3-Colorings of Diameter 2 Graphs

Dylan McKellips

May 2025

Abstract

Coloring graphs is a known NP-Complete problem. However, for certain families of graphs (such as triangle free planar graphs, bipartite graphs, or graphs with maximum degree 3 or less) this problem is solvable in deterministic polynomial time. Due to the strong structural requirements of diameter 2 graphs, we are interested in whether diameter 2 graphs are such a family. We explore various approaches to this problem that utilize a diameter 2 graphs unique structure.

1 Introduction

The question of whether diameter 2 graphs are 3-colorable in polynomial time has a well known open problem in graph theory. Diameter 2 graphs are graphs where each pair of vertices are either neighbors, or share a common neighbor. While graph coloring in general is an NP-complete problem, subexponential time colorings, polynomial time approximations, and polynomial time algorithms on certain classes (or graphs with restricted subgraphs) are known. In this paper, we explore a number of approaches to this difficult open problem. We outline methods for coloring diameter 2 graphs and provide incremental results towards faster 3 coloring for diameter 2 graphs.

In section 2, we provide background on graph theory, spectral theory, and overview some interesting results. In section 3, we overview historic spectral coloring algorithms, and provide additional proofs and modifications. In section 4, we overview seed based coloring, providing bounds and exploring results related to dominating sets in diameter 2 graphs. Finally, in section 5, we overview families of diameter 2 graphs.

2 Spectral Background

2.1 Basics

A graph $G = (V, E)$ is a set of vertices $v_1, v_2 \cdots \subset V$ and edges $(v_i, v_j) \in E$ between them. We may write edges as $v_i v_j$. An undirected graph is a graph such that $(v_i, v_j) \in E \implies (v_j, v_i) \in E$. A simple graph has that we do not

have $(v_i, v_j) \in E$ if $i = j$, and we have at most one edge between any pair of vertices i, j. A finite graph has $|V| < \infty$. For the remainder of the paper, when we say *graph* we refer to a *finite, simple, undirected* graph.

Spectral graph theory studies the relation between a graph and the eigenvalues and eigenvectors corresponding to the various matrices associated with a graph. An *eigenvalue* λ and associated *eigenvector* v with respect to a matrix M have the property that $Mv = \lambda v$. The spectrum of matrices associated with a graph can reveal properties that would otherwise be difficult to obtain.

We explore some spectral properties of various matrices associated with graphs, including the adjacency matrix, Laplacian, edge Laplacian, and signless Laplacian. The spectrum can be an unnatural device, and this section seeks to establish familiarity with the spectrum and possible ways to use it.

Matrices with respect to a graph G will have a subscript denoting this. If context if obvious this subscript will be omitted. First, we define common matrices associated with graphs.

Definition 1. *The adjacency matrix A has entry $a_{ij} = 1$ if there is an edge between vertices v_i, v_j and 0 otherwise.*

Definition 2. *The degree matrix D has entry $d_{ij} = deg(v_i)$ if $i = j$, and 0 otherwise.*

Definition 3. *The vertex-edge incidence matrix R of a graph G is $n \times m$, with entry $r_{ij} = 1$ if vertex v_i is an endpoint of edge e_j, and 0 otherwise.*

Definition 4. *Laplacian L is defined as A-D.*

The Laplacian is as commonly used as the adjacency matrix, if not more so, in spectral graph theory. It is famously used to determine algebraic connectivity, Cheeger's bound (for bottlenecks and sparse cuts), mixing time, among uses. A question to check your understand is: when do the adjacency matrix and the Laplacian provide the same spectral information? Other matrices introduced, such as the signless Laplacian or line graph, are less commonly used. Next, we explore some basic properties of graph associated matrices. The following definition is an often useful property of a matrix, with applications in optimization and other fields.

Definition 5. *We say a symmetric $n \times n$ matrix A is positive semidefinite if $x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$.*

This definition leads to the first result exemplifying the usefulness of the spectrum, as the following result can be approximately calculated in polynomial time by a computer, whereas manually testing all $x \in \mathbb{R}^n$ cannot.

Lemma 2.1. *A symmetric matrix is positive semidefinite if and only if all eigenvalues are nonnegative.*

Lemma 2.2. *The Laplacian is positive semidefinite .*

Proof. We have

$$\begin{aligned} x^T L x &= x^T \left(\sum_{(i,j)\in E} (e_i - e_j)(e_i - e_j)^T \right) x \\ &= \sum_{(i,j)\in E} x^T (e_i - e_j)(e_i - e_j)^T x \\ &= \sum_{(i,j)\in E} (x_i - x_j)^2. \end{aligned}$$

□

Definition 6. *The signless Laplacian $|L|$ is defined as $A+D$. Note this matrix is also positive semidefinite.*

Lemma 2.3. *If R is the vertex-edge incidence matrix of a graph G, then $RR^T = A + D$, and $R^T R = A_{L(G)} + 2I_{|E|}$.*

Proof. The first follows from observing that rows multiplied with a transpose of itself will count all edges adjacent to a vertex (which is its degree), and rows multiplied with transposes of other rows will yield a 1 if they share an edge, and 0 if not. Since we are dealing with simple graphs, we will not count more than one edge.

The second follows similarly, by noting that each edge connects exactly 2 vertices, so the "degree matrix" is $2I_{|E|}$. □

Next, we investigate further spectral objects relating to the preceding matrices, and results relating to them. The characteristic polynomial is a useful tool to represent the spectrum. We also explore notions of equivalence under the spectrum.

Definition 7. *The characteristic polynomial of the adjacency matrix of a graph G is denoted by $P_G(\lambda) = det(\lambda I - A)$.*

Definition 8. *The characteristic polynomial of the signless Laplacian of a graph G is denoted by $Q_G(\lambda) = det(\lambda I - |L|)$.*

Definition 9. *Two graphs are called* cospectral *if they have the same spectrum.*

Definition 10. *Two graphs are called* Q-spectral *if they have the same polynomial $Q(\lambda)$.*

Definition 11. *For a graph G, we define the line (dual) graph $L(G)$ is the adjacency graph of edges for G. Explicitly, we have $V(L(G)) = E(G)$, and vertices in $L(G)$ are adjacent if their corresponding edges in G share an endpoint.*

Definition 12. *Two graphs are called* L-spectral *if their line graphs are cospectral.*

Lemma 2.4. *For a matrix M and its characteristic polynomial P, we have $P_{M+cI}(\lambda) = P_M(\lambda + c)$.*

Proof. Note that $(\det((M + cI) - \lambda I) = \det(M + (c - \lambda)I)$, then complete a change of variable. □

Lemma 2.5. *Let $B \in \mathbb{R}^{n\times m}$, with $n \leq m$ and $\operatorname{rank}(B) = r$. Then,*

$$\det(\lambda I_m - B^\top B) = \lambda^{m-n} \det(\lambda I_n - BB^\top)$$

Proof. Let $M = B^\top B \in \mathbb{R}^{m\times m}$, and $N = BB^\top \in \mathbb{R}^{n\times n}$. We have that $\operatorname{rank}(M) = \operatorname{rank}(N) = r$. Observe that for any vector $x \in \mathbb{R}^m$,

$$x^\top Mx = x^\top B^\top Bx = (Bx)^\top (Bx) = \|Bx\|^2 \geq 0,$$

so M is positive semidefinite. Similarly, N is positive semidefinite.

Now, suppose $\lambda \neq 0$ is an eigenvalue of M with eigenvector $x \in \mathbb{R}^m$. Then:

$$B^\top Bx = \lambda x.$$

Apply B to both sides:

$$BB^\top (Bx) = \lambda(Bx).$$

So either $Bx = 0$, which would imply $\lambda = 0$, a contradiction, or $Bx \neq 0$, and Bx is an eigenvector of N with eigenvalue λ. A similar argument shows that every nonzero eigenvalue of N is also an eigenvalue of M.

Then, let the nonzero eigenvalues of $B^\top B$ and $BB^\top$ be $\mu_1, \ldots, \mu_r$. We have:

$$\det(\lambda I_m - B^\top B) = \prod_{i=1}^{r} (\lambda - \mu_i) \cdot \lambda^{m-r}$$

$$\det(\lambda I_n - BB^\top) = \prod_{i=1}^{r} (\lambda - \mu_i) \cdot \lambda^{n-r}.$$

Therefore,

$$\det(\lambda I_m - B^\top B) = \lambda^{m-n} \det(\lambda I_n - BB^\top).$$

□

Lemma 2.6. *[Cvetkovic et al. [CRS96]] $P_{L(G)}(\lambda) = (\lambda+2)^{m-n} Q_G(\lambda+2)$, and if Spec(G) $= \lambda_1, \ldots, \lambda_t$, then the eigenvalues of L(G) are $\theta_i = \lambda_i - 2$ $i \in [t]$, $\theta_i = -2$ else.*

Proof. This follows from 2.3 and 2.5. □

Theorem 2.7 (Cvetkovic et al. [CRS96])**.** *Q-spectral implies L-spectral.*

Proof. If graphs are $Q-$spectral, then they have the same number of vertices and edges. Then, L-spectrality follows from 2.6. □

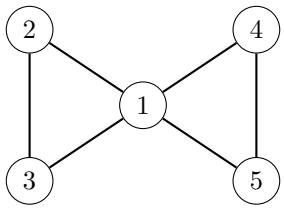

Figure 1: Butterfly Graph

However, note that we can find graphs that share a line graph Laplacian that are not $Q-$spectral. To see this, consider $K_4 - e \cup K_2$ and the butterfly graph 1 with an additional disconnected vertex. Then, both line graphs have characteristic polynomial $\lambda(\lambda^2 - \lambda - 4)(\lambda - 1)(\lambda + 1)^2$, however the first has Q polynomial $\lambda(\lambda - 1)(\lambda - 2)(\lambda - 3)(\lambda^2 - 5\lambda + 2)$, while the latter has $\lambda^2(\lambda - 1)(\lambda - 2)(\lambda - 3)(\lambda^2 - 5\lambda + 2)$.

Lemma 2.8. *An independent set in $L(G)$ corresponds to a matching in G.*

Proof. By selecting edges in a matching, we ensure that these edges are adjacent to no other edges. Thus, these edges are not connected in the line graph. □

Lemma 2.9. *For a graph G, $|E| = -\frac{p_1}{2}$ where p_1 is the coefficient of λ^{n-1} in Q_G.*

Proof. The coefficient of λ^{n-1} is equal to $-tr(|L|)$, and the trace of $|L|$ is equal to the sum of vertex degrees of G. This is equal to double the number of edges, so we have $p_1 = -tr(|L|) = -2|E|$. □

The following definition is another useful tool for interacting with the spectrum, used in many proofs. We present it alongside some results we can deduce by utilizing it.

Definition 13. *The Rayleigh quotient of a symmetric matrix A is the normalized quadratic form, expressed*

$$\frac{x^T A x}{x^T x}$$

By maximizing or minimizing over nonzero x, we can find the largest and smallest eigenvalues of A.

Theorem 2.10 (Cvetkovic et al. [CRS96])**.** *Let H be G after switching edges ab, cd to non edges ad, bc. Let x be a principal eigenvector of G. If $(x_a - x_c)(x_b - x_d) \geq 0$. then $\lambda_1^H \geq \lambda_1^G$, with equality iff both of the two products are 0.*

Proof. First, note that

$$\lambda_1^G = \sup_{x \in R^n - \{0\}} \frac{x^T Q^G x}{x^T x} = \max_{\|x\|=1} x^T Q x$$

, then if we change edges we have, if x is the eigenvector corresponding to λ_1^G, that

$$\lambda_1^H - \lambda_1^G = \max_{\|y\|=1} y^T Q^H y - x^T Q^G x \geq x^T (A^H - A^G)x + x^T (D^H - D^G)x$$

With equality if and only if x is also the principle eigenvector for Q^H. Then, from this we obtain

$$\lambda_1^H - \lambda_1^G \geq 2(x_a - x_c)(x_b - x_d)$$

and our result follows. □

Theorem 2.11. *Let λ_1 be the largest eigenvalue of A. Then,*

$$\lambda_1 \geq \frac{2m}{n}$$

Proof. Consider that $\lambda_1 = \max_{x \in R^n} \frac{x^T A x}{x^T x}$, and let $x = 1_n$. Then, we count each edge twice in the numerator, yielding $2m$ and the denominator will be n summations of 1, so we have our desired lower bound. □

Theorem 2.12. *Let D be the distance matrix, $\bar{A}$ the complement and J of all ones. Then,*

$$D^2 = 4(J - I)\bar{A} + A^2$$
$$D = 2(J - I) + A.$$

2.2 Hamiltonian Cycle

A *Hamiltonian cycle* is a path on a graph that starts at a vertex and traverses edges of the graph, visiting each vertex exactly once, finishing on the vertex it started on, and visiting each edge no more than once. Finding a Hamiltonian cycle on an arbitrary graph is known to be an NP-complete problem [Kar72]. In this section, we explore a result by Veldman, where they prove that the Line graph of a diameter 2 graph is Hamiltonian. In general, solving the Hamiltonian cycle problem is NP-complete, so this is the first instance where the diameter 2 property allots us more power than arbitrary graphs.

Lemma 2.13. *Given a star graph $K_{1,s}$, the line graph $L(K_{1,s})$ has a Hamiltonian cycle.*

Proof. Since all edges are adjacent to the central vertex (and thus each other), we have that $L(K_{1,s}) = K_s$. □

Theorem 2.14. *For all integers $n \geq 4$, the line graph of the complete graph K_n has a Hamiltonian cycle.*

Proof. Assign the vertices some arbitrary order. Then, traverse all edges of v_1 in any order such that we traverse the edge connected v_1 to v_2 last. Then, traverse the edges of v_2 (excluding the one we already traversed earlier) similarly, traversing the edge connecting v_2 to v_3 last. Repeat this until we have traversed the entire graph in this manner, then return from v_n to v_1. Then, we traversed no edge twice, and have touched every edge. □

Definition 14. *A circuit C is a walk in which all edges are distinct, and the first and last vertices are equal. A $D-$circuit is a circuit in which every edge of G is incident with at least one vertex in the $D-$circuit.*

Definition 15. *If C is a circuit and Z a cycle of G such that $V(Z) \cap V(C) \neq \emptyset \neq V(Z) \cap (V(G) - V(C))$ and $G[E(C) \triangle E(Z)]$ (where $\triangle$ denotes symmetric difference) is connected, then Z is called a C-augmenting cycle. Clearly, if C is a circuit and Z is a C-augmenting cycle, then $G[E(C) \triangle E(Z)]$ is also a circuit, and $|V(G[E(C) \triangle E(Z)])| > |V(C)|$, implying that if C is a maximal circuit, we have no $C-$augmenting cycle.*

Definition 16. *Define τ to be P_3 with two new non-adjacent vertices adjacent to one of the end vertices. Similarly, define τ^+ to be P_3 with two new adjacent vertices adjacent to one of the end vertices.*

Lemma 2.15 (Veldman [Vel88])**.** *Let G be a graph and C a maximal circuit. Then, there is no cycle Z with*

$$V(Z) \cap V(Z) \neq \emptyset \neq V(Z) \cap (V(G) - V(C)) \wedge |E(z) \cap E(C)| \leq 1$$

Definition 17. *A block of a graph is a maximal connected subgraph with no cut-vertices, that is no vertices whose removal would disconnect the graph. For example, all complete graphs of size 3 or larger are blocks, while no paths of length 3 or longer are.*

Lemma 2.16 (Veldman [Vel88])**.** *If $x_1, \ldots, x_k$ is a path in G, and for $1 \leq i < j < k$ if we have $x_i x_{i+1}, x_j x_{j+1}$ in the same block B of G, then $x_m x_{m+1} \in E(B) \forall i < m < j$.*

Definition 18. *The neighborhood of a set of vertices $S \subset V$ is*

$$N(S) = \{v_i \in V/S \;\Big|\; e_{ij} \in E, v_j \in V\}.$$

The closed neighborhood of a set of vertices $S \subset V$ is

$$N[S] = \{v_i \in V/S \;\Big|\; e_{ij} \in E, v_j \in V\} \cup S.$$

Theorem 2.17 (Veldman [Vel88])**.** *Let G be a connected graph that is not a tree such that every subgraph isomorphic to τ or τ^+ with $d(a_i) \geq 2, i \in \{1, 2\}$ satisfies at least one of the following:*

1. $|N(a) \cap N(c)| \geq 2$

2. $|N(b) \cap N(c) \geq 1|$

3. *For* $i \in \{1,2\}, |N(b) \cap N(a_i)| \geq \begin{cases} 2 & a_1a_2 \in E(G) \\ 3 & else \end{cases}$

4. *For* $i \in \{1,2\}, |N(a) \cap N(b)| \geq 1$

Then L(G) is Hamiltonian

Proof. Let G satisfy our initial conditions, but assume $L(G)$ is non-Hamiltonian. Let C be a maximal circuit of G. Then C is not a D-circuit of G, as by Harary and Nash-Williams G would otherwise have Hamiltonian L(G). Thus there exists a path u_1uu_2 with $u_1, u_2 \notin V(C)$ and $u \in V(C)$. Let $vv_1, vv_2 \in E(C)$. Then, as a result of 2.15 it follows that some subgraph $H \leq G$ with vertices $\{u_1, u_2, u, u_1, u_2\}$ is isomorphic to τ or τ^+, and $(N(u_1) \cap N(u)) - \{u_2\} = N(u_2) \cap N(u) = \emptyset$. . We must have $d_C(u_1) \geq d_C(u_2) \geq 2$ $(i = 1, 2)$, so we know that H satisfies (3) or (4).

First, assume that H satisfies (3). Then, assume $H = \tau^+$, so $v_1v_2 \in E(G)$. Assume without loss of generality that u_2 and v_1 have a common neighbor w with $w \neq u$. By 2.15, $v_1w \in E(C)$. If $v_1v_2 \in E(C)$, then vu_2wv_1v is a C-augmenting cycle, a contradiction with 15. If $v_1v_2 \in E(G) - E(C)$, then $vu_2wv_1v_2v$ is a C-augmenting cycle, which contradicts that C is maximal.

Thus, we can only have that $u_1u_2 \notin E(G)$, so $H = \tau$. Assume without loss of generality that v_1 and u_2 have common neighbors not equal to v, label these w_1 and w_2. By 2.15, $v_1w_1, v_1w_2 \in E(C)$. If v_1v, v_1w_1, and v_1w_2 are in the same block of C, then $C - \{v_1w_1, v_1w_2\}$ is connected, implying that $u_2w_1v_1w_2u_2$ is a C-augmenting cycle, contradicting 15. If v_1v and v_1w_1 are in different blocks of C, then $C - \{v_1v, v_1w_1\}$ is connected (since every block of C is 2-edge-connected), so $vu_2w_1v_1v$ is a C-augmenting cycle, which contradicts that C is maximal.

So H cannot satisfy (3), and instead must satisfy (4).

Call a path P *special* if it satisfies the following requirements:

- P has origin v,
- $E(P) \subseteq E(C)$,
- each block of C contains at most one edge of P, and
- u_1 and the terminus of P have a common neighbor.

Note that, if P is a special path, then, by the third requirement, $C - E(P)$ is connected.

Since H_1 satisfies (4), G contains a special path of length 1. Let P be a special path of maximum length, x the terminus of P, y the immediate predecessor of x on P, and z a common neighbor of u_1 and x.Then , we know $z \notin V(P)$, otherwise G contains the C-augmenting cycle $Q_1 \cup vu_2u_1z$, where Q_1 denotes the (v, z)-subpath of P. Also, $z \neq u_2$, otherwise $P \cup vu_2x$ is a C-augmenting cycle.

Furthermore, xz is an edge of C, otherwise the cycle Z_1, with $Z_1 = P \cup uu_2u_1zx$ is a C-augmenting cycle. Moreover, the edges xy and xz are in the same block of C; assuming the contrary, by 2.16, all edges of $E(P) \cup \{xz\}$ are in different blocks of C, again yielding the contradiction that Z_1 is a C-augmenting cycle. This contradiction is avoided only if $\{xy, xz\}$ is a 2-edge cut of C. Thus, either $d_C(x) = 2$ or x is a cut vertex of C. If $d_C(x) = 2$, then $G[E(C) \triangle E(Z_1)]$ consists of a trivial component and a component that is a circuit; the latter circuit contains one vertex more than C, contradicting the maximality of C. Thus x is a cut vertex of C.

Let B be a block of C containing x and different from the block that contains xy (and xz). Then, by 2.16, B differs from all blocks of C that contain an edge of P. Let xx_1 and xx_2 be two edges of B and let $H_1 = G[\{u_1, z, x, x_1, x_2\}]$. By 2.15, $u_1x \notin E(G)$. Also, $u_1x_i \notin E(G)$, otherwise $P \cup vu_2u_1x_ix$ is a C-augmenting cycle ($i = 1, 2$). Since P is a longest special path, $zx_i \notin E(G)$ ($i = 1, 2$), so H_1 is isomorphic to τ or τ^+. Since $d_G(x_i), d_C(x_i) \geq 2$ ($i = 1, 2$), H_1 satisfies one of our requirements. We finish by showing that each case yields a contradiction.

First suppose H_1 satisfies (1). Let $z_1 \in (N(u_1) \cap N(x)) - \{z\}$. As in the last paragraph, we have $z_1 \notin V(P) \cup \{u_2\}$, xz_1 is an edge of C, and xz_1 and xy are in the same block of C. Since xz is also in this block, $C - (E(P) \cup \{xz\})$ is connected, and Z is a C augmenting cycle, which is a contradiction.

Now suppose H_1 satisfies (2). Let $y_1 \in N(z) \cap N(x)$. As a result of 2.15, $y_1 \notin \{u_1, u_2\}$. If $y_1 \in V(P)$, then $Q_2 \cup vu_2u_1zy_1$, with Q_2 as the (u, y_1)-subpath of P, is a C-augmenting cycle, whether $zy_1 \in E(C)$ or not. Thus we must have $y_1 \notin V(P)$. If xy_1 and zy_1 are edges of C, then Z_1 is a C-augmenting cycle, else $P \cup vu_2u_1zy_1x$ forms a contradiction.

Next suppose H_1 satisfies (3), with $b = z$ and $x_1, x_2 = a_1, a_2$. Let $x_3 \in (N(z) \cap N(x_1)) - \{x\}$. Earlier arguments imply that x_3 cannot be a vertex in $V(P) \cup \{u_1, u_2\}$. We consider all possible cases for the membership of x_1x_3, x_3z with respect to C. If both x_1x_3 and x_3z are edges of C, then Z_1 is a C-augmenting cycle. If both x_1x_3 and x_3z are in $E(G) - E(C)$, then the cycle Z_1, with $Z_1 = P \cup vu_2u_1zx_3x_1x$ is a C-augmenting cycle. Assume $x_1x_3 \in E(G) - E(C)$ and $x_3z \in E(C)$. By 2.16, x_3z is not an edge of the block B of C containing x_1, so, x_3z is not a cut edge of the connected subgraph $(C + x_1x_3) - (E(P) \cup \{xx_1\})$ of G, since $zx \cup Q_3 \cup x_1x_3$ (Q_3 is an (x, x_1)-path in $B - xx_1$) is a (z, x_1)-path in this subgraph. This implies Z_1 is a C-augmenting cycle. Finally, Now assume $x_1x_3 \in E(C)$ and $x_3z \in E(G) - E(C)$. We finally distinguish two cases.

First, if $x_1x_2 \in E(G)$, then $x_3 \neq x_2$, otherwise H_1 would satisfy (2), which we already proved generates a contradiction. If $x_1x_2 \in E(C)$, then Z_1 is a C-augmenting cycle. If $x_1x_2 \in E(G) - E(C)$, then $P \cup uu_2u_1zx_3x_1x_2x$ is a C-augmenting cycle, so we must have no edge x_1x_2.

Thus, we must have that $x_1x_2 \notin E(G)$. Consider some $x_4 \in (N(z) \cap N(x_1)) - \{x, x_3\}$. Similar to the consideration for x_3, we may assume $x_4 \notin V(P) \cup \{u_1, u_2\}$, $x_1x_4 \in E(C)$, and $x_4z \in E(G) - E(C)$. If both x_1x_3 and x_1x_4 are edges of B, then $B - \{xx_1, x_1x_3\}$ is connected and Z_1 is a C-augmenting cycle.

Else, with $x_1x_4 \notin E(B)$, then by 2.16 all edges of $E(P) \cup \{xx_1, x_1x_4\}$ are in different blocks of C, and hence $P \cup vu_2u_1zx_4x_1x$ is a C-augmenting cycle.

Finally, suppose H_1 satisfies (iv). Assume without loss of generality that $N(u_1) \cap N(x_1) \neq \emptyset$. Then $P \cup xx_1$ is a special path longer than P, our final contradiction.

□

Theorem 2.18 (Veldman [Vel88])**.** *If G is diameter 2 with at least 4 vertices, L(G) is Hamiltonian*

Proof. If G has diameter 1, then G is complete, so L(G) is Hamiltonian by 2.14. Else, every induced subgraph isomorphic to τ or τ^+ satisfies 2.17, so either L(G) is Hamiltonian or G is a tree. If G is a tree, then G is isomorphic to the star graph, and L(G) is Hamiltonian by 2.13. □

2.3 Removals and Interlacing

When considering graphs, we are often interested in removing or adding edges or vertices in order to influence the diameter of the graph. The following theorem gives insight into how vertex removals impact the spectrum of the graph.

Theorem 2.19 (Cauchy, Poincare)**.** *Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric. Let $B \in \mathbb{R}^{m\times m}$ with $m < n$ be a principal submatrix (obtained by deleting both the i-th row and i-th column for some i, $n-m$ times). Suppose A has eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$ and B has eigenvalues $\mu_1 \leq \cdots \leq \mu_m$. Then:*

$$\lambda_k \leq \mu_k \leq \lambda_{k+n-m} \quad \text{for } k = 1, \ldots, m.$$

Proof. Williamson [WD16]

Without loss of generality, assume

$$A = \begin{bmatrix} B & X^T \\ X & Z \end{bmatrix}.$$

Let $\{x_1, \ldots, x_n\}$ be the eigenvectors of A, and $\{y_1, \ldots, y_m\}$ the eigenvectors of B.

Define the following subspaces:

$$V = \text{span}(x_k, \ldots, x_n), \quad W = \text{span}(y_1, \ldots, y_k),$$

$$W' = \left\{ \begin{bmatrix} w \\ 0 \end{bmatrix} \in \mathbb{R}^n \,\middle|\, w \in W \right\}.$$

Since $\dim(V) = n-k+1$ and $\dim(W') = \dim(W) = k$, there exists $w' \in V \cap W'$, and $w' = \begin{bmatrix} w \\ 0 \end{bmatrix}$ for some $w \in W$. Thus,

$$w'^T A w' = \begin{bmatrix} w^T & 0 \end{bmatrix} \begin{bmatrix} B & X^T \\ X & Z \end{bmatrix} \begin{bmatrix} w \\ 0 \end{bmatrix} = w^T B w.$$

Since we have $\lambda_k = \min_{x \in V} \frac{x^T A x}{x^T x}, \quad \beta_k = \max_{x \in W} \frac{x^T B x}{x^T x}$, we can conclude

$$\lambda_k \leq \frac{w'^T A w'}{w'^T w'} = \frac{w^T B w}{w^T w} \leq \beta_k.$$

To prove the second inequality, define:

$$V = \text{span}(x_1, \ldots, x_{k+n-m}), \quad W = \text{span}(y_k, \ldots, y_m),$$

$$W' = \left\{ \begin{bmatrix} w \\ 0 \end{bmatrix} \in \mathbb{R}^n \,\middle|\, w \in W \right\}.$$

Then $\dim(V) = k+n-m$ and $\dim(W') = m-k+1$, so there exists $w' \in V \cap W'$ and $w' = \begin{bmatrix} w \\ 0 \end{bmatrix}$ for some $w \in W$. We again compute $w'^T A w' = w^T B w$, therefore,

$$\lambda_{k+n-m} = \max_{x \in V} \frac{x^T A x}{x^T x} \geq \frac{w'^T A w'}{w'^T w'} = \frac{w^T B w}{w^T w} \geq \min_{x \in W} \frac{x^T B x}{x^T x} = \beta_k.$$

□

3 Spectral Graph Partitioning

With a familiarity in spectral graph theory in hand (for additional reference consult [CDS95] or [Spi25]), we now look to 3-color graphs using the spectrum. We provide the meaning of this statement below, then explore algorithms for achieving this end.

Definition 19. *We say a $k-$coloring is a function $\phi : V \to \{0, \ldots, k\}$. A* proper *coloring is one such that if $vw \in E$, we have $\phi(v) \neq \phi(w)$. Informally, this means that any two vertices that share an edge cannot be the same color. Henceforth, when we say coloring we mean proper coloring.*

Definition 20. *An independent set is a set of vertices $S \subset V$ such that no vertices in S share an edge. Note that partitioning a graph into independent sets is equivalent to coloring it.*

3.1 Aspvall and Gilbert

We look to use the spectrum to color graphs, or, equivalently, partition a graph into independent sets. Aspvall and Gilbert [AG84] provided an algorithm that will partition a graph into independent sets using eigenvectors of the adjacency matrix. It is easy to see that we can do this using all eigenvalues, as this will partition the graph into independent sets of size one. They conjecture that the algorithm will work for arbitrary graphs when considering only the negative eigenvalues, and furthermore Aspvall and Gilbert prove that taking $k-1$ negative eigenvalues of the adjacency matrix will partition a block k regular graph. They do not extend this to general graphs

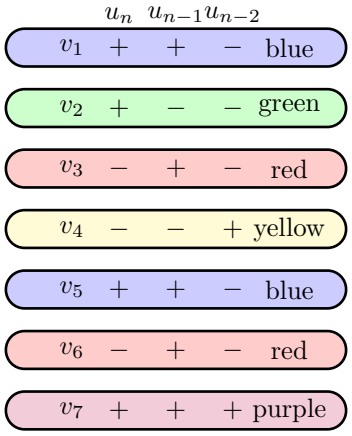

Figure 2: Sign Partitioning

Definition 21. *We say a sign coloring is a partition assigned by eigenvectors, where vertices are the same color if their sign with respect to a subset of eigenvalues is the same. Let 0 be positive.*

For example, consider the following partition of the first 7 vertices of a graph by the smallest 3 eigenvalues. We note that v_1, v_5 have the same sign with respect to these eigenvalues, and so they are in the same partition.

Definition 22. *We say a block regular graph is a graph with a partition such that each vertex in a partition, or block, has the same degree as all other vertices in its block. A block k regular graph has k such blocks.*

Theorem 3.1 (Aspvall and Gilbert [AG84])**.** *We can partition a graph into independent sets using the sign pattern of its eigenvalues.*

Proof. Perron and Frobenius state that for (connected) graphs, the first eigenvector is all positive. We can choose an orthonormal basis for the eigenvectors. If we have multiplicity, we take a linear combination of associated eigenvectors. Call the matrix with columns as eigenvectors U.Then, the rows are the signs. Since the first element in each row is positive, we must have at least one pair of row entries of differing signs in order for their inner product to be 0.

□

Lemma 3.2. *Reordering vertices preserves the spectrum of the adjacency, Laplacian, signless Laplacian, and line graph.*

Proof. Consider some matrix M, and a permutation matrix P that reorders the rows and columns of M into $M' = PMP^T$. Note that P is an orthogonal matrix, so we have $P^T = P^{-1}$, so we have $M' = PMP^{-1}$. Thus, M' and M are similar, and so they have the same eigenvalues. Then, we can have M be any associated matrix with a graph G, and see that reordering matrices is equivalent to applying a permutation matrix PMP^T, and the result follows. □

Definition 23. *A graph B is block regular if for each block, the row and column sums are constant. A tripartite block regular graph has 3 such blocks, where vertices within blocks do not share edges.*

Definition 24. *The degree graph β of a block regular tripartite graph has b_{ij} the row some of block ij and b_{ji} the column sum.*

Theorem 3.3 (Aspvall and Gilbert [AG84])**.** *Let G be a block regular 3- partite graph. Then there is a set of 2 eigenvectors whose sign patterns properly partition vertices of the graphs.*

Proof. Let G be tripartite with partition V_1, V_2, V_3 with each part having size r, s, t respectively. Then, let

$$A = \begin{bmatrix} 0 & A_{12} & A_{13} \\ A_{12}^T & 0 & A_{23} \\ A_{13}^T & A_{23}^T & 0 \end{bmatrix} B = \begin{bmatrix} 0 & b_{12} & b_{13} \\ b_{21} & 0 & b_{23} \\ b_{31} & b_{32} & 0 \end{bmatrix}$$

Where A is the adjacency matrix of G, and B is the block degree matrix. Observe that these share eigenvalues, up to multiplications by 1_i to scale the block degree eigenvector to match the dimension of A (and vice versa), that is $(\alpha 1, \beta 1, \gamma 1)^T$ is an eigenvector of A with eigenvalue λ if and only if (α, β, γ) is an eigenvector of B with eigenvalue λ.

Note that $rb_{12} = sb_{21}, rb_{13} = tb_{31}, and sb_{23} = tb_{32}$, then let D be a diagonal matrix with entries $\sqrt{r}, \sqrt{s}, \sqrt{t}$. Then, let $B' = DBD^{-1}$, which is symmetric and nonnegative with zero diagonal entries, and furthermore B and B' share eigenvalues, with identical sign patterns. These then partition the rows of B' into singletons, and thus B and A. □

3.2 Negative Eigenvalue Partition

For general graphs, Aspvall and Gilbert suggest the following algorithm: repeatedly select and eigenvalue and use the sign of its components (let zero be positive) to refine the coloring so that all vertices with the same signs across multiple eigenvectors will be in the same color class. Aspvall and Gilbert conjecture that this algorithm will find a coloring after only considering negative eigenvalues. This algorithm does not provide the needed eigenvectors, but they can be found in polynomial time, unless one of B's negative eigenvalues has higher multiplicity as an eigenvalue of A than of B. Otherwise, we need not try all pairs of eigenvalues. The negative sum of the corresponding eigenvalues is equal to the spectral radius of A, so we need only try half as many colorings as there are negative eigenvalues of its adjacency matrix.

Note that the conjecture made has not been proved. This conjecture is also a critical part of the proof of the Alon-Kahale algorithm. We explore this conjecture for certain groups of graphs below, namely Cayley graphs for Abelian groups. These graphs have a strong structure that may make the conjecture easier to prove. We provide a brief introduction to group theory and Cayley graphs in Appendix A.

Theorem 3.4. *We can use negative eigenvalues to partition a Cayley graph of* $\mathbb{Z}_2^n$.

Proof. Consider an example of a generating set $H = \{(1, 0, \ldots, 0), \ldots, (0, \ldots, 0, 1)\}$. Then, it is sufficient to prove that for $H \subset \mathbb{Z}_2^n$, we have $\forall h \in H$ there exists some $w \in \mathbb{Z}_2^n$ such that $\sum_{u \in H} (-1)^{u^T w} < 0$, and $h^T w \equiv 1 \mod 2$.

In order to prove this, first select some h, w such that $h^T w \equiv 1 \mod 2$. Then, take the space of all $u \neq h$, and we find that over this space

$$E[(-1)^{u^T w}] = 0,$$

then, we observe that this implies

$$E\left[\sum_{u \in H} (-1)^{u^T w}\right] < 0.$$

From this we can conclude that such an h, w pair exists. □

Theorem 3.5. *We can use negative eigenvalues to partition a Cayley graph of* $\mathbb{Z}_{\ltimes}$.

Proof. Let H be a generator set, so we have $\{a, b\} \in E \iff b - a \in H$.Consider some $H \subset \mathbb{Z}_n / \{0\}$. Define our coloring for a with respect to some eigenvector w as $\chi_w(a) = e^{2\pi i a w / n}$. Observe that if $e^{2\pi i a w / n}$ is an eigenvector, then so is $e^{2\pi i (-a) w / n}$, with the same eigenvalue. This implies that we can take our generating set to be symmetric, so $H = H \cup -H = -H$ (and thus our graph is undirected). We want to determine when these eigenvalues are negative. Recall Euler's identity $e^{ix} = \cos(x) + i \sin(x)$, and refer to the following figure:

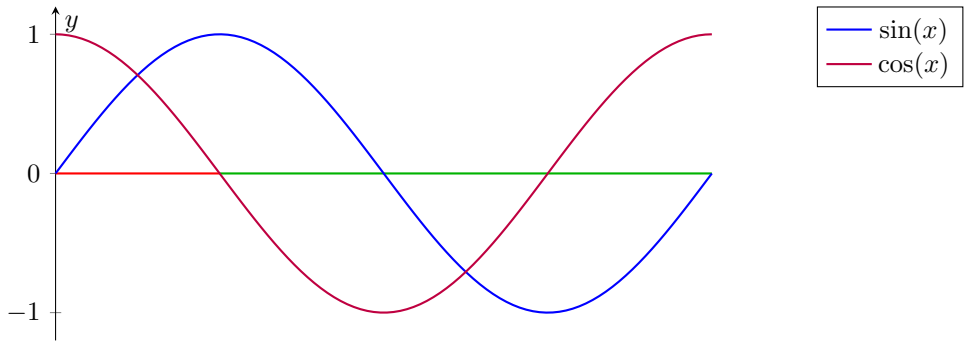

Figure 3: Graph of sine and cosine

We say $2\pi w a / n \in [0, \frac{\pi}{2}]$ means w is "bad" (highlighted red) as neither cos or sin are negative here, and otherwise we say w is "good" (highlighted green). Then, we know there exists some $w \in \{0, \ldots, n-1\}$ that is "good", and by a similar expectation argument as in 3.4, we can conclude that

$$\sum_{a \in H} e^{2\pi i a w / n} < 0$$

So our partition is formed by the eigenvalues corresponding by negative eigenvectors.

□

3.3 Alon-Kahale Algorithm

Alon and Kahale [AK97] take Aspvall and Gilbert's idea and expand it to random graphs where each vertex in a block has the same *expected* degree as other vertices in its block. They prove this for uniform sized blocks, and we show this result can be expanded to unbalanced classes. We outline their algorithm below, and outline our proof for non-uniformity.

The Alon-Kahale assumes the graph is random (edges are added between 3 groups of n vertices with uniform probability p, call this $G_{3n,p,3}$), and implicitly assumes that color classes are balanced (shown by each color class having n vertices). The first step is that we remove high degree vertices (with degree greater than or equal to $5d$, where $d = np$), because we wish to control the spectrum, in order for proofs about eigenvalue partitions to hold. We make the following observation, connecting this process to diameter 2 graphs. Empirically, we notice this algorithm is robust to randomly adding edges until the graph is diameter 2. The relevance of this model to diameter 2 graphs is outlined by the following theorem and corollary, which intuitively states that this random graph will be "close to" diameter 2.

Theorem 3.6. *As $n \to \infty$, $G_{3n,p,3}$ is diameter 2 with probability 1.*

Proof. The odds two vertices do not have a neighbor in a given color class is $(1-p^2)^n$, which goes to 0 as n grows large, for any p. Note that this bound is loose, and we can converge to 0 more rapidly. □

Corollary 3.6.1. *Almost all tripartite random graphs have diameter 2, that is the proportion of tripartite random graphs that are diameter 2 will go to 1 as $n \to \infty$, for all p.*

The algorithm roughly works by (after removing high degree vertices) first finding a linear combination of the smallest two eigenvalues of the adjacency matrix with median zero normalized to have ℓ_2 norm $\sqrt{2n}$, call this t_u. Then, it partitions vectors to be all vertices by norm after multiplication by t_u into three categories - within $\left|\frac{1}{2}\right|$ of 0, strictly greater than $\frac{1}{2}$, and strictly less than $-\frac{1}{2}$. This first stage almost always generates a proper 3 coloring. It then performs a balancing stage, by recoloring algorithms by the least popular color of its neighbors. After this stage, it takes all vertices with few neighbors colored any one color, and attempts to brute force color these final vertices. With high probability, this produces a proper 3 coloring. Implementation details are provided in appendix B.1.

Empirical experimentation shows that the relative size of the color classes are not so important, nor is the exact probability p, although they must both be so that no one color class is "too small", so any failed coloring can usually

be resolved by keeping the same relative sizes of color classes and probability p, and increasing n. This is because large imbalance leads to certain thresholds in the later stages not being met, and the algorithm terminating (even though the first stage usually produces a very close to correct 3-coloring).

Removing high degree vertices was not quite so easily relaxed, and that leaving high degree vertices tended to lead to a failure in coloring. This is because, intuitively, these vertices have more impact on the spectrum, and so they are "overrepresented" in the spectrum.

Given an arbitrary random graph with unknown probability p and unknown color class sizes, by observing the degree distribution we are able to estimate the expected number of vertices in each color class. This informs balancing in later stages of the algorithm.

It is possible to have some control the spectrum of the graph, and to attempt to reduce the degree of high degree vertices. Observe that if we have a triangle lattice, we can add or remove certain edges from this lattice without affecting coloring, as some vertices are forced to be different colors by the lattice, and so an edge between them will have no effect on coloring (even if they are far). Alternatively, if two vertices are at odd distance from each other along some cycle with a chord between them, then we can remove the chord and know that they will still be different colors. This can be done as a preprocessing step to make a graph more likely to fit the eigenvalue constraints of the algorithm.

We prove an alteration of the proof of stage one of the Alon-Kahale algorithm, with the assumption that we have color classes W_1, W_2, W_3 of size , $\alpha n, \beta n$ where $0 < \alpha, \beta, \leq 1$. Add edges with probability p, and let $d = pn$, and let $a = \frac{(1+\alpha+\beta)}{3}$ be the expected relative size of our classes. Let $x = (x_v : v \in V)$ be the vector defined by $x_v = 2$ for $v \in W_1$, $x_v = -\frac{1}{\alpha}$ for $v \in W_2$, and $x_v = -\frac{1}{\beta}$ for $v \in W_3$. Then, let $y = (y_v : v \in V)$ be the vector defined by $y_v = 0$ is $v \in W_1$, $y_v = \alpha$ is $v \in W_2$, and $y_v = -\frac{\alpha}{\beta}$ if $v \in W_3$. .

First we provide Alon and Kahale's proposition 2.1:

Theorem 3.7 ([AK97])**.** *Almost surely,*

1. $\lambda_1 \geq (1 - 2^{\Omega(d)})2d$,
2. $\lambda_{3an} \leq \lambda_{3an-1} \leq -(1 - 2^{-\Omega(d)})d$, *and*
3. $|\lambda_i| \leq O(\sqrt{d}) \quad 2 \leq i \leq 3an - 1$

This proposition is utilized below to prove the correctness of the eigenvector partition, then an modified proof is provided (with some tighter bounds).

First, we prove that almost surely $\|(A+dI)y\|^2 = O(nd)$ and $\|(A+dI)x\|^2 = O(nd)$.

Note that the expectation of $(A+dI)y = \hat{0}$, since each coordinate is the sum of $3an$ random variables, call these P_{ij}, where each represents an edge. These are iid Bernoulli with probability p , with variance $p(1-p) = O(d/n)$. Then, for each coordinate of $(A + dI)y$ we have a random variable C_i for each coordinate, which has mean 0 and variance depending on which set i is in. If $i \in W_1$, we

have variance of C_i equal to $\alpha n(d/n)1^2 + \beta n(d/n)(-1)^2 = (\alpha+\beta)d$, if $i \in W_2$ we have variance βd, and else we have variance αd. Since the expectation is 0, we have that the variance is equal to the square of the coordinate. Then, the expected value of $\sum C_i^2$ is $O((\alpha+\beta+2\alpha\beta)nd)$. We have that the fourth moment of each coordinate on W_1 is $O(d^2)$, so we can use Chebyshev to prove that the sum of squares on W_1 is $O((\alpha+\beta)nd)$ almost surely, and similarly for W_2, W_3 and $O(\beta\alpha nd)$. Then, we have $\|(A+dI)y\|^2 = O((\alpha+\beta)nd)$ almost surely.

Combined with proposition 2.1 we have

$$\begin{aligned}\|(A+dI)y\|^2 &= \sum_{i=1}^{3na} c_i^2(\lambda_i+d)^2 \\ &\geq \Omega(d^2)\sum_{i=1}^{3na-2} c_i^2\end{aligned}$$

So we have $\|\delta\|^2 = O\left(\frac{(\alpha+\beta)n}{d}\right)$.

Then, we have that $x-\epsilon$ and $y-\delta$ are independent, since if we had $\mu(x-\epsilon)+\nu(y-\delta)=0$, we would have $\mu x+\nu y = \mu\epsilon+\nu\delta$. We have $\mu n(4+\frac{1}{\alpha^2}+\frac{1}{\beta^2})+\nu n(\alpha^2+\frac{\alpha^2}{\beta^2}) = \|\mu\epsilon\nu\delta\|^2 \leq |\mu|\|\epsilon\| + |\nu|\|\delta\| = O((\|\mu\|+\|\nu\|)\sqrt{\frac{(\alpha+\beta)n}{d}}$, then $\mu(4+\frac{1}{\alpha^2}+\frac{1}{\beta^2})+\nu(\alpha^2+\frac{\alpha^2}{\beta^2}) = O((\mu^2+\nu^2)(\alpha+\beta)/d)$ implying that $\mu=\nu=0$.

Thus, we have $\sqrt{3an}e_{3an-1}$ and $\sqrt{3an}e_{3an}$ as linear combinations of $x-\epsilon$ and $y-\delta$.

We can write t as a linear combination of these. We can set $t = f+\eta$ with $f \in F$ (where F is the set of vectors constant on color classes whose coefficients sum to 0) and $\|\eta\|^2 = O(n/d)$. Then, at most $O(n/d)$ coordinates of η are greater than .01 in absolute value as a result of our bound on the norm of η. Call the coefficients of f on each color class $\alpha_1 \geq \alpha_2 \geq \alpha_3$. Let α_2 correspond to the largest color class. Since the median of t is 0, we cannot have more than $2n - O(n/d)$ degrees with the same sign, so $|\alpha_2| < 1/4$. As a result of the sum of $\alpha_i = 0$ and the squared sum equal to $\|f\|^2/n = 2 + O(d^{-1})$, we have that $\alpha_1 > 3/4$ and $\alpha_3 < -3/4$. This implies that the colorings defined by our eigenvalue partition agree on all except for at most $O(n/d)$ vertices.

We now prove alternative bounds on eigenvalues.

Lemma 3.8 (Alon Kahale Lemma 2.6 modified)**.** *There exists a constant $\gamma > 0$ such that almost surely the following holds:*

1. *For any two distinct color classes V_1, V_2 and any subset $X \subset V_1$, $Y \subset V_2$ if $|X| = 2^{-\gamma d}|V_1|$ and $|Y| \leq 3|X|$, then $|e(X, V_2 - Y) - d|X|| \leq .001d|X|$*
2. *If J is the set of vertices having more than $1.01d$ neighbors in G in some color class, then $|J| \leq 2^{-\gamma d}\beta n$*

Proof. Take 2 distinct color classes. First, consider V_1 to be our color class if size αn, and V_2 our color class of size βn. Let $\epsilon = 2^{-\gamma d}$. Then, the probability

that our hypothesis for part 1 holds but the conclusion does not follow is at most

$$\binom{\alpha n}{\epsilon n}\sum_{i=0}^{3\epsilon n}\binom{\beta n}{i}2^{-\Omega(\epsilon n d)}$$

So considering all $\binom{3}{2}=6$ ways to choose color classes (let α or $\beta=1$ to consider the largest color class) in this way, we can use Chernoff bounds to bound their sum from above by

$$2^{O\left(\max\{H(\epsilon),\alpha H(\frac{\epsilon}{\alpha}),\beta H(\frac{\epsilon}{\beta},H(3\epsilon),\alpha H(\frac{3\epsilon}{\alpha}),\beta H(\frac{3\epsilon}{\beta})\}\right)n}2^{-\Omega(\epsilon n d)}$$

□

So if we have γ small enough, or $\alpha n, \beta n$ not too small, this will hold. We can bound in a similar manner for the second part.

Similarly, Lemma 3.1 uses b sufficiently large and β' (where β' is the β used in Alon and Kahale's proof) sufficiently small, so this works as well, essentially as in the original paper.

Lemma 3.9 (Alon Kahale 3.1 modified)**.** *There exists a constant γ' such that, almost surely, for any subset X of $2^{-\gamma' d}an$ vertices, we have $e(X,V)\leq 5d|X|$.*

Proof. Considering the proof of the previous lemma, we can bound the ways to select X by $2^{-\Omega(d\epsilon n)}$. Then if $\log(\frac{1}{\epsilon})<d/b$ for large enough b (and thus small enough γ'), this probability will go to 0. □

Now, we consider an proof of an altered version of Alon and Kahale's proposition 2.1. We consider the first bound. Note that the average degree is expected to be $(1+O(1))2d\left(\frac{\alpha\beta+\alpha+\beta}{1+\alpha+\beta}\right)$ instead of $(1+O(1))2d$. Then, we know the average degree influences the first eigenvalue, so in combination with 3.9, we can bound it as follows: $\lambda_1\geq(1-2^{-\Omega(d)})2d\left(\frac{\alpha\beta+\alpha+\beta}{1+\alpha+\beta}\right)$.

The second bound is not used in the approximate coloring, so we omit a modified proof.

We modify the lemmas presented in the Alon Kahale paper to bound the middle eigenvalues in the following manner:

Instead of creating an $n\times n$ matrix B to represent cross class connectivity, we create matrix block sizes that correspond to edges between classes of size $n,\alpha n,\beta n$, call these blocks B_i. Then, instead of bounding by $O(\sqrt{d})$, we note that each group of edges between classes has a different average degree. Thus, we want to prove a bound on $O(\sqrt{d_{max}})$, where d_{max} is the largest average degree, dependent on α,β. In Alon and Kahale's original paper, this is $d_{max}=2d=O(d)$. We can prove Alon and Kahale's Lemma 3.3 for each block (noting that each block will have a different constant c_i bounding its variance), and Lemmas 3.4, 3.5 require no modification. Then, we conclude that the maximum contribution of pairs in C (that is, pairs x,y whose with $|x_u y_v|\leq\sqrt{d_{\max}}/n$) to $|x^T B_i y|$ is $O(\sqrt{d_{max}})$. The contribution of terms whose absolute values exceed $\sqrt{d_{max}}/n$ is bounded as in the original paper.

Alon and Kahale's Lemma 3.6 requires a minor modification in the final equation, by noting that we cannot assume equality across class sizes, and instead must bound the final equation similar to in 3.8. Then, note the constant used will have a dependency on α, β. From this, we conclude that the contribution on terms whose absolute value exceeds $\sqrt{d_{max}}/n$ is $O(\sqrt{d_{max}}$. This implies Lemma 3.3, implying 3.2. Lemma 3.7 needs no modification, other than noting that d_{max} will provide a tighter bound.

Then, by observing that if H is all subspaces of $\mathbb{R}^{3an}$ with codimension 1, we have

$$\lambda_2 = \min_H \max_{x \in H, x \neq 0}.$$

Let H have the sum of coordinates 0. Then, as we did earlier, we can write $x \in H$ as $f + s$ with $f \in F$, then we have

$$\begin{aligned}
x^T A x &= f^T A f + 2 s^T A f + s^T A s \\
&= f^T A f + 2 s^T (A + dI) f + s^T A s \\
&\leq -(1 - 2^{-\Omega(d)}) 2d \left(\frac{\alpha\beta + \alpha + \beta}{1 + \alpha + \beta} \right) \|f\|^2 + 2\|s\|\|(A + d_{max} I) f\| + O(\sqrt{d_{max}})\|s\|^2 \\
&\leq O(\sqrt{d_{max}}\|s\|\|f\|) + O(\sqrt{d_{max}}\|s\|^2) \\
&\leq O(\sqrt{d_{max}}(\|s^2\| + \|f^2\|) \\
&= O(\sqrt{d_{max}}\|x\|^2) \\
&= O(\sqrt{d}\|x\|^2).
\end{aligned}$$

Where we use $O(\sqrt{d_{max}}\|x\|^2) = O(\sqrt{d}\|x\|^2)$ interchangeably for proving bounds, in particular the latter being easier to utilize in the earlier proof of the correctness of partitioning, but note that for precise calculation accounting for the sizes of color classes we use the former. Similarly we can bound $|\lambda_{3n-2}|$, and the third bound in 3.7 follows.

We observe that the rebalancing phases will not work as intended with unbalanced color classes. Instead, we can approximate the class sizes by observing the degree distribution, and attempt to rebalance accordingly. Then, in the third stage, we can identify wrongly colored vertices, and attempt to brute force color them. We do not provide correctness of these ideas beyond empirical success.

4 Seed-Based Algorithms

Another approach to coloring a graph is by first coloring a subset of vertices within a graph, and then using this planted coloring to color the rest of the graph. We call this a *seed* coloring, where the seed is the planted coloring. If the neighborhood of our seed is the entire graph, then 3 coloring the seed will entail that the remainder of the vertices have at most 2 possible colors. This problem is called LIST-2 coloring, and is solvable in polynomial time. A subset of vertices such that each vertex in the ambient graph is either in the set or

neighboring a vertex in the set is called a *dominating set*, and we observe that finding and coloring a dominating set in polynomial time implies that we can color the entire graph in polynomial time.

4.1 Degree Based Coloring

We consider many options for degree based coloring, showing that some work, and that some do not. We provide a number of observations, parameterized by restrictions on degree size.

1. Let G be a graph with minimum degree $\geq \alpha n$. Let $\log(n) = \alpha n$. Select vertices with probability $p = \frac{2\log n}{\alpha n}$, then the odds that a vertex with αn neighbors is not covered by this set is

$$\begin{aligned}(1-p)^{\alpha n} &\leq e^{-p\alpha n} \\ &= e^{-2\log n} \\ &= \frac{1}{n^2}\end{aligned}$$

   Using the union bound, the odds that a vertex is not covered is $\frac{1}{n}$.

2. If we can select one vertex of maximum degree $c\log n$, then we can color this vertex and its neighborhood by brute force (time $3^{c\log n} = O(n3^c)$, then color the remaining neighbors by LIST-2 coloring.

However, we are not guaranteed a vertex with small or large degree. We show that without a degree bound requirement, we cannot naively brute force in a similar manner. Select $\log(n)$ vertices, call this set S. We can color S using brute force colorings in time $3^{\log n} = O(n)$. Consider an arbitrary vertex i, and consider the odds that this vertex or none of its neighbors have been colored, with the assumption this node has $\log n$ neighbors. We are have $\binom{n}{\log n}$ total possible options for S , and $\binom{n-\log n}{\log n}$ of these will avoid the $\log n$ neighbors of our vertex. Then, we bound the odds that vertex i will not be in $N[S]$ as follows:

$$\begin{aligned}P(i \notin N[S]) &\leq \frac{\binom{n-\log n}{\log n}}{\binom{n}{\log n}} \\ &= \frac{(n-\log n)!^2}{(n-2\log n)!n!}\end{aligned}$$

By Stirling

$$\begin{aligned}
&\approx \frac{\left(\sqrt{2\pi(n-\log n)}\left(\frac{n-\log n}{e}\right)^{n-\log n}\right)^2}{\sqrt{2\pi(n-2\log n)}\left(\frac{n-2\log n}{e}\right)^{n-2\log n}\cdot\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} \\
&\approx \frac{\left(\frac{n-\log n}{e}\right)^{2(n-\log n)}}{\left(\frac{n-2\log n}{e}\right)^{n-2\log n}\left(\frac{n}{e}\right)^n} \\
&\approx_{n\to\infty} \left(\frac{n-\log n}{n}\right)^n
\end{aligned}$$

So this bound does not quite work.

A second approximation considers the odds a vertex with $\log n$ neighbors has any of these $\log n$ neighbors selected. The odds the first neighbor is not selected is $\left(1-\frac{\log n}{n}\right)$, and since each neighbor is independent, we have our end result as $\left(1-\frac{\log n}{n}\right)^{\log n}$. This goes to 1 as $n\to\infty$. So we cannot prove our result with this method. However, we have parameters. If A = the number of neighbors that our vertex has, we consider our probability as $\lim_{n\to\infty}\left(1-\frac{\log n}{n}\right)^A$.

Increasing by a constant factor, if we have $\log n$ vertices with degree $\frac{cn}{\log n}$, then the odds that any vertex is not colored approaches is $\left(1-\frac{c}{\log n}\right)^{\log n}\to\frac{1}{e^c}$ as n grows large. Then, the odds that at least one of our $n-\log n$ vertices are not in a neighborhood is $1-(1-e^{-c})^{n-\log n}$, which approaches 1 as $n\to\infty$, so this does not work either.

Say we, similarly, take $f(n)$ samples of $\log n$ nodes with degree $\frac{n}{\log n}$, for some polynomial f. Then, the odds that all nodes are hit in a sample is

$$\left(1-\left(\frac{n-\frac{n}{\log n}}{n}\right)^{\log n}\right)^{n-\log n}$$

by a similar calculation as previous paragraphs. Then, the odds that 1 sample fails is 1 minus this. So the probability that $f(n)$ samples fail is

$$\left(1-\left(1-\left(\frac{n-\frac{n}{\log n}}{n}\right)^{\log n}\right)^{n-\log n}\right)^{f(n)},$$

which goes to 1 for polynomial f, and thus we can say with probability 1 all samples will fail, and we will not obtain a dominating set. This illustrates why we cannot naively select vertices and find a dominating set in arbitrary graphs.

4.2 Size of Dominating Set

A natural question is whether we can find a logarithmic size dominating set, or if our random selection was doomed from the start. If we can do this efficiently, then we can color the dominating set in polynomial time, and LIST-2 color the remaining graph in polynomial time. We provide results from Henning and Yeo [Ha07] that show this cannot, in general, be done for diameter 2 graphs.

Theorem 4.1 (Henning and Yeo [Ha07])**.** *Dominating number of diameter 2 graphs is bounded above by* $\left(\frac{1+\ln(\delta)}{\delta}\right) n$.

Theorem 4.2 (Henning and Yeo [Ha07])**.** *For diameter 2 graphs, if the min degree is larger than* $\ln(n)\sqrt{n}$*, then the size of the dominating set is bounded by* $1+\sqrt{n}$.

Proof. Assume $n \geq 3$. Then, let our domination number be $\gamma_t(G)$. We have by 4.1 abd by noting that $\frac{1+\ln(n)}{n}$ is decreasing in n, that

$$\begin{aligned}
\gamma_t(G) &\leq \left(\frac{1+\ln(\delta)}{\delta}\right) n \\
&\leq \left(\frac{1+\ln(\sqrt{n}\ln(n))}{\sqrt{n}\ln(n)}\right) n \\
&= \left(\frac{1+\frac{1}{2}\ln(n)+\ln(\ln(n))}{\ln(n)}\right)\sqrt{n} \\
&= \frac{\sqrt{n}}{2} + \frac{1+\ln(\ln(n))}{\ln(n)}\sqrt{n}
\end{aligned}$$

So we show that $\frac{1+\ln(\ln(n))}{\ln(n)}\sqrt{n} \leq 1+\frac{\sqrt{n}}{2}$. This holds for $n \geq 213$ as the LHS is decreasing and less than $\frac{1}{2}$, and computer verification resolves all other cases. □

Theorem 4.3 (Henning and Yeo[Ha07])**.** *For diameter 2 graphs, the dominating set is bounded by* $1+\sqrt{n\ln(n)}$*, and this bound is tight.*

Proof. We set up a similar inequality to 4.2

$$\begin{aligned}
\gamma_t(G) &\leq \left(\frac{1+\ln(\delta)}{\delta}\right) n \\
&\leq \left(\frac{1+\ln(\sqrt{n\ln(n)})}{\sqrt{n\ln(n)}}\right) n \\
&= \left(\frac{1+\frac{1}{2}\ln(n)+\ln(\ln(n))}{\sqrt{\ln(n)}}\right)\sqrt{n} \\
&= \frac{\sqrt{n\ln(n)}}{2} + \frac{1+\ln(\ln(n))}{\ln(n)}\sqrt{n\ln(n)}
\end{aligned}$$

So we need prove that

$$\left(\frac{1+\frac{1}{2}\ln(\ln(n))}{\ln(n)}\sqrt{n\ln(n)} \leq 1+\frac{\sqrt{n\ln(n)}}{2}\right)$$

Which holds when $n \geq 24$, as the LHS is decreasing and less than $\frac{1}{2}$. The following cases follow from computer verification. □

Theorem 4.4 (Henning and Yeo [Ha07])**.** *If the domination number is greater than* $1+\sqrt{n}$ *, then the min. degree is between* $\sqrt{n}$ *and* $\ln(n)\sqrt{n}$*.*

Proof. This follow from 4.2 and by noting that if we have minimum degree of some vertex v less than $\sqrt{n}$, we can take $N(v)$ as a dominating set with size less than or equal to $1+\sqrt{n}$. □

Thus, even if min degree is $\frac{n}{\ln n}$, we have the size of the dominating set bounded by $\ln^2(n)$, which is too large to brute force color.

4.3 Claw Free Dominating Set

Since we cannot find a dominating set in arbitrary diameter 2 graphs, we investigate restricted classes of diameter 2 graphs. We overview results provided by Bouqet et al. in regards to this question , where they prove that if we forbid certain subgraphs, we can find dominating sets for diameter 2 graphs efficiently.

Definition 25. *The claw graph is* $K_{1,3}$*.*

Definition 26. *A vertex* v *is complete so a set* S *if it is adjacent to every vertex in* S*, and anticomplete if it is adjacent to none of the vertices in* S*.*

Definition 27. *A W-join is a pair of disjoint non-empty sets of vertices (A,B) such that* $|A|+|B|>2$*, both are cliques,* A *is neither complete nor anticomplete to* B*, and every vertex of* $V(G)\backslash(A\cup B)$ *is either complete or anticomplete to* A *and complete or anticomplete to* B*.*

Definition 28. *Neighborhoods of vertices* u, v *are distinct if we do not have* $N[u]\subset N[v]$*, or vice versa. If all such vertices have this property in a graph, we say the graph has distinct neighborhoods.*

Definition 29. *A proper circular arc graph is defined by the following characteristics: each vertex corresponds to an arc on a circle, and vertices are adjacent if their arcs intersect. Furthermore, what makes this graph proper is that no arc is entirely contained within another.*

Theorem 4.5 (Martin et al. [MPvL20])**.** *Every claw free graph with distinct neighborhoods, maximal independent set number at least 3, and more than 13 vertices is either a proper circular-arc graph or a line graph.*

Theorem 4.6 (Bouqet et al. [BDPR24])**.** *Finding a dominating set is polynomial time solvable for line graphs with diameter 2.*

Proof. Bouqet et al. prove that $2K_2$ free graphs (which have diameter 2 line graphs) have maximal stable sets that can be enumerated in polynomial time. Then, as matchings in the original graph correspond to a dominating set in the dual, the result follows. □

Theorem 4.7 (Hsu and Tsai [HT91])**.** *Constructing a dominating set for a proper circular arc graph can be done in polynomial time.*

Proof. First, cut at an arbitrary point to make the circular arc graph an interval graph. Then, use a sweep algorithm to process arcs, and record all intersections. Then, run a greedy algorithm that picks an interval covering the leftmost arc that extends the furthest to the right. □

Theorem 4.8 (Bouqet et al. [BDPR24])**.** *Finding a dominating set in claw-free diameter two graphs is solvable in polynomial time.*

Proof. Let G be a claw-free diameter 2 graph. Then, assume the size of the dominating set is larger than 4, as else we can brute force it in polynomial time. It follows that G has no $W-$join, and since the dominating set size is less than or equal to the maximal independent set size, the size of the maximal independent set is at least 3. We can also assume that $|V| > 13$, as else we can find a dominating set in constant time.

Next, assume that there is a pair of adjacent vertices u, v such that the neighborhood of u without v is a subset of the neighborhood of v without u. Then, for ever dominating set D of $G-u$, we have $N(u) \cap D \neq \emptyset$, so a dominating set of $G-u$ also dominates G. Also, removing u does not increase the diameter. Thus, we can search for all such pair u, v and remove them from G in polynomial time.

Thus, we can apply 4.5. If G is a line graph (which can be checked in polynomial time) we use 4.6, else we use 4.7. □

Definition 30. *A vertex Cover of a graph is a subset of vertices such that each edge has at least one endpoint in this subset. Finding such a cover is known to be $NP-$complete for arbitrary graphs.*

Theorem 4.9 (Bouqet et al [BDPR24])**.** *Finding a dominating set is NP-Complete for $K_{1,4}$ free graphs with diameter 2.*

Proof. This follows from a polynomial time reduction from vertex cover. From $I = (G = (V, E), k)$ an instance of Vertex Cover, we build an instance $I' = (G', \gamma)$ where G' is $K_{1,4}$-free with diameter 2 and $\gamma = k$.

We start by constructing $G' = (V', E')$. The vertices of V' are partitioned into V_1, E_1, E_2, S, s. We define these sets and the edges of G' as follows:

- For each vertex $v \in V$, there is a vertex $v_1 \in V_1$, that is, $V_1 = v_1 \mid v \in V$;
- For each edge $uv \in E$, there is a vertex $e^1_{uv} \in E_1$ and $e^2_{uv} \in E_2$, that is, $E_1 = e^1_{uv} \mid uv \in E$ and $E_2 = e^2_{uv} \mid uv \in E$;

- For each $u_1 \in V_1$, the vertices $u_1 \cup e^1_{uv} \mid u = u_1$ and $u_1 \cup e^2_{uv} \mid u = u_1$ form two cliques;
- $V_1 \cup S \cup s$ is a clique;
- For each pair $e, e' \in E_1 \cup E_2$ such that $N(e) \cap N(e') \cap V_1 = \emptyset$, there is a vertex $s_{e,e'} \in S$ and the two edges $s_{e,e'}e, s_{e,e'}e' \in E'$. Note that these edges e, e' correspond to copies of non-incident edges in G.

Since $V_1 \cup S \cup s$ is a clique and that every pair of vertices $e, e' \in E_1 \cup E_2$ has a common neighbor in $V_1 \cup S$, it follows that $\text{diam}(G') = 2$. We show that G' is $K_{1,4}$-free. For each vertex of G', we give a partition of its neighborhood into at most three cliques. For $u_1 \in V_1$: $N(u_1) \cap E_1$, $N(u_1) \cap E_2$, and $N(u_1) \cap (S \cup V_1 \cup s)$. For $e^i_{uv} \in E_i$, $i \in 1, 2$: $N(e^i_{uv}) \cap N(u_1)$, $N(e^i_{uv}) \cap N(v_1)$, and $N(e_{uv}) \cap S$. For $s_{e,e'} \in S$: $s_{e,e'}, e$, $s_{e,e'}, e'$, and $V_1 \cup S \cup s$. For the vertex s: $N(s) = S \cup V_1$. Therefore G' is $K_{1,4}$-free.

Let C, $|C| \leq k = \gamma$, be a vertex cover of G. Then its copy in V_1 is a dominating set of G' of size at most γ.

Let $I' = (G', \gamma)$ be a positive instance, so there exists Γ, $|\Gamma| \leq \gamma$ a dominating set of G'. From Γ we will construct a dominating set Γ' such that $\Gamma' \subseteq V_1$. Since $N(s) = V_1 \cup S$ we can assume that $s \notin \Gamma$. Let S_i be the vertices of S with two neighbors in E_i, that is, $S_i = s_{e,e'} \mid e, e' \in E_i$, $i = 1, 2$. Let $\Gamma_i = \Gamma \cap (E_i \cup S_i)$. Without loss of generality $|\Gamma_1| \leq |\Gamma_2|$. Let $\Gamma' = \Gamma \setminus \Gamma_2$. For each $e^1 \in \Gamma_1$, we add e^2 to Γ', and for each $s_{e^1,e'^1} \in \Gamma_1$, we add s_{e^2,e'^2} to Γ'. Since $G'[E_1 \cup S_1 \cup V_1]$ is isomorphic to $G'[E_2 \cup S_2 \cup V_1]$, it follows that Γ' is a dominating set of G' such that $|\Gamma'| \leq \gamma$.

Let E^0_i be the vertices $e^i_{uv} \in E_i$ such that $\Gamma' \cap N(e^i_{uv}) \cap V_1 = \emptyset$, $i = 1, 2$. For each $e^1_{uv} \in E^0_1$, there is $e^2_{uv} \in E^0_2$, and vice versa, because $N(e^1_{uv}) \cap V_1 = N(e^2_{uv}) \cap V_1$. Since $N(e^1_{uv}) \cap N(e^2_{uv}) \cap S = s_{e,e'}$, with $e = e^1_{uv}, e' = e^2_{uv}$, and that each vertex of S has exactly two neighbors in $E_1 \cup E_2$, it follows that $|E^0_1| \leq |S \cap \Gamma'|$. Then we remove the vertices of S from Γ' and we replace them by $u_1 \in V_1$ for each $e^1_{uv} \in E^0_1$. It follows that Γ' is a dominating set of G' such that $|\Gamma'| \leq \gamma$. Note that $\Gamma' \subseteq V_1$.

Let C be the copies of the vertices of $\Gamma' \cap V_1$ in G. Since each vertex $e^1_{uv} \in E_1$ has a neighbor in $\Gamma' \cap V_1$, it follows that C is a vertex cover of G such that $|C| \leq k$. □

Definition 31. *A split graph S is a graph whose vertices can be partitioned into $S = (K, I)$, where K is a clique and I is an independent set.*

Theorem 4.10. *Given a split graph, we can find the split in polynomial time.*

Proof. First, we sort all vertices by degree. Find the largest k for which we have k vertices of degree $k - 1$ or larger. Consider all vertices with degree exactly $k - 1$. Then, these vertices must be connected to all vertices in the k clique, and we can pick any to be in the clique. □

Definition 32. *A vertex v is called simplicial when $N(v)$ is a clique.*

Lemma 4.11 (Bouqet et al. [BDPR24]). *If u, v are vertices of a graph such that $N(u) \subset N(v)$, and v is simplicial, then the dominating set of G is not affected by the removal of v.*

Proof. For any graph containing a simplicial vertex, there is a dominating set that does not contain this vertex. Let S be such a set that does not contain v. Then S is a dominating set of $G - v$. Then, it follows that u is simplicial, and that $uv \notin E$. Thus, there is a set $|S'|$ that dominates and does not contain u, with $|S| = |S'|$. □

Theorem 4.12 (Bouqet et al. [BDPR24]). *Finding a dominating set is NP-complete for triangle-free graphs with diameter 2.*

Proof. We give a polynomial transformation from Dominating Set, which is NP-complete for split graphs with diameter 2 (see [9]). From $I = (G, k)$ an instance of Dominating Set, we build an instance $I' = (G', k')$.

In $I = (G, k)$, $G = (K \cup S, E)$ is a split graph with diameter 2 where K is a clique and S is a stable set. Let $u, v \in S$. First, since the vertices of S are simplicial, it follows from 4.11 that we can suppose that $N(u)N(v)$ and $N(v)N(u)$. Second, since $\text{diam}(G) = 2$, there exists $w \in K$ such that $u - w - v$ is a path in G.

From G we build $G' = (V', E')$ as follows. We take a copy K_1 of K and two copies S_1, S_2 of S. For the sake of simplicity, for $v \in K$, its copy in K_1 is denoted by v_1, whereas for $v \in S$, its copies in S_1, S_2 are denoted by v_1, v_2, respectively. We then add two vertices t and s. For each pair $u \in K$, $v \in S$, if $uv \in E$, then we add the edge u_1v_1; otherwise, we add the edge u_1v_2. For every $v \in S$, we add the edge v_1v_2. Then we make t complete to K_1 and s complete to S_2. Last, we add the edge st. Note that t, K_1, s, S_1, S_2 is a partition of G' into stable sets. Finally, we take $k' = k + 1$.

We show that G' is triangle-free. Since $N(t) = K_1 \cup s$ and $N(s) = S_2 \cup t$ are two stable sets, it follows that t and s cannot be in a triangle. Thus, if a triangle exists, it has one vertex $u_1 \in K_1$, one vertex in S_1, and one vertex in S_2. So this triangle contains the edge v_1v_2. But when u_1v_1 is an edge, u_1v_2 is not an edge, and vice versa. So G' is triangle-free.

We show that $\text{diam}(G') = 2$. We observe that t and s are at distance at most two from any vertex of the graph. So we can focus on the vertices of $K_1 \cup S_1 \cup S_2$. Since t is complete to K_1 and s is complete to S_2, for any pair $v_1, u_1 \in K_1$ (respectively $v_2, u_2 \in S_2$), there exists the path $v_1 - t - u_1$ (respectively $v_2 - s - u_2$).

Since $\text{diam}(G) = 2$, for any pair $v_1, u_1 \in S_1$, there exists $w_1 \in K_1$ such that $v_1 - w_1 - u_1$ is a path of G'. Now let $u_1 \in S_1$, $v_1 \in K_1$ (respectively $u_2 \in S_2$, $v_1 \in K_1$), such that $uv \notin E$ (respectively $uv \in E$). Then $u_2v_1 \in E'$ (respectively $u_1v_1 \in E'$), so $u_1 - u_2 - v_1$ (respectively $u_2 - u_1 - v_1$) is a path of G'. Now let $u_1 \in S_1$, $v_2 \in S_2$, $u \neq v$. From Lemma 4.1, we can assume that there exists $w \in N(u)$, $w \notin N(v)$. Therefore $u_1w_1, v_2w_1 \in E'$, and $u_1 - w_1 - v_2$ is a path in G'. So $\text{diam}(G') = 2$.

Let D be a dominating set of G with $|D| \leq k$. Let D' be the set of the copies of the vertices of D in $K_1 \cup S_1$. Then $D' \cup t$ is a dominating set of G' and $|D' \cup t| \leq k + 1 = k'$.

Conversely, let D' be a dominating set of G' with $|D'| \leq k' = k + 1$. Since $N(s) = S_2 \cup t$, it follows that $|D' \cap (t, s \cup S_2)| \geq 1$.

First, suppose $S_2 \cap D' = \emptyset$. So $|D' \cap t, s| \geq 1$. For each $v_1 \in D' \cap S_1$, if any, let a unique $u_1 \in N(v_1) \cap K_1$. Then let

Second, if $|S_2 \cap D'| \geq 1$ and $t \in D'$, then for each $v_2 \in D' \cap S_2$, let a unique $u_1 \in N(v_1) \cap K_1$ (where v_1 is the neighbor of v_2 in S_1). Then let

Third, if $|S_2 \cap D'| \geq 1$ and $t \notin D'$, then for each $v_1 \in S_1$ which is not dominated by a vertex of K_1 or by itself, we have that v_1 is dominated by v_2, its neighbor in S_2. Let any $w \in N(v_1) \cap K_1$. Since $t \notin D'$, we have that w is dominated either by $u_1 \in N(w) \cap S_1$, $u_1 \neq v_1$, or by $u_2 \in S_2$, $u_2 \neq v_2$. In the first case, we replace u_1 by w in D', in the second case we replace u_2 by w in D'. Then we take $\bar{D} = D' \cap (K_1 \cup S_1)$.

In all cases, we take D to be the copies of the vertices of $\bar{D}$ in G. We have that D is a dominating set of G with $|D| \leq k$. □

5 Families of Diameter 2 Graphs

Another perspective on coloring is investigating the family of all diameter 2 graphs. If we can identify certain structures, such as a finite family of forbidden subgraphs that force a 3 coloring to not be possible, then we can use this to inform coloring algorithms. We explore results related to counting various families of graphs, then construct an example to show that there is not a finite family of forbidden subgraphs for diameter 2 graphs.

5.1 Well Quasi Ordering

A well quasi ordering is a relation defined on an infinite set that informs us on certain structural properties of the set. We are interested in when diameter 2 graphs are well quasi ordered, as this institutes some structure to the family of all diameter 2 graphs.

Definition 33. *A well-quasi-order on a set S is a preorder $(P, \leq)$, such that for any infinite sequence $\{x_i\}_{i \in I} \subset S$, there exists some $i < j$ with $x_i \leq x_j$.*

Alternatively, this is expressed by satisfying two conditions. First, there must be no infinite antichains (set in which no two elements are comparable), so for any infinite sequence in P, there is some pair of elements that are comparable. Second, there must be no strictly decreasing infinite sequence in P.

Definition 34. *A minor H of a graph G is H can be formed from G by deleting vertices, edges, and by contracting edges.*

Theorem 5.1 (Robertson and Seymour [RS04])**.** *Undirected graphs are well quasi ordered by the minor relation.*

Definition 35. *A cograph is a graph which does not contain P_4 as an induced subgraph.*

Theorem 5.2 (Damaschke [Dam90])**.** *Cographs are well-quasi ordered under the subgraph relation.*

Theorem 5.3. *The family of claw-free and C_5 free diameter 2 graphs are well-quasi ordered.*

Proof. If a diameter 2 graph has an induced path $P_4 = v_1v_2v_3v_4$, then we know there must be edges $v_1v_5, v_4v_5 \in E$ in order for $d(v_1, v_4) \leq 2$ to hold. Then, we could still have this graph not be a cograph if v_2v_5 or $v_3v_5 \in E$, as either would imply that $v_1v_2v_3v_4v_5v_1$ is not an induced 5 cycle. Forbidding claws and 5 cycles thus ensures that we cannot have P_4 as a subgraph. Then, all remaining graphs are cographs, and so the remaining family is a collection of cographs, and thus is well-quasi ordered. □

5.2 Strongly Regular Graphs

Strongly regular graphs are a family of graphs with strong structural properties, that are guaranteed to be diameter 2 if given certain parameters. Similar to how Alon and Kahale generalized coloring random graphs from coloring regular graphs, it is possible to generalize coloring random graphs from coloring strongly regular graphs. In this subsection we provide an overview of results relating to coloring strongly regular graphs.

Definition 36. *We call a graph G strongly regular of degree k if it can be described as (n, d, p, q), where every pair of adjacent neighbors have p neighbors in common, and every nonadjacent pair has q neighbors in common. Such a graph is primitive if both itself and its complement are connected. Note that these graphs are diameter 2 (and distance regular) if $q > 0$.*

Lemma 5.4 (Haemers [Hae79])**.** *Let f_n be the multiplicity of λ_n. Then,*

$$\gamma(G) \geq \max\{1 + f_n, 1 - \frac{\lambda_n}{\lambda_2}\}$$

Proof. Let $\gamma \leq f_n$. Then, $\lambda_n = \lambda_{n-\gamma+1}$. By observing that

$$(\gamma - 1)\lambda_{k+1} \geq -\lambda_{n-k(\gamma-1)}$$

the result follows, with $k = 1$. □

Lemma 5.5 (Haemers [Hae79])**.** *If G is primitive and strongly regular, and not the pentagon or the complete $\gamma-$bipartite graph, then*

1. $d \leq -\lambda_n(\gamma(G) - 1)$
2. $-\lambda_n \leq \lambda_2(\gamma(G) - 1)$
3. $\lambda_2 \leq \gamma(G) - 1$

Proof. First, we prove that $\gamma(G) \geq \max\{1 - \frac{\lambda_1}{\lambda_n}, 1 - \frac{\lambda_n}{\lambda_2}\}$. If $n \leq 28$, we verify this by computer checking. If $\lambda_2 < 2$, then either it is equal to 1, or G is the conference graph (and thus n<25). Else, strongly regular graphs with $\lambda_2 = 1$ were proven by Seidel [Sei75] to satisfy $n \leq 28$, be a ladder, complement of a lattice, of complement of a triangular graph, all of which satisfy the bound on $\gamma(G)$. Finally, let $\lambda_2 \geq 2$. If G is imprimitive, the result follows, so assume G is primitive. Then, if the result did not hold, we have $\lambda_1 < n$, and $f_n\lambda_n + (n - 1 - f_n\lambda_2 + \lambda_1 = 0$ implies

$$\begin{aligned} f_n^2 &< -\frac{f_n\lambda_n}{\lambda)2} \\ &= n - 1 - f_n + \frac{\lambda_1}{\lambda_2} \\ &< \frac{3}{2}n - f_n \end{aligned}$$

So $f_n^2 + 3f_n < \frac{3}{2}n + 2\sqrt{\frac{3}{2}n}$, thus $n < 24$, we have a contradiction, and our bound on $\gamma(G)$ follows.

Then, we can deduce (1) and (2).

Since G is primitive, $0 < q = d - \lambda_2\lambda_n$, so by (1) we have $\gamma(G) - 1 \geq -\frac{d}{\lambda_n} > \lambda_2$. □

Lemma 5.6 (Haemers [Hae79])**.** *For a strongly regular graph G, we have* $q - d = \lambda_2\lambda_n$

Theorem 5.7 (Haemers [Hae79])**.** *For any* $n \in \mathbb{N}$*, the number of primitive strongly regular graphs with chromatic number n is finite.*

Proof. If G is primitive, then $q \geq 1$, and so by 5.5

$$n \leq nq = (d - \lambda_2)(d - \lambda_n) \leq d(d - \lambda_n) \leq d(d - \lambda_n) < \gamma(\gamma - 1)^5.$$

□

5.3 Moore Graphs

We provide a well known result about Moore Graphs, from Hoffman and Singleton. This shows the relation between diameter, girth, and degree, as well as an example on observing that certain families of graphs are finite, if a strong enough restriction is imposed.

Definition 37. *The girth of a graph is the length of its smallest subgraph that is a cycle.*

Lemma 5.8. *A diameter 2 graph cannot have girth 6 or larger.*

Proof. Say a diameter 2 graph has girth 6, and find the smallest cycle. Then, consider antipodal vertices. These vertices must be at most two two steps away,

else our graph is diameter 2. If they share an edge, we have found a C_4, if they share a common neighbor we have found a C_5, either way, the assumption on the graph having girth 6 is contradicted. Similar arguments can be made for larger girth graphs. □

Theorem 5.9. *A diameter 2 graph with maximum degree Δ cannot have more than $1 + \Delta^2$ vertices. This bound is called the* Moore bound.

Proof. Find the maximal degree vertex v, then it has Δ neighbors. Furthermore, each of its neighbors can have at most Δ neighbors, yielding at most Δ^2 vertices in $N[N(v)]$. Then, we have $\Delta^2 + 1$ vertices in $N[N[v]] = G$. □

Theorem 5.10 (Hoffman and Singleton [HS60])**.** *Every graph with diameter 2 and achieves the Moore bound is girth 5 is called a* Moore graph*, it is k-regular with $k^2 + 1$ vertices, with $k \in \{2, 3, 7, 57\}$. Furthermore, the existence of $k = 57$ is a mystery.*

5.4 Mycielskian

The Mycielskian is a construction that preserves many structural properties of a graph, notably diameter, while not preserving others, notable $k-$colorability. Systematic constructions with provable properties can be a useful tool for considering large families of graphs. In this subsection, we explore and prove certain properties relating to the Mycielskian.

Definition 38. *The Mycielskian M(G) of a graph G is a construction obtained by adding n+1 vertices such that, if the original vertices are $v_1, \ldots, v_n$, we have for each edge v_iv_j new edges u_iv_j and u_jv_i. Call $u_1 \ldots, u_n$ auxiliary vertices. Finally, connect the n+1th new vertex to all auxiliary vertices.*

As an example, we show the Mycielskian of C_4.

Theorem 5.11. *If G is diameter 2, then M(G) is as well.*

Proof. Consider all vertices in $G \leq M(G)$. Then, these can reach all vertices in G in 2 steps by assumption, and all new auxiliary vertices by either directly traveling to it or by traveling through a common vertex. Furthermore, they can reach the final new vertex by traveling through any auxiliary vertices. The new vertices form a star, so they can all reach each other as well. □

Theorem 5.12. *The Mycielskian construction increases the chromatic number.*

Proof. Color $G \leq M(G)$ as you normally would. Then, we must use all of the colors in the auxiliary vertices, and the final new vertex increases the chromatic number by 1. □

Theorem 5.13 (Mycielski [Myc55])**.** *The Mycielskian construction does not increase the clique number of the graph.*

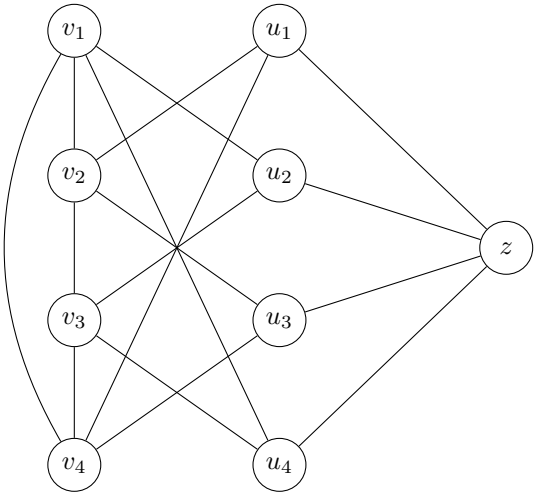

Figure 4: Mycielskian of C_4

Proof. The only new triangles must be of the form $v_iv_ju_k$, where $v_iv_jv_k$ is a triangle in G. □

Lemma 5.14. *If $G \subset G'$, then $M(G) \subset M(G')$.*

Proof. This follows from the construction of $M(G)$. □

5.5 C_i and P_j Freeness

Coloring diameter 2 graphs without certain subgraphs, such as small cycles or paths, is an avenue that has shown promise. Results for polynomial time 3 colorings for some such families has been proven, and is likely to be possible for further families.

We consider an *induced path* P_j to be a path such that no vertices in the path share an edge with other vertices in the path that they are not adjacent to with respect to the path. Likewise, in an induced cycle C_i we cannot have any chords, or edges between vertices that are not adjacent with respect to the cycle. A P_j free graph is a graph that does not include any induced P_j. To gain understanding of this, note that a P_2 free diameter two graph must be K_n for some n, since the P_2 free condition forbids vertices sharing a neighbor without sharing an edge, however all vertices must share a neighbor or share an edge, and thus all vertices share an edge.

Polynomial time colorings is open for P_t free graphs when $t \geq 8$. For $t \geq 2$, we have that P_t free graphs are a subclass of $C_{<t}$ free graphs (both due to Matin et al). Rojas and Stein prove polynomial time coloring is possible for diameter 2 graphs. It is also polynomial time solvable for the following diameter 2 graphs:

- $(C^{odd}_{<t-3}, P_t)$ free [Rojas and Stein [RS20]]

- (C_3, C_4)-free [Martin et al. [MPS19]]
- $K_{2,1,r}$-free for every $r \geq 1$ [Martin et al. [MPS19]]
- $S_{1,2,2}$-free [Martin et al. [MPS19]]
- C_5 free [Martin et al. [MPS21]]
- C_6 free [Martin et al. [MPS21]]
- (C_4, C_t) free for $t \in \{3, 5, 6, 7, 8, 9\}$ [Martin et al. [MPS21]]
- (C_4, C_t) free for $t \geq 10$ [Klimosova and Sahlot [KS23]]

Note that (C_3, C_s) with $s \geq 8$ remains open. Proving polynomial time coloring for diameter 2 graphs without small cycles generally works as follows:

1. Assert the existence of some cycle that is not forbidden
2. Color this cycle
3. Safely reduce the possible colors for the remainder of the graph, starting with its direct neighbors
4. Make observations about graph structure; construct more safe coloring rules
5. Conclude that the remainder of the graph must be easy to color

We investigate coloring a (C_3, C_8) free graph G using this method. Then, we can find an induced C_7, as otherwise the graph would be C_3, C_7 free, and coloring it could be done in polynomial time. While we do not complete this proof, we provide useful observations that can guide next steps.

We have three cases for coloring this graph up to permutation. Without a loss of generality, let us color the first vertex a. Then, if there is only one vertex colored a, then there is one possible coloring - alternating b and c. If there are two vertices colored a, we have two possibilities up to permutation - either the third vertex is also colored a, or the fourth is. We examine each of the three cases below. Let $L(v) \subset \{a, b, c\}$ be the set of colors that v could be. In each case, we consider $N_1 = \{v \in N(C_7) \Big| |L(v)| \leq 2\}$, $L_3 = \{v \in N(N_1) \Big| |L(v)| = 3\}$, so N_1 are all neighbors of the cycle with list 2 colors, and L_3 are all neighbors of neighbors of the cycle with list 3 colors.

We can apply propogation rules as described by [MPS19], [KS23]. These rules will reduce the size of $L(v)$ for certain vertices, getting us closed to an instance of LIST-2 coloring. Our goal to reduce the size of L_3 to less than $\log(n)$, or otherwise make it easy to color.

We refer to $A_i, B_i, C_i \subset N_1$ where the subscript is a string that denotes which vertices in our C_7 they are connected to, so for example A_1 is connected to vertex 1 in the cycle, and C_{357} is connected to vertices $3, 5, 7$. We have $A = \cup_{i \in I} A_i$, and similar for B, C. All A_i, B_i, C_i are independent sets (by the triangle free

constraint). Furthermore, there are no edges between $A_i, A_j \quad i \cap j \neq \emptyset$ by the same constraint. We also have that each A_i is not connected to any 1-colored vertex that is not colored a, similar for B_i, C_i. Generally, if two vertices share a neighbor, they cannot be connected (triangle free).

If we can prove that L_3 has a dominating set of size $\log(n)$, then we can brute force color this in linear time, and can 3 color each coloring instance in polynomial time.

Observe that If A_i, A_j has $|i| = |j| = 1$, and $|i - j| = 2$, then A_i, A_j do not share edges by the C_8 free constraint. Similar for B, C. We provide observations for the three possible cases of an initial coloring of C_7.

1. One vertex colored a
   Let $C_7 = (1, 2, 3, 4, 5, 6, 7, 1)$, and let $c(1) = a, c(2) = b, c(3) = c, c(4) = b, c(5) = c, c(6) = b, c(7) = c$.

   (a) We know the following:
      i. $B_4 = \emptyset$
      ii. $C_5 = \emptyset$

   (b) There are no edges between:
      i. A, C_{3x}
      ii. A_1, B_6
      iii. $B_2,\ B_4$
      iv. C_5, C_7

   (c) All vertices in L_3 must connect to $A_1 = A$.

   (d) No vertices connect to C_{57}, C_5, B_4, B_{24}

   (e) Each vertex has at most one neighbor in B_{26}, C_{57}.

   (f) Let $av \in E$. If $vv' \in E$ and $v'c_{35} \in E$, then $c_{35}a \in E$

2. Two vertices colored a, with two vertices between them.
   Let $C_7 = (1, 2, 3, 4, 5, 6, 7, 1)$, and let $c(1) = a, c(2) = b, c(3) = a, c(4) = b, c(5) = c, c(6) = b, c(7) = c$.

   (a) All vertices in L_3 have at most one edge in each of $B_{24}, B_{46}, A_{13}, C_{57}$.

3. Two vertices colored a, with three vertices between them.

   Let $C_7 = (1, 2, 3, 4, 5, 6, 7, 1)$, and let $c(1) = a, c(2) = b, c(3) = c, c(4) = a, c(5) = b, c(6) = c, c(7) = b$. We have the following observations, noting the symmetry between A and C:

   (a) Each vertex in A has an edge to a vertex in C, and vice versa.

   (b) Each vertex in L_3 cannot have an edge in both A_1 and A_4, so it must have at least one in A_{14}.

   (c) Each vertex in L_3 cannot have an edge in both C_3 and C_6, so it must have at least one in C_{36}.

(d) Each induced $P_2 = vv'v'' \subset L_3$ completes a C_4, with v, v'' sharing a neighbor in A_{14} or C_{36} (by C_8 free requirement combined with triangle free).

(e) Whichever neighbor v, v'' share in A_{14} or C_{36} cannot have an edge shared with neighbors of any other neighbors of v, v''.

(f) If $(a, c) \in E$, we cannot have a P_2 with (v, a) or (v'', c).

(g) There are $\sum_{v \in L_3} \binom{deg(v)}{2}$ P_2s in L_3.

Ideally, this can help us institute a relation between the number of edges in L_3, and how difficult L_3 is to color.

We hope these observations can inform future consideration of this problem, whether by the author or others.

5.6 Counting Forbidden Subgraphs

In this section, we prove that there is not a finite family of forbidden subgraphs. In doing this, we provide an example of a graph that resists the coloring techniques we mentioned earlier, despite seeming to be a prime candidate for spectral algorithms.

Theorem 5.15 (Pan, Stefankovic)**.** *There is an infinite family of forbidden diameter two subgraphs that force a 4 or higher chromatic number.*

Proof. Consider a diameter 2 graph G with 3^k vertices. Let $V = \{\{0, 1, 2\}^k\}$, and let our edge set satisfy $E = \{\{a, b\} \big| \ \forall i \ \ a_i \neq b_i\}$. This graph is 3 colorable and diameter 2 with $3k$ colorings. To see that G satisfies the diameter 2 constraint, consider two vertices without an edge u, v. Then, they differ by at least one coordinate, however there is at least one third node w that differs from all coordinates from both nodes, so there is an edge between u, w and w, v. To find a coloring, partition all vertices by their ith coordinate, then each set will be independent. Next we generate forbidden subgraphs. We add k edges between vertices sharing a_i, b_i, and the ith edge removes the ith coloring. Each edge addition creates a graph that is not a subgraph of prior graphs, and we can do this for any k, creating an arbitrarily large family. □

Coloring our original graph with no additional edges added resists spectral methods and seed based coloring.However, we provide a combinatorial argument to find the original structure.

Given an arbitrary graph that has been formed in the above manner, we can efficiently recover the original graph with the following algorithm. We use the terms string and vertex interchangeably. Say we have 3^k vertices. First, pick and arbitrary triangle. Set the vertices to be $\bar{0}, \bar{1}, \bar{2}$. Observe that $N(\bar{0})$ is all strings in $\{1, 2\}^k$, and we find other neighborhoods similarly.

Next, pick an arbitrary string s in $N(\bar{0})$, and note that we can deduce the number of 1s and 2s it contains by counting the number of edges it shares with

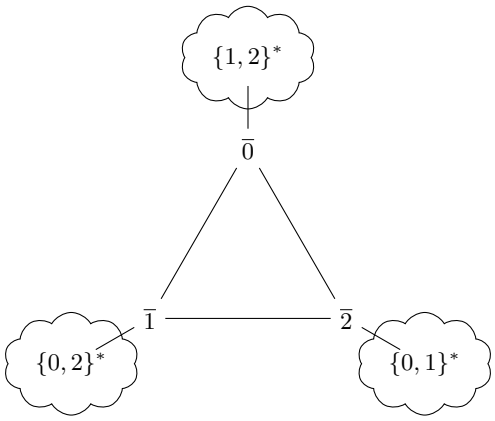

Figure 5: Initial Declaration as a subset of our graph

$N(\bar{1}), N(\bar{2})$.Note that all vertices in $N(\bar{1})$ sharing an edge with s have a 0 where s has a 2. Then, each digit that is a 1 in s can be either a 0 or 2 in a vertex in $N(\bar{1})$ that shares an edge with s. From this, we can conclude that if s has 2^l edges with $N(\bar{1})$, then s has l digits that are 1, and $n-1$ digits that are 2.

Find all vertices in $N(\bar{0})$ and $N(\bar{2})$ that have exactly one digit that is a 1. Fix some vertex v in $N(\bar{0})$ that satisfies this condition. Note that there will be exactly one vertex $u \in N(\bar{2})$ with only one 1 that does not share an edge with v. Without a loss of generality, designate that both of these vertices have a 1 in their first digit. Refer to 6. We can do this since the order of the digits does not matter, only if strings are equal or are not equal with respect to a particular digit matters. Continue this process for all pairs of vertices that have exactly one of some number in their string representation. Using this method, we can identify all strings with a number that appears in exactly one digit. Consider that we have found all strings $\{100\bar{0}, 010\bar{0}, 001\bar{0}, \dots\}$. From this, we can find the string $110\bar{0}$. Observe that it is the only string in $N(\bar{2})$ with two 1s that does not share an edge with $100\bar{0}\dots$ or $010\bar{0}\dots$. We can similarly identify all strings in $N(\bar{0}), N(\bar{1}), N(\bar{2})$. Then, we can deduce the remaining graph, and color it as described in the proof of 5.15.

We also conjecture a spectral algorithm that works for certain eigenbases, namely up to permutation of the standard eigenbasis formed from ordering the strings in ascending order with respect to ternary representation before calculating eigenspaces. This has empirical correctness, although we do not provide a proof. The algorithm is described below:

1. Find eigenvectors e_{3^k}, e_{3^k-1} with respect to the smallest eigenvalue $(-1)2^{k-1}$. Observe that these are in $\{-1, 0, 1\}^*$.

2. Select two such eigenvectors that do not have any -1s in the same digit. Then, if the vth digit of e_{3^k} is -1, color v red. Similarly, color digits of e_{3^k-1} with value -1 blue. This will give 2/3 of class red, blue.

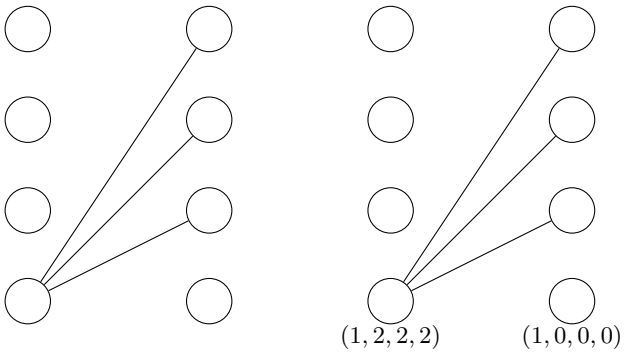

Figure 6: Selecting an ordering of digits

3. Use the adjacency matrix to find vertices that have both red vertices and yellow vertices as neighbors, color these yellow

4. Observe that we are left with an instance of LIST-2 coloring.

Note that neither of these algorithms are robust to adding edges. This provides examples of hard diameter 2 graphs that resist our algorithms, and we leave coloring algorithms for these graphs as an open problem for future works.

6 Conclusion

We investigate coloring of diameter 2 graphs and related interesting results via spectral and combinatorial approaches. We provide results relating to spectral partitioning, seed coloring, and graph construction, and outline existing results relating to spectral theory, Hamiltonian cycles in line graphs, dominating sets, and families of diameter 2 graphs.

As open problems, we leave Aspvall and Gilbert's conjecture on partitioning negative eigenvalues as an interesting direction. Proving it for all Cayley graphs of Abelian groups would be a nice stepping stone. We also leave (C_3, C_t) freeness for $t \geq 8$, and hope that our observations are insightful. Finally, we leave coloring our difficult example from the final section as a problem that can guide robust spectral coloring algorithms.

7 Acknowledgments

I would to thank my advisor Daniel Stefankovic for introducing me to theoretical computer science through both engaging courses and by guiding me during this project, with an approach to problems and problem solving that ensures our work is fun, valuable, and interesting. For his insight into graph theory and infectious enthusiasm for mathematics, I would like to my co advisor Jonathan

Pakianathan. I would also like to thank Alex Iosevich for serving on my honors committee and always providing equal parts laughter and insight. I would like to thank Yekai Pan, who worked alongside myself and Daniel for the past year as we tackled this problem, providing companionship as we both understood and did not understand various concepts together. I would like to thank my peers Kumar Anuraag, Jacob Trokel, and Leo Sciortino for humouring my near-constant discussion of this paper for the previous few weeks and providing valuable advice and friendship for the duration of my undergraduate career. I would like to thank my buddy from home Matt, who has bean asking for a reference in a paper I write. I would like to thank my sister, who has had little to no impact on my mathematics career, but is cool. Finally, I would like to thank my parents for raising me and encouraging a pursuit of mathematics since I was young. While I am the sole author on this paper, this work is not all my own, and I owe much to everyone who I've interacted with throughout my undergraduate studies.

References

[AG84] Bengt Aspvall and John R. Gilbert. Graph coloring using eigenvalue decomposition. *SIAM Journal on Algebraic and Discrete Methods*, 5(4):526–538, 1984.

[AK97] Noga Alon and Nabil Kahale. A spectral technique for coloring random 3-colorable graphs. *SIAM Journal on Computing*, 26(6):1733–1748, 1997.

[BDPR24] Valentin Bouquet, François Delbot, Christophe Picouleau, and Stéphane Rovedakis. On the complexity of dominating set for graphs with fixed diameter. *Theoretical Computer Science*, 956:114561, 2024.

[CDS95] Dragoš M. Cvetković, Michael Doob, and Horst Sachs. *Spectra of Graphs: Theory and Applications*. Academic Press, 3rd edition, 1995. First published in 1980; third edition includes revisions and additions.

[CRS96] Dragoš Cvetković, Peter Rowlinson, and Slobodan Simić. A spectral generalization of line graphs on graphs with least eigenvalue -2. *Journal of Graph Theory*, 23(3):233–240, 1996.

[Dam90] Peter Damaschke. Induced subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 14(4):427–435, 1990.

[Ha07] Michael A. Henning and Anders Yeo and. A transition from total domination in graphs to transversals in hypergraphs. *Quaestiones Mathematicae*, 30(4):417–436, 2007.

[Hae79] Willem H. Haemers. *Eigenvalue techniques in design and graph theory*. PhD thesis, Eindhoven University of Technology, 1979.

[HS60] Alan J. Hoffman and Robert R. Singleton. On moore graphs with diameters 2 and 3. *IBM Journal of Research and Development*, 4(5):497–504, 1960.

[HT91] Wen-Lian Hsu and Kuo-Hui Tsai. Linear time algorithms on circular-arc graphs. *Information Processing Letters*, 40(3):123–129, 1991.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Richard E. Miller, editor, *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[KS23] Tereza Klimošová and Vibha Sahlot. 3-coloring c_4 or c_3-free diameter two graphs. In Pat Morin and Subhash Suri, editors, *Algorithms and Data Structures*, volume 14079 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2023.

[MPS19] Barnaby Martin, Daniel Paulusma, and Siani Smith. Colouring h-free graphs of bounded diameter. In *Proceedings of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.

[MPS21] Barnaby Martin, Daniël Paulusma, and Siani Smith. Colouring graphs of bounded diameter in the absence of small cycles. In Tiziana Calamoneri and Federico Corò, editors, *Algorithms and Complexity: 12th International Conference, CIAC 2021, Proceedings*, volume 12701 of *Lecture Notes in Computer Science*, pages 367–380. Springer, 2021.

[MPvL20] B. Martin, D. Paulusma, and E. J. van Leeuwen. Disconnected cuts in claw-free graphs. *Journal of Computer and System Sciences*, 113:60–75, 2020.

[Myc55] J. Mycielski. Sur le graphe de l'ordre d'un graphe donné. *Colloquium Mathematicum*, 3:161–162, 1955.

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

[RS20] Alberto Rojas and Maya Stein. 3-colouring p_t-free graphs without short odd cycles, 2020. Accessed: 2025-05-06.

[Sei75] R. Seidel. Strongly regular graphs with $\lambda_2 = 1$. *Journal of Combinatorial Theory, Series B*, 18(1):1–13, 1975.

[Spi25] Daniel A. Spielman. Spectral and algebraic graph theory. http://cs-www.cs.yale.edu/homes/spielman/sagt/sagt.pdf, 2025. Incomplete Draft, April 2, 2025. Current version available at http://cs-www.cs.yale.edu/homes/spielman/sagt.

[Vel88] H. J. Veldman. A result on hamiltonian line graphs involving restrictions on induced subgraphs. *Journal of Graph Theory*, 12(3):413–420, 1988.

[WD16] David P. Williamson and Kun Dong. Orie 6334 spectral graph theory lecture 4: Eigenvalue interlacing theorem. https://people.orie.cornell.edu/dpw/orie6334/lectures/lec4.pdf, 2016. Lecture notes, Cornell University.

A Group Theory

A.1 Groups

Definition 39. *A group is a set G equipped with a binary operation $\cdot$ (often written as multiplication or addition) satisfying the following:*

- ***Closure:*** *For all $a, b \in G$, the result $a \cdot b \in G$.*
- ***Associativity:*** *For all $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*
- ***Identity:*** *There exists an element $e \in G$ such that for all $a \in G$, $e \cdot a = a \cdot e = a$.*
- ***Inverses:*** *For all $a \in G$, there exists $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.*

Definition 40. *A subset $S \subseteq G$ is called a generating set if every element of G can be written as a product of elements from S and their inverses.*

A.2 Cayley Graph

Given a finite group G and a subset $S \subseteq G$ that generates G, we have the Cayley graph $\mathrm{Cay}(G, S)$ is a graph defined as follows:

- Each group element $g \in G$ is a vertex.
- For each $g \in G$ and each $s \in S$, draw a directed edge from g to $g \cdot s$.
- If S is symmetric (i.e., $s \in S \Rightarrow s^{-1} \in S$), the graph can be considered undirected.

We provide an example, the Cayley Graph of $\mathbb{Z}_4$

Let $G = \mathbb{Z}_4 = \{0, 1, 2, 3\}$, the integers modulo 4 under addition. Let $S = \{1\}$, which generates G. We have vertices as $\{0, 1, 2, 3\} = G$, and edges defined as follows: for each $g \in \mathbb{Z}_4$, add a directed edge from g to $g + 1 \mod 4$. This gives a directed cycle on 4 vertices.

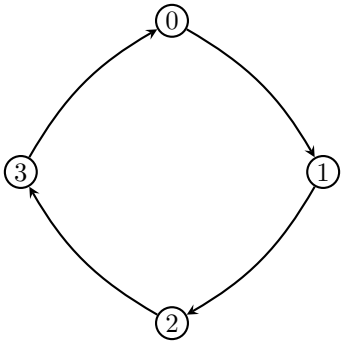

Figure 7: Cayley Graph for $\mathbb{Z}_4$ generated by $\{1\}$

B Code

We include code for various algorithms discussed in the body of the paper. Note that various helper functions are omitted.

B.1 Alon Kahale Algorithm

```
    import org.apache.commons.math3.linear.EigenDecomposition;
import org.apache.commons.math3.linear.RealVector;

import java.util.*;

    public static int[][] preprocessMat(int[][] matrix, double
        threshold) {
        int n = matrix.length;
        List<Integer> toRemove = new ArrayList<>();

        // Identify rows (and corresponding columns) to be
            removed
        for (int i = 0; i < n; i++) {
            int count = 0;
            for (int j = 0; j < n; j++) {
                if (matrix[i][j] > 0) {
                    count++;
                }
            }
            if (count > threshold) {
                System.out.println("Vertex " + i + " slated for
                     removal");
                toRemove.add(i);
            }
        }

```

```
        // Create a new matrix with the filtered rows and columns
        int newSize = n - toRemove.size();
        int[][] newMatrix = new int[newSize][newSize];
        int newRow = 0;

        for (int i = 0; i < n; i++) {
            if (toRemove.contains(i)) continue;
            int newCol = 0;
            for (int j = 0; j < n; j++) {
                if (toRemove.contains(j)) continue;
                newMatrix[newRow][newCol] = matrix[i][j];
                newCol++;
            }
            newRow++;
        }

        return newMatrix;
    }


    public static RealVector get_t(RealVector e3n_1, RealVector e3n, int n) {
        RealVector t = e3n_1; // Start with  e 3 n 1
        double alpha = 1.0, beta = 1.0; // Coefficients for linear combination

        while (true) {
            t = e3n_1.mapMultiply(alpha).add(e3n.mapMultiply(beta));

            double[] components = t.toArray();
            java.util.Arrays.sort(components);
            double median = components[n / 2];

            if (Math.abs(median - 0.0) < 1e-4) break;
            if (median > 0) {
                beta -= 0.01;
            } else {
                alpha -= 0.01;
            }
        }
        t = t.mapDivide(t.getNorm()).mapMultiply(Math.sqrt((double) (2 * e3n.getDimension()) / 3));
        return t;
    }


    public static Partition phase1(RealVector t) {
```

```
        Partition partition = new Partition();

        for (int i = 0; i < t.getDimension(); i++) {
            double value = t.getEntry(i);

            if (value > 0.5) {
                partition.V1.add(i);
            } else if (value < -0.5) {
                partition.V2.add(i);
            } else {
                partition.V3.add(i);
            }
        }
        return partition;
    }

    public static Partition phase2(int[][] matrix, Partition
        partition) {
        List<Partition> partitions = new ArrayList<>();
        partitions.add(partition); // V^0_1, V^0_2, V^0_3

        for (int i = 1; i < Math.ceil(Math.log(matrix.length));
             i++) {
            Partition prevPartition = partitions.get(i - 1);
            Partition currentPartition = new Partition();
            for (int v = 0; v < matrix.length; v++) {
                int countV1 = countNeighborsInSet(matrix, v,
                    prevPartition.V1);
                int countV2 = countNeighborsInSet(matrix, v,
                    prevPartition.V2);
                int countV3 = countNeighborsInSet(matrix, v,
                    prevPartition.V3);
                // Find the least popular color class
                if (countV1 < countV2 && countV1 < countV3) {
                    currentPartition.V1.add(v);
                } else if (countV2 < countV1 && countV2 <
                    countV3) {
                    currentPartition.V2.add(v);
                } else if (countV3 < countV1 && countV3 <
                    countV2) {
                    currentPartition.V3.add(v);
                } else {
                    ArrayList<Integer> tiedIndices = new
                        ArrayList<>();
                    if (countV1 == countV2 && countV1 ==
                        countV3) {
                        tiedIndices.add(1);
                        tiedIndices.add(2);
                        tiedIndices.add(3);
                    } else if (countV1 == countV2) {
```

```
                        tiedIndices.add(1);
                        tiedIndices.add(2);
                    } else if (countV1 == countV3) {
                        tiedIndices.add(1);
                        tiedIndices.add(3);
                    } else if (countV2 == countV3) {
                        tiedIndices.add(2);
                        tiedIndices.add(3);
                    }
                    // Select randomly from the tied options
                    if (!tiedIndices.isEmpty()) {
                        Random random = new Random();
                        int selected = tiedIndices.get(random.
                            nextInt(tiedIndices.size()));
                        switch (selected) {
                            case 1:
                                currentPartition.V1.add(
                                    selected);
                                break;
                            case 2:
                                currentPartition.V2.add(
                                    selected);
                                break;
                            case 3:
                                currentPartition.V3.add(
                                    selected);
                                break;
                            default:
                                System.out.println("No Color
                                    Class Selected in tiebreaker
                                    ");
                                currentPartition.V1.add(
                                    selected);
                                break;
                        }
                    }
                }
            }
            partitions.add(currentPartition);
        }
        return partitions.getLast();

    }

    public static int countNeighborsInSet(int[][] matrix, int
        vertex, Set<Integer> colorClass) {
        int count = 0;
        for (int neighbor = 0; neighbor < matrix.length;
            neighbor++) {
```

```
            if (matrix[vertex][neighbor] > 0 && colorClass.
                contains(neighbor)) {
                count++;
            }
        }
        return count;
    }

    public static boolean phase3(int[][] matrix, Partition
        partition, int threshold) {
        int n = matrix.length;
        // Repeatedly uncolor vertices
        boolean changed;
        do {
            changed = false;
            for (int j = 1; j <= 3; j++) { // Iterate over the
                three sets in the partition
                Set<Integer> vertices = partition.getSet(j);
                Set<Integer> toUncolor = new HashSet<>();
                for (int v : vertices) {
                    boolean shouldUncolor = false;
                    for (int l = 1; l <= 3; l++) {
                        if (l != j) {
                            int countNeighborsL =
                                countNeighborsWithColor(matrix,
                                partition.getSet(l), v);
                            if (countNeighborsL < threshold /
                                2) {
                                shouldUncolor = true;
                                break;
                            }
                        }
                    }
                    if (shouldUncolor) {
                        toUncolor.add(v);
                        changed = true;
                    }
                }
                for (int v : toUncolor) {
                    partition.remove(v); // Remove uncolored
                        vertices from the partition
                }
            }

        } while (changed);

        //get indices of all uncolored vertices
        Set<Integer> uncoloredVertices = getUncoloredVertices(
            partition, n);
```

```
191         int[][] uncolored_part = new int[uncoloredVertices.size
                ()][uncoloredVertices.size()];
192         int newRow = 0;
193 
194         for (int i = 0; i < n; i++) {
195             if (!uncoloredVertices.contains(i)) continue;
196             int newCol = 0;
197             for (int j = 0; j < n; j++) {
198                 if (!uncoloredVertices.contains(j)) continue;
199                 uncolored_part[newRow][newCol] = matrix[i][j];
200                 newCol++;
201             }
202             newRow++;
203         }
204 
205         List<Set<Integer>> components = findConnectedComponents
                (uncolored_part);
206         int log3n = (int) Math.ceil(Math.log(n) / Math.log(3));
207         for (Set<Integer> component : components) {
208             //If there is a strongly connected component that
                    is too large, give up
209             if (component.size() > log3n) {
210                 System.out.println(" Component: " + component +
                            " is too large");
211                 return false;
212             }
213             //If you cannot brute force color, give up
214             if (!bruteForceColorComponent(matrix, component,
                    partition)) {
215                 System.out.println("No brute force coloring");
216                 return false;
217             }
218         }
219         partition.print_partition();
220         System.out.println("Is valid partition? " + partition.
                is_valid_col(matrix));
221         return true;
222     }
223 
224     public static Set<Integer> getUncoloredVertices(Partition
            partition, int n) {
225         Set<Integer> uncolored = new HashSet<>();
226         for (int i = 0; i < n; i++) {
227             if (!partition.V1.contains(i) && !partition.V2.
                    contains(i) && !partition.V3.contains(i)) {
228                 uncolored.add(i);
229             }
230         }
231         return uncolored;
232     }
```

```
233
234     private static int countNeighborsWithColor(int[][]
            adjacencyMatrix, Set<Integer> colorSet, int vertex) {
235         int count = 0;
236         for (int neighbor = 0; neighbor < adjacencyMatrix.
                length; neighbor++) {
237             if (adjacencyMatrix[vertex][neighbor] > 0 &&
                    colorSet.contains(neighbor)) {
238                 count++;
239             }
240         }
241         return count;
242     }
243
244     private static void dfs(int[][] graph, int v, boolean[]
            visited, Set<Integer> component) {
245         visited[v] = true;
246         component.add(v);
247         // Visit all adjacent vertices
248         for (int i = 0; i < graph.length; i++) {
249             if (graph[v][i] == 1 && !visited[i]) {
250                 dfs(graph, i, visited, component);
251             }
252         }
253     }
254
255     // Function to find all connected components in the graph
256     public static List<Set<Integer>> findConnectedComponents(
            int[][] graph) {
257         int n = graph.length;
258         boolean[] visited = new boolean[n];
259         List<Set<Integer>> components = new ArrayList<>();
260
261         // Perform DFS for every unvisited vertex
262         for (int i = 0; i < n; i++) {
263             if (!visited[i]) {
264                 Set<Integer> component = new HashSet<>();
265                 dfs(graph, i, visited, component);
266                 components.add(component);
267             }
268         }
269         return components;
270     }
271
272     private static boolean bruteForceColorComponent(int[][]
            adjacencyMatrix, Set<Integer> component, Partition
            partition) {
273         int[] colors = {1, 2, 3};
274         List<Integer> vertices = new ArrayList<>(component);
```

```
275         return assignColors(adjacencyMatrix, vertices,
                partition, colors, 0);
276     }
277
278     private static boolean assignColors(int[][] adjacencyMatrix
            , List<Integer> vertices, Partition partition, int[]
            colors, int index) {
279         if (index == vertices.size()) {
280             return true; // All vertices colored successfully
281         }
282
283         int v = vertices.get(index);
284         for (int color : colors) {
285             if (isValidColor(adjacencyMatrix, partition.getSet(
                    color), v)) {
286                 partition.add(v, color);
287                 if (assignColors(adjacencyMatrix, vertices,
                        partition, colors, index + 1)) {
288                     return true;
289                 }
290                 partition.remove(v); // Backtrack
291             }
292         }
293         return false; // No valid coloring found
294     }
295
296     private static boolean isValidColor(int[][] adjacencyMatrix
            , Set<Integer> colorSet, int vertex) {
297         for (int neighbor = 0; neighbor < adjacencyMatrix.
                length; neighbor++) {
298             if (adjacencyMatrix[vertex][neighbor] > 0 &&
                    colorSet.contains(neighbor)) {
299                 return false; // Conflict with neighbor
300             }
301         }
302         return true;
303     }
304
305     //given a preprocessed matrix, will run the alon-kahale
            algorithm.
306     public static void alon_alg(int[][] matrix, double
            threshold) {
307         EigenDecomposition eigenDecomposition = Main.calc_es(
                matrix);
308         double[] evals = eigenDecomposition.getRealEigenvalues
                ();
309         RealVector t = get_t(eigenDecomposition.getEigenvector(
                matrix.length - 2), eigenDecomposition.
                getEigenvector(matrix.length - 1), evals.length);
310         Partition initial = phase1(t);
```

```
311         initial.print_partition();
312         Partition second = phase2(matrix, initial);
313         System.out.println("Alon alg result: " + phase3(matrix,
                second, (int) (threshold / 2.0)));
314     }
315
316
317
318     public static class Partition {
319         public Set<Integer> V1 = new HashSet<>();
320         public Set<Integer> V2 = new HashSet<>();
321         public Set<Integer> V3 = new HashSet<>();
322
323         public Set<Integer> getSet(int j) {
324             switch (j) {
325                 case 1:
326                     return V1;
327                 case 2:
328                     return V2;
329                 case 3:
330                     return V3;
331                 default:
332                     System.out.println("Empty partition,
                        algorithm failed");
333                     System.exit(0);
334                     return null;
335             }
336         }
337
338         public void print_partition() {
339             System.out.print("Partition 1: [");
340             for (Integer i : V1) {
341                 System.out.print(i + ", ");
342             }
343             System.out.println("]");
344             System.out.print("Partition 2: [");
345             for (Integer i : V2) {
346                 System.out.print(i + ", ");
347             }
348             System.out.println("]");
349             System.out.print("Partition 3: [");
350             for (Integer i : V3) {
351                 System.out.print(i + ", ");
352             }
353             System.out.println("]");
354         }
355
356         public void remove(int v) {
357             if (V1.contains(v)) {
358                 V1.remove(v);
```

```
            } else if (V2.contains(v)) {
                V2.remove(v);
            } else if (V3.contains(v)) {
                V3.remove(v);
            } else {
                System.out.println("Tried to remove an invalid
                    vertex from partition");
            }
        }

        public void add(int v, int color) {
            switch (color) {
                case 1:
                    V1.add(v);
                    break;
                case 2:
                    V2.add(v);
                    break;
                case 3:
                    V3.add(v);
                    break;
                default:
                    System.out.println("Tried to add an invalid
                         vertex to partition");
            }
        }

        public boolean is_valid_col(int[][] mat) {
            for (Integer i : V1) {
                for (Integer j : V1) {
                    if (!Objects.equals(i, j)) {
                        if (mat[i][j] == 1) {
                            return false;
                        }
                    }
                }
            }
            for (Integer i : V2) {
                for (Integer j : V2) {
                    if (!Objects.equals(i, j)) {
                        if (mat[i][j] == 1) {
                            return false;
                        }
                    }
                }
            }
            for (Integer i : V3) {
                for (Integer j : V3) {
                    if (!Objects.equals(i, j)) {
                        if (mat[i][j] == 1) {
```

```
                            return false;
                        }
                    }
                }
            }

            return true;
        }


    }
```

B.2 Difficult Example Algorithm

```

    //given eigenvectors corresponding to the most negative
        eigenvalue, return only those in {-1,0,1}^*
    public static int[][] preproc_evs(int[][] ev){
        ArrayList<int[]> good = new ArrayList<>();
        for (int[] ints : ev) {
            boolean bad = false;
            for (int j = 0; j < ev[0].length; j++) {
                if (!(ints[j] == 0) && !(ints[j] == 1) && !(
                    ints[j] == -1)) {
                    bad = true;
                    break;
                }
            }
            if (!bad) {
                good.add(ints);
            }
        }
        int[][] goodtogo = new int[good.size()][ev.length];
        for(int i = 0; i<good.size(); i++){
            goodtogo[i] = good.get(i);
        }
        return goodtogo;
    }

        //given a preprocessed set of eigenvectors, return two
            that do not overlap -1s.
    public static int[][] find2(int[][] ev){
        for (int i = 0; i<ev.length; i++){
            for(int j = i+1; j<ev.length; j++){
                int[] v1 = ev[i];
                int[] v2 = ev[j];
                boolean next = false;
                for(int k = 0; k<v1.length; k++){
```

```
                    if(v1[k] == -1 && v2[k] == -1){
                        next = true;
                        break;
                    }
                }
                if(next){ break;}
                return new int[][]{v1, v2};
            }
        }
        System.out.println("None found... something is amiss");
        return null;
    }

    //given an adjacency matrix and a list of preprocessed
        eigenvectors, will return a spectral partitioning
    public static  Partition counterex_partition(int[][] mat,
        int[][] ev){
        int[][] evs = Main.find2(ev);
        Partition init = initial_partition(mat, evs);
        Partition second = second_partition(mat, init);
        Partition third = third_partition(mat, second);
        return third;
    }


    public  static Partition initial_partition(int[][] mat, int
        [][] evs){
        Partition partition = new Partition();
        for(int i = 0; i < mat.length; i++){
            if(evs[0][i] == -1){
                partition.add(i, 1);
            }else if(evs[1][i] == -1){
                partition.add(i, 2);
            }
        }
        return partition;
    }

    public  static Partition second_partition(int[][] mat,
        Partition init){
        Partition second = init;
        for(int i = 0; i < mat.length; i++){
            if(!second.V1.contains(i) && !second.V2.contains(i)
                ){
                boolean NOTA = false;
                boolean NOTB = false;
                for(int j = 0; j < mat[i].length; j++){
                    if(mat[i][j] == 1){
                        if(second.V1.contains(j)){
                            NOTA = true;
```

```
                        }
                        if(second.V2.contains(j)){
                            NOTB = true;
                        }
                    }
                    if(NOTA && NOTB){
                        second.add(i, 3);
                    }
                }
            }
        }
        return second;
    }

    public static Partition third_partition(int[][] mat,
        Partition party){
        Partition third = party;
        for (int i = 0; i< mat.length; i++){
            if(!third.V1.contains(i) && !third.V2.contains(i)
                && !third.V3.contains(i)){
                boolean NOTA = false;
                boolean NOTB = false;
                boolean NOTC = false;
                for(int j = 0; j < mat[i].length; j++){
                    if(mat[i][j] == 1){
                        if(third.V1.contains(j)){
                            NOTA = true;
                        }
                        if(third.V2.contains(j)){
                            NOTB = true;
                        }
                        if(third.V3.contains(j)){
                            NOTC = true;
                        }
                    }

                }
                if(NOTA && NOTB && NOTC){
                    System.out.println("Coloring Failed, no
                        color available");
                    return  null;
                }else if(NOTA && NOTB){
                    third.add(i, 3);
                }else if(NOTA && NOTC){
                    third.add(i, 2);
                }else if(NOTB && NOTC){
                    third.add(i, 1);
                }else{
                    System.out.println("List 2");
```

```
123                     //System.out.println("Coloring failed, more
                            than 1 available");
124                 }
125 
126             }
127         }
128         return third;
129     }
```