'*. We can generalize this question as the following problem:

Problem 1.3 (The Vertex Cover Problem)**.**
Input: A graph $G(V, E)$
Parameter: $k \in \mathbb{N}$
Question: Does G have a vertex cover of size at most k?

This is one of Karp's 21 $\mathcal{NP}$-Complete Problems [12].

In this paper, we shall be focusing mainly on this problem. Being a problem in $\mathcal{NP}$-Complete, this is quite a difficult problem.

We have some rules to help determine the whether G has a vertex cover of at most $k = 2$. These rules shall be covered in Section 1.3, however for this particular instance of the problem, we shall use the Degree k Rule (Rule 3 from Section 1.3).

In simple terms, this rule, from [2], works by finding a vertex v with a degree of at least $k + 1$. If v is not in a vertex cover of G, then all of its neighbours must be in the cover, pushing the size of the vertex cover over k. Thus, for the size of the cover to be *at most* k, v must be in the cover.

In the case for $k = 2$ and the graph $G(V, E)$ in Figure 1, we can apply this rule as follows:

1. $b \in V$ has a degree of 3, which is greater than $k = 2$, so by applying this rule, we have converted the problem instance with $(G, k = 2)$ to the instance with $(G', k' = 1)$ where G' is the original graph without the vertex B.
2. In G', since the edge bd was removed in the conversion from G, d now has a degree of size 3. This is greater than $k' = 1$, so we can apply the same rule and remove d from G', call this new graph G''. Now we have the problem instance of $(G'', k'' = 0)$.
3. G'' still has 3 edges remaining: $\{cf, ef, fg\}$, thus it cannot be covered with $k'' = 0$ vertices.
4. Thus we know the Vertex Cover Problem with $(G'', k'' = 0)$ returns a No-Instance, which, by Rule 3, implies the instance with $(G', k' = 1)$ also returns a No-Instance, which similarly implies the problem instance with $(G, k = 2)$ also returns a No-Instance.

Thus we have determined that G can not be covered with $k = 2$ vertices, and that the minimum vertex cover of G has a size of 3, and we are done.

1.1.2 Applications

The vertex cover of a graph can be used to solve many substantial problems. For instance, consider the Bar Fight problem from [7].

You are a bouncer at a bar, and this bar is known for the excessive amounts of brawling among its customers. Say you as the bouncer know all of these customers, and, for each pair of customers, know whether there exists a potential for a brawl. Your task is to only admit customers such that

Figure 2: A conflict between Alan (A) and Bob (B)

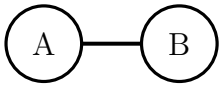


Figure 3: The conflict in Figure 2 represented as a graph

there exists no potential conflict among any of the patrons. You are also tasked with turning away the least possible number of people, ensuring that the profits of the day are maximised.

We can convey the information of this conflict on a graph. Say we have a conflict between two people: Alan (lets call then A) and Bob (lets call them B), like in Figure 2. We can represent this conflict as a graph with vertices A and B with an edge between A and B, like in Figure 3. This is typically how problems such as these are modelled as graphs.

Given such a graph, to avoid any conflict, we must progressively remove vertices until no edges are present, since the edges represent conflicts in our original problem. This means each edge will have at least one of its endpoints in the set of removed vertices. The set of removed vertices thus forms a vertex cover of the graph. If we want to minimise the number of people removed, we would want to find the minimum vertex cover of the graph.

For example, consider once again the graph $G(V, E)$ in Figure 1. We know the minimum vertex cover of the graph can be given by $C = \{\mathrm{b}, \mathrm{d}, \mathrm{f}\}$. Removing these vertices results in a graph, call this graph $G'(V', E')$, with no edges $E' = \emptyset$, since no pair of vertices in $V' = \{\mathrm{a}, \mathrm{c}, \mathrm{e}, \mathrm{g}\}$ shared an edge in the original graph G. Thus, the vertices V' in the altered graph G' represent a set of customers among whom there exist no potential conflict (Figure 4). By finding the minimum vertex cover of G, we have found the *largest* set of vertices from V that do not share an edge. This set is called the Maximum Independent Set of G. We can also find this connection from the theorem by Gallai (compiled in [5]) which states, for an minimum vertex cover C, and a maximum independent set I, $|C| + |I| = |V|$.

Although the Bar Fight Problem seems like a fairly basic application of Vertex Cover the notion of Conflict Resolution actually transcends to many sophisticated problems in the areas

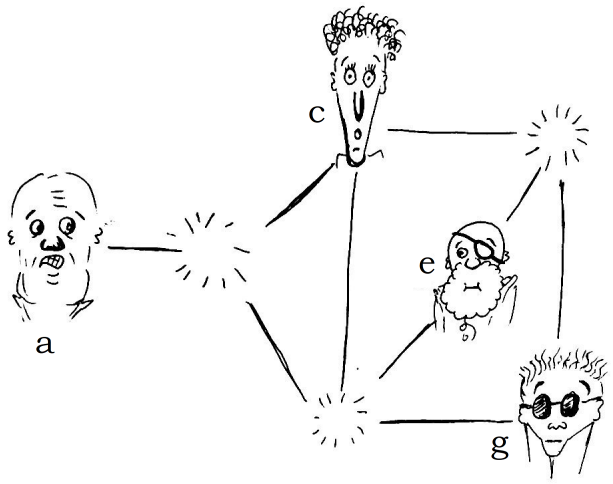


Figure 4: Using the minimum vertex cover from Figure 1 to solve the Bar Fight problem *(This is the first night without a brawl, leaving everyone visibly shocked!).*

of bioinformatics.

Vertex Cover is not only able to resolve conflicts in bar fights, but also evolution trees, as researched by U. Stege in [17]. Stege investigates the area of 'Conflict Resolution *in molecular sequence data*', specifically resolving conflict graphs encountered when computing Multiple Sequence Alignments (MSA), which can be used to build evolutionary trees and predict protein structure.

Abu-Khzam et al [1]. also applied the Vertex Cover Problem to the area of computational biology, specifically, to find *'phylogenetic trees based on protein domain information'*, which was a technique pioneered by Cheetham et al. [3], including U. Stege, the author mentioned above.

Using this technique, Abu-Khzam et al. were able to test out various Vertex Cover kernelization algorithms, (these algorithms, among others shall be covered in the coming sections).

1.2 Classical and Parameterized Complexity Theory

1.2.1 Classical Complexity

An important characteristic of a problems is its difficulty, which can be measured by some sort of cost associated with solving the problem, be it a cost of resources, of money, or even of sweat and tears.

Problems in computer science are no different. These problems typically work on a certain input, and produce a corresponding output. A useful measure of difficulty for such a problem is the *time taken* to solve the problem given an input of a certain measure.

We can think of this as a function $f(n)$, where n is a measure of the input to a problem, and $f(n)$ is the time taken to solve the problem on an input with such a measure. A useful notion to consider

is the *upper bound* of the function f, to represent the *"worst case scenario"*. We can formalize such a notion:

Definition 1.4 (Big $\mathcal{O}$)**.**
Given functions $f, g : \mathbb{Z}^+ \mapsto \mathbb{R}$, we say $f \in \mathcal{O}(g)$ if and only if:

There exists some $m \in \mathbb{R}$ and $k \in \mathbb{Z}^+$ such that
$|f(n)| \leq m|g(n)|,\ \forall n \in \mathbb{Z}^+,\ n > k$.

For an algorithm ϕ that runs in time $f \in \mathcal{O}(g)$, we can say $\phi \in \mathcal{O}(g)$.

For an algorithm ϕ with a running time linear to the size of the input, we can say $\phi \in \mathcal{O}(n)$, or that ϕ has *linear complexity*. An example would be determining if a collection is sorted in C++ [6].
Also, problems that have linear complexity are also in $\mathcal{O}(n^2)$, and that consequently $\mathcal{O}(n) \subset \mathcal{O}(n^2)$. We can also say, for $F \in \mathcal{O}(f)$ and $G \in \mathcal{O}(g)$, $F+G \in \mathcal{O}(f) \cup \mathcal{O}(g)$. Thus for $f(n) = n^2+4n$, $f \in O(n^2)$.

We can establish the following *complexity classes* $\mathcal{P}$ and $\mathcal{NP}$:

Definition 1.5 ($\mathcal{P}$)**.**
A problem Q is in $\mathcal{P}$ if and only if there exists an algorithm ϕ that solves Q on a Deterministic Turing Machine, and $\phi \in \mathcal{O}(n^c)$, for some constant $c \in \mathbb{R}$.

Simply put, $\mathcal{P}$ is the class of algorithms that run in time polynomial of the input. The example above from [6] would be in $\mathcal{P}$.

Definition 1.6 ($\mathcal{NP}$)**.**
A problem Q is in $\mathcal{NP}$ if and only if there exists an algorithm ϕ that solves Q on a Non-Deterministic Turing Machine, and $\phi \in \mathcal{O}(n^{\mathcal{O}(1)})$.

Alternatively, this is the class of problems where a YES-Instance is *verifiable* in $\mathcal{P}$ given also the solution to the problem instance.
We can say $\mathcal{P} \subset \mathcal{NP}$, as solving a problem implicitly verifies its solution. Problems in $\mathcal{NP}$ are considered, for the most part, more difficult than those in $\mathcal{P}$. Whether $\mathcal{P} = \mathcal{NP}$ remains an open question[2].

Proof 1.7 (The VERTEX COVER Problem (Problem 1.3) is in $\mathcal{NP}$)**.**
Given an instance of the problem (G, k), and its solution C as the vertex cover of size $|C| \leq k$, we can verify that C is a solution to the instance (G, k) in time polynomial of the size of the input $n = |V| + |E|$, by the following:

1. Verify C is of size k. This task is linear on $|C|$, and since $|C| \leq k < n$, this task is in $\mathcal{O}(n)$
2. Verify each edge is incident on a vertex in C. This task checks whether all of the $|E|$ edges are incident with any of the *at most* k vertices of C, thus, since $|E| \cdot k \leq n^2$, this task is in $\mathcal{O}(n^2)$

[2]And a large source of contention (`https://www.win.tue.nl/~gwoegi/P-versus-NP.htm`)

In total, this process takes the order of $n^2 + n$ steps, thus we can say the verification of (G, k) with solution C is in $\mathcal{O}(n^2)$. Thus, The VERTEX COVER Problem is *verifiable* in $\mathcal{P}$, and thus it is in $\mathcal{NP}$ □.

VERTEX COVER is a fairly difficult problem to solve, as we can see by its complexity in $\mathcal{NP}$. There exists, however, a richer class of problems in which VERTEX COVER resides, however first we must confront the concept of *reduction*.

When facing difficult problems, it often helps to translate the current problem into another, hopefully simpler problem-space.

For example: A sports captain is tasked with deciding which team members shall be playing at a certain match. Say there are 10 available spots left, and 11 team members to choose from.
Rather than the captain deciding the top 10 out of 11 team members, it is much easier for the captain to decide which team member shall *not* be playing in the match. Thus converting the task of choosing the top 10 team members to the task of choosing the single worst team member.

When converting these problem instances, one must be wary of the complexity of converting the input. When these conversions having polynomial complexity, we can define the following:

Definition 1.8 (Karp Reduction[12])**.**
Consider the languages L and M.
We say $L \leq_p^m M$ (L is Karp Reducible to M) if and only if:
There exists some $f \in \mathcal{P}$ such that $f(x) \in M \iff x \in L$
Then we say f is a Karp Reduction (or a Polynomial Reduction) of L to M.
Additionally, if $L \leq_p^m M$ and $M \leq_p^m L$, then L and M are said to be *equivalent*.

In broader terms, $L \leq_p^m M$ when we can convert the problem in L to a problem in M, in polynomial time. The notation of $L \leq_p^m M$ is used to indicate this is a *polynomial-time* ($\leq_p$) *many-one* ($\leq^m$) reduction, which is equivalent to Karp Reduction.

With this, we can define $\mathcal{NP}$-Hard as such:

Definition 1.9 ($\mathcal{NP}$-Hard)**.** $f \in \mathcal{NP}\text{-Hard} \iff \forall\, g \in \mathcal{NP}, g \leq_p^m f$

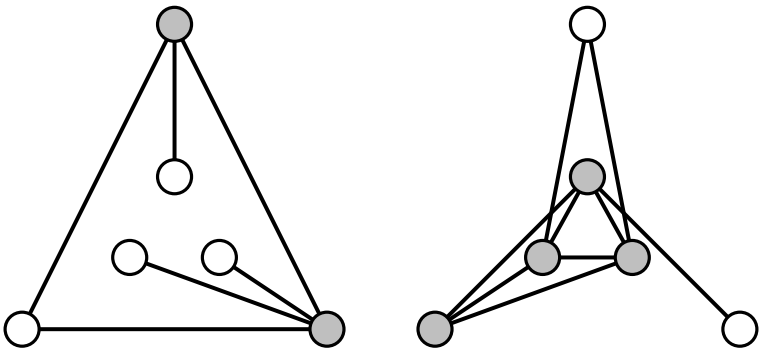
A problem may be in $\mathcal{NP}$-Hard, and not in $\mathcal{NP}$, such as the Maximum INDEPENDENT SET Problem mentioned in Section 1.1.2. As of 2017, this problem can be solved in $\mathcal{O}(1.1996^n n^{\mathcal{O}(1)})$ [18]. These problems are *pretty hard*[3], at least compared to VERTEX COVER.
The VERTEX COVER Problem *does* belong to $\mathcal{NP}$-Complete, which is defined as such:

Definition 1.10 ($\mathcal{NP}$-Complete)**.** f is in $\mathcal{NP}$-Complete if and only if:

1. $f \in \mathcal{NP}$
2. $f \in \mathcal{NP}$-Hard

Karp's 21 Problems are perhaps the cornerstone of $\mathcal{NP}$-Complete [12]. He also provides the proof for VERTEX COVER $\leq_p^m$ CLIQUE, as is detailed below:

[3] As the name dictates



(a) Vertex Cover on $(G, k = 2)$ (b) Clique on $(\overline{G}, l = 4)$

Figure 5: An instance of the Vertex Cover Problem, and its reduction to the Clique Problem

Proof 1.11 (Vertex Cover $\leq_p^m$ Clique)**.**
On an instance of The Vertex Cover Problem (G, k), on input of size $n = |V| + |E|$:

1. Compute the complement $\overline{G}$ of G, which takes $|V|^2$ steps, thus, since $|V| \leq n$, this task is in $\mathcal{O}(n^2)$
2. Compute $l = |V| - k$, which can take at most on the order of $|V|$ steps, this, since $|V| \leq n$, this task is in $\mathcal{O}(n)$.

The reduction is Clique on $(\overline{G}, l) \iff$ Vertex Cover on (G, k). Computing $(\overline{G}, l)$ takes on the order of $n^2 + n$ steps, thus this reduction is in $\mathcal{P}$ $\square$.

Thus Vertex Cover $\leq_p^m$ Clique, and we are done. We can similarly prove Clique $\leq_p^m$ Vertex Cover, showing Clique and Vertex Cover are *equivalent*, at least within Classical Complexity.

1.2.2 Parameterized Complexity

Downey and Fellows broke new ground in [10] by establishing the area of Parameterized Complexity Theory. In essence, it seeks to observe the complexity of a problem given some sort of parameter. We have already seen such a problem in Problem 1.3, where the parameter k represented the largest acceptable vertex cover size. In this section we shall explore the theory behind the Parameterized Complexity class of *fixed-parameter tractable* (FPT). We define FPT from [9].

Definition 1.12 (Fixed Parameter Tractable)**.**
A problem with input of size n and parameter k is said to be *fixed-parameter tractable* (FPT) if there exists an algorithm ϕ and computable function f such that $\phi \in \mathcal{O}(f(k)\ n^c)$.
This can also be denoted as $\mathcal{O}^*(f(k))$.

The main motivation behind Parameterized Complexity comes from finding and exploring alternate ways to parameterize a problem, and exploring it's new complexity. As quoted from [9]:

> The future of algorithmics is multivariate.

Classical complexity only considers the size of the input, n, or more specifically, the size of the description, or representation, of the input. For problems with numerical input, such as INTEGER FACTORIZATION (which is in $\mathcal{NP}$), or determining whether a number is a prime ($\mathcal{O}(\sqrt{n})$), this type of measure make sense, however this notion of n solely representing the input starts to lose meaning once we deal with problems with more abstract inputs, such as graphs. With problems such as The VERTEX COVER problem (Problem 1.3), not only are we able to give a graph instance on which to find the vertex cover, but we are also able to specify a parameter k, which in this case represents the maximum size of the cover.

Indeed, we can parameterize any problem by introducing some parameter that specifies some aspect of the solution, or even an aggregate of information or constraint concerning the input and the solution to the problem [9].

We can show that The VERTEX COVER Problem is FPT using a proof adapted from Cygan et. al [7] [4]. Consider the following observations:

Observation 1.13. For a VERTEX COVER problem instance $(G(V, E), k = 0)$:
This is a YES-Instance $\iff E = \emptyset$.

Observation 1.14. Given a graph $G(V, E)$ and a vertex cover C of G of size $|C| = k$:
For each $uv \in E$ either $v \in C$ or $u \in C$.

A common method to solve FPT problems is by using *branching* algorithms. We can take advantage of these two observations and construct the following recursive branching algorithm $\phi(< G, k >)$ to solve The VERTEX COVER Problem:

1. If $|E| = \emptyset$ then, by Observation 1.13, return YES if $k = 0$
2. Alternatively if $k = 0$ then, again by Observation 1.13, return YES if $|E| = \emptyset$
3. Else, pick an edge $uv \in E$, and, by Observation 1.14 branch to return:
   $\phi(G - u, k - 1) \wedge \phi(G - v, k - 1)$,
   where $G - q$ for vertex $q \in V$ is the graph G with q removed, along with its incident edges.

This algorithm can only add up to k vertices, since at each step, when this algorithm branches, it adds one of the potential vertices of the vertex cover. So the maximum depth of the recursion tree is bounded by k. This tree is a binary tree, branching two potential solutions at each step, so the number of steps taken to traverse this recursion is on the order of 2^k.

At each step of the traversal, the emptiness of E is checked, and an edge $uv \in E$ is selected, which takes on the order of n^c steps, for some $c \in \mathbb{R}$. Thus $\phi \in \mathcal{O}(2^k\ n^c)$, equivalently $\phi \in \mathcal{O}^*(2^k)$. [5]

Thus VERTEX COVER is FPT, and we are done $\square$.

1.3 Kernelization of the Vertex Cover Problem

1.3.1 Kernelization

We have looked at Karp Reduction in Section 1.2, which dealt with converting one problem instance into another. We also know that there exists a Karp Reduction that reduces any $\mathcal{NP}$-Complete problem to another [12].

[4] [7] also provide a proof for how CLIQUE is not FPT

[5] The formulation of a slightly different proof from [7] find a $\phi \in \mathcal{O}^*(\varphi^k)$, where φ is the golden ratio.

A similar concept exists for problems that are in FPT, namely, algorithms that convert an instance of a parameterized problem into another, such that the size of the converted instance is bounded by the parameter. This concept is called *kernelization*.

Kernelization is defined as follows, from Downey and Fellows [8]:

Definition 1.15 (Kernelization)**.**
For a parameterized problem with instance I and parameter k, a *kernelization* of the problem is an algorithm ϕ that converts (I, k) to (I', k') such that:

1. $k' \leq k$
2. $|I'| \leq g(k)$, for some function g only of k
3. (I, k) is a YES-Instance $\iff$ (I', k') is a YES-Instance.
4. ϕ is computable in time polynomial in $|I| + |k|$, or more succinctly $\phi \in \mathcal{O}((|I| + |k|)^{\mathcal{O}(1)})$

Downey and Fellows go further to show that:

Theorem 1.16 (FPT and Kernel)**.**
A problem on (I, k) is in FPT $\iff$ the problem is kernelizable, i.e. there exists a kernelization ϕ on (I, k).

This proof is given in [9].

Since $k-$VERTEX COVER is in FPT, it is kernelizable. This means there exist Kernelization Algorithms for this problem.

1.3.2 Reduction Rules and Kernelization Algorithms for $k-$Vertex Cover

According to Fellows et. al in [11]:

> The VERTEX COVER problem is often referred to as the *drosophila* of parameterized complexity.

The $k-$VERTEX COVER Problem has many *Reduction Rules*, reductions of the problem instance (G, k) to another instance (G', k') in polynomial time, similar to a kernelization. That being said, Reduction Rules do not always provide a bound on the problem instance the way a kernelization does, however they can be used as building blocks to form kernelization algorithms.

These rules are especially useful and simple, allowing us to take advantage of the tractable nature of The $k-$VERTEX COVER Problem, which we area able to observe through the application of individual rules, and rules used in tandem with one another.
There are ten Reduction Rules of particular interest that shall be studied in this paper, and shall be outlined in the remainder of this section. Collectively these shall be referred to as The Ten Reduction Rules (or *Ten RRs*). As a shorthand, we shall be referring to specific rules either by their rule number (eg. Rule 10) or by their abbreviated name (**LP**). The abbreviate names shall be used when describing rule sequences.

We define the following two rules from Buss and Goldsmith [2].

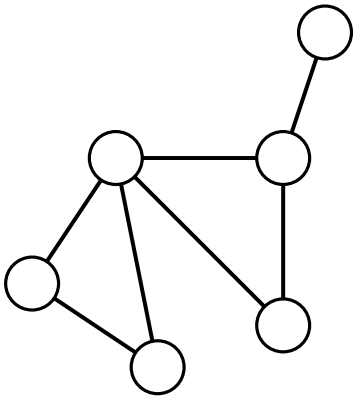


Figure 6: On a VERTEX COVER problem instance with this graph and $k = 3$, we can apply the Pendant Rule (Rule 2) k times. (Try it!)

Rule 1 (*Isolated Vertex Rule* or *Degree 0* - **D0**)**.**
If there exists some $v \in V$ such that $\deg(v) = 0$ then:
(G, k) is a YES-Instance $\iff$ $(G - v, k)$ is a YES-Instance
We are also able to determine that v is not in the optimal VERTEX COVER of G.

This comes from the observation that vertices that are not adjacent to any other vertex are not in the optimal vertex cover [6].

Rule 2 (*Pendant Rule* or *Degree 1* - **D1**)**.**
If there exists some $u \in V$ such that $N(u) = \{v\}$, then:
$(G, k) \iff (G - N[u], k - 1)$
We are also able to determine that $N(u) = \{v\}$ is in the optimal VERTEX COVER of G, and u is not.

Essentially, if G contains a vertex v with a single neighbour u, then u is in the MINVC and v is not. The graph in Figure 6 can be completely kernelized using this rule.
We used the next rule to determine the MINVC size of the graph in Figure 1 from Section 1.1.

Rule 3 (k Degree Rule - **DK**)**.**
Given an instance of the Parameterized VERTEX COVER Problem with $(G(V, E), k)$, if there exists some $v \in V$ such that $\deg(v) \geq k$ then:

The instance (G, k) is a YES-Instance $\iff$ $(G', k - 1)$ is a YES-Instance, where G' is given by the vertices $V \setminus \{v\}$ and the edges $E \setminus \{e \in E \mid v \in e\}$.
We are also able to determine that, should a VERTEX COVER of size k exist, then v is in that cover.

[6] As a side note, we can also make an observation for the related CLIQUE problem, as mentioned in the previous section. The complementary version of a disconnected vertex would be a vertex connected to all other vertices. Surely this vertex *would* be included in a Maximum CLIQUE, just as an isolated vertex *would not* be in a MINVC.

Theorem 1.17 (From Buss and Goldsmith [2])**.** *Using Rule 1 and Rule 3 exhaustively on an instance* $(G(V, E), k)$*:*
The size of the resulting kernel $(G'(V', E'), k')$ *is bounded by* $|V'| \leq k^2 + k$ *and* $|E'| \leq k^2$*.*

Applying a Kernelization Rule *exhaustively* means until it can't be applied any further, or until it isn't required to be applied. For instance, after each application of Rule 1, the parameter of the kernel is decremented. Thus in this case it can not be applied more than k times. The graph in Figure 6 with $k = 3$ is one such instance.

The following proof is adapted from adapted from [11, 7].

Proof 1.18 (of Theorem 1.17)**.** After applying both Rules (1, 3) exhaustively, we have the resulting kernel instance $(G'(V', E'), k')$.
Through Rule 3, G' has a maximum degree of k. Since it is not possible to cover more than k^2 edges with k vertices of degree at most k, $|E'| > k^2 \implies (G, k)$ is a No-Instance.
Similarly, $|V'| > k^2 + k \implies (G, k)$ is a No-Instance.

Thus from [2] we can say that the Vertex Cover Problem admits a kernel with at most k^2 edges and at most $k^2 + k$ vertices.

By considering a pendant vertex and its neighbour as a clique of size 2, the concept from Rule 1 can be extended to cliques of size 3 with the following rule:

Rule 4 (Degree 2 Adjacent - **2A**)**.**
If there exists some $u \in V$ such that $N(u) = \{v, w\}$ and $(v, w) \in E$, then:
$(G, k) \iff (G - N[u], k - 2)$
We are also able to determine that $N(u)$ is in the optimal Vertex Cover of G, and u is not.

Which can be further be generalized to cliques of any size:

Rule 5 (Complete Neighbourhood - **CN**)**.**
This is a generalization of Rule 2 and Rule 4.

If there exists some $u \in V$ such that $N(u)$ is a complete neighbourhood, then:
$(G, k) \iff (G - N[u], k - |N(u)|)$

Notice how Rules 1, 2, and 4 are all covered by the Complete Neighborhood Rule (5), which means if Rules 1, 2, or 4 are applicable to a particular problem instance, then so is Rule 5.
For cases where there exists non edges among the neighbourhood of a vertex, the following rules are of interest.

Rule 6 (Degree 2 Non-Adjacent - **2N**)**.**
If there exists some $u \in V$ such that $N(u) = \{v, w\}$ and $(v, w) \notin E$, then:
Fold $N[u]$ in G into u' in G', then: $(G, k) \iff (G' - u', k - 1)$
We are also able to determine that
If u' is in the optimal Vertex Cover of G' then $N(u)$ is in the optimal Vertex Cover of G, otherwise u is in the optimal Vertex Cover of G. An example of this rule is show in Figure 7.

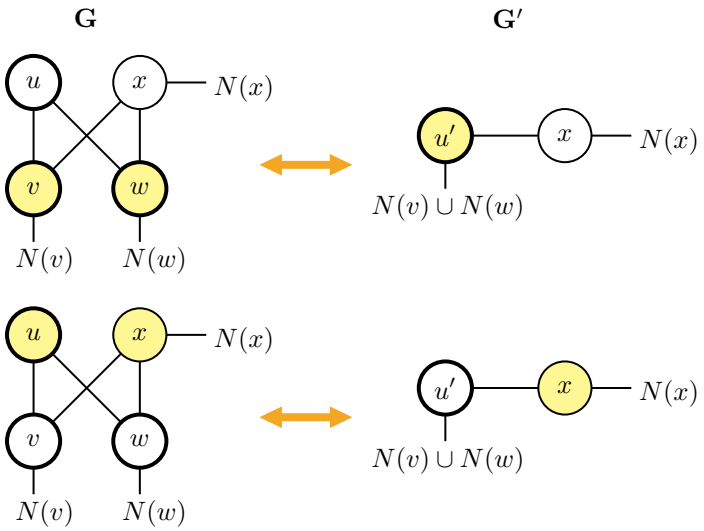


Figure 7: Rule 6 in action. The left shows the the pre-folded graph G, the right shows the graph G', folding $\{u, v, w\} \in V$ to u'. The top shows the case where u' is in the cover of G', the bottom showing if u' is not in it's cover.

Rule 7 (Struction - **St**)**.** This is a generalization of Rule 6.

If there exists some $u \in V$ such that the induced subgraph $G[N(u)]$ contains at most $|N(u)| - 1$ non edges, then:

For each such non-edge (x, y) in the neighbourhood of u, fold x, y into $v_{x,y}$, such that:

$N(v_{x,y}) = N(x) \cup N(y) \setminus \{u\}$. Let the graph G' be the graph containing all the folded vertices from $N(u)$. Then:

$(G, k) \iff (G', k - 1)$

Rule 8 (General Fold - **GF**)**.** This is a generalization of Rule 7 and Rule 5.

If there exists some independent set $I \subset V$ such that $|N(I)| = |I| + 1$, and $|N(S)| > |S|$ for every subset $\emptyset \subset S \subseteq I$, then:

i) If $N(I)$ is not an independent set of G, then $(G, k) \iff (G - N[I], k - |N(I)|)$.

ii) If $N(I)$ is an independent set of G, then construct G' from G by folding $N[I]$ to the vertex $u \in V'$, such that $N(u) = N(N(I)) - I$, then $(G, k) \iff (G', k - |I|)$.

Rule 9 (Crown Reduction - **Cr**)**.**

This rule is introduced in [1]

Given two disjoint set of vertices $H, I \subset V$, where:

1. H is the neighbourhood of I ($H = N(I)$) which is to say
   $H = \{v \in V \mid uv \in E \text{ for } u \in I \text{ and } v \notin I\}$

2. I is a non-empty and independent set, that is:

   $I \neq \emptyset$ and

   $\nexists\ uv \in E$ such that $u \in I$ and $v \in I$.

3. The set of edges connecting H and I contain a *matching* such that every element in H is matched.

   This means: from the set of edges $E' = \{uv \in E \mid u \in I,\ v \in H\}$, there is a *matching* $M \subseteq E'$ (a set of non-adjacent edges) where $\forall\ v \in H,\ \exists\ uv \in M$ (and implicitly this means $u \in I$).

Then there exists an optimal vertex cover containing all of H and none of I, as proved by [1].
Then $(G, k) \iff (G', k')$, where the kernelized graph G' is G with the vertices in H and I removed, along with any edges containing an endpoint in H or I, and $k' = k - |H|$.

Li and Zhu [14] have recently found a $2k$ kernelization using only *Crown Reduction*.

Since The Vertex Cover Problem is $\mathcal{NP}$-Complete, it can be reduced to any other $\mathcal{NP}$-Complete problem in polynomial time.

We can therefore formulate a reduction from Vertex Cover on an instance $(G(V, E), k)$ to a decision on an Integer Linear Programming Problem, or ILP.

Assign a variable $X_u \in \{0, 1\}$ for each $u \in V$ such that:

1. $\sum_{v \in V} X_v$ is *minimized*
2. $X_u + X_v$ is satisfied for each $\in E$

Then Vertex Cover on $(G, k) \iff \sum_{v \in V} X_v \leq k$.

Unfortunately ILP is also $\mathcal{NP}$-Complete, however we can relax the variables to $X_u \in [0, 1]$ to generate the following Linear Programming (LP) Kernelization Rule:

Rule 10 (Linear Programming - **LP** [1, 11, 7])**.** We formulate the following Linear Programming problem:
Assign a variable $X_u \in [0, 1]$ for each $u \in V$ such that:

1. $\sum_{v \in V} X_v$ is *minimized*
2. $X_u + X_v \geq 1, \forall (u, v) \in E$

The reduction then becomes:
$(G, k) \iff (G - V', k - |C|)$, where $V' = \{u \in V | X_u = 1 \vee X_u = 0\}$, and $S = \{u \in V | X_u = 1\}$

Cygan et. al [7] provide a proof showing Vertex Cover admits a kernel with at most $2k$ vertices using only LP Kernelization.

1.3.3 Experimentation on Vertex Cover Kernelizations

Abu-Khzam et. al [1] performs a comparison of various kernelization techniques for Vertex Cover evaluating how each technique performs in the context of computational biology, where solving the Maximum Clique Problem is quite relevant. Since Vertex Cover is easily reducible to Clique, Abu-Khzam et. at are able to use Vertex Cover kernelization techniques to solve Maximum Clique Problems in this area.

Algorithm	run time	kernel size (n')	parameter size (k')
High Degree with Preprocessing	0.58	181	43
Linear Programming	1.15	0	0
Network Flow	1.25	36	18
Crown Reduction	0.23	328	98

Table 1: *"Table 1"* from [1], the performance of each of the methods.

Algorithm	run time	kernel size (n')	parameter size (k')
Linear Programming	0.05	0	0
Network Flow	0.02	0	0
Crown Reduction	0.03	69	23

Table 2: *"Table 2"* from [1], Preprocessing (including the high-degree algorithm, Rule 3) was performed before each of the other 3 methods.

Abu-Khzam et. al look at the following kernelization algorithms: Linear Programming kernelization (Rule 10), Crown Reduction (Rule 9), Network Flow kernelization, a method based off Linear Programming kernelization, and the Degree k Rule (Rule 3), which they refer to as *High Degree*. They also look at, what they refer to, various *Preprocessing* Rules, of which include the Isolated Vertex Rule (Rule 1), and the Pendant Rule (Rule 2), among various others.

They explore how using the kernelization algorithms both by on their own, as well as in conjunction with the Preprocessing Rules, affects the bounds and the run time of the various algorithms. Table 1 shows the results from running each of the techniques on their own, whereas Table 2 shows the results of running Linear Programming Kernelization, Network Flow, and Crown Reduction *after* running the Preprocessing techniques with the Degree k Rule.

They conclude the best approach is to run the Preprocessing Rules with the Degree k Rule, and then Crown Reduction, an exceptionally fast technique. If the remaining kernel is sparse, Linear Programming Kernleization or Network Flow should be applied, otherwise a branching technique (effectively a *brute-force* technique) should be applied to solve the remaining kernel.

2 Aims and Objectives

Our thesis is as follows:
The algorithmically optimal sequencing of The Ten Reduction Rules depends on the structural properties of the problem instance.

We define a Reduction Rule sequence to be an ordered sequence of rules from the 10 RRs. The order dictates the order in which they are applied to a problem instance.
For example, consider the rule sequence: $[CN, St, LP]$. If we say this sequence was applied to an initial problem instance (G, k), we mean:

1. First CN was applied to (G, k) until exhaustion (i.e. until it can not be applied any further) to obtain kernel (G', k'),
2. Then St was applied to (G', k') until exhaustion to obtain kernel (G'', k''),
3. And finally LP was applied to (G'', k'') until exhaustion to obtain kernel (G''', k'''), which is the resultant kernel from the sequence $[CN, St, LP]$ and initial problem instance (G, k).

This system of characterising the application of multiple rules is limited in that we do not repeat the entire sequence after applying the final rule. This limitation is useful limitation comparing similar rule sequences, in that we know a rule sequence $R_1 = [r_1, r_2, ..., r_n]$ is no better in terms of kernel size than a rule sequence $R_2 = [r_1, r_2, ..., r_n, r_{n+1}]$, which is the rule sequence R_1 with rule r_{n+1} added on the end.
Without this limitation, R_1 and R_2 would not be comparable in this way, as by inserting rule r_{n+1} after r_n, we are also inserting rule r_{n+1} *before* rule r_1, which has a change to significantly alter the kernel provided to rule r_1 in sequence R_2, compared to that provided to rule r_1 in sequence R_1.

This limitation also means we are able to progressively search through the space rule sequences for a $k-$Vertex Cover problem instance to find which is most effective. This method is described in Section 3.2.
In our study we shall focus on three graph classes: Erdős-Rényi graphs, planar graphs, and Watts-Strogatz graphs.

An Erdős-Rényi graph $G(V, E)$ considers two parameters (n, p), where $|V| = n$, and the edge $(u, v) \in E$ with a probability of p, where $u, v \in V, u \neq v$.
A planar graph is a graph that permits an embedding on Euclidean 2D space, where no edges cross. Equivalently a Planar graph is such that does not contain a K_5 or $K_{3,3}$ minor, as shown by Kuratowski in 1930 [13].
Watts-Strogatz graphs modelled to have low density (most pairs of vertices are not neighbours) but high clustering (there is a high probability two vertices are neighbours if they share a neighbour). Watts-Strogatz graphs are modelled by (n, κ, β), where for a graph is formed by a ring lattice of n vertices, where each vertex is shares an edge with its κ nearest vertices on the lattice. This way there are $\frac{nk}{2}$ edges. Each edge is then rewired with a probability of β, that is, for an edge (i, j), where j is to the right of i on the lattice, replace it with the new edge (i, l), where l is chosen uniformly at random while avoiding self loops and multi-edges.

We consider Erdős-Rényi graphs and Planar graphs along two key structural parameters: graph size ($|V|$), and graph density, for Erdős-Rényi graphs this is $\frac{2|E|}{|V|(|V|-1)}$, and for planar graphs this is $\frac{|E|}{3|V|-6}$.
For Watts-Strogatz graphs, we shall consider their size ($|V|$), their mean degree κ, and their rewiring probability β.

Our central aim is to investigate the relationship between these parameters, and which types of rule sequences are most effective.

3 Methodology

3.1 Graph Datasets

We consider three classes of graphs: Erdős-Rényi, planar, and Watts-Strogatz graphs.
We generate 1200 Erdős-Rényi and 700 planar graphs of sizes $|V| \in [1, 100]$, and respective densities $d \in (0, 1]$, uniformly at random. We also generate 500 Watts-Strogatz graphs of sizes $|V| \in [1, 100]$, discretely across $\kappa \in \{2, 4, 6\}$ and $\beta \in \{0.05, 0.125, 0.25, 0.5, 0.75, 0.875\}$, uniformly at random. We chose the latter parameters for κ and β as it more closely follows the idea of local clustering, where with $\kappa = 2$, we have effectively no clustering, $\kappa = 4$ we get slightly more, and $\kappa = 6$, we reach quite a tight clustering. We also made the decision to keep κ independent from $|V|$, since we wanted to explore how the small-world nature of Watts-Strogatz graphs played a role with deciding rule ordering. By keeping κ independent from $|V|$, we ensure that average path length is directly controlled by β.

For graph generation, we use the Python graph theory library NetworkX[7] [15]. This library already contains a method for generating Erdős-Rényi graphs : `nx.erdos_renyi_graph(n,p)`, and for generating Watts-Strogatz graphs: `nx.watts_strogatz_graph(n,k,p)`.

NetworkX does not have a method to generate uniformly random planar graphs, so we implemented a method to generate a planar graph using Delaunay Triangulation by the following in Python, using the mathematics library SciPy [16]:

For a planar graph of size $|V| = n$, and density $d \in [0, 1]$:

1. Create an empty graph $G(V, E)$ where $|E| = 0$ and $V = \{v_1, v_2, ..., v_n\}$
2. Generate n uniformly random points $P = \{p_1, p_2, ..., p_n\}$ on a 2D Euclidean plane.
3. Using SciPy's `scipy.spatial.Delaunay` method, obtain a set of 3-tuples corresponding to each triangle in the Delaunay triangulation of P.
4. For each 3-tuple of points obtained from the triangulation (p_a, p_b, p_c), add the edges (v_a, v_b), (v_b, v_c), and (v_a, v_c) if they do not already exist in E.
5. Finally, for each edge $e \in E$, remove e from the E with probability $(1 - d)$.

[7] Using Python version 3.8.3, NetworkX version 2.5

We are then able to provide the reduction rules these graphs, with its $k-$Vertex Cover parameter k, which we obtain via CPLEX, since all of these graphs are relatively small (finding a minimum vertex cover).

3.2 Rule Ordering Sequence Search Space

To find an effective rule ordering for a given graph, we consider a structure we call a **Rule Kernel**, which we define as (G, k, R, t), for graph G, integer parameter k, finite ordered sequence of Reduction Rules R, and time t, where (G, k) is the $k-$Vertex Cover problem instance kernel after applying the sequence of Reduction Rules R on some initial problem instance (G_0, k_0), and t is the time taken in milliseconds to reach the current kernel (G, k) from (G_0, k_0) after applying the Rules of R (t can be any unit of time as long as there is consistency, however in this paper we shall consider milliseconds for t).

This way we are able to consider how a problem instance is kernelized at every sequential application of the rules from R, and are able to know how fast a kernel was found. We are further able to calculate which Rule Kernel provides the most effective kernelization of the initial problem instance, first by preferring the Rule Kernel with fewer vertices in it's kernel graph, then by fewer edges, then by the lower time taken t.

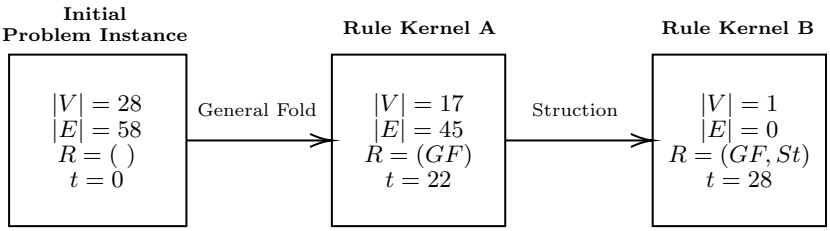


Figure 8: Given the left Initial Problem Instance, applying General Fold gives Rule Kernel A, and then applying Struction gives Rule Kernel B, which is close to being fully kernelized.

For example, considering the scenario from Figure 8:
Beginning with an *Initial Problem Instance*, notably with $R = (\ )$ (an empty sequence of Reduction Rules, having applied nothing) and $t = 0$ (applying no rules takes no time), we apply General Fold to reach *Rule Kernel A*, the first Rule Kernel, with 9 fewer vertices and 13 fewer edges, having taken 22 milliseconds since the *Initial Problem Instance*.

Further applying Struction on *Rule Kernel A*, we see the remaining kernel is a graph of only a single vertex, no edges, having taken 28 milliseconds in total.

By this, since *Rule Kernel B* has a smaller kernel size ($|V| = 1$), than *Rule Kernel A* ($|V| = 17$), we can say *Rule Kernel A* has the more effective kernelization.

It is important to note that, the only way one can reach *Rule Kernel B* is by passing through *Rule Kernel A*, as it requires having applied General Fold first. As such, we can think of *Rule Kernel B* as the child of *Rule Kernel A*, and thus *Rule Kernel A* the child of the *Initial Problem Instance*.

Based on this type of hierarchy, we are able to consider the search-space of rule orderings by considering the Rule Kernel tree of a particular graph, where, starting at the *Initial Problem*

Instance as the root Rule Kernel, construct all of its children Rule Kernels by applying each of the Reduction Rules once to the root. This would generate all the kernels of the *Initial Problem Instance* after applying any of the Reduction Rules exactly once. Applying the same operation on each of the children would result in the kernels after applying exactly two rules. An example of such a tree for a particular *Initial Problem Instance* is given in Figure 9.

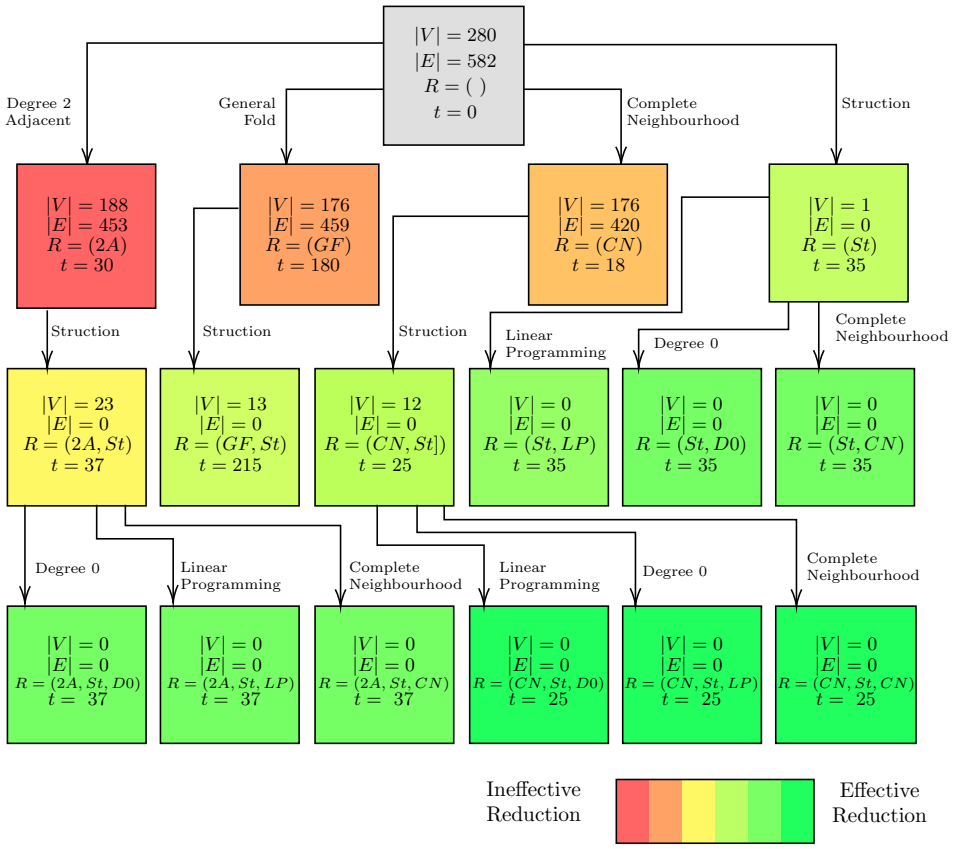


Figure 9: An example of a rule order search tree of a problem instance.

In the above example we see that the Reduction Rule sequence (St, CN) reaches full kernelization with only two rules used, however the Reduction Rule sequence (CN, St, CN) although being realised in the *next* depth of the search tree, is deemed a more effective kernelization because it reaches full kernelization faster (has a lower t).

The Rule Kernels at a depth of n of this type of search tree (where the root, being the *Initial Problem Instance*, has a depth of 0) can be characterised as all the Rule Kernels in the search tree where $|R| = n$, and can be collectively considered the n^{th} generation, each being a child of a Rule Kernel from the $(n-1)^{th}$ generation.

This way, for each $k-$Vertex Cover problem instance, we are able to rank each Rule Kernel across all depths of the tree by effectiveness, and consider the best Reduction Rule sequences for

each graph to find an optimal rule ordering for a graph, given its structural properties. To account for equivalently effective Rule Kernels, we consider a standard competition "1224" ranking system, where the rank of a candidate is $1+$(# Candidates ranked higher). This way we are able to threshold the candidate Rule Kernels to be above a certain rank, and are able to retain information about equivalent

By bounding the breadth of this search tree to b, we are only considering the top b most effective Rule Kernels for a given generation. Similarly, by bounding the depth of the search tree to d, we are considering Rule Kernels across no more than d generations since the *Initial Problem Instance*, and thus are only considering up to d rules for a Reduction Rule sequence.

By allowing a large bound on the breadth, we allow for scenarios where a Rule Kernel from an early generation may be ineffective when compared to others in its generation, but its descendants contain a top Rule Kernel. If the bound on the breadth was limited enough, the ineffective ancestor from the earlier generation

We consider this method for traversing the search space of ordered Reduction Rule sequences for a given graph, checking no more than $d \cdot b$ rule orderings (at most d generations, each with at most b Rule Kernels) for depth-bound d and breadth bound b, which would check Reduction Rule sequences of up to d rules. This is opposed to iterating thorough all ordered Reduction Rule sequences of up to d rules, which would require exactly $|\text{Rules}| \cdot (|\text{Rules}| - 1)^{d-1}$ checks, since we do not consider Reduction Rule sequences where the same rule is applied consecutively, since applying each rule means we apply until exhaustion.

For our study, across the 10 Rules we consider, we limit $d < 5$, and the breadth of the search tree to $b < 7$, we check at most 35 Reduction Rule sequences per problem instance, as opposed to checking $10 \cdot 9^4 = 65,610$ Reduction Rule sequences by iterating through every sequence. Of these candidate Rule Kernels, we consider Rule Kernels of ranks 1 to 4. This means that, in the case that there are over five joint-first-placed Rule Kernels, followed by a sixth-placed Rule Kernel, we only consider the Rule Kernels in joint-first-place.

For the example in Figure 9, we find in joint-first-place the three Rule Kernels with sequences: (CN, St, $D0$), (CN, St, LP), and (CN, St, CN). Then, by the "1224" ranking system, we find the next best joint-fourth-place Rule Kernels with sequences: ($2A$, St, $D0$), ($2A$, St, LP), and ($2A$, St, CN). These six Rule Kernels shall be considered the best for this problem instance.

By collecting all the top performing Rule Kernels across all the graphs in our data-set, we are able to determine the best rule sequences across the various structural parameters of our graphs.

It should be noted this is run in Java 1.8.0, and that for Rule 10 we use IBMs CPLEX Studio 20.1.0 to generate our Linear Program. This is relevant when considering the timing results t of the Rule Kernels.

3.3 Determining The Best Rule Sequences across Structural Parameters

Because some Reduction Rules are subsumed by other Reduction Rules (for example, Degree 0 being subsumed by Complete Neighbourhood), or because some rules perform similar actions on certain graph substructures, we would not always expect a single ordering to be determined as most

effective by the previous searching method, instead we would expect a *family* of rule sequences to be effective in similar ways.

For example, the sequences $[CN, St, D0]$ would have an identical effect as $[CN, St, CN]$ and $[CN, St, LP]$ and for small enough graphs, these three rule sequences rank equivalently, for their small enough Rule Kernel. This is exactly the case in Figure 9, where a three-way tie for the most effective Rule Kernel sees identical operations. In this scenario, we know from the search tree that the predecessor of the $[CN, St, \{D0/CN/LP\}]$ Rule Kernel, $[CN, St]$, contained no edges, thus all of its vertices were isolated. This meant that any rule that removed these isolated vertices would be sufficient to completely reduce this kernel, which Degree 0, Complete Neighbourhood, and Linear Programming were able to achieve.

Because of this, we consider grouping sets of high-performing rule sequences into classes or *communities*. We posit that rule sequences within the same communities are effective on the same types of graphs, that is, on graphs with the same structural properties.

To generate these communities for a given type of graph, we consider the set of rule sequences of all Rule Kernels that were ranked 1 (either solely or jointly) across all our problem instances of the given type of graph, call this set S. We shall construct a $|S| \times |S|$ matrix M, where $M_{i,i} = 0$, and $M_{i,j} =$*(the number of initial problem instances where both sequences* $R_i, R_j \in S$ *ranked as first)*, for $i, j \in \{1, 2, ..., |S|\}$. This means if $M_{a,b}$ is valued highly, then sequences R_a and $R_b \in S$ consistently take the top rank in terms of effectiveness across many of the same problem instances, which in turn implies they act very similarly.

We then consider the edge-weighted graph $G_S(V_S, E_S)$ generated by using M as an adjacency matrix, where each vertex $v_a \in V_S$ corresponds to a rule sequence $R_a \in S$, and the weight of the edge $(v_a, v_b) \in E_S$ corresponds to how many problem instances both sequences R_a and $R_b \in S$ were considered the best.

Upon G_S, we apply a community finding algorithm. In our study we employed NetworkX's greedy modularity community detecting algorithm:

`nx.algorithms.community import greedy_modularity_communities(G, weights)`.

This finds communities in G_S using Clauset-Newman-Moore greedy modularity maximization [4], generating communities of vertices where vertices within the same community are more likely to be adjacent, exhibiting denser clusters, whereas connections from one community to another are much more sparse.

This means we are able to obtain communities of rule sequences, where rule sequences in the same community performed highly on the same problem instances, and rule sequences from different communities rarely worked well simultaneously on a given problem instance. For each rule class, we shall generate a corresponding set of rule communities, and investigate across which values of the various parameters we consider do the rules from these different communities rank highly in terms of effectiveness.

4 Results

A full description of all the rule communities for each graph class is below in the Appendix.

4.1 Erdős-Rényi Graph Results

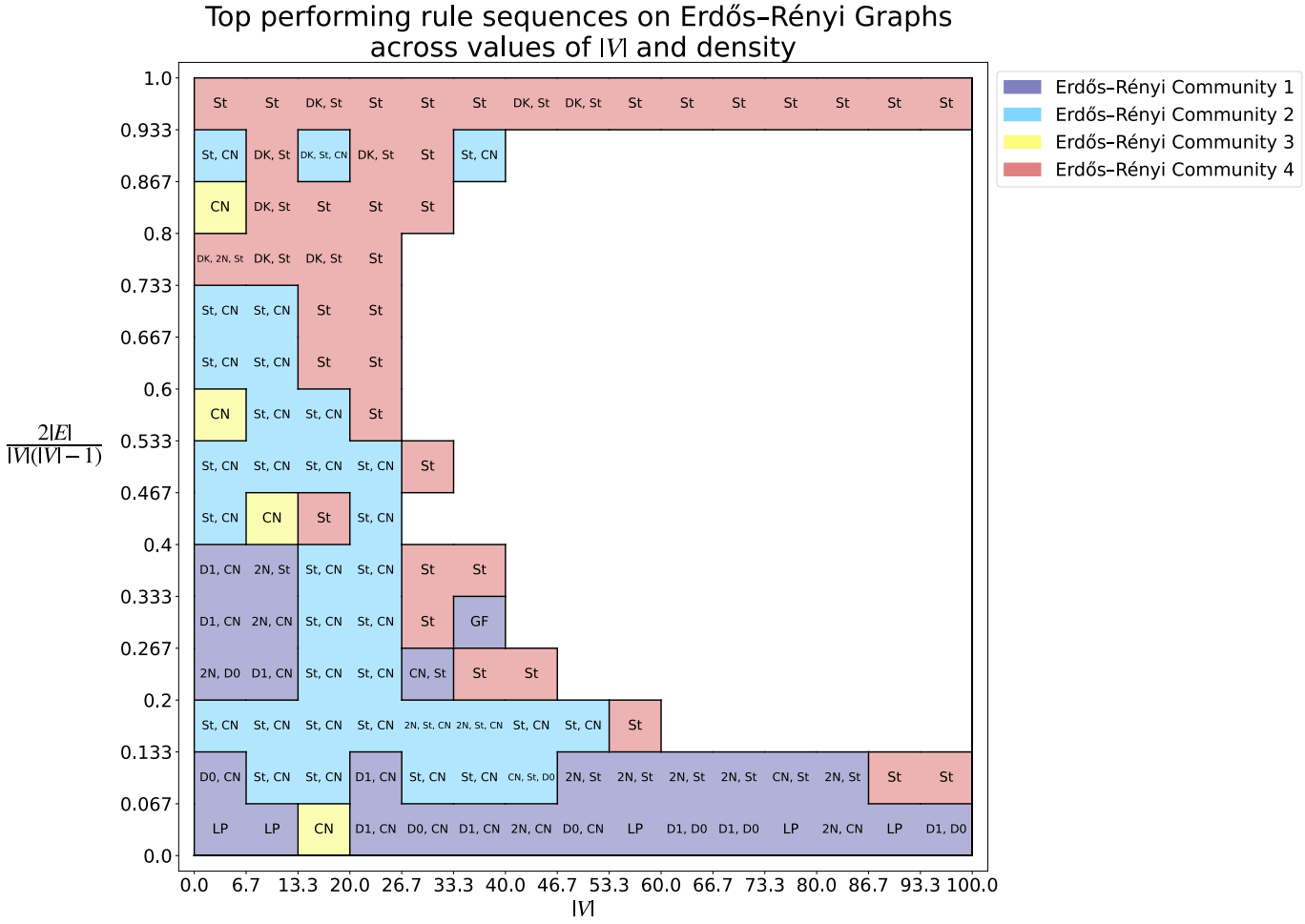
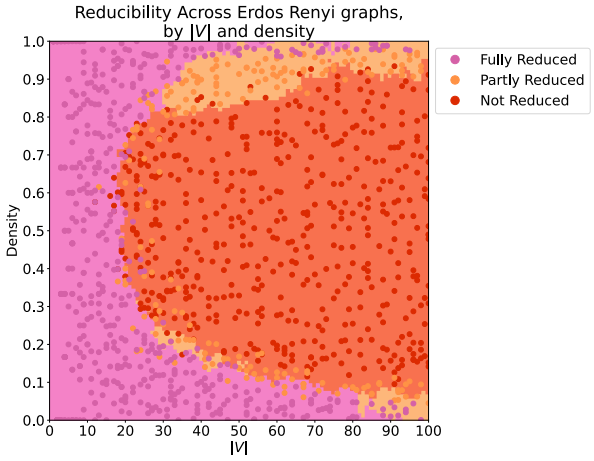
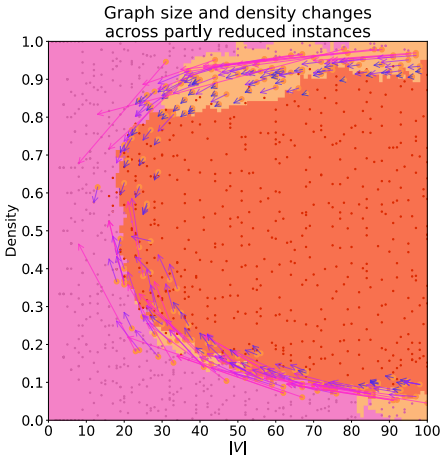


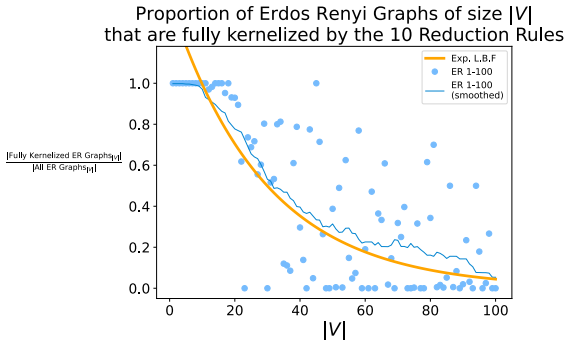
Figure 10: Binning across 15 ranges of $|V|$, and 15 ranges of density $\frac{|E|}{|V|(|V|-1)}$, this is how the different rule sequence communities of Erdős-Rényi graphs are situated across values of these two parameters.



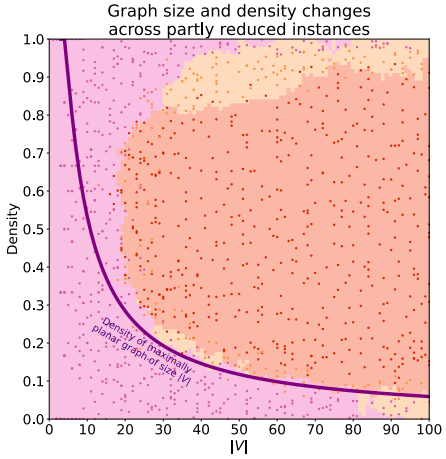
(a) Reducibility of Erdős-Rényi graphs by graph size $|V|$ and density $\frac{|E|}{|V|(|V|-1)}$. Graphs in the red zone are almost always completely unkernelizable.



(b) The changes in graph size $|V|$ and density $\frac{|E|}{|V|(|V|-1)}$ from the initial problem instance to the best kernel found.



(c) The proportion of graphs that are kernelized at size $|V|$, trend line in yellow is approximately $P_k = 10^{0.148-|V|\cdot 0.015}$



(d) The regions of unreducibility follow the boundary of maximum planar density.

Figure 12: The unreducibility chasm, appearing beyond $|V| = 20$. The change in kernel size and density follows the border around the chasm. As the size of the problem instance increases, the proportion of graphs that are kernelized follow an exponential decay.

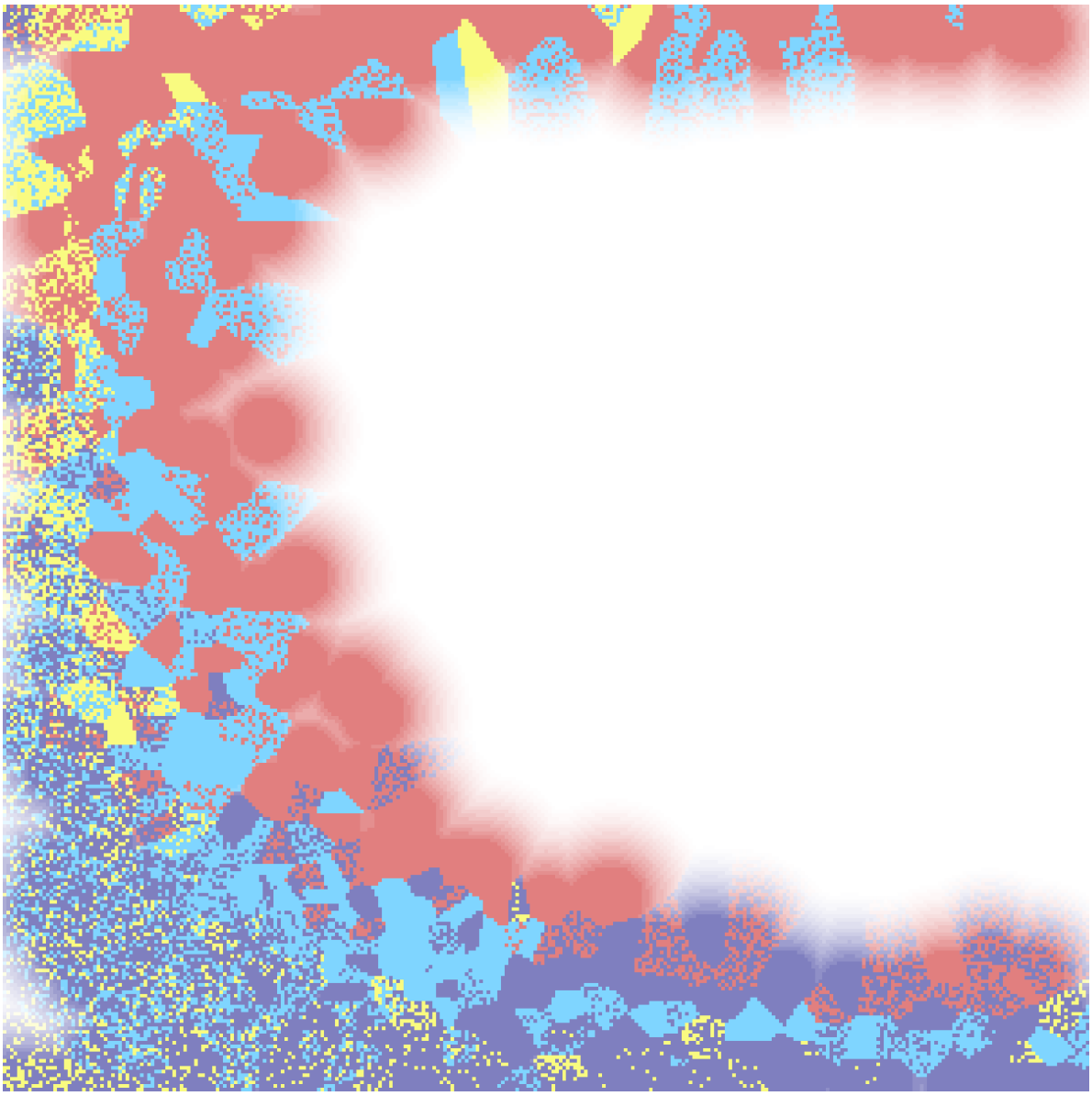


Figure 11: A render of what the Figure 10 would look like if the space of $|V|$ and Erdős-Rényi Density was calculated continuously. Although not shown in the render, this image follows the same axis and colouring as Figure 10, where purple, blue, yellow, and red correspond to Erdős-Rényi Communities 1, 2,3, and 4, and the horizontal axis for $|V|$ spans the range $[1, 100]$, and the vertical axis for Erdős-Rényi density spans the range $(0, 1)$. This figure was originally made for purely aesthetic purposes, however provides valuable insights beyond Figure 10.

4.2 Planar Graph Results

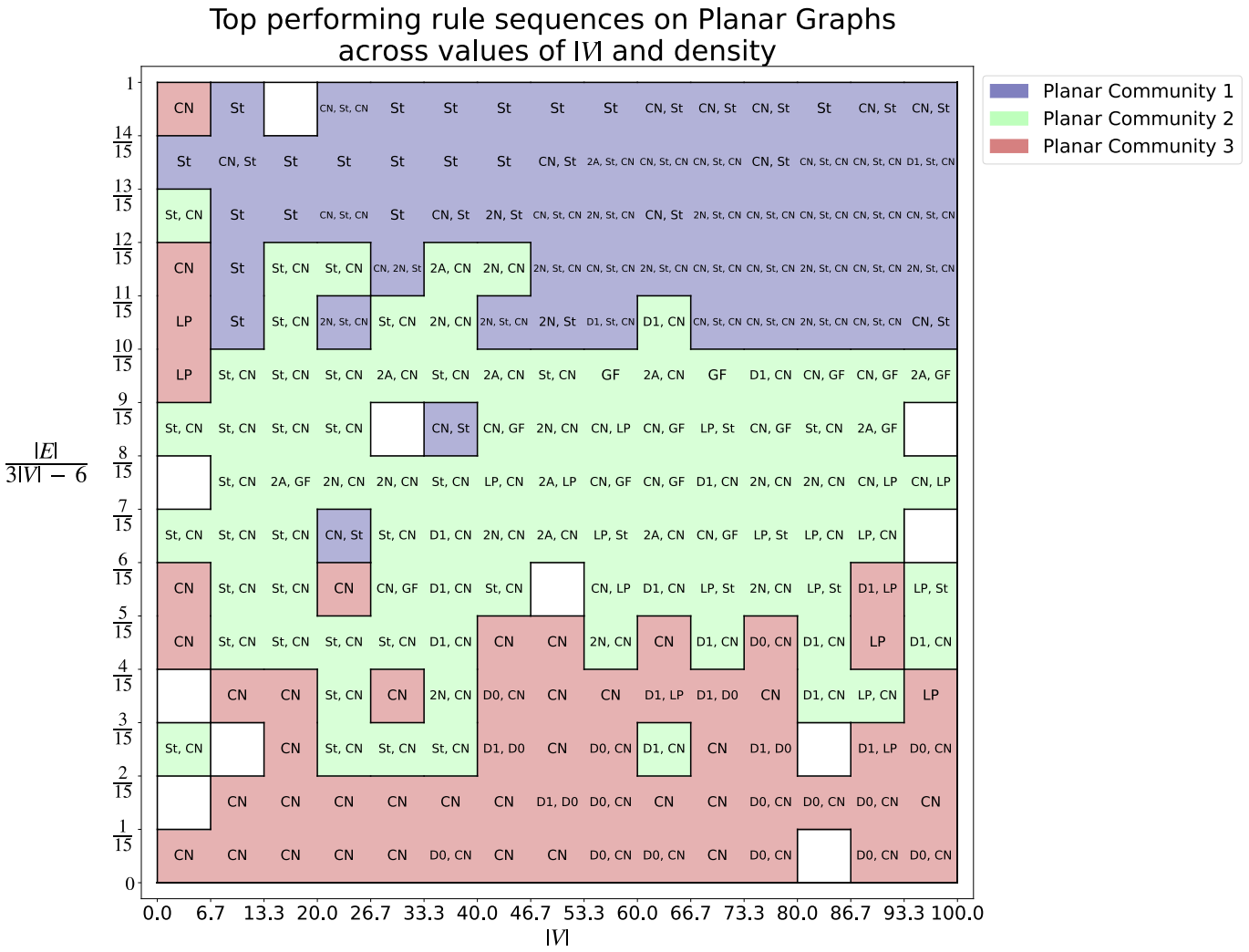
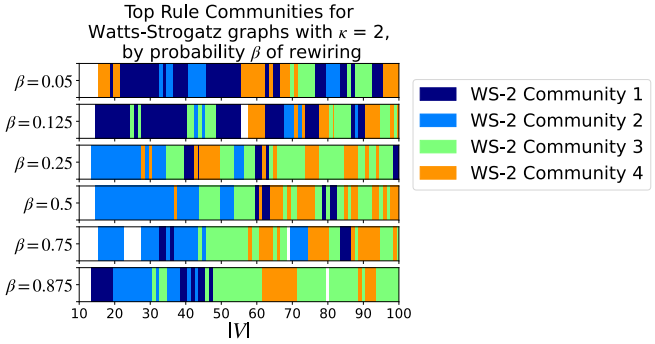
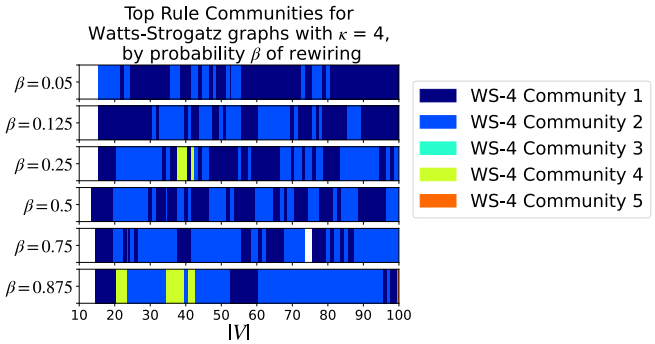


Figure 13: Binning across 15 ranges of $|V|$, and 15 ranges of planar density $\frac{|E|}{3|V|-6}$, this is how the different rule sequence communities of planar graphs are situated across values of these two parameters.

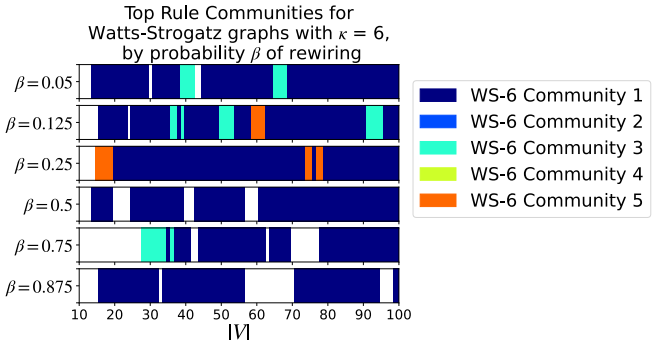
4.3 κ-Watts-Strogatz Graph Results



(a) Watts-Strogatz, $\kappa = 2$



(b) Watts-Strogatz, $\kappa = 4$



(c) Watts-Strogatz, $\kappa = 6$

Figure 14: Watts-Strogatz Graphs from $|V| \in [20, 100]$, across discrete values of β, for $\kappa = 2, 4, 6$

5 Discussion

5.1 Erdős-Rényi Discussion

The first thing we notice is, from Figure 12(a), there exist a large portion of Erdős-Rényi graphs in the range $|V| \in [1, 100]$, density $\in (0, 1)$ that are completely unreducible by the Ten Reduction Rules. These graphs lie in the goldilocks zone of being too big, and being dense enough that many of the rules that rely on non edges or small degrees do not apply, such as Rule 6 and Rule 1, but *just* sparse enough that Rule 3 can not be applied. Further investigation into this *chasm* would be interesting, as well as clearly defining its boundary.

Also from Figure 12(b) we see that most partly reduced instances, i.e. instances that were able to be reduced to a smaller kernel, but not solved completely, seemed to reduce by moving across $|V|$ and density *around* the edge of the chasm. The endpoints of each partly reduced instance is where its final kernel lies on the axis described above, where it seems graphs of this nature with a density over 0.5 reduced to more sparse graphs, up to a density of 0.5, and graphs with a density below 0.5 reduced to something more dense.

By the nature of reduction rules, these kernels always had fewer vertices than the original problem instance. We find that those closer to the chasm seemed to have a slower trajectory (across $|V|$ and density) than instances further away from the chasm. An investigation as to how the magnitude of change relates to the distance from the chasm would be worth investigating.

As for effective rule sequences, we can see, mostly just by sheer area, that Planar Community 2 and 4 dominate the space, particularly for dense graphs, although both communities have a sizable presence among sparse graphs too. In particular, $[St]$ seems to be the dominating rule sequence among dense Erdős-Rényi graphs, at least in this reducible zone.

At very low densities, it is unsurprising to see rule sequences that begin with *D0*, *D1*, and *2N*, among Planar Community 1. These sequences tend to work well with extremely sparse, almost tree-like graphs, based on their low density.

Figure 11 shows how the coverage of these rule sequence communities are spread across the two parameters $|V|$ and density. We see that Erdős-Rényi Community 4 in red, which is mainly Rule 7 and some Rule 3, is strongly present at the border between the reducible region and unreducible chasm of Erdős-Rényi graphs. It is also interesting to note that Erdős-Rényi Community 2 seems to always be sandwiched in between Erdős-Rényi Community 4 and Erdős-Rényi Community 1, which when consulting the Appendix, makes sense since the rule sequences in this community seem to be employing a mix of folding techniques by Rules 7 and 6, while also employing rules used on sparse and simpler graphs, such as Rules 5, 10, and 1.

All the sequences in Erdős-Rényi Community 3 are either kernelizing on a high degree before cleaning up the rest with either *LP* or *CN*, or they are employing a sequence equivalent to *CN*. It therefore makes sense that rule sequences from this community reduce small, dense graphs, the most effectively, seemingly for $|V| \leq 20$.

5.2 Planar Discussion

From Figure 13, the three rule sequence communities found form three layers across the planar density parameter. It is interesting to note that the area of influence of these rule sequence communities seems to be independent of $|V|$, and instead by heavily reliant on density. This is possibly

due to the nature of planar graphs generated by Delaunay Triangulation, where local substructures are not always indicative of global structures, that is, when comparing a neighbourhood in a particularly large planar graph of a given density with a neighbourhood of a smaller planar graph of the same density, both substructures would be almost identical. Compare this to an Erdős-Rényi graph, where a graph of density d with $|V| = 1000$ would have dramatically different types of neighbourhoods when compared to a graph of density d and $|V| = 100$, where there would be an average ten-fold difference in the size of the neighbourhood.

Observing the communities themselves, it seems once again that Struction is best at solving the more dense instances. Among Planar Community 1, smaller graphs are able to do the job with the single-rule sequence $[St]$, whereas larger graphs prefer $[CN, St, CN]$. It seems densities above $\frac{2}{3}$ start requiring rule sequences from Planar Community 1, in particular $[St]$ for smaller graphs of $|V| < 40$, and $[CN, St, CN]$ for larger graphs. For graphs with a sparsity under $\frac{1}{3}$, these are sparse enough that we are able to solve them using only $[CN]$, almost regardless of the size. For very large sparse graphs, using $[LP]$ also proves to be useful.

According to Figure 12(d), it seems the boundary of maximum planar density along $|V|$ tends to coincide with the lower boundary of the unreducible chasm from the figures in Figure 12. This relationship is further shown when comparing Figure 10 to Figure 13, it seems Planar Communities 1, 2, and 3 overlap with Erdős-Rényi Communities 4, 2, and 1 respectively, going so far as to reflect the sandwiching of Erdős-Rényi Community 2 by Communities 4 and 1 as the sandwiching of Planar Community 2 by 1 and 3.

Ultimately it seems that perhaps the non-global nature of substructures within planar graphs allow $|V|$ to be independent of the rule sequence effectiveness among the planar communities found, and this is slightly reflected among similarly dense planar graphs. It would be interesting to investigate whether this is an inherent property of planar graphs, or whether this independence from $|V|$ extends to other graphs generated by a proximity measure, as with Delaunay Triangulation.

5.3 Watts-Strogatz Discussion

Not much could be gleamed from the Watts-Strogatz results. Graphs with $\kappa = 6$ were almost always solved with $[St]$. For $\kappa = 2$, the Rules 6 and 4 were the first rules in the majority of sequences in WS-4 Communities 1 and 2, the two communities that dominated the $\kappa = 4$ graphs across all value of β.

We know that for $\kappa = 6$ graphs, the average degree is exactly 6. Comparing this to planar graphs, where the average degree never exceeds 6, but approaches it in the limit as $|V|$ increases for planar graphs of density 1, we see that Watts-Strogatz graphs with $\kappa = 6$ are quite similar to extremely dense planar graphs. This explains why $[St]$ is an extremely efficient sequence for both types of graphs.

The underlying patterns behind the results of the $\kappa = 2$ graphs remain unclear. It seems that WS-2 Communities 1 and 2 are effective on smaller graphs, and WS-2 Communities 3 and 4 on larger ones, however corresponding with the rule sequences that make up these communities (in the Appendix), we find it difficult to understand why this is the case. It is possible that alternate structural properties must be considered to understand what is going on for $\kappa = 2$ graphs, such as clustering.

6 Conclusion

We observe effective Reduction Rule Sequences, as well as communities of such sequences, across Erdős-Rényi graphs, planar graphs, and Watts-Strogatz graphs, investigating where these rule sequences and rule sequence communities were most effective across various structural properties of these graphs, in particular by graph size and density for Erdős-Rényi and planar graphs, and across mean degree and rewiring probability for Watts-Strogatz graphs.

We find a strong relationship between which rule sequence communities were most effective for a given problem instance, and the size and density of the problem instance for Erdős-Rényi and planar graphs, as well as showing how graph size can often be irrelevant for planar graphs when determining which rule sequence is the most effective for planar graphs.

We also find that determining between which rule sequence communities were most effective for a Watt-Strogatz graphs heavily depended on κ, and that for $\kappa = 2$, the relationship between dominating rule sequence communities and $\beta|V|$ is not clear.

It would be interesting to investigate how other graph properties correlate with the effectiveness of these rule sequences.

It would also be interesting to look at other reduction rules such as reduction by Network Flow as covered in Abu-Khza et al. [1], and other graph classes, particularly parameterized classes such as $k-$outer-planar and $k-$tree-width.

References

[1] Faisal N Abu-Khzam et al. "Kernelization algorithms for the vertex cover problem". In: (2017).

[2] Jonathan F Buss and Judy Goldsmith. "Nondeterminism within P^". In: *SIAM Journal on Computing* 22.3 (1993), pp. 560–572.

[3] James Cheetham et al. "Solving large FPT problems on coarse-grained parallel machines". In: *Journal of Computer and System Sciences* 67.4 (2003), pp. 691–706.

[4] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. "Finding community structure in very large networks". In: *Physical review E* 70.6 (2004), p. 066111.

[5] Ernest J Cockayne, Stephen T Hedetniemi, and R Laskar. "Gallai theorems for graphs, hypergraphs, and set systems". In: *Discrete mathematics* 72.1-3 (1988), pp. 35–47.

[6] cppreference.com. *list::insert - cppreference.com*. Accessed: 2020-06-05. URL: `https://en.cppreference.com/w/cpp/algorithm/is_sorted#Complexity`.

[7] Marek Cygan et al. *Parameterized algorithms*. Vol. 4. 8. Springer, 2015.

[8] Rod Downey. "A basic parameterized complexity primer". In: *The Multivariate Algorithmic Revolution and Beyond*. Springer, 2012, pp. 91–128.

[9] Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*. Vol. 4. Springer, 2013.

[10] Rodney G Downey and Michael R Fellows. "Parameterized computational feasibility". In: *Feasible mathematics II*. Springer, 1995, pp. 219–244.

[11] Michael R. Fellows et al. *What is known about Vertex Cover Kernelization?* 2018. arXiv: `1811.09429 [cs.DS]`.

[12] Richard M Karp. "Reducibility among combinatorial problems". In: (1972), pp. 85–103.

[13] Kazimierz Kuratowski. "Sur le probleme des courbes gauches en topologie". In: *Fundamenta mathematicae* 15.1 (1930), pp. 271–283.

[14] Wenjun Li and Binhai Zhu. "A 2k-kernelization algorithm for vertex cover based on crown decomposition". In: *Theoretical Computer Science* 739 (2018), pp. 80–85.

[15] *NetworkX*. 2021. URL: `https://networkx.org/`.

[16] *SciPy*. 2021. URL: `https://docs.scipy.org/doc/scipy/reference/index.html`.

[17] Ulrike Stege. "Resolving conflicts in problems from computational biology". PhD thesis. ETH Zurich, 1999.

[18] Mingyu Xiao and Hiroshi Nagamochi. "Exact algorithms for maximum independent set". In: *Information and Computation* 255 (2017), pp. 126–146.

Appendix

Erdős-Rényi Communities

Community: 1
- 2N, D0
- GF
- D1, CN
- 2N, CN
- D1, D0
- D0, CN
- Cr, D0
- D1, LP
- Cr, CN
- D0
- LP
- D0, St
- 2N, LP
- CN, St
- 2N, St
- Cr, LP
- D0, LP
- Cr, GF

Community: 2
- St, CN
- St, D0
- 2N, St, LP
- St, LP
- CN, St, LP
- CN, GF
- DK, St, D0
- DK, St, CN
- 2N, St, D0
- 2N, St, CN
- CN, St, CN
- CN, St, D0
- DK, St, LP

Community: 3
- CN
- DK, LP
- 2A, D0
- 2A, LP
- 2A, CN
- DK, CN
- DK, D0

Community: 4
- St
- DK, 2N, CN
- DK, St
- DK, 2N, St

Planar Communities

Community: 1
- CN, St
- CN, St, CN
- St
- CN, 2N, St
- 2N, CN, St
- 2A, CN, St
- 2N, St, CN
- 2N, St
- D1, St, CN
- 2A, St, CN
- CN, 2N, St, CN
- D1, CN, St
- 2N, CN, 2N, St

Community: 2
- D1, GF

- St, CN
- D1, CN
- 2N, CN
- 2A, CN
- LP, CN
- GF
- CN, GF
- LP, St
- 2A, GF
- 2A, LP

Community: 3

- CN, LP
- CN
- Cr, GF
- Cr, CN
- D0, CN
- LP
- D1, D0
- D1, LP
- Cr, LP

Watts-Strogatz Communities

$\kappa = 2$

Community: 1

- CN, GF, CN
- 2N, 2A, D0
- 2A, GF
- Cr, St
- D1, GF
- CN, GF, LP
- CN, St
- GF
- 2N, St
- St
- CN, GF
- 2N, GF
- CN, GF, D0
- 2N, 2A, CN

Community: 2

- St, CN
- Cr, CN
- 2A, LP
- 2A, CN
- Cr, GF
- LP, St
- LP, GF
- LP, CN

Community: 3

- D1, D0
- D1, LP
- 2N, D0
- St, D0
- 2N, LP
- St, LP
- Cr, LP
- CN, LP

Community: 4

- 2N, D1, LP
- LP
- 2N, D1, CN
- 2N, D1, D0
- GF, CN
- GF, D0
- GF, LP

$\kappa = 4$

Community: 1
- St
- 2A, CN, 2N, St
- 2N, CN, 2N, St
- CN, St
- 2A, St, CN
- CN, 2N, St
- 2N, CN, St
- 2A, 2N, CN, St
- 2A, 2N, St, D0
- 2A, CN, St
- 2A, 2N, St, CN
- 2A, 2N, St, LP
- 2A, D0, 2N, St
- 2A, D0, St

Community: 2
- St, CN
- St, D0
- 2N, St, CN
- 2N, St, D0
- 2N, St
- CN, St, D0
- CN, St, CN
- 2N, GF
- St, LP
- CN, St, LP
- 2N, St, LP
- 2N, CN
- GF

Community: 3
- CN, 2N, CN
- CN, 2N, 2A, CN
- CN, 2N, 2A, D1, CN
- CN, 2N, 2A, D0, CN
- CN, 2N, 2A, St, CN
- CN, 2N, 2A, D1, D0
- CN, 2N, 2A, St, LP
- CN, 2N, 2A, LP
- CN, 2N, 2A, St, D0
- CN, 2N, 2A, D0, St

Community: 4
- 2A, GF
- 2A, D0, CN
- 2A, St, D0
- 2A, D1, CN
- 2A, St, LP
- 2A, CN
- CN

Community: 5
- 2N, GF, St, CN
- 2N, GF, St, LP
- 2N, GF, St, D0

$\kappa = 6$

Community: 1
- St

Community: 2
- St, D0

Community: 3
- St, CN

Community: 4
- St, LP

Community: 5
- CN, St