Applying Computational Resources to the Down-Arrow Problem

by

Johnathan Koch

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in the

Mathematics

Program

YOUNGSTOWN STATE UNIVERSITY

May, 2023

Applying Computational Resources to the Down Arrow Problem

Johnathan Koch

Signature:

Johnathan Koch, Student May 2023

Approvals:

Dr. Alexis Byers, Thesis Advisor May 2023

Dr. Alina Lazar, Committee Member May 2023

Dr. Anita O'Mellan, Committee Member May 2023

Dr. Salvatore A. Sanders, Dean of Graduate Studies May 2023

Abstract

A graph G is said to arrow a graph H if every red-blue edge coloring of G presents a monochromatic H, and is written $G \to H$. The down-arrow Ramsey set reports all subgraphs H of a graph G for which $G \to H$. Formally, the down-arrow Ramsey set is a graph G is $\downarrow G := \{H \subseteq G : G \to H\}$. Calculating this set by way of scientific computing is computationally prohibitive with the resources commonly available to graph theorists and other academics. Using existing research into complete graphs, the down-arrow Ramsey sets for small complete graphs (K_n for $2 \leq n \leq 7$) can be generated quickly. For larger complete graphs (K_n for $8 \leq n \leq 11$) specific pre-processing steps are leveraged to speed up calculations in addition to existing data sets. Presented is work on the development of a Python script to generate the down-arrow Ramsey set of a graph through efficient memory management and parallel computing methodologies. The *down-arrow generator* is used to report new results on complete graphs as well as complete bipartite graphs, and assorted other graphs.

Table Of Contents

Introduction

Computers have no doubt transformed much of research today. Presented by Appel, Haken, and Koch, [1, 2], the proof that "every planar map is four colorable" in 1976 was a leap forward in research as the first computer-assisted mathematical proof accepted by most. In their paper, they present the reduction of "every planar graph" to a finite number of cases, each of which likely requiring the computational power of computers to determine four colorability. Many of those who doubt the proof of the four color problem follow Tymoczko who argues that the four color problem cannot be called a theorem due to the philosophical ramifications. Additionally, Tymoczko states "no mathematician has seen a proof to the [four color theorem]" due to the reliance on computer calculations [23].

This paper presents both calculations and proofs. Computers are used to provide insight into conjectures. By the use of rigorous error checking and many tests of viability, the down-arrow Ramsey sets reported by the down arrow generator are highly likely to be correct.

1.1 Introduction to Ramsey Theory

Graham and Butler put it quite cleanly: "Ramsey theory is that branch of combinatorics which deals with structure which is preserved under partitions" [13]. Classic theorems in Ramsey theory are:

Theorem 1.1.1 (Van Der Waerden)**.** *In any partition of the integers into finitely many classes, some class always contains arbitrarily long arithmetic progressions.*

Theorem 1.1.2 (Ramsey)**.** *For any partition of the k-element subsets of an infinite set S into finitely many classes, there is always an infinite subset of S with all its k-elements subsets in a single class.*

Both of these theorems take a mathematical object (integers or sets) and inspect what kinds of substructures must always remain in at least one of the classes after arbitrary partitioning.

A classical example of Ramsey theory in motion is the following statement [14]:

Theorem 1.1.3. *In any collection of six people, there will be either three mutual acquaintances, or three mutual strangers.*

It should be noted here that "being an acquaintance" in this regard is binary ("jib" is either an acquaintance or not an acquaintance of "jab"), symmetric (if "jib" is an acquaintance of "jab", then "jab" is an acquaintance of "jib"), but not transitive (if "jib" is an acquaintance of "jab" and "jab" is an acquaintance of "jeb" then "jib" is not necessarily an acquaintance of "jeb").

Proof. Following the structure of Graham et. al [14], let us fix one person, say person A. By the pigeonhole principle, person A must either be acquaintances with or strangers with at least three other people, say persons B, C, and D. Without loss of generality, suppose person A is acquaintances with each of persons B, C, and D. If one of persons B, C, or D is acquaintances with another of persons B, C, or D, then they would form a group of three mutual acquaintances with person A. If none of persons B, C, or D are acquaintances, then they form a group of three mutual strangers. □

This theorem has a direct translation into a graph theoretic problem that will be discussed later, once we build the fundamental definitions required.

1.2 Introduction to Graph Theory

Let us begin with some preliminary structure to graph theory. A *graph* G is two sets: a vertex set $V(G)$ and an edge set $E(G)$. For a *simple, undirected graph*, an *edge* is an unordered tuple of distinct vertices. For a *simple, directed graph*, an *edge* is an ordered tuple of distinct vertices. For vertices u and v, the edge between them can be written as (u, v) or more simply uv. In this way, edges are sometimes said to live in the set $V(G) \times V(G)$ where order may or may not matter, depending on directedness.

The *order* of a graph G is the number of vertices in $V(G)$. The *size* of a graph G is the number of edges in $E(G)$. Two vertices are *adjacent* to one another when there exists an edge between them. The *degree* of a vertex is the number of other vertices adjacent to it. A vertex is *isolated* if it has degree 0. An edge is *indicent* to the vertices it contains. For example, please refer to Fig. 1.1, pictured is the Zim graph, a simple undirected graph of order 11 and size 15 with no isolated vertices. Vertex i is adjacent to vertices e, f, g, j, and k. Edge ab is incident to the vertices a and b.

The *adjacency matrix* of a graph G of order n is an $n \times n$ matrix encoding vertex adjacency. This matrix

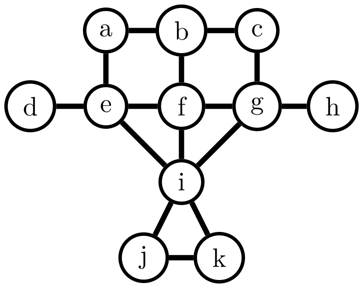


Figure 1.1: The Zim graph

is defined entry-by-entry as follows [11]:

$$a_{ij} = \begin{cases} 1 \text{ if } v_i v_j \in E(G) \\ 0 \text{ if } v_i v_j \notin E(G) \end{cases}$$

Importantly, multiple adjacency matrices can represent the same graph by ordering the vertices differently. Consider the two matrices in Fig. 1.2 that both represent the Zim graph G in Fig. 1.1. Highlighted in these two matrices are where vertices d and h are transposed. Also of note is that for simple, undirected graphs the adjacency matrix representation is symmetric about the main diagonal–which contains only zeros. This fact will be leveraged later to efficiently store a simple, undirected graph in computer memory.

The *complement* of a graph G, written as $\overline{G}$, is given by the graph where $V\left(\overline{G}\right) = V(G)$ and $E\left(\overline{G}\right) = \overline{E(G)}$. We are using that $E(G)$ resides in $V(G) \times V(G)$, so $\overline{E(G)}$ is the set of edges not in $E(G)$. In other words: vertices u and v are adjacent in $\overline{G}$ if and only if u and v are not adjacent in G. Fig. 1.3 shows a graph G and its complement $\overline{G}$.

A graph G is a *path* on n vertices if $V(G)$ can be written as $\{v_1, v_2, ..., v_n\}$ and $E(G) = \{v_i v_{i+1} : 1 \leq i \leq n-1\}$. In this way, a path on n vertices can be written as P_n or $v_1 v_2 ... v_n$. A graph G is a *cycle* on n vertices if $V(G)$ can be written as $\{v_1, v_2, ..., v_n\}$, $v_1 v_2 ... v_n$ is a path P, and $E(G) = E(P) \cup \{v_1 v_n\}$. The notation C_n or $v_1 v_2 ... v_n v_1$ is used for cycles on n vertices. The *complete graph* on n vertices, K_n, is given by a vertex set with n vertices and $uv \in E(K_n)$ if and only if $u, v \in V(K_n)$ are distinct. A *complete bipartite graph* is a graph G where $V(G)$ can be partitioned into two disjoint sets S and T where $uv \in E(G)$ if and only if $u \in S$ and $v \in T$. To denote the size of each partition, we use $K_{s,t}$ for complete bipartite graphs with underlying partite sets of order s and t respectively. Refer to Fig. 1.4 for some examples of these classes of graphs. This paper investigates the properties of complete and complete bipartite graphs, while recognizing the analysis already done on paths and cycles.

We now have the foundational definitions to begin discussing how graphs relate to each other. Like many mathematical objects, there is a concept of isomorphism for graphs. A *graph isomorphism* between graphs G

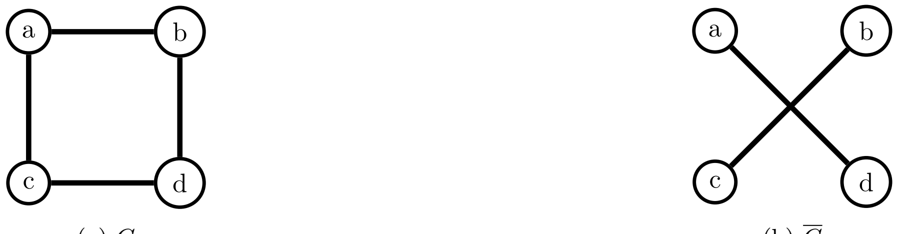
vertex	a	b	c	d	e	f	g	h	i	j	k
a	0	1	0	0	1	0	0	0	0	0	0
b	1	0	1	0	0	1	0	0	0	0	0
c	0	1	0	0	0	0	1	0	0	0	0
d	0	0	0	0	1	0	0	0	0	0	0
e	1	0	0	1	0	1	0	0	1	0	0
f	0	1	0	0	1	0	1	0	1	0	0
g	0	0	1	0	0	1	0	1	1	0	0
h	0	0	0	0	0	0	1	0	0	0	0
i	0	0	0	0	1	1	1	0	0	1	1
j	0	0	0	0	0	0	0	0	1	0	1
k	0	0	0	0	0	0	0	0	1	1	0

(a) Vertices ordered lexicographically

vertex	a	b	c	h	e	f	g	d	i	j	k
a	0	1	0	0	1	0	0	0	0	0	0
b	1	0	1	0	0	1	0	0	0	0	0
c	0	1	0	0	0	0	1	0	0	0	0
h	0	0	0	0	0	0	1	0	0	0	0
e	1	0	0	0	0	1	0	1	1	0	0
f	0	1	0	0	1	0	1	0	1	0	0
g	0	0	1	1	0	1	0	0	1	0	0
d	0	0	0	0	1	0	0	0	0	0	0
i	0	0	0	0	1	1	1	0	0	1	1
j	0	0	0	0	0	0	0	0	1	0	1
k	0	0	0	0	0	0	0	0	1	1	0

(b) Vertices d and h are transposed

Figure 1.2: Two adjacency matrices for graph G in Fig. 1.1



(a) G (b) $\overline{G}$

Figure 1.3: A graph G and its complement $\overline{G}$

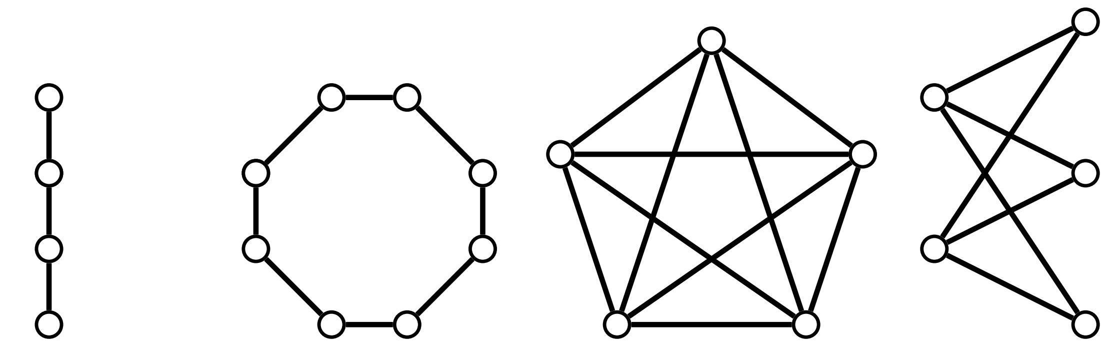
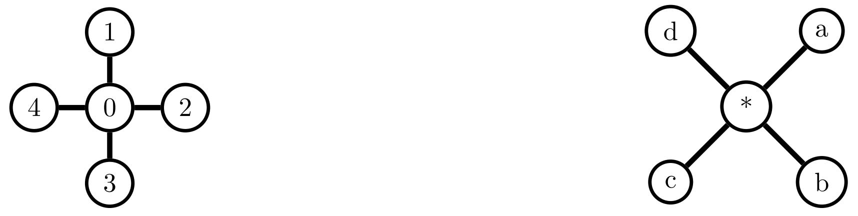

(a) P_4 (b) C_8 (c) K_5 (d) $K_{2,3}$

Figure 1.4: Four classes of graphs



(a) G (b) H

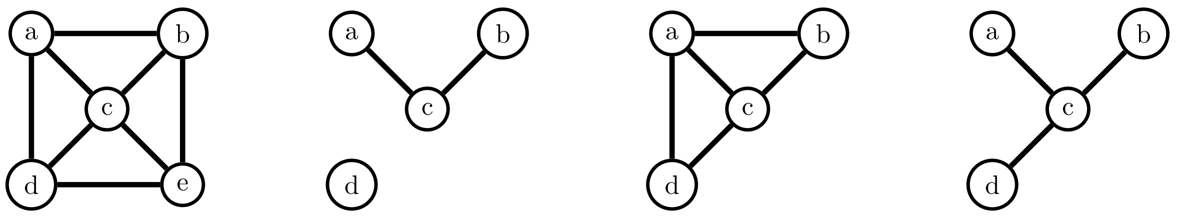
Figure 1.5: Isomorphic graphs G and H

and H is a bijective function $f : V(G) \to V(H)$ where $uv \in E(G)$ if and only if $f(u)f(v) \in E(H)$. We write G is isomorphic to H as $G \cong H$. For the purposes of this paper, graphs will be unique up to isomorphism. Also of note is how "up to isomorphism" allows differing graph drawings to represent the same graph. In particular, without the inherent "up to isomorphism", graph G and graph H in Fig. 1.5 would be distinct, though for all intents within this paper they are treated as the same graph.

A graph H is a *subgraph* of another graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, and we write $H \subseteq G$. An *induced subgraph* H of G is written as $H = G[I]$ where $I \subseteq V(G) \cup E(G)$. In this way, I is the inducing set and is allowed to contain any number of vertices or edges. The resulting graph H has all vertices in I, all vertices incident to an edge in I, and all edges in I.

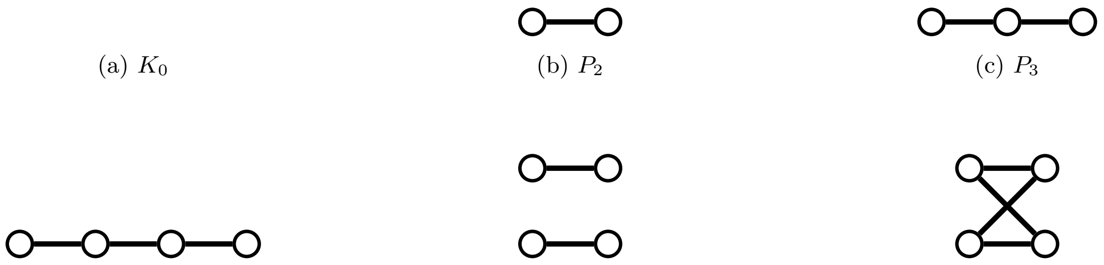
There are two special cases of an induced subgraph. When I contains only edges, $G[I]$ is an *edge-induced subgraph*. A *vertex-induced subgraph* H is a special type of induced subgraph of G where I contains only vertices, but $E(H)$ contains all edges in G incident to two vertices in I. Fig. 1.6 shows a graph G with three subgraphs, one of each type of induced subgraph. Graph H_1 is an induced subgraph of $G[I]$ where $I = \{d, ac, bc\}$. Graph H_2 is the vertex-induced subgraph $G[I]$ where $I = \{a, b, c, d\}$. Graph H_3 is the edge-induced subgraph $G[I]$ where $I = \{ac, bc, cd\}$.

This marks one of the recurring themes of the development of the down arrow generator: software developers use different implied terminology than graph theorists. Software developers use the shorthand



(a) G (b) $H_1 \subseteq G$ (c) $H_2 \subseteq G$ (d) $H_3 \subseteq G$

Figure 1.6: A graph G and three subgraphs


(a) K_0 (b) P_2 (c) P_3

(d) P_4 (e) $2P_2$ (f) $K_{2,2}$

Figure 1.7: $\langle K_{2,2} \rangle$

"subgraph" to refer to vertex-induced subgraphs while graph theorists use the shorthand to refer to edge-induced subgraphs. This paper will be diligent to make the distinction in every place where a subgraph, induced subgraph, vertex-induced subgraph, or edge-induced subgraph are needed. As this paper has roots more in graph theory than computer science, edge-induced subgraphs will be the most prolific type of subgraph discussed.

Consider "is a subgraph" to be a relation on graphs. It falls quickly from the underlying set operations that "is a subgraph" satisfies reflexivity, antisymmetry, and transitivity. Hence we may view "is a subgraph" as a partial ordering on a set of graphs. In a similar manner, all three forms of induced subgraphs provide a partial ordering on a set of graphs. For our purposes, the *poset* of a graph G is given to be the set of edge-induced subgraphs of G with the partial ordering "is an edge-induced subgraph of". We denote the poset of G under this relation with $\langle G \rangle$.

Definition 1.2.1. $\langle G \rangle = \{G[A] : A \subseteq E(G)\}$

Only to point out the different terminology between graph theorists and computer scientists, "is an edge-induced subgraph of" translates to "is subgraph monomorphic to" in the computer science literature [15].

Fig. 1.7 shows the poset $\langle K_{2,2} \rangle$ to contain, up to isomorphism, the 6 graphs $K_0, P_2, P_3, P_4, 2P_2$, and $K_{2,2}$. As a note, prefacing a graph H with a positive integer indicates how many distinct copies of H are present as subgraphs of the reported graph. This is to say that there are two distinct copies of P_2 in $2P_2$.

This formulation of $\langle G\rangle$ is a slight departure from work done by Byers and Olejniczak as the poset of a graph in their paper uses the relation "is a subgraph" [5, 6]. The key distinction is that graphs with isolated vertices are elements in a graph's poset for Byers and Olejniczak , whereas they are omitted in work done in this paper. This is done as a run-time optimization for the down arrow generator program because adding isolated vertices to a graph does not change the structure red-blue edge colorings. To translate posets from this paper into posets as Byers and Olejniczak use, use the following process:

For $1 \leq i \leq |V(G)|$, for $H \in \langle G\rangle$ where $|V(H)| + i = |V(G)|$: add a copy of H with i isolated vertices to $\langle G\rangle$.

1.3 Introduction to Graph Theoretic Ramsey Theory

One of the special cases of Theorem 1.1.2 applicable in graph theory is as follows [11]:

Theorem 1.3.1 (Ramsey 1930)**.** *For every $r \in \mathbb{N}$ there exists an $n \in \mathbb{N}$ such that every graph of order at least n contains either K_r or $\overline{K_r}$ as an edge-induced subgraph.*

The common way to approach this is not to look at all possible graphs on n vertices, but instead to inspect the complete graph on n vertices and consider all possible red-blue edge colorings of K_n. A *red-blue edge coloring* $\mathcal{C}$ of a graph G is the assignment of either "red" or "blue" to each edge of G.

Definition 1.3.2. *For a graph G, a* red-blue edge coloring *is $\mathcal{C} : E(G) \to \{$"red", "blue"$\}$.*

There is no restriction to which edges are colored red or blue in red-blue edge colorings. The special case of a graph where all edges are assigned the same color is a *monochromatic graph.* A red-blue edge coloring $\mathcal{C}$ of a graph G induces two monochromatic edge-induced subgraphs:

- $\mathcal{C}_r = G[\mathcal{C}^{\leftarrow}(\text{"red"})]$–the edge-induced subgraph of G given by red edges.
- $\mathcal{C}_b = G[\mathcal{C}^{\leftarrow}(\text{"blue"})]$–the edge-induced subgraph of G given by blue edges.

Superscripts will be used to index red-blue edge colorings when multiple are being considered. A red (blue) graph is simply a monochromatic red (blue) graph where every edge is colored red (blue). Two colorings are isomorphic if they induce isomorphic red and blue monochromatic edge-induced subgraphs. See Fig. 1.8 for a particular red-blue edge coloring of K_5 with $\mathcal{C}_r$ and $\mathcal{C}_b$ explicitly shown.

We can then restate Theorem 1.1.2 in the context of red-blue edge colorings.

Theorem 1.3.3. *For any $r \in \mathbb{N}$, there exists some $n \in \mathbb{N}$ such that whenever $m \geq n$, every red-blue edge coloring of K_m induces a monochromatic K_r.*

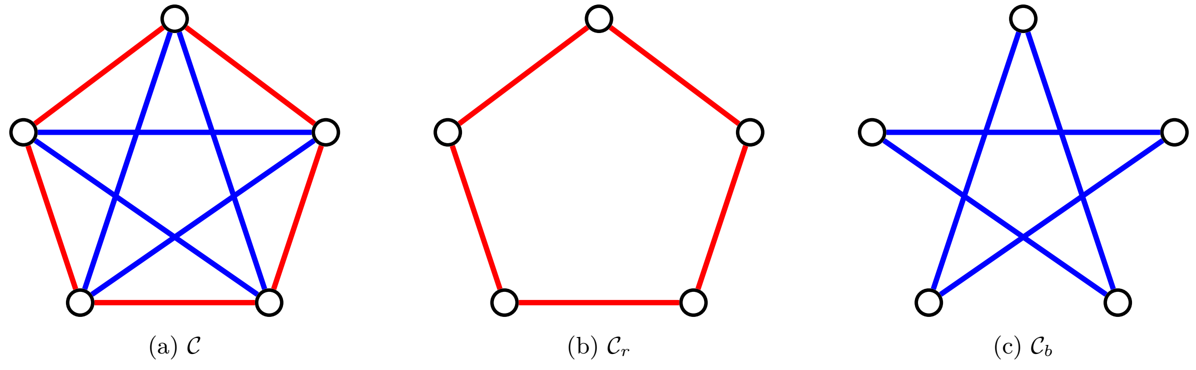

(a) $\mathcal{C}$ (b) $\mathcal{C}_r$ (c) $\mathcal{C}_b$

Figure 1.8: A red-blue edge coloring of K_5

In this way we can view the red subgraph to be the K_r and the blue subgraph to be the K_s (or vice versa) in Theorem 1.1.2. This allows the inspection of a single graph instead of all graphs on n vertices because considering all red-blue edge colorings does this implicitly. As every red-blue edge coloring of K_n is considered, the red subgraph (and the blue subgraph) ranges over every graph on n vertices.

Now, stating Theorem 1.1.3 in a graph theoretic framework results in:

Theorem 1.3.4. *Any red-blue edge coloring of K_6 must induce a monochromatic K_3*

We see that the gathering of six people in Theorem 1.1.3 correlates to K_6 in Theorem 1.3.4, and being an acquaintance or a stranger in Theorem 1.1.3 correlates to an edge being colored red or blue respectively in a red-blue edge coloring in Theorem 1.3.4. In fact, proofs of Theorem 1.3.4 follow directly that of Theorem 1.1.3 with appropriate translation of terminology [7].

The *Ramsey number* $R(s,t) = n$ for integers s and t is given to be the minimum n for which all red-blue edge colorings $\mathcal{C}$ of K_n induce a subgraph of $\mathcal{C}_r$ that is isomorphic to K_s or a subgraph of $\mathcal{C}_b$ that is isomorphic to K_t [7]. Proofs of Ramsey numbers come in two pieces–first it must be shown that every coloring of K_n induces a red K_s or a blue K_t, and second a coloring of K_{n-1} must demonstrate neither a red K_s nor a blue K_t. Fig. 1.8 shows a coloring of K_5 that does not contain a monochromatic K_3, hence showing $R(3,3) \geq 6$. Fig. 1.8 in combination with Theorem 1.3.4 constitute a proof that $R(3,3) = 6$, one of the most famous Ramsey numbers.

Ramsey numbers extend to graphs as inputs as well, where $R(F,H) = n$ is the smallest n for which every red-blue edge coloring $\mathcal{C}$ of K_n induces a subgraph of $\mathcal{C}_r$ isomorphic to F or a subgraph of $\mathcal{C}_b$ isomorphic to H. $R(F,H)$ is referred to as the *graph Ramsey number*. Looking at $R(F,H) = n$ from the perspective of the graph being colored, it is said that K_n *arrows* graphs F and H and is written $K_n \to (F,H)$ for $n = R(F,H)$ [4]. When F and H are isomorphic, we say simply that $K_n \to H$. Of special interest to work in this paper is determining what certain graphs G, that are not K_n, arrow. In effect: we will determine for a fixed graph

G, all edge-induced subgraphs H that satisfy $G \to H$.

1.4 Introduction to Python

Before discussing Python specifically, it must be mentioned the widespread use of network analysis tools on computers. The igraph tool [10] is available in R, Python, Mathematica, and C/C# and is readily updated. The JgraphT library [18] is a java specific implementation of many network analytic tools initially released in 2020. There is also the standalone implementation SageMath [21] that many graph theorists will be immediately familiar with. SageMath, or Sage for short, is used not only for graph theory however, as it was initially developed for number theory. As with many open-source projects, Sage has grown in scope as its user base has assisted with its development [22].

These predominantly researcher-focused languages and environments are excellent places to start if you are not familiar with any particular programming language as they all have their own syntax flavor. This paper uses Python and the package NetworkX for better control over the underlying computational resources [15]. While NetworkX is available for Sage, the flexibility of Python's multiprocessing capabilities was more useful than writing in Sage. The NetworkX package allows all of the required functions for graphs: reading and writing graphs, determining graph isomorphisms [9], and even the NP-complete problem of determining if a graph H is a subgraph of another graph G [8].

With the language and package selected, next is a short discussion on the many ways in which to store graphs on a computer. Easily accessible are markup-style formats (GEXF, GraphML), serialized-object-style formats (like JSON), and most notably the standalone formats GML and graph6/sparsegraph6. The down arrow generator utilizes both the graph6 and the GML formats for their specialized functionality.

The GML (Graph Modeling Language) format can store directed graphs by storing each vertex and each edge separately [16]. It also allows for the storing of vertex labels so long as they can be converted to an ACII string. This will be quite useful in the creation of the poset of a graph, because the node labels will be graphs themselves and the edges will be the "is a subgraph of" relation in the GML file.

The graph6/sparsegraph6 formats are quite similar, though for the types of graphs used in this paper the graph6 format was used as recommended by the documentation [16]. The graph6 format also includes a useful feature that not many other formats allow for, it can store multiple graphs within a single file, and NetworkX will read them in one by one. This uniquely qualifies the graph6 format for storing sets of graphs.

The way a graph G is converted into the graph6 representation is by encoding the upper triangle[1] of the graph's adjacency matrix into bytes and interpreting these as ASCII printable characters. This is how

[1]The upper triangle of a matrix being the entries of a matrix, read left to right, that are above the main diagonal.

a graph is stored as a node in the poset GML, but comes with the downside that only simple undirected graphs may be stored with no edge or vertex labels. Another complication to the graph6 format is that one graph may have multiple graph6 representations because of the ambiguity in selecting an ordering on $V(G)$ to generate the adjacency matrix. This is shown in the two different adjacency matrices Fig. 1.2a and Fig. 1.2b. The table in Fig. 1.2a correlates to the graph6 string: “JqWX@WC?OA_” while the table in Fig. 1.2b correlates to the graph6 string: “JqWWHW_?OA_”. Both of these graph6 strings decode to the Zim graph in Fig. 1.1.

The final observation is noticing that calculating $\downarrow G$ for a graph G is highly parallelizable. Because all of the possible colorings of G must be considered, each of these processes can be done simultaneously. In essence, the set of all colorings is broken up into as many pieces as the computer can process at once, and each of those pieces are processed simultaneously. Once the disparate pieces are analyzed a final pass to merge all the results is completed.

Due to the limitations of the Python global interpreter lock (GIL) [3], the multiprocessing package must be used to run these tasks concurrently. Unlike many languages, Python is both compiled and interpreted. If Python were simply compiled, then there would be no need for a lock of the interpreter because there would be no global interpreter required to execute individual instructions. The interpreter must be replicated for each process that is to be parallelized, which the multiprocessing package implements.

The Down-Arrow Problem

2.5 Statement

The down-arrow problem, as laid out by Byers and Olejniczak is as follows [6]: for a fixed graph G, determine all edge-induced subgraphs H of G for which $G \to H$. The set of all such H is given to be $\downarrow G$, and is called the down-arrow Ramsey set of a graph.

Definition 2.5.1. $\downarrow G := \{H \subseteq G : G \to H\}$

For a particular coloring $\mathcal{C}$ of G, the posets $\langle \mathcal{C}_r \rangle$ and $\langle \mathcal{C}_b \rangle$ contain all of (and perhaps more than) the elements in $\downarrow G$. This is formalized by Proposition 2.5.2.

Proposition 2.5.2. *For a graph G, $\downarrow G \subseteq \langle \mathcal{C}_r \rangle \cup \langle \mathcal{C}_b \rangle$ for a red-blue edge coloring $\mathcal{C}$.*

Consider the graph G in Fig. 2.9. Now consider the two red-blue edge colorings $\mathcal{C}^1$ and $\mathcal{C}^2$ in Fig. 2.9.

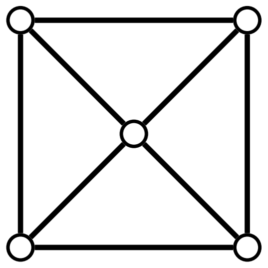
In the case of $\mathcal{C}^1$, calculating $\langle \mathcal{C}_r^1 \rangle \cup \langle \mathcal{C}_b^1 \rangle$ falls quickly since $\mathcal{C}_r \cong \mathcal{C}_b$, so $\langle \mathcal{C}_r^1 \rangle \cup \langle \mathcal{C}_b^1 \rangle = \langle \mathcal{C}_r^1 \rangle$. This extends to anytime where $\mathcal{C}_r \subseteq \mathcal{C}_b$ or vice versa, as $\langle \mathcal{C}_r \rangle \cup \langle \mathcal{C}_b \rangle = \langle \mathcal{C}_b \rangle$ or vice versa.

In the case of $\mathcal{C}^2$, neither $\mathcal{C}_r$ nor $\mathcal{C}_b$ is a subgraph of the other, hence the union of their posets must be listed out explicitly as

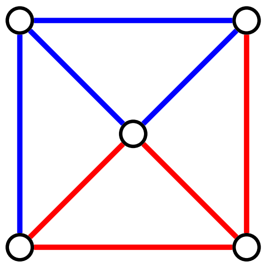
$$\langle \mathcal{C}_r^2 \rangle \cup \langle \mathcal{C}_b^2 \rangle = \{K_0, P_2, P_3, K_{1,3}, K_{1,4}\} \cup \{K_0, P_2, 2P_2, P_3, P_4, C_4\} = \{K_0, P_2, 2P_2, P_3, P_4, C_4, K_{1,3}, K_{1,4}\}.$$

But $\mathcal{C}^1$ does not present with neither a monochromatic C_4 nor does $\mathcal{C}^2$ present a monochromatic $K_{1,4}$! So both $\langle \mathcal{C}_r^1 \rangle \cup \langle \mathcal{C}_b^1 \rangle$ and $\langle \mathcal{C}_r^2 \rangle \cup \langle \mathcal{C}_b^2 \rangle$ have more edge-induced subgraphs than are truly present in $\downarrow G$.

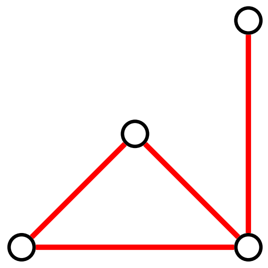
The finesse to the work done by Byers and Olejniczak was finding particular colorings of the fixed G that heavily restrict $\downarrow G$. In particular, the coloring $\mathcal{C}$ of K_6 shown in Fig. 2.10 is used to show that the $K_{2,3}$ is excluded in $\downarrow K_6$ [6]. Many of the colorings emphasized in Byers' and Olejniczak's work were either symmetric or extremal.



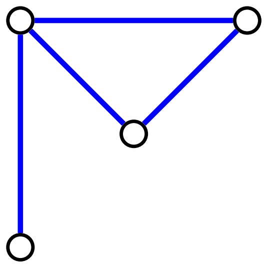
(a) G



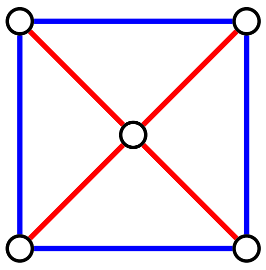
(b) $\mathcal{C}^1$



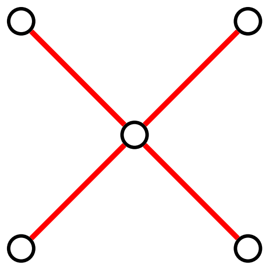
(c) $\mathcal{C}^1_r$



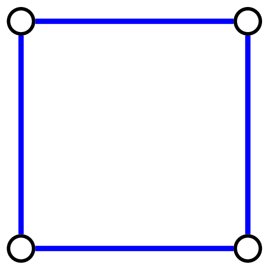
(d) $\mathcal{C}^1_b$



(e) $\mathcal{C}^2$

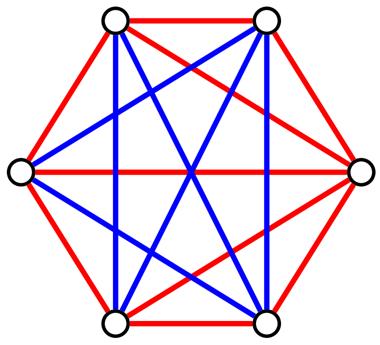


(f) $\mathcal{C}^2_r$

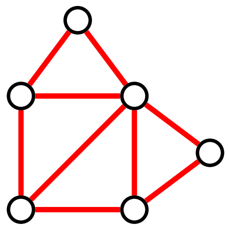


(g) $\mathcal{C}^2_b$

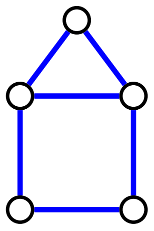
Figure 2.9: A graph G and two colorings $\mathcal{C}^1$ and $\mathcal{C}^2$.



(a) $\mathcal{C}$



(b) $\mathcal{C}_r$



(c) $\mathcal{C}_b$

Figure 2.10: A coloring $\mathcal{C}$ of K_6

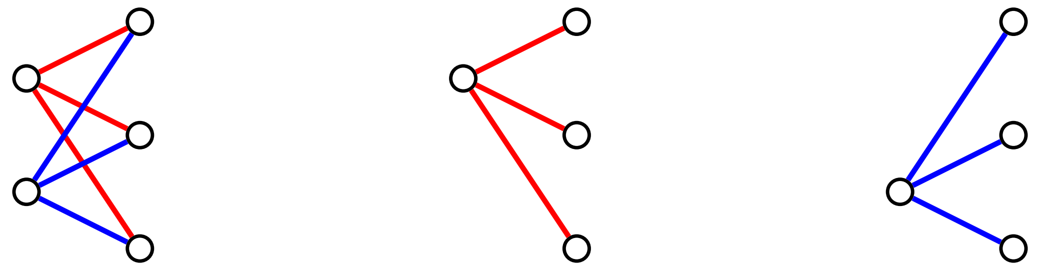

(a) $\mathcal{C}_1$ (b) $\mathcal{C}_r$ (c) $\mathcal{C}_b$

Figure 2.11: $\mathcal{C}_1$ of $K_{2,3}$

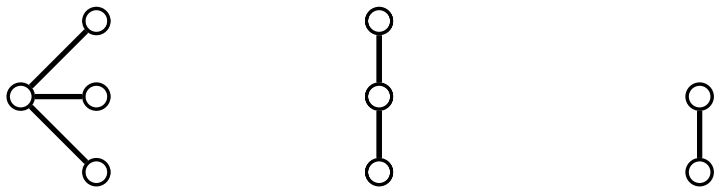

(a) $K_{1,3}$ (b) P_3 (c) P_2 (d) K_0

Figure 2.12: $\langle K_{1,3} \rangle$

When a coloring is symmetric, $\mathcal{C}_r \cong \mathcal{C}_b$. We see these kinds of colorings pop up in classical Ramsey numbers as well— as with Fig. 1.8 where $\mathcal{C}_r \cong \mathcal{C}_b \cong C_5$. With symmetric colorings like these, we have $\langle \mathcal{C}_r \rangle = \langle \mathcal{C}_b \rangle$, so $\downarrow G \subseteq \langle \mathcal{C}_r \rangle \cup \langle \mathcal{C}_b \rangle$ reduces quickly to $\downarrow G \subseteq \langle \mathcal{C}_r \rangle$. A similar reduction occurs when $\mathcal{C}_r \subseteq \mathcal{C}_b$ or vice versa, leading to the use of these types of colorings as well.

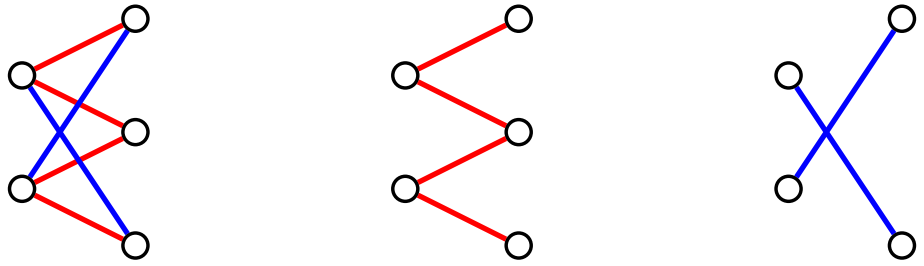
For extremal colorings, these are on a case-by-case basis because they are constructed so as to remove particular graphs from the down-arrow Ramsey set. Constructions such as these have the flavor of forbidden subgraph problems. Extremal colorings are like $\mathcal{C}^2$ in Fig. 2.9 where $\mathcal{C}_b$ and $\mathcal{C}_r$ are very different from each other.

For a small example, let us walk through determining $\downarrow K_{2,3}$.

Theorem 2.5.3. $\downarrow K_{2,3} = \langle P_3 \rangle$.

Proof. Consider the symmetric coloring $\mathcal{C}^1$ as in Fig. 2.11. In this case, we see that $\mathcal{C}^1_r \cong \mathcal{C}^1_b \cong K_{1,3}$. Hence $\langle \mathcal{C}^1_r \rangle \cup \langle \mathcal{C}^1_b \rangle = \langle K_{1,3} \rangle$, so $\downarrow K_{2,3} \subseteq \langle K_{1,3} \rangle$. Considering $\langle K_{1,3} \rangle$ in Fig. 2.12, we search for other colorings to determine if all 4 of these graphs are present in those as well. The coloring $\mathcal{C}^2$ in Fig. 2.13 does not present with a monochromatic $K_{1,3}$, but does present with monochromatic P_3, P_2, and K_0. Hence $K_{1,3} \not\in \downarrow K_{2,3}$, and $\downarrow K_{2,3} \subseteq \{P_3, P_2, K_0\}$. Now observe that $\{P_3, P_2, K_0\} = \langle P_3 \rangle$, so $\downarrow K_{2,3} \subseteq \langle P_3 \rangle$.

What remains to show is that every coloring $\mathcal{C}$ of $K_{2,3}$ must present some K_0, P_2, or P_3. Since $K_0 \subseteq P_2 \subseteq P_3$, we can simply show that every coloring $\mathcal{C}$ must present some P_3. We do this by a classical Ramsey-like argument. Consider $V(K_{2,3}) = \{u_1, u_2, v_1, v_2, v_3\}$ with the obvious bipartite sets. Now, let us inspect the



(a) $\mathcal{C}_2$ (b) $\mathcal{C}_r$ (c) $\mathcal{C}_b$

Figure 2.13: $\mathcal{C}_2$ of $K_{2,3}$

edges incident to u_1 in an arbitrary red-blue edge coloring $\mathcal{C}$. By the pigeonhole principle, u_1 is connected to two of $\{v_1, v_2, v_3\}$ by edges of the same color. Without loss of generality, u_1 is connected to v_1 and v_2 by edges of the same color. Hence $\mathcal{C}$ presents with a monochromatic P_3 since the path $v_1u_1v_2$ is monochromatic.

Thus $\downarrow K_{2,3} = \langle P_3 \rangle$. □

Theorem 2.5.3 concurs with the down-arrow Ramsey set of $K_{2,t}$, once observing that $P_3 \cong K_{1,2}$, presented by Byers and Olejniczak [5]:

$$\downarrow K_{2,t} = \langle K_{1,\lceil \frac{t}{2} \rceil} \rangle$$

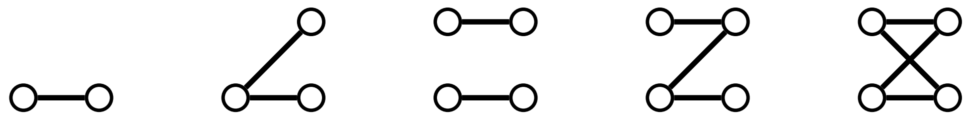
2.6 Exhaustive Procedure

To go about determining the down-arrow Ramsey set for more complicated graphs, we need to talk about what is going on under the hood of all examples up until now.

For a particular coloring $\mathcal{C}^1$, the posets $\langle \mathcal{C}^1_r \rangle$ and $\langle \mathcal{C}^1_b \rangle$ are combined into some superset of $\downarrow G$ as this represents all possible monochromatic subgraphs of G that are present under $\mathcal{C}^1$. We are leveraging set theory here by treating $\langle \mathcal{C}^1_r \rangle$ and $\langle \mathcal{C}^1_b \rangle$ as the underlying sets, and simply performing set union. Next, another coloring $\mathcal{C}^2$ is considered yielding another superset of $\downarrow G$, namely $\langle \mathcal{C}^2_r \rangle \cup \langle \mathcal{C}^2_b \rangle$. With both of these supersets containing $\downarrow G$, it must be that $\downarrow G$ resides in their intersection. From this perspective arises another formulation of the down-arrow Ramsey set definition in Theorem 2.6.1 [6].

Theorem 2.6.1. *For a graph G, $\downarrow G = \bigcap_{\mathcal{C}} \Big(\langle \mathcal{C}_r \rangle \cup \langle \mathcal{C}_b \rangle \Big)$ for all red-blue edge colorings $\mathcal{C}$.*

This formulation of the down-arrow Ramsey set yields insight into how to automate the calculation for an arbitrary graph G. If we were able to iteratively generate the unique red-blue edge colorings of G then we could calculate the down-arrow Ramsey set by way of Theorem 2.6.1. In fact, we have already discussed such an iterative procedure. We can induce all unique red-blue edge colorings of G by considering $\langle G \rangle$.



(a) H_0 (b) H_1 (c) H_2 (d) H_3 (e) H_4 (f) H_5

Figure 2.14: $\langle K_{2,2} \rangle$

Coloring $\mathcal{C}^i$	$\mathcal{C}_r^i$	$\mathcal{C}_b^i$	$\langle \mathcal{C}_r^i \rangle \cup \langle \mathcal{C}_b^i \rangle$
$\mathcal{C}^1$	H_0	H_5	$\langle H_5 \rangle$
$\mathcal{C}^2$	H_1	H_4	$\langle H_4 \rangle$
$\mathcal{C}^3$	H_2	H_2	$\langle H_2 \rangle$
$\mathcal{C}^4$	H_3	H_3	$\langle H_3 \rangle$
$\mathcal{C}^5$	H_4	H_1	$\langle H_4 \rangle$
$\mathcal{C}^6$	H_5	H_0	$\langle H_5 \rangle$

Figure 2.15: All red-blue edge colorings $\mathcal{C}$ of $K_{2,2}$

Theorem 2.6.2. *The set $C = \{\mathcal{C} : \exists H \in \langle G \rangle$ with $\mathcal{C}_r \cong H\}$ contains all unique red-blue edge colorings of a graph G.*

This falls directly from Definition 1.3.2. By way of example, let us calculate $\downarrow K_{2,2}$ in an exhaustive manner by way of Theorem 2.6.1 and Theorem 2.6.2.

Theorem 2.6.3. $\downarrow K_{2,2} = \langle P_2 \rangle$.

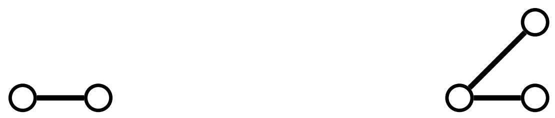
Proof. Fig. 2.14 shows $\langle K_{2,2} \rangle$, and more importantly it gives the basis for all possible colorings of $K_{2,2}$. Consider coloring each edge-induced subgraph of $K_{2,2}$ red, and consequently coloring the remaining edges blue. This is shown in Fig. 2.15, along with $\langle \mathcal{C}_r \rangle \cup \langle \mathcal{C}_b \rangle$ for each coloring. This constitutes all possible red-blue edge colorings of $K_{2,2}$. Consider each coloring C^i in turn:

1. Case $\mathcal{C}^1$: $\mathcal{C}_r^1$ is an edge-induced subgraph of $\mathcal{C}_b^1$, so $\langle \mathcal{C}_r^1 \rangle \cup \langle \mathcal{C}_b^1 \rangle = \langle \mathcal{C}_b^1 \rangle$
2. Case $\mathcal{C}^2$: $\mathcal{C}_r^2$ is an edge-induced subgraph of $\mathcal{C}_b^2$, so $\langle \mathcal{C}_r^2 \rangle \cup \langle \mathcal{C}_b^2 \rangle = \langle \mathcal{C}_b^2 \rangle$
3. Case $\mathcal{C}^3$: $\mathcal{C}_r^3 \cong \mathcal{C}_b^3$, so $\langle \mathcal{C}_r^3 \rangle \cup \langle \mathcal{C}_b^3 \rangle = \langle \mathcal{C}_r^3 \rangle$
4. Case $\mathcal{C}^4$: $\mathcal{C}_b^4 \cong \mathcal{C}_b^4$, so $\langle \mathcal{C}_r^4 \rangle \cup \langle \mathcal{C}_b^4 \rangle = \langle \mathcal{C}_r^4 \rangle$
5. Case $\mathcal{C}^5$: $\mathcal{C}_b^5$ is an edge-induced subgraph of $\mathcal{C}_r^5$, so $\langle \mathcal{C}_r^5 \rangle \cup \langle \mathcal{C}_b^5 \rangle = \langle \mathcal{C}_r^5 \rangle$
6. Case $\mathcal{C}^6$: $\mathcal{C}_b^6$ is an edge-induced subgraph of $\mathcal{C}_r^6$, so $\langle \mathcal{C}_r^6 \rangle \cup \langle \mathcal{C}_b^6 \rangle = \langle \mathcal{C}_r^6 \rangle$

Now, $\downarrow K_{2,2}$ is as follows:

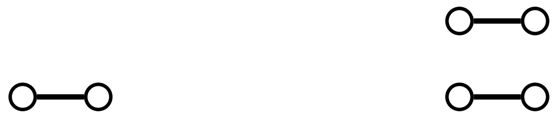
$$\begin{aligned}
\downarrow K_{2,2} &= \Big(\langle \mathcal{C}_r^1\rangle \cup \langle \mathcal{C}_b^1\rangle\Big) \cap \Big(\langle \mathcal{C}_r^2\rangle \cup \langle \mathcal{C}_b^2\rangle\Big) \cap \Big(\langle \mathcal{C}_r^3\rangle \cup \langle \mathcal{C}_b^3\rangle\Big) \cap \Big(\langle \mathcal{C}_r^4\rangle \cup \langle \mathcal{C}_b^4\rangle\Big) \quad \cap \Big(\langle \mathcal{C}_r^5\rangle \cup \langle \mathcal{C}_b^5\rangle\Big) \cap \Big(\langle \mathcal{C}_r^6\rangle \cup \langle \mathcal{C}_b^6\rangle\Big) \\
&= \Big(\langle H_5\rangle\Big) \cap \Big(\langle H_4\rangle\Big) \cap \Big(\langle H_2\rangle\Big) \cap \Big(\langle H_3\rangle\Big) \cap \Big(\langle H_4\rangle\Big) \cap \Big(\langle H_5\rangle\Big) \\
&= \Big(\langle H_5\rangle \cap \langle H_5\rangle\Big) \cap \Big(\langle H_4\rangle \cap \langle H_4\rangle\Big) \cap \Big(\langle H_2\rangle \cap \langle H_3\rangle\Big) && \text{by associativity} \\
&= \Big(\langle H_5\rangle\Big) \cap \Big(\langle H_4\rangle\Big) \cap \Big(\langle H_2\rangle\Big) \cap \Big(\langle H_3\rangle\Big) && \text{by identity} \\
&= \Big(\langle H_5\rangle \cap \langle H_4\rangle\Big) \cap \Big(\langle H_2\rangle\Big) \cap \Big(\langle H_3\rangle\Big) && \text{by associativity} \\
&= \Big(\langle H_4\rangle\Big) \cap \Big(\langle H_2\rangle\Big) \cap \Big(\langle H_3\rangle\Big) && \text{by } H_4 \subseteq H_5 \\
&= \Big(\langle H_4\rangle \cap \langle H_2\rangle\Big) \cap \Big(\langle H_3\rangle\Big) && \text{by associativity} \\
&= \Big(\langle H_2\rangle\Big) \cap \Big(\langle H_3\rangle\Big) && \text{by } H_2 \subseteq H_4
\end{aligned}$$

What is left is to calculate $\langle H_2\rangle \cap \langle H_3\rangle$. Figs. 2.16 and 2.17 show explicitly $\langle H_2\rangle$ and $\langle H_3\rangle$. By observation, $\langle H_2\rangle \cap \langle H_3\rangle = \{P_2, K_0\}$. Now observe that $\langle P_2\rangle = \{P_2, K_0\}$. Hence $\langle H_2\rangle \cap \langle H_3\rangle = \langle P_2\rangle$, and moreover $\downarrow K_{2,2} = \langle P_2\rangle$.



(a) K_0 (b) P_2 (c) $K_{1,2}$

Figure 2.16: $\langle H_2\rangle$



(a) K_0 (b) P_2 (c) $2P_2$

Figure 2.17: $\langle H_3\rangle$

□

This exhaustive procedure, taking each coloring of an input graph G, is what the down arrow generator uses to ensure that the correct down-arrow Ramsey set is returned.

The Down-Arrow Generator

At the highest level, the down arrow generator is precisely what is depicted in Algorithm 1. The nuances come in exactly how $\langle G \rangle$ is calculated. Generating the entire poset structure as a pre-processing step allows redundant work to be eliminated when $\langle H \rangle$ and $\langle \overline{H} \rangle$ are computed. Constructing the underlying set of edge-induced subgraphs of G is step one, and calls on the VF2 algorithm to determine graph isomorphisms. Identifying the "is an edge-induced subgraph" relationship is arguably the hardest step in generating $\langle G \rangle$ since it is NP-complete.

Algorithm 1 High-level overview of the down-arrow generator

```
1: procedure DOWN-ARROW GENERATOR(G)
2:     $D \leftarrow \langle G \rangle$                          ▷ Initialize the down-arrow set
3:     for $H \in \langle G \rangle$ do
4:         $S \leftarrow \langle H \rangle \cup \langle \overline{H} \rangle$      ▷ Calculate the union of $\langle \mathcal{C}_r \rangle$ and $\langle \mathcal{C}_b \rangle$
5:         $D \leftarrow S \cap D$              ▷ Remove the extraneous elements from $\downarrow G$
6:     end for
7:     return $D$
8: end procedure
```

3.7 Making the Subgraphs

The first primary function of the down-arrow generator is the make_subgraphs() function given in Fig. 3.18. For now, the important piece is that make_subgraphs() calls make_subgraph_part() function on line 6. This in turn calls the subgraph_generator() function shown in Fig. 3.19.

The key point of subgraph_generator() is to be a special type of function in Python called a generator. On line 4 of subgraph_generator(), the yield keyword is used so as to return the given value to make_subgraph_part() while letting it know that there will be more values returned when the generator is called again. This is a memory management optimization, so make_subgraph_part() only holds one item at a time in memory. For larger graphs, this is necessary as **all** edge-induced subgraphs are generated requiring a large amount of physical memory, not just the edge-induced subgraphs **up to isomorphism**.

```
1  def _make_subgraphs(Graph_name):
2      _make_graph_directory(Graph_name)
3      num_workers = max(1, multiprocessing.cpu_count()-1)
4      workers = []
5      for id in range(num_workers):
6          job = multiprocessing.Process(target=_make_part_subgraphs, args=(Graph_name,id,
       num_workers))
7          job.start()
8          workers.append(job)
9      for worker in workers:
10         worker.join()
11     workers = []
12     for id in range(num_workers):
13         job = multiprocessing.Process(target=_filter_subgraphs, args=(Graph_name,id,
       num_workers))
14         job.start()
15         workers.append(job)
16     for worker in workers:
17         worker.join()
18     _finish_subgraphs(Graph_name)
19     return
```

Figure 3.18: The make_subgraphs() function

```
def _subgraph_generator(Graph):
    for num_edges in range(Graph.number_of_edges()+1):
        for edges in it.combinations(Graph.edges(), num_edges):
            yield Graph.edge_subgraph(edges)
```

Figure 3.19: The subgraph_generator() function

It isn't until finish_subgraphs() on line 18 of make_subgraphs() shown in Fig. 3.18 that the subgraphs are filtered so as to only report the unique edge-induced subgraphs.

The make_subgraphs() function can be bypassed in the unique case of complete graphs. There is no innovative workaround however; the edge-induced subgraphs of K_n have already been generated by McKay [17] and is readily available in graph6 format. The data provided by McKay was used to construct the set of edge-induced subgraphs of K_n for $n \leq 11$.

3.8 Making the Poset

After generating the set of unique edge-induced subgraphs, the down-arrow generator calls the make_poset() function shown in Fig. 3.21. Because this is the first required step that is anticipated to take a long time for large graphs, the implementation of parallel computing methodologies is highly important. Lines 6 through 9 of make_poset() set up unique processes, each with their own Python interpreters, and task them with determining different parts of $\langle G \rangle$ by way of make_poset_part shown in Fig. 3.22.

Line 3 of make_poset_part() shows the use of the most important generator for the down-arrow generator: split_work(). This is a simple generator that returns, one at a time, every n-th element of an input generator

```python
def _finish_subgraphs(Graph_name):
    unique_subgraphs = []
    for file in os.listdir(f"Graphs/{Graph_name}/Parts/UniqueSubgraphs"):
        if file.endswith(".g6"):
            for subgraph in nx.read_graph6(f"Graphs/{Graph_name}/Parts/UniqueSubgraphs/{file}"):
                for unique_subgraph in unique_subgraphs:
                    if nx.is_isomorphic(subgraph, unique_subgraph):
                        break
                else:
                    unique_subgraphs.append(subgraph)
    with open(f"Graphs/{Graph_name}/{Graph_name}.Unique.Subgraphs.g6", "wb") as output_file:
        [output_file.write(nx.to_graph6_bytes(subgraph, header=False)) for subgraph in unique_subgraphs]
    return
```

Figure 3.20: The finish_subgraphs() function

```python
def _make_poset(Graph_name):
    if not os.path.exists(f"Graphs/{Graph_name}/{Graph_name}.Unique.Subgraphs.g6"):
        _make_subgraphs(Graph_name)
    num_workers = max(1, multiprocessing.cpu_count()-1)
    workers = []
    for id in range(num_workers):
        job = multiprocessing.Process(target=_make_part_poset, args=(Graph_name,id, num_workers))
        job.start()
        workers.append(job)
    for worker in workers:
        worker.join()
    _finish_poset(Graph_name)
    return
```

Figure 3.21: The make_poset() function

```python
def _make_poset_part(Graph_name, ID, Num_workers):
    poset_graph = nx.empty_graph(create_using=nx.DiGraph)
    for target in _split_work(nx.read_graph6(f"Graphs/{Graph_name}/{Graph_name}.Unique.Subgraphs.g6"), ID, Num_workers):
        for source in nx.read_graph6(f"Graphs/{Graph_name}/{Graph_name}.Unique.Subgraphs.g6"):
            if nx.algorithms.isomorphism.GraphMatcher(target, source).subgraph_is_monomorphic():
                poset_graph.add_edge(f"{nx.to_graph6_bytes(source, header=False).strip()}", f"{nx.to_graph6_bytes(target, header=False).strip()}")
    nx.write_gml(poset_graph, f"Graphs/{Graph_name}/Parts/Poset/Poset.Part.{ID}.gml")
    return
```

Figure 3.22: The make_poset_part() function

```
1 def _split_work(Generator, ID, Num_workers):
2     for job_number,job in enumerate(Generator):
3         if (job_number % Num_workers) == ID:
4             yield job
5         else:
6             continue
```

Figure 3.23: The split_work() function

by way of the modular arithmetic on line 3 of Fig. 3.23. This is used to allocate elements of the input generator in a round-robin manner to different processes.

Lastly we see how graphs can be used as vertex labels is used in the GML format on line 6 of make_poset_part(). The graph is first converted to its graph6 string, which is then converted into an ASCII string from the byte string. As per the NetworkX documentation and the graph6 format [15, 16], the newline character and the header must be omitted to store the raw graph6 string as the vertex label.

3.9 Making the Down-Arrow Ramsey Set

All the work the algorithm has done up to this point is to make the next function, make_down_arrow_set(), execute as fast as possible. In fact, this function takes on a similar order of runtime as make_subgraphs. This is due to leveraging the poset on lines 7 and 8 of make_down_arrow_part() shown in Fig. 3.24. Because the poset has been generated, constructing $\langle \mathcal{C}_r \rangle$ and $\langle \mathcal{C}_b \rangle$ are just a matter of finding the source vertices on edges that terminate at $\mathcal{C}_r$ and $\mathcal{C}_b$ respectively.

One point that was the cause of an early bug is seen on lines 4 and 15 of make_down_arrow_part(). To decode the graph6 string from the poset vertex label, one must recover any lost backslash characters that Python omitted when converting from a string of bytes to a string of ASCII characters. This is done with the “replace(b’////’,b’//’)” logic.

3.10 Reporting the Ideals

Once the down-arrow Ramsey set for an input graph is generated, the same mechanism to generate a poset is used, because ideals are generated by maximal elements in the poset. Finally, images are generated to show the down-arrow Ramsey set as well as the ideals.

At each part in the process, the down-arrow generator leaves all remnants of the generation process. This is helpful in restarting the process from power-loss because for graphs larger than K_6, runtime is expected to be more than an hour. In particular, $\downarrow K_7$ takes about an hour, $\downarrow K_8$ takes about 3 days, and $K_{5,5}$ takes about 4 days to determine.

```python
def _make_down_arrow_set_part(Graph_name, ID, Num_workers):
    down_arrow_set = None
    for red_subgraph_graph6_string in _split_work(_poset_iterator(Graph_name), ID, Num_workers):
        red_subgraph = nx.from_graph6_bytes(bytes(red_subgraph_graph6_string[2:len(red_subgraph_graph6_string)-1], "utf-8").replace(b"\\\\",b"\\"))
        blue_subgraph = _Complement(red_subgraph, _get_graph_from_name(Graph_name))
        blue_subgraph_graph6_string = _find_in_poset(nx.to_graph6_bytes(blue_subgraph, header=False).strip(), Graph_name)
        red_subgraphs = _subgraphs_of(red_subgraph_graph6_string, Graph_name)
        blue_subgraphs = _subgraphs_of(blue_subgraph_graph6_string, Graph_name)
        coloring_union = _union(red_subgraphs, blue_subgraphs)
        if down_arrow_set == None:
            down_arrow_set = coloring_union
        else:
            down_arrow_set = _intersection(down_arrow_set, coloring_union)
    if not down_arrow_set == None:
        down_arrow_set = [nx.from_graph6_bytes(bytes(subgraph_graph6_string[2:len(subgraph_graph6_string)-1], "utf-8").replace(b"\\\\",b"\\")) for subgraph_graph6_string in down_arrow_set]
        with open(f"Graphs/{Graph_name}/Parts/DownArrowSet/{Graph_name}.Down.Arrow.Set.Part.{ID}.g6", "wb") as output_file:
            [output_file.write(nx.to_graph6_bytes(subgraph, header=False)) for subgraph in down_arrow_set]
    return
```

Figure 3.24: The make_down_arrow_part() function

Results

In this chapter, specific results are given by the down arrow generator. This list provides the output as maximal ideals for many interesting input graphs. While paths, cycles, and complete bipartite graphs $K_{s,t}$ for $s = 1, 2$ have been completely classified [6], this chapter provides the foundation to begin developing generalized patterns. We expand on complete graphs, complete bipartite graphs $K_{s,t}$ for $s, t \leq 6$, and some favored miscellaneous graphs.

4.11 Complete Graph Results

For the complete graphs K_n with $1 \leq n \leq 8$, the down arrow generator concurs with Byers and Olejniczak [6]. For the down-arrow Ramsey set of complete graphs, this has direct application to graph-theoretic Ramsey theory. In particular, we need to define the diagonal graph Ramsey number. First, the *graph Ramsey number* for simple, undirected, connected graphs G and H, $R(G, H)$, is the minimum n where $K_n \to (G, H)$. Recall, this means to say that every red-blue edge coloring of K_n admits a red G or a blue H. When G and H are isomorphic, $R(G, H)$ is simply put $R(G, G)$ and denotes the *diagonal graph Ramsey number.* At this time, there are few diagonal graph Ramsey numbers known, as catalogued by [20].

By the nature of down-arrow Ramsey sets, $\downarrow K_n$ classifies all graphs for which any red-blue edge coloring admits a monochromatic subgraph. Hence the set $\downarrow K_n \setminus \downarrow K_m, \forall m < n$ classifies all graphs with diagonal graph Ramsey number n.

4.11.1 Down-Arrow Ramsey Sets of Complete Graphs

Listed below are first $\downarrow K_n$ for $1 \leq n \leq 9$ by way of reporting the poset of maximal ideals, then the entries that extend the diagonal graph Ramsey numbers in the Atlas of Graphs [20].

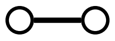


Figure 4.25: $\downarrow K_2 = \langle P_2 \rangle$

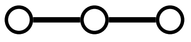


Figure 4.26: $\downarrow K_3 = \langle P_3 \rangle$

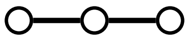


Figure 4.27: $\downarrow K_4 = \langle P_3 \rangle$

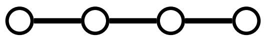
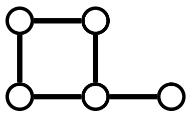


Figure 4.28: $\downarrow K_5 = \langle P_4 \rangle$


(a) H_1

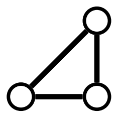
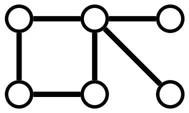
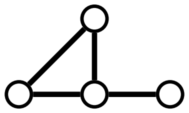

(b) H_2

Figure 4.29: $\downarrow K_6 = \langle H_1 \rangle \cup \langle H_2 \rangle$


(a) H_1


(b) H_2

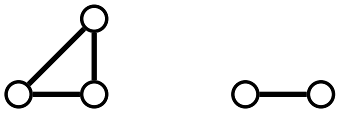
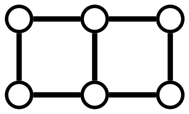
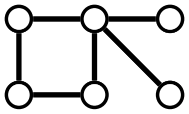
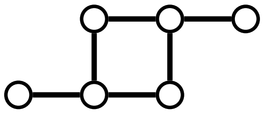
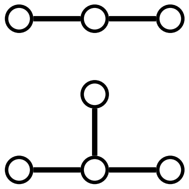

(c) H_3

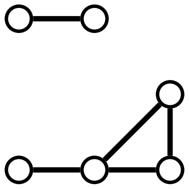
Figure 4.30: $\downarrow K_7 = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$


(a) H_1


(b) H_2


(c) H_3


(d) H_4


(e) H_5

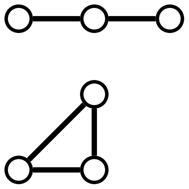

(f) H_6

Figure 4.31: $\downarrow K_8 = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle \cup \langle H_4 \rangle \cup \langle H_5 \rangle \cup \langle H_6 \rangle$

For the complete graph K_9, this is novel and undiscovered before the use of the down-arrow generator.

Figure 4.32: $\downarrow K_9 =$ still running

4.11.2 Graphs G with $R(G,G) \leq 9$

Listed below are all graphs with at least one edge, no isolated vertices, and diagonal graph Ramsey number less or equal to 9. These were determined by inspecting the down-arrow Ramsey sets of complete graphs. Instead of naming the graphs individually, each of the following graphs are identified with their assigned numbers as given by the graph atlas [20].

Graphs G with $R(G,G) = 2$:

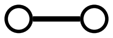


Figure 4.33: G3

Graphs G with $R(G,G) = 3$:

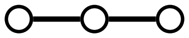
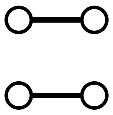
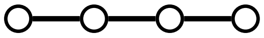


Figure 4.34: G6

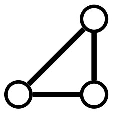
There are no graphs G with $R(G, G) = 4$.

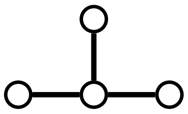
Graphs G with $R(G, G) = 5$:

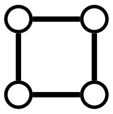

(a) G11

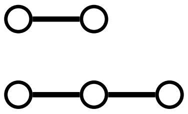

(b) G14

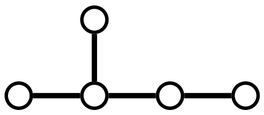
Graphs G with $R(G, G) = 6$:

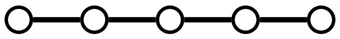

(a) G7

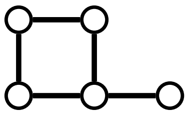

(b) G13


(c) G16

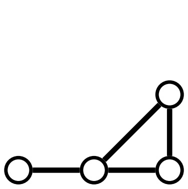

(d) G26

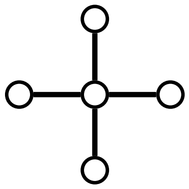

(e) G30

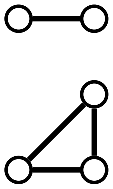

(f) G31

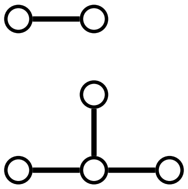

(g) G37

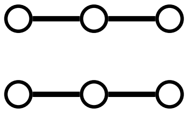
Graphs G with $R(G, G) = 7$:

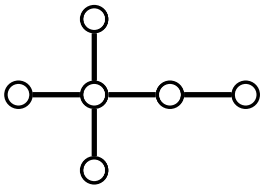

(a) G15

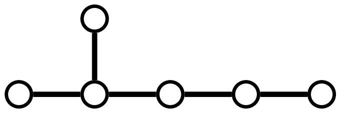

(b) G29

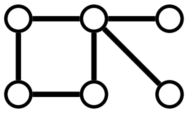

(c) G32


(d) G68

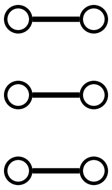

(e) G70

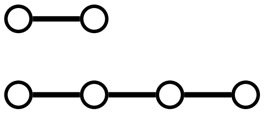

(f) G78

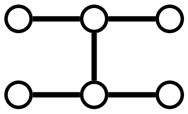

(g) G81

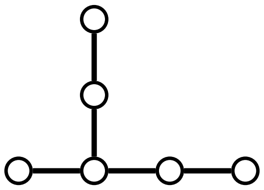

(h) G96

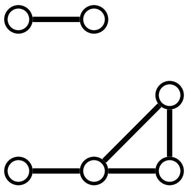
Graphs G with $R(G, G) = 8$:

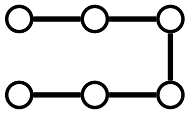

(a) G61

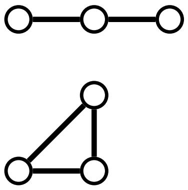

(b) G69

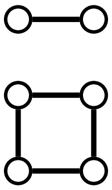

(c) G79

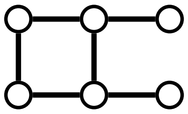

(d) G80

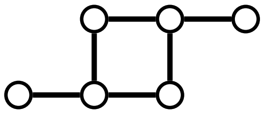

(e) G82

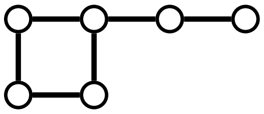

(f) G83

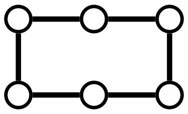

(g) G84

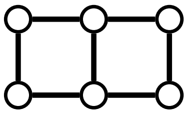

(h) G85

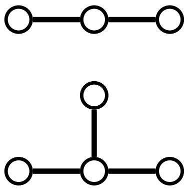

(i) G98


(j) G99


(k) G103

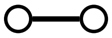

(l) G105

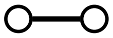

(m) G128

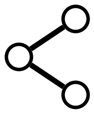

(n) G245

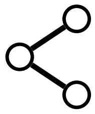
4.12 Complete Bipartite Graph Results

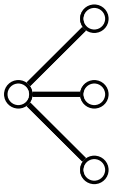
For the complete bipartite graphs $K_{1,t}$ and $K_{2,t}$ with $1 \leq t \leq 6$, the down arrow generator concurs with Byers and Olejniczak [5].


(a) $\downarrow K_{1,1} = \langle K_{1,1} \rangle$


(b) $\downarrow K_{1,2} = \langle K_{1,1} \rangle$


(c) $\downarrow K_{1,3} = \langle K_{1,2} \rangle$


(d) $\downarrow K_{1,4} = \langle K_{1,2} \rangle$


(e) $\downarrow K_{1,5} = \langle K_{1,3} \rangle$

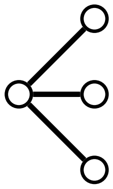
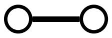
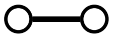
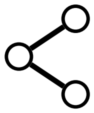
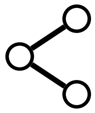

(f) $\downarrow K_{1,6} = \langle K_{1,3} \rangle$

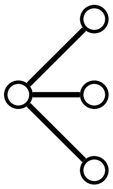
Figure 4.39: $\downarrow K_{1,t}$ for $t \leq 6$


(a) $\downarrow K_{2,1} = \langle K_{1,1} \rangle$


(b) $\downarrow K_{2,2} = \langle K_{1,1} \rangle$


(c) $\downarrow K_{2,3} = \langle K_{1,2} \rangle$


(d) $\downarrow K_{2,4} = \langle K_{1,2} \rangle$


(e) $\downarrow K_{2,5} = \langle K_{1,3} \rangle$

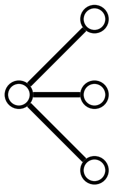

(f) $\downarrow K_{2,6} = \langle K_{1,3} \rangle$

Figure 4.40: $\downarrow K_{2,t}$ for $t \leq 6$

For the complete bipartite graphs $K_{3,t}$ and $K_{4,t}$ with $1 \leq t \leq 6$, the down arrow generator concurs with Olejnizcak's poster presented at the Fall Indiana MAA section meeting [19].

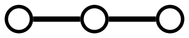


Figure 4.41: $\downarrow K_{3,1} = \langle K_{1,2} \rangle$

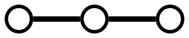
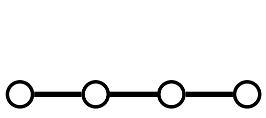


Figure 4.42: $\downarrow K_{3,2} = \langle K_{1,2} \rangle$


(a) H_1

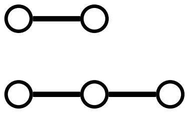
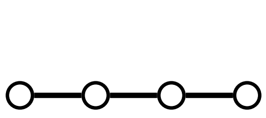

(b) H_2

Figure 4.43: $\downarrow K_{3,3} = \langle H_1 \rangle \cup \langle H_2 \rangle$


(a) H_1

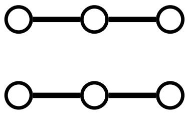
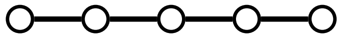
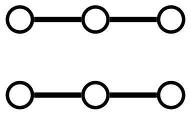

(b) H_2

Figure 4.44: $\downarrow K_{3,4} = \langle H_1 \rangle \cup \langle H_2 \rangle$


(a) H_1


(b) H_2

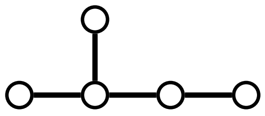
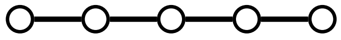
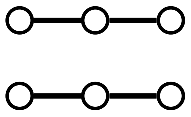

(c) H_3

Figure 4.45: $\downarrow K_{3,5} = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$


(a) H_1


(b) H_2

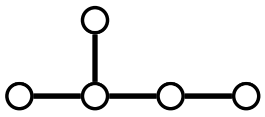

(c) H_3

Figure 4.46: $\downarrow K_{3,6} = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$

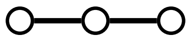


Figure 4.47: $\downarrow K_{4,1} = \langle K_{1,2} \rangle$

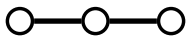
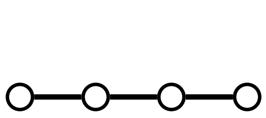


Figure 4.48: $\downarrow K_{4,2} = \langle K_{1,2} \rangle$


(a) H_1

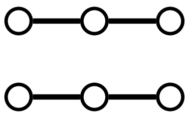
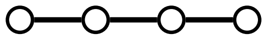

(b) H_2

Figure 4.49: $\downarrow K_{4,3} = \langle H_1 \rangle \cup \langle H_2 \rangle$


(a) H_1

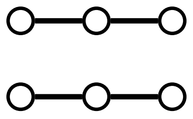
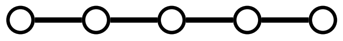
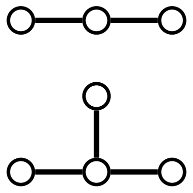

(b) H_2

Figure 4.50: $\downarrow K_{4,4} = \langle H_1 \rangle \cup \langle H_2 \rangle$


(a) H_1


(b) H_2

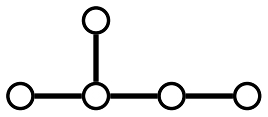

(c) H_3

Figure 4.51: $\downarrow K_{4,5} = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$

For the complete bipartite graphs $K_{5,t}$, these are novel and undiscovered before the use of the down arrow generator.

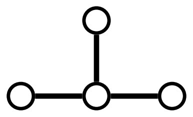


Figure 4.52: $\downarrow K_{5,1} = \langle K_{1,3} \rangle$

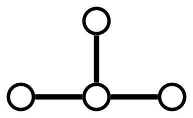
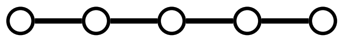
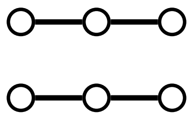


Figure 4.53: $\downarrow K_{5,2} = \langle K_{1,3} \rangle$


(a) H_1


(b) H_2

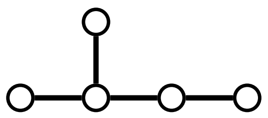
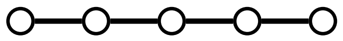
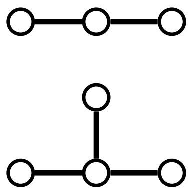

(c) H_3

Figure 4.54: $\downarrow K_{5,3} = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$


(a) H_1


(b) H_2

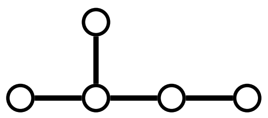
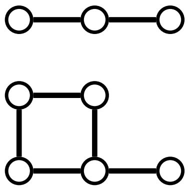
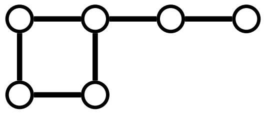

(c) H_3

Figure 4.55: $\downarrow K_{5,4} = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$


(a) H_1


(b) H_2

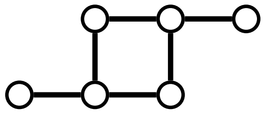

(c) H_3

Figure 4.56: $\downarrow K_{5,5} = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$

4.13 Complete k-partite graphs

Complete bipartite graphs are a special case of k-partite graphs. The down-arrow Ramsey set of complete k-partite graphs have yet to be investigated before the down-arrow generator. A complete k-partite graph is a graph G where $V(G)$ can be partitioned into k disjoint sets A_i each of size a_i such that $E(G)$ contains uv if and only if $u \in A_i$ and $v \notin A_i$. The notation $K_{a_1,a_2,\ldots,a_i}$ is used to denote complete k-partite graphs. Given below are the maximal ideals of the down-arrow Ramsey set for some complete k-partite graphs where $k \geq 3$.

Down-arrow Ramsey sets of complete 3-partite graphs:

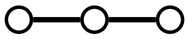


Figure 4.57: $\downarrow K_{1,1,1} = \langle P_3 \rangle$

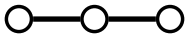


Figure 4.58: $\downarrow K_{1,1,2} = \langle P_3 \rangle$

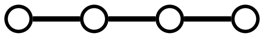


Figure 4.59: $\downarrow K_{1,2,2} = \langle P_4 \rangle$

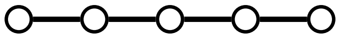


Figure 4.60: $\downarrow K_{2,2,2} = \langle P_5 \rangle$

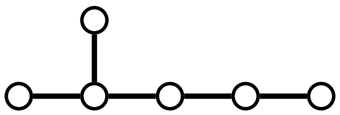


Figure 4.61: $\downarrow K_{2,2,3} = \langle \text{G81} \rangle$

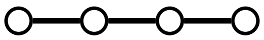


Figure 4.62: $\downarrow K_{1,2,2} = \langle P_4 \rangle$

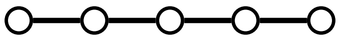
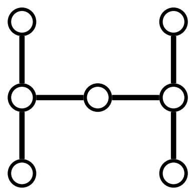
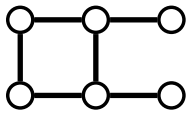
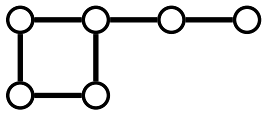


Figure 4.63: $\downarrow K_{2,2,2} = \langle P_5 \rangle$


(a) H_1


(b) H_2


(c) H_3

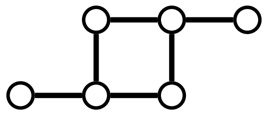

(d) H_4

Figure 4.64: $\downarrow K_{2,3,3} = \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle \cup \langle H_4 \rangle$

Down-arrow Ramsey sets of complete 4-partite graphs:

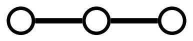


Figure 4.65: $\downarrow K_{1,1,1,1} = \langle P_3 \rangle$

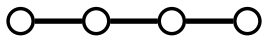
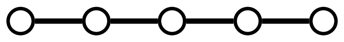


Figure 4.66: $\downarrow K_{1,1,1,2} = \langle P_4 \rangle$


(a) H_1

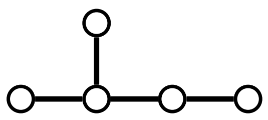

(b) H_2

Figure 4.67: $\downarrow K_{1,1,2,2} = \langle H_1 \rangle \cup \langle H_2 \rangle$

4.14 Friendship Graph results

A *friendship graph*, F_n, is a graph given by adjoining n copies of C_3 at a single shared vertex. The down-arrow Ramsey set of fan graphs have yet to be investigated before the down-arrow generator. Particular colorings of F_n for $2 \leq n \leq 8$ is given below in Fig. 4.68.

Theorem 4.14.1. $\downarrow F_n = \langle K_{1,n}\rangle \cup \langle \lceil \frac{n}{2} \rceil P_2\rangle$

Proof. Let the vertex adjoining the n copies of C_3 in F_n be called v.

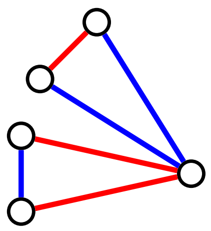
Consider F_n as component pieces: the edge-induced subgraph H_1 induced by edges incident to v, and the edge-induced subgraph H_2 induced by edges not incident to v. First observe that $H_1 \cong K_{1,2n}$, so $K_{1,2n} \subseteq F_n$ and hence $\downarrow K_{1,2n} = \langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle \subseteq \downarrow F_n$. Now observe that $H_2 \cong nP_2$, half of which ($\lceil \frac{n}{2} \rceil$ in the case that n is odd) must be colored the same. Hence $\langle \lceil \frac{n}{2} \rceil P_n\rangle \subseteq \downarrow F_n$. Combining both these ideas then yields $\langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle \cup \langle \lceil \frac{n}{2} \rceil P_2\rangle \subseteq \downarrow F_n$.

The following color scheme achieves $\downarrow F_n \subseteq \langle K_{1,n}\rangle \cup \langle \lceil \frac{n}{2} \rceil P_2\rangle$. Let $\mathcal{C}$ be given such that $\lceil \frac{n}{2} \rceil$ of the C_3 are colored with edges incident to v assigned red and the remaining blue. $\mathcal{C}$ then colors the remaining $\lfloor \frac{n}{2} \rfloor$ of the C_3 with edges incident to v assigned blue and the remaining red. This leaves $\mathcal{C}_r \cong K_{1,2\lceil \frac{n}{2}\rceil} \cup \lfloor \frac{n}{2} \rfloor P_2$ and $\mathcal{C}_b \cong K_{1,2\lfloor \frac{n}{2}\rfloor} \cup \lceil \frac{n}{2} \rceil P_2$. For $2 \leq n \leq 8$, this coloring $\mathcal{C}$ is given for F_n in Fig. 4.68. We then consider $\langle \mathcal{C}_r\rangle \cup \langle \mathcal{C}_b\rangle$ as follows:

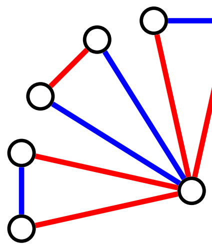
$$\begin{aligned}
\Big(\langle \mathcal{C}_r\rangle\Big) \cup \Big(\langle \mathcal{C}_b\rangle\Big) &= \Big(\langle K_{1,2\lceil \frac{n}{2}\rceil} \cup \lfloor \frac{n}{2} \rfloor P_2\rangle\Big) \cup \Big(\langle K_{1,2\lfloor \frac{n}{2}\rfloor} \cup \lceil \frac{n}{2} \rceil P_2\rangle\Big) \\
&= \Big(\langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle\Big) \cup \Big(\langle \lfloor \frac{n}{2} \rfloor P_2\rangle\Big) \cup \Big(\langle K_{1,2\lfloor \frac{n}{2}\rfloor}\rangle\Big) \cup \Big(\langle \lceil \frac{n}{2} \rceil P_2\rangle\Big) \\
&= \Big(\langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle \cup \langle K_{1,2\lfloor \frac{n}{2}\rfloor}\rangle\Big) \cup \Big(\langle \lfloor \frac{n}{2} \rfloor P_2\rangle\Big) \cup \Big(\langle \lceil \frac{n}{2} \rceil P_2\rangle\Big) && \text{by associativity} \\
&= \Big(\langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle\Big) \cup \Big(\langle \lfloor \frac{n}{2} \rfloor P_2\rangle\Big) \cup \Big(\langle \lceil \frac{n}{2} \rceil P_2\rangle\Big) && \text{by } K_{1,2\lfloor \frac{n}{2}\rfloor} \subseteq K_{1,2\lceil \frac{n}{2}\rceil} \\
&= \Big(\langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle\Big) \cup \Big(\langle \lfloor \frac{n}{2} \rfloor P_2\rangle \cup \langle \lceil \frac{n}{2} \rceil P_2\rangle\Big) && \text{by associativity} \\
&= \Big(\langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle\Big) \cup \Big(\langle \lceil \frac{n}{2} \rceil P_2\rangle\Big) && \text{by } \lfloor \frac{n}{2} \rfloor P_2 \subseteq \lceil \frac{n}{2} \rceil P_2
\end{aligned}$$

Hence $\downarrow F_n \subseteq \langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle \cup \langle \lceil \frac{n}{2} \rceil P_2\rangle$.

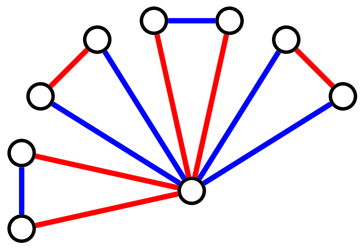
With both halves of set inclusion, we may conclude $\downarrow F_n = \langle K_{1,2\lceil \frac{n}{2}\rceil}\rangle \cup \langle \lceil \frac{n}{2} \rceil P_2\rangle$. ☐



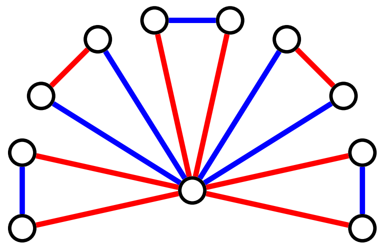
(a) F_2



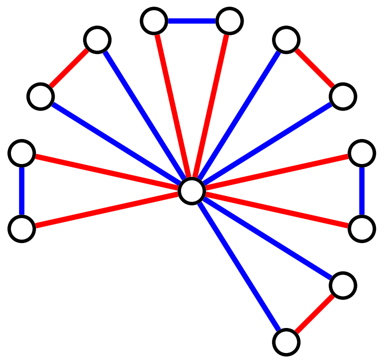
(b) F_3



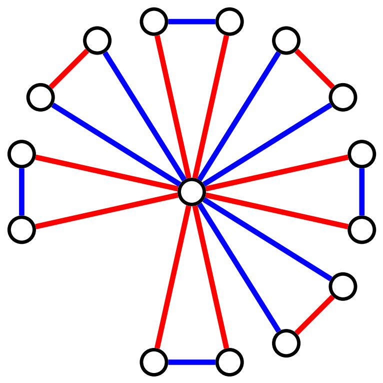
(c) F_4



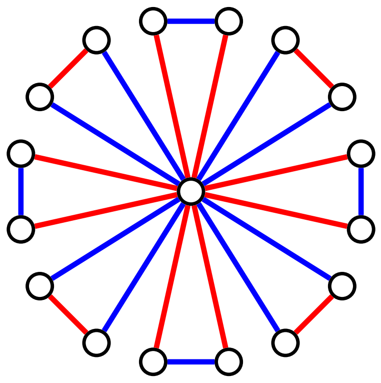
(d) F_5



(e) F_6



(f) F_7



(g) F_8)

Figure 4.68: Red-blue edge colorings of F_n for $2 \leq n \leq 8$.

4.15 Miscellaneous Graphs

The down-arrow Ramsey set of the following graphs have yet to be investigated before the down-arrow generator. The primary reason for selecting these particular graphs primarily comes from previous work I have done, or by one-off questions I have received during my work on this paper.

The Zim graph was created for the use in power domination, however it has become a kind of favorite toy graph of mine.

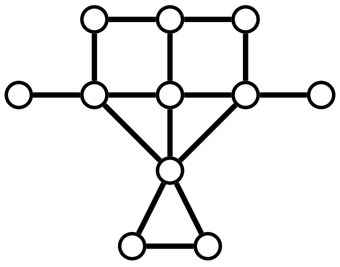
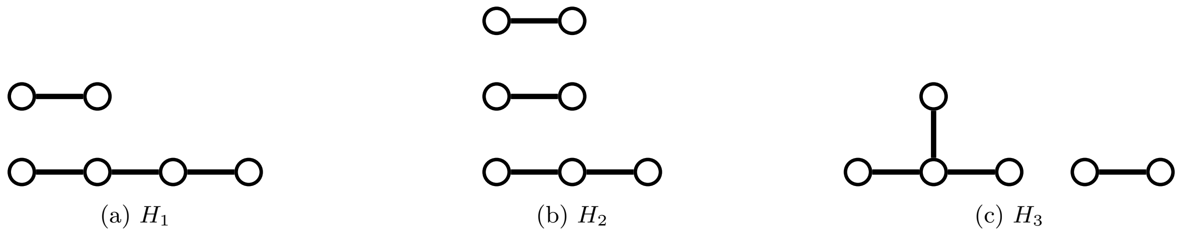


Figure 4.69: The Zim graph

The down-arrow Ramsey set of the Zim graph is given below.



(a) H_1 (b) H_2 (c) H_3

Figure 4.70: $\downarrow$ Zim graph $= \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$

K_4 snake was the focus of a year-long project of mine in π-Harmonious labelings that started my graph theoretic research path.

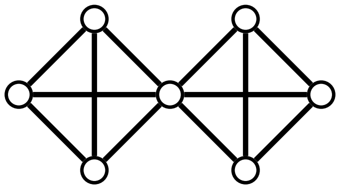
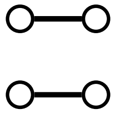


Figure 4.71: K_4 Snake

The down-arrow Ramsey set of K_4 snake is given below.


(a) G11

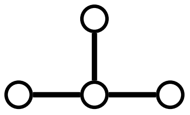

(b) G13

Figure 4.72: $\downarrow K_4$ snake $= \langle \text{G11} \rangle \cup \langle \text{G13} \rangle$

The Petersen graph is named after Julius Petersen, and is used as a counterexample for many graph properties.

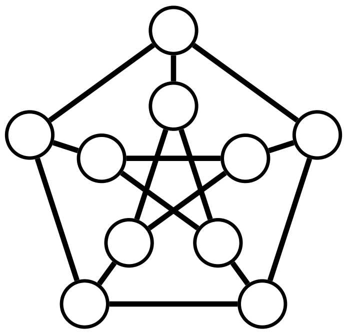
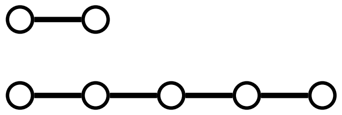
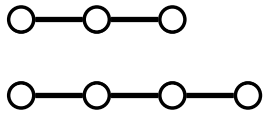


Figure 4.73: The Petersen graph

The down-arrow Ramsey set of the Petersen graph is given below.


(a) H_1


(b) H_2

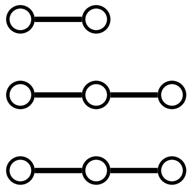

(c) H_3

Figure 4.74: $\downarrow$ Petersen graph $= \langle H_1 \rangle \cup \langle H_2 \rangle \cup \langle H_3 \rangle$

The Barioli-Fallat tree is named after Francesco Barioli and Shaun Fallat, and was used as a "spectacular counterexample" for the spectral arbitrariness of trees [12]. In power domination and zero-forcing, this graph has come up as a counterexample for many conjectures.

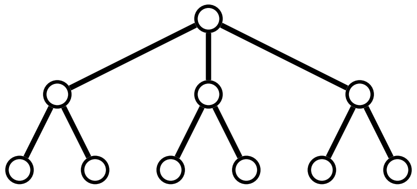


Figure 4.75: The Barioli-Fallat tree

The down-arrow Ramsey set of the Barioli-Fallat tree is given below.

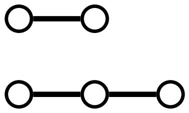


Figure 4.76: $\downarrow$ Barioli-Fallat tree = $\langle$G26$\rangle$

Conclusion

5.16 Limitations and Use Cases for the Down Arrow Generator

The calculations given by the down arrow generator are just that, calculations. They serve as the guiding light to determining the down-arrow Ramsey set of a given graph. What remains for the mathematician to do is to:

1. Prove that nothing outside of the reported down-arrow Ramsey set exists by locating red-blue edge colorings of G that exclude larger graphs.

2. Prove that each of the reported maximal graph ideals are in every red-blue edge coloring of G.

To generate the results laid out in this paper, the down arrow generator was left to run on a dedicated machine for over a month. If high performance computers are leveraged with a large amount of time dedicated to the search, many more down-arrow Ramsey sets could be generated. The down-arrow Ramsey sets of complete graphs are likely the easiest to compute because of the data set that supplies all of the unique edge-induced subgraphs of K_n for $n \leq 11$, which gives a stepping stone for the down arrow generator.

5.17 Future work

Current versions of the down arrow generator use more and more system memory as the input graph grows. In fact, it requires very nearly 128 gigabytes of system RAM to inspect and report the down-arrow Ramsey set of K_8, and much more than that to inspect the structure of K_9. Future efforts will be placed into a dynamic memory management routine so systematically cache lists to system storage. A preliminary version of this style of algorithm is available on my GitHub page: `https://github.com/JibJibFlutterhousen/Down-Arrow-Ramsey-Sets`.

While brute-force calculations are elegant calculations in-and-of-themselves, we must only use them as calculations and not proofs lest we fall into the trap that Tymoczko describes in the four color problem [23].

Further algorithms can be added to inspect each of the individual colorings to generate a list of graphs that forbid certain graphs from the down-arrow Ramsey set, as in one-half of the proof in Theorem 2.5.3.

Bibliography

[1] Kenneth Appel and Wolfgang Haken. “Every planar map is four colorable. Part I: Discharging”. In: *Illinois Journal of Mathematics* 21 (1977), pp. 429–490.

[2] Kenneth Appel, Wolfgang Haken, and Jürgen Koch. “Every planar map is four colorable. Part II: Reducibility”. In: *Illinois Journal of Mathematics* 21 (1977), pp. 491–567.

[3] David Beazley. “Understanding the Python GIL”. In: *PyCon US 2010*. URL: `https://archive.org/details/pyvideo_353___understanding-the-python-gil-82`. 2010.

[4] Stefan A. Burr. “On the Computational Complexity of Ramsey—Type Problems”. In: *Mathematics of Ramsey Theory*. Ed. by Jaroslav Neetil and Vojtch Rödl. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 46–52. ISBN: 978-3-642-72905-8. DOI: `10.1007/978-3-642-72905-8_5`. URL: `https://doi.org/10.1007/978-3-642-72905-8_5`.

[5] Alexis Byers and Drake Olejniczak. “The Down Arrow Ramsey Set”. In: *33rd Midwest Conference on Combinatorics and Combinatorial Computing*. 2019.

[6] Alexis Byers and Drake Olejniczak. “The Down-Arrow Ramsey Set of a Graph”. In: *Journal of Combinatorial Mathematics and Combinatorial Computing* (2020).

[7] G. Chartrand, L. Lesniak, and P. Zhang. *Graphs & Digraphs*. Discrete Mathematics and Its Applications Series. CRC Press, Taylor & Francis Group, 2016. ISBN: 9781498735766. URL: `https://books.google.com/books?id=vkQwjgEACAAJ`.

[8] Stephen A. Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the third annual ACM symposium on Theory of computing - STOC ’71*. ACM Press, 1971. DOI: `10.1145/800157.805047`.

[9] L.P. Cordella et al. “A (sub)graph isomorphism algorithm for matching large graphs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.10 (2004), pp. 1367–1372. DOI: `10.1109/TPAMI.2004.75`.

[10] Gabor Csardi and Tamas Nepusz. *The igraph software package for complex network research.* 2006. URL: `https://igraph.org`.

[11] Reinhard Diestel. *Graph Theory - Graduate Texts in Mathematics.* Springer-Diestel, Jan. 2016. 444 pp. ISBN: 3961340056. URL: `https://www.ebook.de/de/product/26866671/reinhard_diestel_graph_theory_graduate_texts_in_mathematics.html`.

[12] Shaun M. Fallat et al. *Spectral arbitrariness for trees fails spectacularly.* 2023. DOI: `10.48550/ARXIV.2301.11073`. URL: `https://arxiv.org/abs/2301.11073`.

[13] R. Graham and S. Butler. *Rudiments of Ramsey Theory.* CBMS Regional Conference Series in Mathematics. Conference Board of the Mathematical Sciences, 2015. ISBN: 9780821841563. URL: `https://books.google.com/books?id=WpelCgAAQBAJ`.

[14] R.L. Graham, B.L. Rothschild, and J.H. Spencer. *Ramsey Theory.* Wiley Series in Discrete Mathematics and Optimization. Wiley, 1991. ISBN: 9780471500469. URL: `https://books.google.com/books?id=55oXT60dC54C`.

[15] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference.* Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[16] Michael Himsolt. *GML: A portable Graph File Format.* url: `http://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf`.

[17] Brendan McKay. "Graphs". url: `http://users.cecs.anu.edu.au/~bdm/data/graphs.html`.

[18] Dimitrios Michail et al. "JGraphT–A Java Library for Graph Data Structures and Algorithms". In: *ACM Trans. Math. Softw.* 46.2 (May 2020).

[19] Drake Olejniczak. "The down-arrow Ramsey set of a graph". In: Fall MAA Indiana Section 2022. 2022.

[20] Ronald C. Read and Robin J. Wilson. *An Atlas of Graphs.* Oxford University Press, USA, 2005, p. 466. ISBN: 9780198526506.

[21] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.1).* `https://www.sagemath.org`. 2023.

[22] William Stein. "SAGE Days 4". In: URL: `https://web.archive.org/web/20070627235122/http://www.sagemath.org/why/stein-sd4.pdf`. 2007.

[23] Thomas Tymoczko. "The Four-Color Problem and Its Philosophical Significance". In: *The Journal of Philosophy* 76.2 (Feb. 1979), p. 57. DOI: `10.2307/2025976`.