Determining a Graph's Chromatic Number for Part Consolidation in Axiomatic Design

Jeffery A. Cavallaro
Graduate Student
Mathematics Department
San Jose State University
jeffery.cavallaro@sjsu.edu

October 23, 2019

1 Axiomatic Design

The axiomatic design framework was developed in the late 20^{th} century by Professor Nam P. Suh while at MIT and the NSF [5]. This was in response to concern in the engineering community that *design* was being practiced almost exclusively as an ad-hoc creative endeavor with very little in the way of scientific discipline. In the words of Professor Suh:

> It [design] might have preceding the development of natural sciences by scores of centuries. Yet, to this day, design is being done intuitively as an art. It is one of the few technical areas where experience is more important than formal education.

It is important to note that Professor Suh was not making these claims in an educational vacuum, but in the shadow of several recent major design failures such as the Union Carbide plant disaster in India, nuclear power plant accidents at Three Mile Island and Chenobyl, and the Challenger space shuttle O-ring failure. Furthermore, Professor Suh asserts that design-related issues resulting in production problems and operating failures were increasingly happening in everything from consumer products to big-ticket items.

The following sections provide an overview of axiomatic design as specified in detail by Professor Suh in [5], and summarized in [1, 2, 6]. Following the overview is a description of how an algorithm proposed by this research can be a helpful tool to a designer using the axiomatic design framework.

1.1 Design

Design is defined as the process by which it is determined *what* needs to be achieved and then *how* to achieve it. Thus, the decisions on what do to are just as important as how to do it. Creativity is the process by which experience and intuition are used to generate solutions to perceived needs. This includes pattern matching to and adapting existing solutions and synthesizing new solutions. Thus, creativity plays a vital role in design. Since different designers may approach the same problem differently, their level of creativity may lead to very different, yet plausible, solutions. Thus, there needs to be a design-agnostic method for comparing different designs with the goal of selecting the best one.

This discussion will sound familiar to mathematicians, since creativity is a very important part of solving math problems, and in particular, writing proofs. Starting with the work of Peano in the 19^{th} century, the field of mathematics has established various tests on what constitutes a good proof. For example:

- Does every conclusion result by proper implication from existing definitions, axioms, and previously proved conclusions?
- Is direct proof, contrapositive proof, proof by contradiction, or proof by induction the best approach for a particular problem?
- Do proofs by induction contain clear basic, assumptive, and inductive steps?
- Are all subset and equality relationships properly proved via membership implication?

- Are all necessary cases included and stated in a mutually exclusive manner?
- Are degenerate cases sufficiently highlighted?
- Are all equivalences proved in a proper circular fashion?
- Are key and reused conclusions highlighted in lemmas?

In short, Professor Suh was looking for a similar framework for the more general concept of design.

1.2 The Framework

The *best* design among a set of choices is the design that:

1. exactly satisfies a clearly defined set of needs.
2. has the greatest probability of success.

In a desire to not hinder the creative element needed for design, yet provide some methodology to distinguish bad designs from good designs from better designs, the diagram in Figure 1.2 establishes the overall framework for axiomatic design.

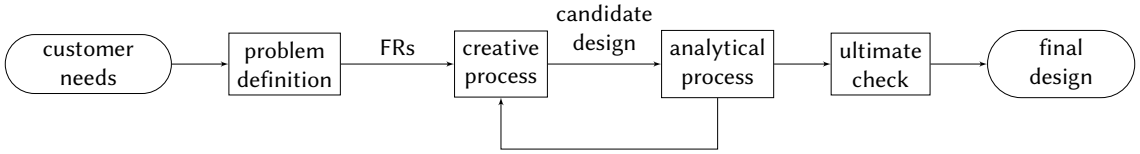


Figure 1: Axiomatic Design Framework

Design starts with the desire to satisfy a set of clear *customer needs*. The term *customer* refers to any entity that expresses needs, and can be as varied as individuals, organizations, or society. The designer, in the *problem definition* phase, determines how these customer needs will be meet by generating a list of *functional requirements* (FRs). It is this list of FRs that determines exactly *what* is to be accomplished.

Once the set of FRs has been determined, the designer begins the *creative process* by mapping the FRs into solutions that are embodied in so-called *design parameters* (DPs). The design parameters contain all of the information on *how* the various FRs are satisfied: parts lists, drawings, specifications, etc. The FRs exist in a design-agnostic *functional space* and the DPs exist in a solution-specific *physical space*. It is the designer's job to provide the most efficient mapping between the two spaces. This process is represented by Figure 1.2.

Simple problems may require only one level of FRs; however, more complicated designs may require a hierarchial structure of FRs, from more general to more detailed requirements. This type of design if often referred to as *top-down* design. Each individual FR layer has its own DP mapping. In fact, the mapping process on one level should be completed prior to determining the FRs for the next level. This is because DP choices on one level may affect requirements on the next level. For example, consider an FR related to a moving part in a design. The DP for this

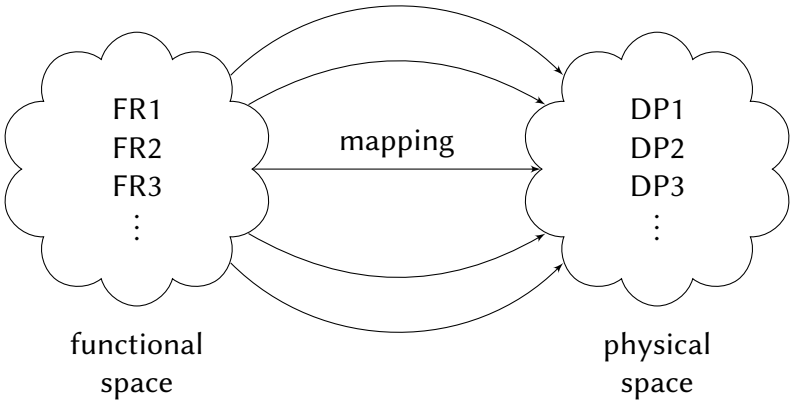


Figure 2: Mapping FRs to DPs

FR could specify that the part be moved manually or automatically. Each choice would result in different FRs for the actual mechanism selected by the DP.

The FR/DP mapping at each level in the design hierarchy is described by the *design equation*, which is shown in Equation 1.

$$[\text{FR}] = [\text{A}][\text{DP}] \tag{1}$$

The design equation is a matrix equation that maps a vector of m FRs to a vector of n DPs via an $m \times n$ design matrix A. As will be shown, good designs require $m = n$, and thus the design matrix is a square $n \times n$ matrix. A full discussion of the design matrix element values is beyond the scope of this research. Instead, we will use the following two values:

$$A_{ij} = \begin{cases} X, & \text{FR}_i \text{ depends on DP}_j \\ 0, & \text{FR}_i \text{ does not depend on DP}_j \end{cases}$$

Since the FR/DP mapping is non-unique, there needs to be a method to compare different plausible designs so that the best design can be selected as the final design. Thus, the framework in Figure 1.2 includes an *analytical process* where designs are judged by a set of axioms, corollaries, and theorems that specify the properties common to all good designs. Once the best design, according to this analysis, is selected, it undergoes an *ultimate check* to make sure that it sufficiently meets all of the customer's needs. If so, then that design is selected as the final design.

1.3 The Axioms

The analytical process is based on two main axioms: the independence axiom and the information axiom. This section describes these axioms and their related corollaries and theorems.

The independence axiom imposes a restriction on the FR/DP mapping:

Axiom 1: Independence

An optimal design always maintains the independence of the FRs. This means that the FRs and DPs are related in such a way that a specific DP can be adjusted to satisfy its corresponding FR without affecting other FRs.

The ideal case is when the design matrix is a diagonal matrix, and so each FR is mapped to and is satisfied by exactly one DP. This is referred to as an *uncoupled* design, which is demonstrated in Figure 1.3. Uncoupled designs completely adhere to the independence axiom.

$$\begin{bmatrix}\text{FR}1\\ \text{FR}2\\ \text{FR}3\end{bmatrix} = \begin{bmatrix}X & 0 & 0\\ 0 & X & 0\\ 0 & 0 & X\end{bmatrix} \begin{bmatrix}\text{DP}1\\ \text{DP}2\\ \text{DP}3\end{bmatrix}$$

Figure 3: An Uncoupled Design

The next best situation is when the design matrix is a lower-triangular matrix. The idea is to finalize the first DPs before moving on to the later DPs. Thus, DP_i can be adjusted without affected FR_1 through FR_{i-1}. This is referred to as a *decoupled* design, which is demonstrated in Figure 1.3. Although decoupled designs do not completely adhere to the independence axiom, they may be reasonable compromises in designs that address complex problems.

$$\begin{bmatrix}\text{FR}1\\ \text{FR}2\\ \text{FR}3\end{bmatrix} = \begin{bmatrix}X & 0 & 0\\ X & X & 0\\ X & X & X\end{bmatrix} \begin{bmatrix}\text{DP}1\\ \text{DP}2\\ \text{DP}3\end{bmatrix}$$

Figure 4: A Decoupled Design

The worst solution is a non-triangular matrix where every change in a DP affects multiple FRs in an unconstrained fashion. This is referred to as a *coupled* design, which is demonstrated in Figure 1.3. Coupled designs are in complete violation of the independence axiom and generally should be decoupled by reworking the FRs or adding additional DPs. Unfortunately, adding additional DPs runs counter to the second axiom.

$$\begin{bmatrix}\text{FR}1\\ \text{FR}2\\ \text{FR}3\end{bmatrix} = \begin{bmatrix}X & X & X\\ X & X & X\\ X & X & X\end{bmatrix} \begin{bmatrix}\text{DP}1\\ \text{DP}2\\ \text{DP}3\end{bmatrix}$$

Figure 5: A Coupled Design

Axiom 2: Information

The best design is a functionally uncoupled design that has the minimum information content.

The *Information* contained in a particular DP is inversely related to the probability that the DP

can successfully satisfy its corresponding FR(s) by Equation 2.

$$I = \log_2 \left(\frac{1}{p} \right) \tag{2}$$

where p is the probability of success and I is measured in bits. This probability must take into consideration such things as tolerances, ease of manufacture, failure rates, etc. The information content of a design is then the sum of the information content of the individual DPs.

From these two axioms come the following seven corollaries:

Corollary 1: Decouple or separate parts or aspects of a solution if the FRs are coupled or become interdependent in the designs proposed.

Corollary 2: Minimize the number of FRs.

Corollary 3: Integrate design features in a single physical part if FRs can be independently satisfied in the proposed solution.

Corollary 4: Use standardized or interchangeable parts if the use of these parts is consistent with the FRs.

Corollary 5: Use symmetrical shapes and/or arrangements if they are consistent with the FRs.

Corollary 6: Specify the largest allowable tolerance in stating FRs.

Corollary 7: Seek an uncoupled design that requires less information than coupled designs in satisfying a set of FRs.

We will now consider the theorems, arising from these axioms and corollaries, which prove that an optimal design results from a square design matrix. In other words, the number of FRs m should be equal to the number of DPs n. First, consider the case where $m > n$. Intuitively, this forces a single DP to be mapped to multiple FRs. Otherwise, some FRs cannot be satisfied by the DPs. This result is stated in Theorem 1.3.

Theorem

When the number of DPs is less that the number of FRs, either a coupled design results or the FRs cannot be satisfied.

The solution to this problem is given by Theorem 1.3.

Theorem

A coupled design due to more FRs than DPs can be decoupled by adding new DPs if the additional DPs result in a lower triangular design matrix.

An example is show in Figure 1.3. Note that the addition of DP3 results in a decoupled design.

The case where the number of FRs is less than the number of DPs is covered by Theorem 1.3.

$$\begin{bmatrix} \text{FR1} \\ \text{FR2} \\ \text{FR3} \end{bmatrix} = \begin{bmatrix} X & 0 \\ X & X \\ X & X \end{bmatrix} \begin{bmatrix} \text{DP1} \\ \text{DP2} \end{bmatrix} \implies \begin{bmatrix} \text{FR1} \\ \text{FR2} \\ \text{FR3} \end{bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{bmatrix} \text{DP1} \\ \text{DP2} \\ \text{DP3} \end{bmatrix}$$

Figure 6: Decoupling by Adding DPs

Theorem

When there are less FRs than DPs then the design is either coupled or redundant.

Finally, the previous three theorems lead to the conclusion in Theorem 1.3.

Theorem

In an ideal design, the number of FRs is equal to the number of DPs.

1.4 Part Consolidation

One particular area of design focus is the number of parts in a product design. According to Professor Suh:

> Poorly designed products often cost more because they use more materials or parts than do well-designed products. They are often difficult to manufacture and maintain.

Decreasing the number of parts in a design while maintaining the independence of the FRs, consistent with Corollary 1.3, lowers the information content of the design. An informative example is the combination can/bottle opener shown in Figure 1.4.

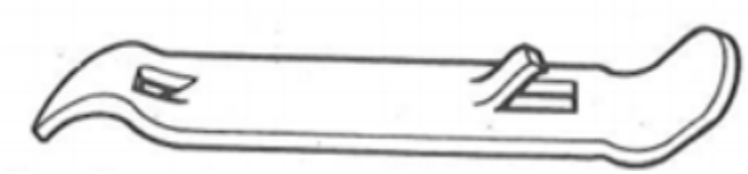


Figure 7: Part Consolidation Example

The design of this handy utensil has two FRs:

FR1: Open beverage cans

FR2: Open beverage bottles

Both FRs are consolidated into a single part, yet remain independent as long as there is no desire to open a can and a bottle simultaneously.

The goal of this research is to provide designers with a tool that they can use to determine the minimum number of parts required to realize a particular design at a particular level of the FR

hierarchy. The designer is required to construct a graph where the vertices are the FRs and the edges indicate that the endpoint FRs need to be realized by separate parts due to independence or other reason. Note that how the edges are determined is beyond the scope of this research. Adjacent FRs are candidates for part consolidation. The goal is to find the chromatic number of the resulting graph, which corresponds to the minimum number of parts.

Unfortunately, finding the chromatic number of a graph is known to be an NP-hard problem. Thus, an algorithm is proposed with improved runtime complexity that can be run on a computer in order to provide a designer with an answer in a reasonable amount of time. Designs with different minimum part requirements can then be compared during the analytical process as part of the overall process of selecting the best design.

2 Graph Theory

This section presents the concepts, definitions, and theorems from the field of graph theory that are needed in the development of the proposed algorithm. This material is primarily taken from the textbooks used by the author during his undergraduate and graduate graph theory classes at SJSU [3, 7].

2.1 Simple Graphs

The problem of part consolidation is best served by a class of graphs called *simple graphs*. A *simple graph* is a mathematical object represented by an ordered pair $G = (V, E)$ consisting of a non-empty and finite set of *vertices* (also called *nodes*): $V(G)$, and a finite and possibly empty set of edges: $E(G)$. Each edge is represented by a two-element subset of $V(G)$ called the *endpoints* of the edge: $E(G) \subseteq \mathcal{P}_2(V(G))$. For the remainder of this work, the use of the term "graph" implies a "simple graph." Thus, a part consolidation problem can be represented by a graph whose vertices are the functional requirements (FRs) of the design and whose edges indicate which endpoint FRs should never be combined into a single part.

In addition to the vertices and edges, a graph may be associated with one or more relations. Each relation has $V(G)$ or $E(G)$ as its domain and is used to associate vertices or edges with problem-specific attributes such as labels or colors. Some authors include these relations and their codomains as part of the graph tuple; however, since these extra tuple elements don't affect the structure of the graph, we will not do so.

The choice of two-element subsets of $V(G)$ for the edges has certain ramifications that are indeed characteristics that differentiate a simple graph from other classes of graphs:

1. Every two vertices of a graph are the endpoints of at most one edge; there are no so-called *multiple* edges between two vertices.
2. The two endpoint vertices of an edge are always distinct; there are no so-called *loop* edges on a single vertex.
3. The two endpoint vertices are unordered, suggesting that an edge provides a bidirectional connection between its endpoint vertices.

When referring to the edges in a graph, the common notation of juxtaposition of the vertices will be used instead of the set syntax. Thus, edge $\{u, v\}$ is simply referred to as uv.

Graphs are often portrayed visually using labeled or filled circles for the vertices and lines for the edges such that each edge line is drawn between its two endpoint vertices. An example graph is shown in Figure 2.1.

When two vertices are the endpoints of the same edge, the vertices are said to be *adjacent* or are called *neighbors*, and the edge is said to *join* its two endpoint vertices. Furthermore, an edge is said to be *incident* to its endpoint vertices. In the example graph of Figure 2.1, vertex a is adjacent to vertices b, d, and e.

As demonstrated by vertex c in Figure 2.1, there is no requirement that every vertex in a graph be

$$V = V(G) = \{a, b, c, d, e\}$$
$$E = E(G) = \big\{\{a, b\}, \{a, d\}, \{a, e\}, \{b, e\}\big\}$$

Figure 8: An Example Graph (labeled and unlabeled)

an endpoint for some edge. In fact, a vertex that is not incident to any edge is called an *isolated* vertex.

We can also speak of adjacent edges, which are edges that share an endpoint:

Definition: Adjacent Edges

Let G be a graph and left $e, f \in E(G)$. To say that e and f are *adjacent* edges means that they share some endpoint $v \in E(G)$:

$$\exists\, v \in V(G), e \cap f = \{v\}$$

or similarly:

$$|e \cap f| = 1$$

Note that two edges in a simple graph can only share one endpoint; otherwise, the two edges would be multiple edges, which are not allowed in simple graphs.

In the example graph of Figure 2.1, notice that ab is adjacent to ad, ae, and be; and ae is adjacent to be.

2.2 Order and Size

Two of the most important characteristics of a graph are the number of vertices in the graph, called the *order* of the graph, and the number of edges in the graph, called the *size* of the graph:

Definition: Order

Let G be a graph. The *order* of G, denoted by n or $n(G)$, is the number of vertices in G:

$$n = n(G) = |V(G)|$$

Definition: Size

Let G be a graph. The *size* of G, denoted by m or $m(G)$, is the number of edges in G:

$$m = m(G) = |E(G)|$$

In the example graph of Figure 2.1, notice that $n = 5$ and $m = 4$.

Since every two vertices can have at most one edge between them, the number of edges has an upper bound:

Theorem

Let G be a graph of order n and size m:

$$m \leq \frac{n(n-1)}{2}$$

Proof. Since each pair of distinct vertices in $V(G)$ can have zero or one edges joining them, the maximum number of possible edges is $\binom{n}{2}$, and so:

$$m \leq \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$$

■

Some choices of graph order and size lead to certain degenerate cases that serve as important termination cases for the the proposed algorithm:

Definition: Degenerate Cases

- The *null* graph is the non-graph with no vertices $(n = m = 0)$.
- The *trivial* graph is the graph with exactly one vertex and no edges $(n = 1, m = 0)$. Otherwise, the graph is *non-trivial*.
- An *empty* graph is a graph with possibly some isolated vertices but with no edges $(m = 0)$.

Hence, both the null and trivial graphs are empty.

2.3 Graph Tuple Relations

Various problems in graph theory require that vertices and edges be assigned values of particular attributes. This is accomplished by adding relations to the graph tuple that map the vertices and/or edges to their attribute values. Note that there are no particular limitations on the nature of such a relation — everything from a basic relation to a bijective function is possible, depending on the problem.

In practice, when a graph theory problem requires a particular vertex or edge attribute, the presence of some corresponding relation $\mathscr{R}$ is assumed and we say something like, "vertex v has attribute a," instead of the more formal, "vertex v has attribute $\mathscr{R}(v)$."

The following sections describe the two relations used by the proposed algorithm.

2.3.1 Labels

One of the possible relations in a graph tuple is a bijective function that assigns each vertex an identifying label. When such a function is present, the graph is said to be a *labeled* graph:

Definition: Labeled Graph

To say that a graph G is *labeled* means that its vertices are considered to be distinct and are assigned identifying names (labels) by adding a bijective labeling function to the graph tuple:

$$\ell : V(G) \to L$$

where L is a set of labels (names). Otherwise, the vertices are considered to be identical (only the structure of the graph matters) and the graph is *unlabeled*.

The vertices in a labeled graph are typically draw as open circles containing the corresponding labels, whereas the vertices in an unlabeled graph are typically drawn as filled circles. This is demonstrated in the example graph of Figure 2.1: the graph on the left is labeled and the graph on the right is unlabeled.

Since the labeling function ℓ is bijective, a vertex $v \in V(G)$ with label "a" can be identified by v or $\ell^{-1}(a)$. In practice, the presence of a labeling function is assumed for a labeled graph and so a vertex is freely identified by its label. This is important to note when a proof includes a phrase such as, "let $v \in V(G) \ldots$" since v may be a reference to any vertex in $V(G)$ or may call out a specific vertex by its label; the intention is usually clear from the context.

The design graphs that act as the inputs to the proposed algorithm are labeled graphs, where the labels represent the various functional requirements:

$$FR_1, FR_2, FR_3, \ldots, FR_n$$

2.3.2 Vertex Color

Other graph theory problems require that the graph's vertex set be distributed into some number of sets based on some problem-specific criteria. Usually, this distribution is a true partition (no empty sets), but this is not required depending on the problem. One popular method of performing this distribution is by adding a *coloring* function to the graph tuple:

$$c : V(G) \to C$$

where C is a set of *colors*; vertices with the same color are assigned to the same set in the distribution. Although the elements of C are usually actual colors (red, green, blue, etc.), a

graph coloring problem is free to select any value type for the color attribute. Note that there is no assumption that c is surjective, so the codomain C may contain unused colors, which corresponds to empty sets in the distribution.

The most popular coloring scheme for a graph requires that adjacent vertices be assigned different colors:

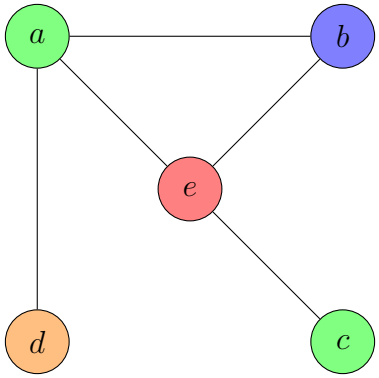
Definition: Proper Coloring

A coloring c on a graph G is called *proper* when no two adjacent vertices are assigned the same color:

$$\forall\, u, v \in V(G), uv \in E(G) \implies c(u) \neq c(v)$$

A proper coloring c with $|C| = k$ is called a k-*coloring* of G and G is said to be k-*colorable*, meaning the actual coloring (range of c) uses *at most* k colors.

An example of a 4-coloring is shown in Figure 2.3.2.



$$c(a) = green$$
$$c(b) = blue$$
$$c(c) = green$$
$$c(d) = orange$$
$$c(e) = red$$

Figure 9: A Graph with a 4-coloring

Since there is no requirement that a coloring c be surjective, the codomain C may contain unused colors. For example, the codomain of the coloring shown in Figure 2.3.2 might be:

$$C = \{green, blue, red, orange\}$$

and hence c is surjective and G is 4-colorable. But we can always add an unused color to C:

$$C = \{green, blue, red, orange, brown\}$$

Now, c is no longer surjective, and according to the definition: G is 5-colorable — the coloring c uses at most 5 colors (actually only 4), which is the cardinality of the codomain.

Thus, we can make statement in Proposition 1.

Proposition 1

Let G be a graph:

G is k-colorable $\implies$ G is $(k+1)$-colorable

By inductive application of Proposition 1, one can arrive at the conclusion in Proposition 2.

Proposition 2

Let G be a graph:

G is k-colorable $\implies$ G is $(k+r)$-colorable for some $r \in \mathbb{N}$.

Furthermore, for a graph G of order n, if $n \leq k$ we can conclude that G is k-colorable, since there are sufficient colors such that each vertex can be assigned its own color. This is stated in Proposition 3, which will turn out to be an important termination case for the proposed algorithm.

Proposition 3

Let G be a graph of order n and let $k \in \mathbb{N}$:

If $n \leq k$ then G is k-colorable.

Since $k \in \mathbb{N}$, by the well-ordering principle, there exists some minimum k such that a graph G is k-colorable:

Definition: Chromatic Coloring

The minimum k such that a graph G is k-colorable is called the *chromatic number* of G, denoted by $\chi(G)$. A k-coloring for a graph G where $k = \chi(G)$ is called a k-*chromatic* coloring.

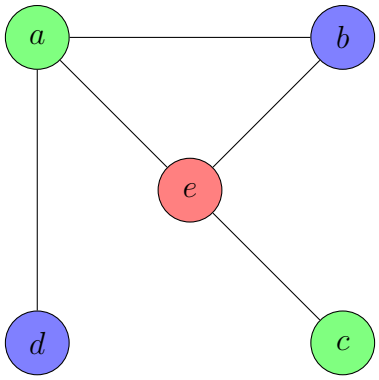
Returning to the example 4-coloring of Figure 2.3.2, note that vertex d can be colored blue and then orange can be excluded from the codomain, resulting in a 3-coloring. This is shown in Figure 2.3.2. Since there is no way to use less than 3 colors to obtain a proper coloring of the graph, the coloring is 3-chromatic. Note that when a coloring is chromatic, there are no unused colors (empty sets) and hence the distribution is a true partition.

The primary purpose of a k-coloring of a graph G is to distribute the vertices of G into k so-called *independent* (some possibly empty) sets:

Definition: Independent Set

Let G be a graph and let $S \subseteq V(G)$. To say that S is an *independent* set means that all of the vertices in S are non-adjacent in G:

$$\forall\, u, v \in S, uv \notin E(G)$$



$$c(a) = green$$
$$c(b) = blue$$
$$c(c) = green$$
$$c(d) = blue$$
$$c(e) = red$$

Figure 10: A Graph with a Chromatic 3-coloring

Since a k-chromatic coloring of a graph G is surjective, there are no unused colors (empty sets) and so the coloring partitions the vertices of G into exactly k independent sets. The goal of the proposed algorithm is to find a chromatic coloring of a design graph so that the resulting independent sets indicate how to consolidate the FRs into a minimum number of parts: one part per independent set.

2.4 Subgraphs

The basic strategy of the proposed algorithm is to arrive at a solution by mutating an input graph into simpler graphs such that a solution is more easily determined. The algorithm utilizes three particular mutators: vertex deletion, edge addition, and vertex contraction. Before describing these mutators, it will be helpful to describe what is meant by graph equality and a *subgraph* of a graph.

2.4.1 Graph Equality

Graph equality follows from equality of the vertex and edge sets:

Definition: Graph Equality

Let G and H be graphs. To say that G is equal to H, denoted $G = H$, means that $V(G) = V(H)$ and $E(G) = E(H)$.

Note that this definition of equality ignores any additional relations that may be added to the graph tuples since those relations tend to be added by specific problems and do not reflect the actual parts of the graphs.

2.4.2 Subgraphs

Since graph equality follows from vertex and edge set equality, there should also be a concept of a *subgraph* resulting from the subsets of those sets:

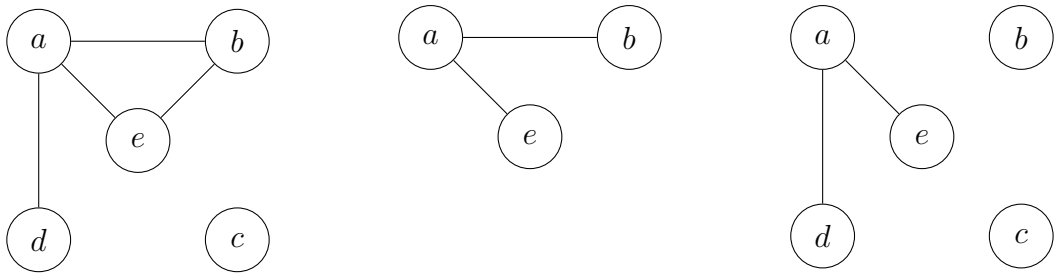
Definition: Subgraph

Let G and H be two graphs:

- To say that H is a *subgraph* of G, denoted $H \subseteq G$, means that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.
- To say that H is a *proper subgraph* of G, denoted $H \subset G$, means that $H \subseteq G$ but $H \neq G$: $V(H) \subset V(G)$ or $E(H) \subset E(G)$.
- To say that H is a *spanning subgraph* of G means that H is a subgraph of G such that $V(H) = V(G)$ and $E(H) \subseteq E(G)$.

Thus, given a graph G and a subgraph H, there should be a sequence of zero or more vertex and/or edge removals to obtain H from G. Likewise, there should be a sequence of zero or more vertex and/or edge additions to obtain G from H. If H is a proper subgraph of G then H and G differ by at least one removed vertex or one removed edge. If H is a spanning subgraph of G then H contains all of the vertices in G but may differ by removed edges only. Per the definition, a graph is always a subgraph of itself ($G \subseteq G$) and the null graph is a subgraph of every graph.

The concept of subgraphs is demonstrated by graphs G, H, and F in Figure 2.4.2. H is a proper subgraph of G by removing vertices c and d and edges ad and be. F is a proper spanning subgraph of G because F contains all of the vertices in G but is missing edges ab and be.



G $\quad$ $H \subset G$ (proper) $\quad$ $F \subset G$ (spanning)

Figure 11: Subgraph Examples

2.4.3 Induced Subgraphs

An *induced* subgraph is a special type of subgraph:

Definition: Induced Subgraph

Let G be a graph and let S be a non-empty subset of $V(G)$. The subgraph of G *induced* by S, denoted $G[S]$, is a subgraph H such that:

- $V(H) = S$
- $u, v \in V(H)$ and $uv \in E(G) \implies uv \in E(H)$

Such a subgraph H is called an *induced subgraph* of G.

Note that when a vertex is removed to make an induced subgraph then all of that vertex's incident edges are also removed. However, for every pair of vertices included in an induced subgraph, if the vertices are the endpoints of a particular edge then that edge must also be included in the subgraph. In the examples of Figure 2.4.2, H is not an induced subgraph of G because it is missing edge be. Likewise, a proper spanning subgraph like F can never be induced due to missing edges. In fact, the only induced spanning subgraph of a graph is the graph itself. Figure 2.4.3 adds edge be so that H is now an induced subgraph of G.

G $\qquad\qquad$ $H = G[\{a, b, e\}]$

Figure 12: Induced Subgraph Example

2.5 Mutators

The following sections describe the graph mutators used by the proposed algorithm.

2.5.1 Vertex Removal

Let G be a graph and let $S \subseteq V(G)$. The induced subgraph obtained by removing all of the vertices in S (and their incident edges) is denoted by:

$$G - S = G[V(G) - S]$$

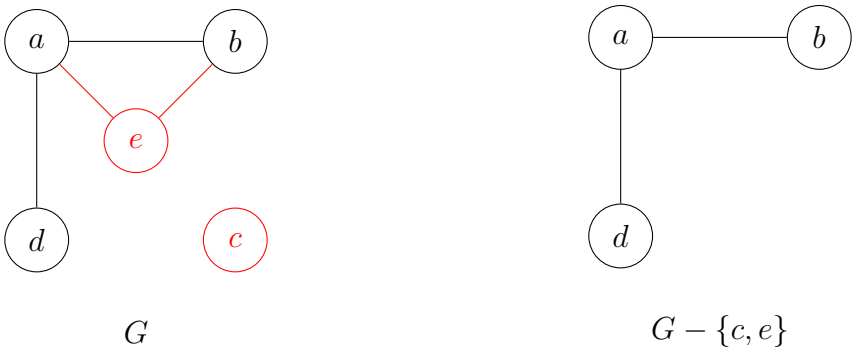
If $S \neq \emptyset$ then $G - S$ is a proper subgraph of G. If $S = V(G)$ then the result is the null graph.

Figure 2.5.1 shows an example of vertex removal: vertices c and e are removed, along with their incident edges ae and be.

If $|S| = 1$ then an alternate syntax can be used. Assume $v \in V(G)$:

$$G - v = G - \{v\}$$

The proposed algorithm uses vertex removal to simplify a k-colorable graph into a simpler subgraph that is still k-colorable.



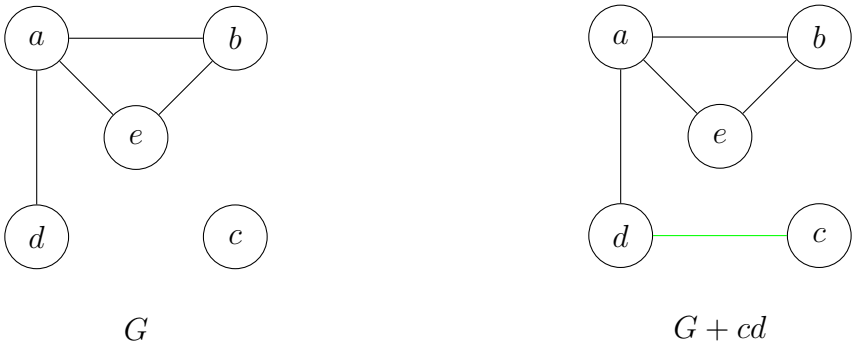
G $\qquad$ $G - \{c, e\}$

Figure 13: Vertex Removal Example

2.5.2 Edge Addition

Let G be a graph and let $u, v \in V(G)$ such that $uv \notin E(G)$. The graph $G + uv$ is the graph with the same vertices as G and with edge set $E(G) \cup \{uv\}$. Note that G is a proper spanning subgraph of $G + uv$.

Figure 2.5.2 shows an example of edge addition: edge cd is added.



G $\qquad$ $G + cd$

Figure 14: Edge Addition Example

The proposed algorithm uses edge addition to prevent two non-adjacent FRs from being consolidated into the same part.

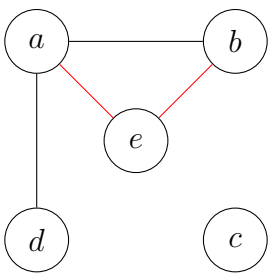
2.5.3 Edge Removal

The proposed algorithm does not use edge removal; however, a number of related algorithms do rely on this mutator so it is presented here. Let G be a graph and let $X \subseteq E(G)$. The spanning subgraph obtained by removing all of the edges in X is denoted by:

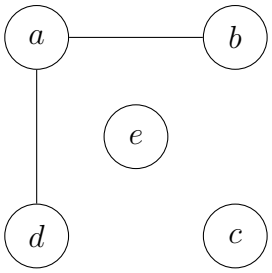
$$G - X = H\left(V(G), E(G) - X\right)$$

Thus, only edges are remove — no vertices are removed. If $X \neq \emptyset$ then $G - X$ is a proper subgraph of G. If $X = E(G)$ then the result is an empty graph (no edges).

Figure 2.5.3 shows an example of edge removal: edges ae and be are removed.



G



$G - \{ae, be\}$

Figure 15: Edge Removal Example

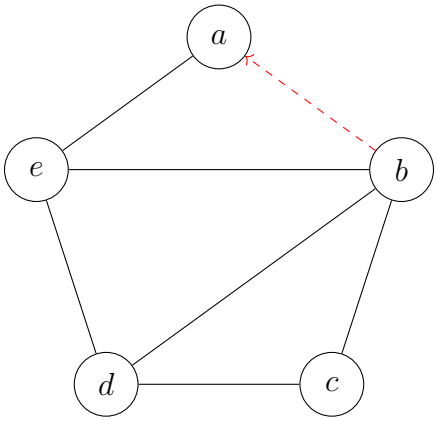
If $|X| = 1$ then an alternate syntax can be used. Assume $e \in E(G)$:

$$G - e = G - \{e\}$$

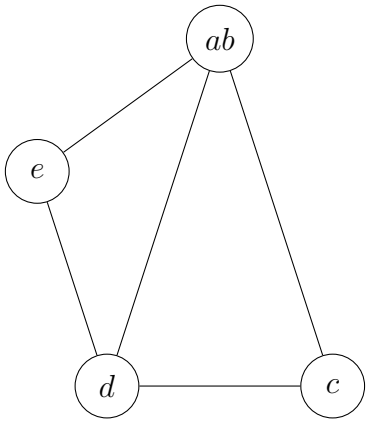
2.5.4 Vertex Contraction

Vertex contraction is a bit different because it does not involve subgraphs. Let G be a graph and let $u, v \in V(G)$. The graph $G \cdot uv$ is constructed by identifying u and v as one vertex (i.e., merging them). Any edge between the two vertices is discarded. Any other edges that were incident to the two vertices become incident to the new single vertex. Note that this may require supression of multiple edges to preserve the nature of a simple graph.

Figure 2.5.4 shows an example of vertex contraction: vertices a and b are contracted into a single vertex. Since edges ae and be would result in multiple edges between a and e, one of the edges is discarded. Edges bc and bd also become incident to the single vertex.



G



$G \cdot ab$

Figure 16: Vertex Contraction Example

For the operation $G \cdot uv$, if $uv \in E(G)$ then the operation is also referred to as *edge contraction*. If $uv \notin E(G)$ then the operation is also referred to as *vertex identification*. The proposed algorithm uses vertex identification to consolidate two non-adjacent FRs into the same part.

2.6 Connected Graphs

The edges of a graph suggest the ability to "walk" from one vertex to another along the edges. A graph where this is possible for any two vertices is called a *connected* graph. The concept of connectedness is an important topic in graph theory; however, an ideal coloring algorithm should work regardless of the connected nature of an input graph. The concept of connectedness and how it impacts coloring is described in this section.

2.6.1 Walks

The undirected edges in a simple graph suggest bidirectional connectivity between their endpoint vertices. This leads to the idea of "traveling" between two vertices in a graph by following the edges joining intermediate adjacent vertices. Such a journey is referred to as a *walk*:

Definition: Walk

A $u - v$ *walk* W in a graph G is a finite sequence of vertices $w_i \in V(G)$ starting with $u = w_0$ and ending with $v = w_k$:

$$W = (u = w_0, w_1, \ldots, w_k = v)$$

such that $w_i w_{i+1} \in E(G)$ for $0 \leq i < k$.

To say that W is *open* means that $u \neq v$. To say that W is *closed* means that $u = v$. The *length* k of W is the number of edges traversed: $k = |W|$.

A *trivial* walk is a walk of zero length — i.e, a single vertex: $W = (u)$.

The bidirectional nature of the edges in a simple graph suggests the following proposition:

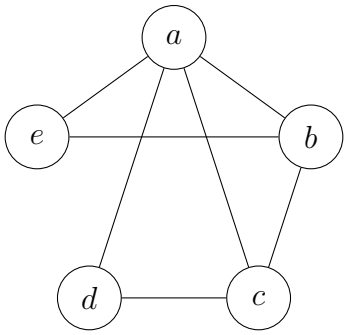
Proposition 4

Let G be a graph and let $u - v$ be a walk of length k in G. G contains a $v - u$ walk of length k in G by traversing $u - v$ in the opposite direction.

An example of two walks of length 4 is shown in Figure 2.6.1. Note that W_1 is an open walk because it starts and ends on distinct vertices, whereas W_2 is a closed walk because it starts and ends on the same vertex.

Note that in the general case, vertices and edges are allowed to be repeated during a walk. Certain special walks can be defined by restricting such repeats:

Definition: Special Walks



$W_1 = (a, b, e, a, c)$ is open
$W_2 = (a, e, b, c, a)$ is closed

$|W_1| = |W_2| = 4$

Figure 17: Open and Closed Walks in a Graph

trail	An open walk with no repeating edges	(a, b, c, a, e)
path	A trail with no repeating vertices	(a, e, b, c)
circuit	A closed trail	(a, b, e, a, c, d, a)
cycle	A closed path	(a, e, b, c, a)

The example special walks stated above refer to the graph in Figure 2.6.1.

When discussing the connectedness of a graph, the main concern is the existence of paths between vertices:

Definition: Connected Vertices

Let G be a graph and let $u, v \in V(G)$. To say that u and v are *connected* means that G contains a $u - v$ path.

But if there exists a $u - v$ walk in a graph G, does this also mean that there exists a $u - v$ path in G — i.e. a walk with no repeating edges or vertices? The answer is yes, as shown by the following theorem:

Theorem

Let G be a graph and let $u, v \in V(G)$. If G contains a $u - v$ walk of length k then G contains a $u - v$ path of length $\ell \leq k$.

Proof. Assume that G contains at lease one $u - v$ walk of length k and consider the set of all possible $u - v$ walks in G; their lengths form a non-empty set of positive integers. By the well-ordering principle, there exists a $u - v$ walk P of minimal length $\ell \leq k$:

$$P = (u = w_0, \ldots, w_\ell = v)$$

We claim that P is a path.

Assume by way of contradiction that P is not a path, and thus P has at least one repeating vertex. Let $w_i = w_j$ for some $0 \leq i < j \leq \ell$ be such a repeating vertex. There are two possibilities:

Case 1: The walk ends on a repeated vertex ($j = \ell$). This is demonstrated in Figure 2.6.1.

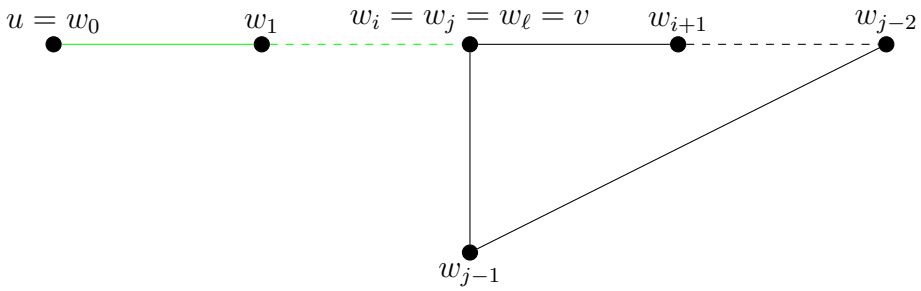


Figure 18: Repeated Vertex at End Case

Let $P' = (u = w_0, w_1, \ldots, w_i = v)$ be the walk shown in green in the figure. P' is a $u - v$ walk of length $i < \ell$ in G.

Case 2: A repeated vertex occurs inside the walk ($j < \ell$). This is demonstrated in Figure 2.6.1.

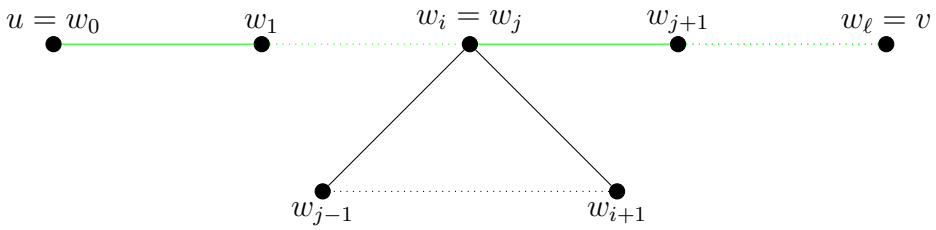


Figure 19: Repeated Vertex Inside Case

Let $P' = (u = w_0, w_1, \ldots, w_i, w_{j+1}, \ldots, w_\ell = v)$ be the walk shown in green in the figure. P' is a $u - v$ walk of length $\ell - (j - i) < \ell$ in G.

Both cases contradict the minimality of the length of P.

$\therefore$ P is a $u - v$ path of length $\ell \leq k$ in G. ∎

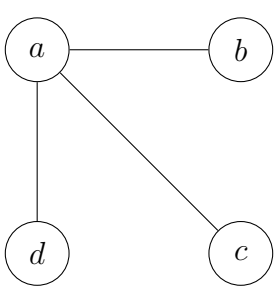
2.6.2 Connected

A *connected* graph is a graph whose vertices are all connected:

Definition: Connected Graph

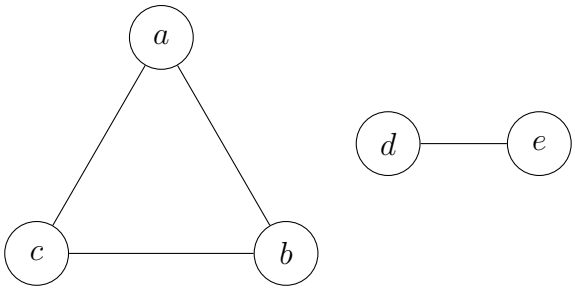
To say that a graph G is *connected* means that for all $u, v \in V(G)$ there exists a $u - v$ path. Otherwise, G is said to be *disconnected*.

Examples of connected and disconnected graphs are shown in figure 2.6.2.



(a, b)
(a, c)
(a, d)
(b, a, c)
(b, a, d)
(c, a, d)

CONNECTED



No path from any of a, b, c to any of d, e

DISCONNECTED

Figure 20: Connected and Disconnected Graphs

By definition, the trivial graph is connected since the single vertex is connected to itself by a trivial path (of length 0).

2.6.3 Components

It would seem that a disconnected graph is composed of some number of connected subgraphs that partition the graph's vertex set under a connected equivalence relation. Each such subgraph is called a *component* of the graph:

Definition: Component

Let G be a graph and let $\mathscr{G}$ be the set of all connected subgraphs of G. To say that a graph $H \in \mathscr{G}$ is a *component* of a G means that H is not a subgraph of any other connected subgraph of $\mathscr{G}$:

$$\forall\, F \in \mathscr{G} - \{H\}, H \not\subset F$$

The number of distinct components in G is denoted by:

$$k = k(G)$$

For a connected graph: $k(G) = 1$.

Each component of a graph G is denoted by G_i where $1 \leq i \leq k(G)$. We also use union notation to denote that G is composed of its component parts:

$$G = \bigcup_{0 \leq i \leq k(G)} G_i$$

Furthermore the G_i are induced by the vertex equivalence classes of the connectedness relation:

Theorem

Let G be a graph with component G_i. G_i is an induced subgraph of G.

Proof. By definition, G_i is a maximal connected subgraph of G. So assume by way of contradiction that G_i is not an induced subgraph of G. Thus, G_i is missing some edges that when added would result in a connected induced subgraph H of G. But then $G_i \subset H$, contradicting the maximality of G_i.

$\therefore$ G_i is an induced subgraph of G. ∎

2.6.4 Impact on Coloring

The impact of disconnectedness on coloring depends on the selected algorithm. One might assume that the selected algorithm should be run on each component individually in order to determine each $\chi(G_i)$ and then use Proposition 2 to conclude that the maximum such value is sufficient for $\chi(G)$:

$$\chi(G) = \max_{1 \leq i \leq k(G)} \chi(G_i)$$

For example, consider the disconnected graph in Figure 2.6.2. The graph contains two components, so number the components from left-to-right:

$$\chi(G_1) = 3$$
$$\chi(G_2) = 2$$

$$\chi(G) = \max\{3, 2\} = 3$$

Using this technique requires application of an initial algorithm to partition the graph into components. Such an algorithm is well-known and is described by Hopcroft and Tarjan, 1973 [4]. The algorithm is recursive. It starts by pushing a randomly selected vertex on the stack and walking the vertex's incident edges, removing each edge as it is traversed. As each unmarked vertex is encountered, it is assigned to the current component. Vertices with incident edges are pushed onto the stack and newly isolated vertices are popped off the stack. Once the stack is empty, any previously unmarked vertex is selected to start the next component and the process continues until all vertices are marked. This algorithm has a runtime complexity of $\mathcal{O}(\max(n, m))$.

Alternatively, an ideal coloring algorithm could be run on the entire graph at once regardless of the number of components in the graph. The proposed algorithm is such a solution, and therefore saves the needless work of partitioning the graph into components first.

2.7 Vertex Degree

Besides a graph's order and size, the next most important parameter is the so-called *degree* of each vertex. In order to define the degree of a vertex, we need to define what is meant by a vertex's *neighborhood* first:

Definition: Neighbor

Let G be a graph and let $u, v \in V(G)$. To say that u is a *neighbor* of v (and vice-versa) means that $uv \in E(G)$.

Thus, neighbor vertices are adjacent. Note that for simple graphs, a vertex is never a neighbor of itself.

The set of all neighbors for a vertex is referred to as the vertex's neighborhood:

Definition: Neighborhood

Let G be a graph and let $u \in V(G)$. The *neighborhood* of u, denoted by $N(u)$, is the set of all neighbors of u in G:

$$N(u) = \{v \in V(G) \mid uv \in E(G)\}$$

The degree of a vertex is then defined to be the cardinality of its neighborhood:

Definition: Degree

Let G be a graph and let $u \in G$. The *degree* of u, denoted by $\deg_G(u)$ or $\deg(u)$, is the cardinality of the neighborhood of u:

$$\deg(u) = |N(u)|$$

Note that the degree of a vertex can be viewed as the number of neighbor vertices or the number of incident edges.

When considering the degrees of all the vertices in a graph, the following limits are helpful:

Notation

Let G be a graph:

$$\delta(G) = \min_{v \in V(G)} \deg(v)$$
$$\Delta(G) = \max_{v \in V(G)} \deg(v)$$

And so, for a graph G or order n, it must be the case that for every vertex $v \in G$:

$$0 \leq \delta(G) \leq \deg(v) \leq \Delta(G) \leq n - 1$$

Intuitively, as $\delta(G)$ increases, a graph becomes denser (more edges) resulting in more adjacencies, making it harder to find a proper coloring at lower values of k.

Vertices can be classified based on their degree:

Definition: Vertex Types

Let G be a graph of order n and let $u \in V(G)$:

$\deg(u)$	TYPE
0	isolated
1	pendant, end, leaf
$n-1$	universal
even	even
odd	odd

The degrees of the vertices in a graph and the number of edges in the graph are related by the so-called First Theorem of Graph Theory:

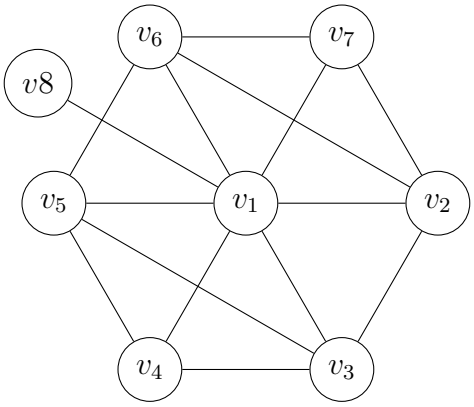
Theorem: First Theorem of Graph Theory

Let G be a graph of size m:

$$\sum_{v \in V(G)} \deg(v) = 2m$$

Proof. When summing all the degrees, each edge is counted twice: once for each endpoint. ∎

These concepts are demonstrated by the graph in Figure 2.7; note that the sum of the vertex degrees is 30, which is twice the number of edges in the graph.



vertex	degree	type
v_1	7	universal,odd
v_2	4	even
v_3	4	even
v_4	3	odd
v_5	4	even
v_6	4	even
v_7	3	odd
v_8	1	pendant,odd
total	30	

$n = 8$ $\quad m = 15 = \frac{30}{2}$
$\delta(G) = 1$ $\quad \Delta(G) = 7$
$\operatorname{diam}(G) = 2$

Figure 21: Vertex Degrees

2.8 Special Graphs

We conclude this introductory section on graph theory with a discussion of some special classes of graphs that are important to the execution of the proposed algorithm.

2.8.1 Empty Graphs

An *empty* graph of order n, denoted by E_n, is a graph with one or more vertices ($n > 1$) and no edges ($m = 0$). An empty graph is connected iff $n = 1$. Examples of empty graphs are shown in Figure 2.8.1.

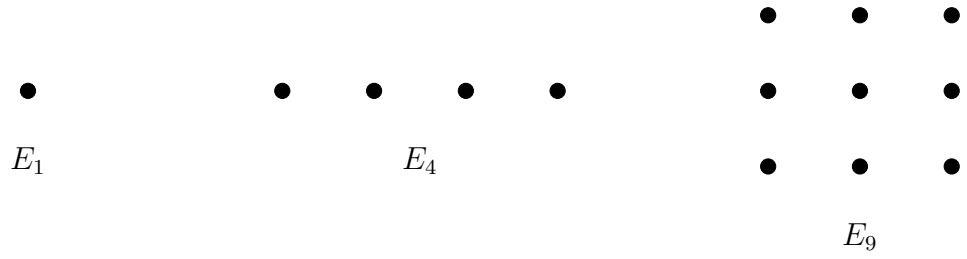


Figure 22: Empty Graphs

The null graph ($n = 0$) is denoted by E_0 and is defined to be 0-chromatic. All other empty graphs are 1-chromatic and thus are important termination conditions for the proposed algorithm.

2.8.2 Paths

A *path* graph of order n and length $n - 1$, denoted by P_n, is a connected graph consisting of a single open path. Examples of path graphs are shown in Figure 2.8.2.

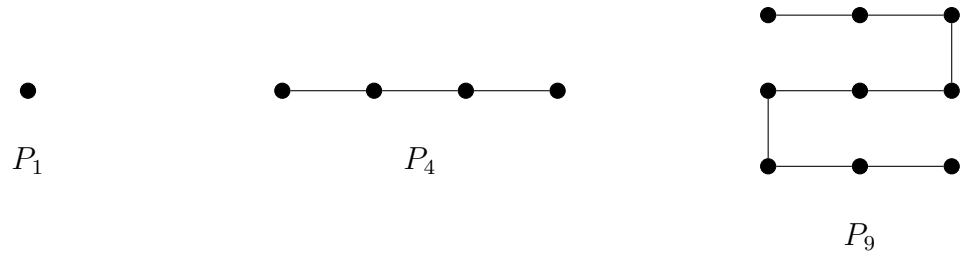


Figure 23: Path Graphs

Note that $P_1 = E_1$ is 1-chromatic, whereas $P_n, n > 1$ is 2-chromatic.

Paths are not particularly important to the proposed algorithm; however, they play a part in the definition of cycles.

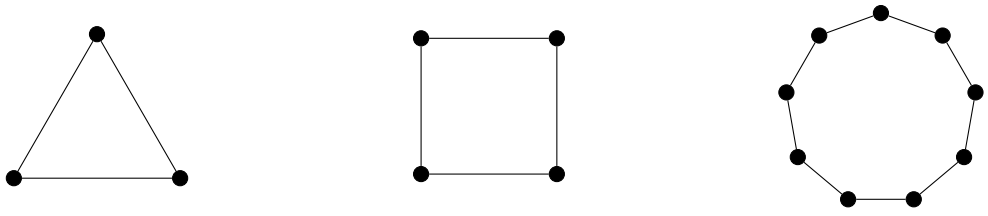
2.8.3 Cycles

A *cycle* graph of order n and length n for $n \geq 3$, denoted by C_n, is a connected graph consisting of a single closed path. When n is odd then C_n is called an *odd* cycle and when n is even then C_n is called an *even* cycle.

Examples of cycle graphs are shown in Figure 2.8.3.

Note that even cycles are 2-chromatic; however, odd cycles are 3-chromatic.

Cycles are not particularly important to the proposed algorithm; however, they play a part in the definition of trees, which are important to the later Zykov analysis of coloring algorithms.



C_3 (odd) C_4 (even) C_9 (odd)

Figure 24: Cycle Graphs

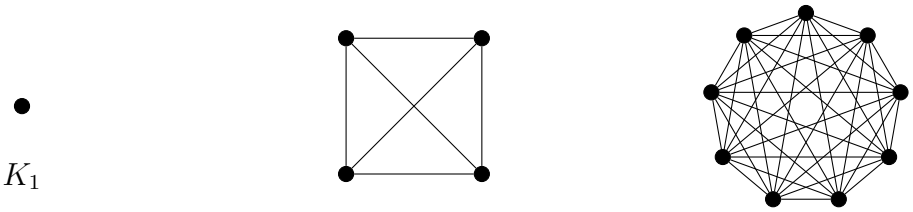
2.8.4 Complete Graphs

A *complete* graph of order n and size $\frac{n(n-1)}{2}$, denoted by K_n, is a connected graph that contains every possible edge:

$$E(G) = \mathcal{P}_2(V(G))$$

and thus all n vertices are adjacent to each other.

Examples of complete graphs are shown in Figure 2.8.4.



K_1 K_4 K_9

Figure 25: Complete Graphs

Note that $K_1 = P_1 = E_1$.

Since all of the vertices in a complete graph are adjacent to each other, each vertex requires a separate color in order to achieve a proper coloring. Thus, K_n is n-chromatic and is also an important termination condition for the proposed algorithm.

2.8.5 Trees

A *tree* is a connected graph that contains no cycles as subgraphs. Typically, one vertex of the tree is selected as the *root* vertex and then the tree is depicted in layers that contain vertices that are equidistant from the root vertex. Note that the bottom layer is composed entirely of pendant vertices, but pendant vertices can exist in the other layers as well. Such pendant vertices are usually referred to as *leaves* in this context.

An example tree is shown in Figure 2.8.5. The root vertex r is shown in red and the leaf vertices b, e, g, h, i, j, k are shown in green.

Trees are important because they can be used to represent the "paper tape" for recursive Turing machine-type algorithms. Each vertex represents a state of the problem and the edges represent

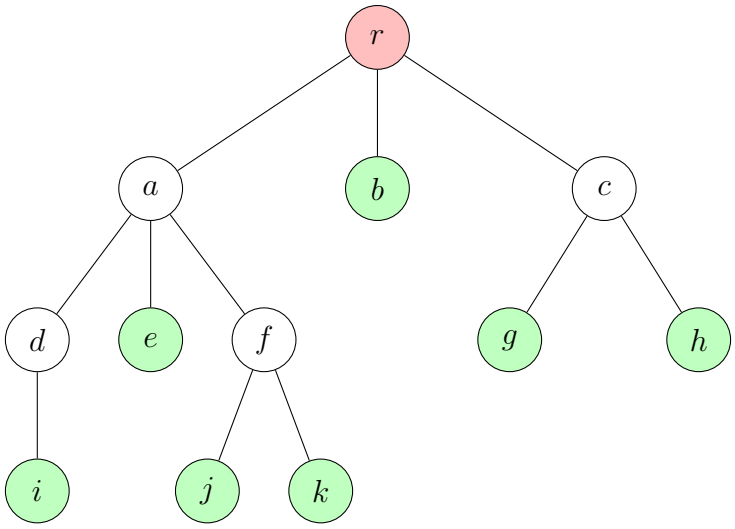


Figure 26: A Tree Organized from Root to Leaves

"movement" of the tape to select the current state. All states can be visited using a so-called *depth-first* walk. In the example in Figure 2.8.5, such a depth-first walk would be:

$$(r, a, d, i, d, a, e, a, f, j, f, k, f, a, r, b, r, c, g, c, h, c, r)$$

Note that this walk guarantees that each vertex is visited at least once.

When such a tree is applied to the problem of exhaustively finding the chromatic number of a graph, the tree is called a *Zykov* tree. These concepts are described in detail in the next section.

3 Traditional Approaches

Determining the chromatic number of a graph is of a class of problems called *NP-hard*.

3.1 Runtime Complexity

3.2 Lower Bound

3.3 Upper Bound

3.4 Zykov Algorithms

4 The Proposed Algorithm

The exhaustive algorithm described in the previous section subjects each pair of non-adjacent vertices in a graph G to the two choices of like (vertex contraction) or different (edge addition) color assignment in a recursive manner. The leaves of the resulting Zykov tree that tracks these choices represent complete graphs that describe all of the possible k colorings of G, with the complete graphs of smallest order representing chromatic colorings of G.

The major advantages of the exhaustive algorithm are that it isn't dependent on the structure of the graph (e.g., connectedness), an example of a chromatic coloring is readily available, and the fact that the algorithm, due to its Turing machine nature, can be coded rather easily to run on a computer. Its major disadvantage is its high runtime complexity due to the need to generate the entire Zykov tree using an exponentially growing number of recursive calls.

Thus, the goals of the proposed algorithm are as follows:

1. It does not depend on the structure of a graph.
2. An example of a chromatic coloring should be available.
3. It can be easily coded for execution on a computer.
4. It has significantly better runtime complexity than the exhaustive algorithm.

To accomplish these goals, the proposed algorithm takes a somewhat different tack: it loops on successively higher values of k. For each candidate k value, a graph is assumed to be k-colorable and a modified version of the exhaustive algorithm is executed to either prove or disprove this assumption. Since a candidate k value is known, certain reversible steps can be applied to mutate G into simpler graphs with equivalent colorability and test for early termination of the current Zykov tree. The first k for which G (or one of its simplifications) is found to be k-colorable is the chromatic number of G. As will be shown, the tradeoff of looping on k and shallow execution of each corresponding Zykov tree far outweighs the need to generate an entire Zykov tree, resulting in a much better runtime complexity.

One slight disadvantage of the proposed algorithm is that whereas the exhaustive algorithm readily provides examples of actual chromatic colorings, the proposed algorithm requires a reverse traversal of its reversible steps in order to construct such a coloring. Nevertheless, this technique still has the advantage over common coloring algorithms such as greedy coloring because it is not heuristic. However, as was stated earlier, during the axiomatic design phase it is more important to know the minimum number of parts as opposed to actual functional requirement allocation to those parts. Thus, a little effort to reconstruct a chromatic coloring after the fact is not of major importance.

This algorithm was first proposed by the author and his advisor in collaboration with a team of mechanical engineering researchers from SUNY Buffalo [1]. It accepts a graph G as input, provides $\chi(G)$ as output, and is composed of an outer loop on values of k and a subroutine called by the outer loop to determine if G is k-colorable. The outer loop and called subroutine are summarized in the following sections. A complete description of the theorems that support the various steps in algorithm and the application of the algorithm to a sample graph then follow.

4.1 Outer Loop

The outer loop accepts a graph G as input and returns $\chi(G)$. It initially checks for some degenerate cases and then loops on increasing values of k. For each value of k, the called subroutine executes the modified exhaustive algorithm to determine if G is k-colorable. The first such successful return identifies $\chi(G)$.

The steps of the outer loop are as follows:

1. If $n = 0$ then return 0, thus handling the degenerate case of a null graph.
2. If $m = 0$ then return 1, thus handling the degenerate case of an empty graph.
3. Initialize k to 2.
4. Call the subroutine to determine if G is k-colorable. The subroutine returns a possibly simplified G called G' and a boolean value R that reports the result of the test.
5. If G is k-colorable ($R =$ true) then return k.
6. Replace G with G'. As will be seen, doing this avoids needless reapplication of certain steps in the called subroutine.
7. Increment k.
8. Go to step 4.

A flowchart of these steps is shown in Figure 4.1.

The outer loop is guaranteed to complete because k will eventually be greater than or equal to n. Thus, by Proposition 3, the current state of G is k-colorable, causing the called subroutine to return true.

4.2 Called Subroutine

The called subroutine executes a modified version of the exhaustive algorithm that determines whether a graph is k-colorable. It accepts the current state of G of order n and size m and the current value of $k \geq 2$ as inputs. It returns a possibly simplified version of G and a boolean value indicating whether or not G is k-colorable. Internally, various tests are applied to trim the corresponding Zykov tree or abandon it all together based on the current value of k.

The steps of the called subroutine and references to their associated theorems are as follows:

1. If $n \leq k$ then return true (Proposition 3).
2. Calculate a maximum edge threshold:

$$a = \frac{n^2(k-1)}{2k}$$

3. If $m > a$ then return false (Corollary 4.3.1).

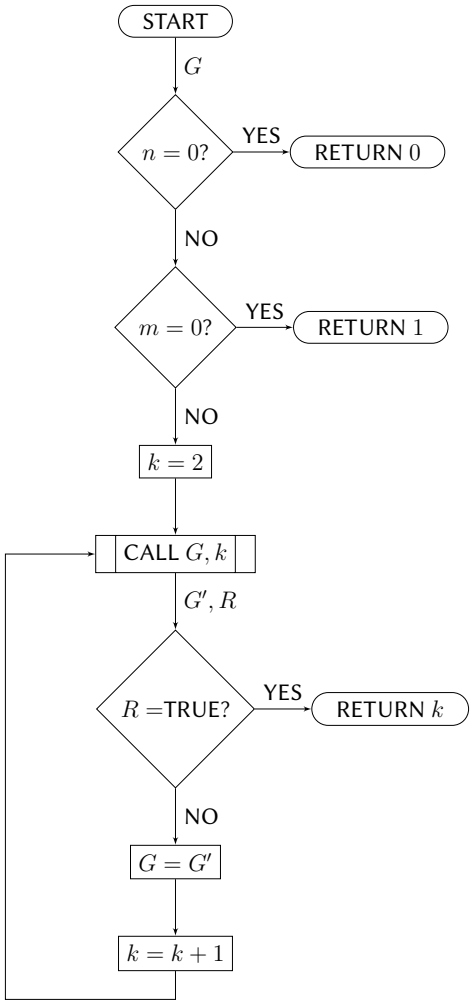


Figure 27: Algorithm Outer Loop

4. Construct the set X of all vertices with degree less than k:

$$X = \{v \in V(G) \,|\, \deg(v) < k\}$$

5. $X \neq \emptyset$ then replace G with $G - X$ and go to step 1 (Corollary 4.3.2).
6. If G has vertices u and v such that $N(u) \subseteq N(v)$ then replace G with $G - u$ and go to step 1 (Theorem 4.3.2).
7. Select two vertices $u, v \in V(G)$ with the smallest number of common neighbors and let

$$b = |N(u) \cap N(v)|$$

8. Calculate an upper bound for the minimum number of common neighbors for all vertices in G:

$$c = n - 2 - \frac{n-2}{k-1}$$

9. If $b > c$ then return false (Corollary 4.3.3).
10. Select two non-adjacent vertices $u, v \in V(G)$ with the smallest number of common neighbors. It will be shown below that such a pair of vertices is guaranteed to exist in the current state of G.
11. Assume that u and v are assigned the same color by letting $G' = G \cdot uv$. Recursively call this routine to see if G' is k-colorable. If so, then return true (Theorem 4.3.4).
12. Assume that u and v are assigned different colors by letting $G' = G + uv$. Recursively call this routine to see if G' is k-colorable. If so, then return true (Theorem 4.3.4).
13. Since neither of the assumptions in steps 11 and 12 hold, conclude that G is not k-colorable and return false.

A flowchart of these steps is shown in Figure 4.2.

Step 1 of the called subroutine is the success termination condition. Success occurs when G is simplified by removing sufficent vertices (steps 4–6) or when the outer loop has sufficiently incremented k (step 7) such that $n \leq k$.

Steps 4–6 of the called subroutine attempt to remove vertices to achieve a simpler graph that is equivalently k-colorable. Each time a vertex is removed, the subtrees associated with that vertex in the corresponding Zykov tree for G are ignored. Since these same steps would just be repeated for $k + 1$, the subroutine returns the current state of the possibly simplified G to the outer loop as a starting point for the next candidate value of k.

Steps 2–3 and 7–9 of the called subroutine apply tests that attempt to disprove that the current state of G is k-colorable for the current value of k. If so, then the current Zykov tree is abandoned and the subroutine returned false. This allows the outer loop to continue with $k + 1$.

The remaining steps of the called subroutine, steps 10–13, constitute the recursive portion of the modified exhaustive algorithm. The subroutine is guaranteed to return because either there will be sufficient vertex contractions such that $n \leq k$, resulting in a true return, or sufficient edge additions such that the graph becomes complete and (as will be shown) is rejected by step 3, resulting in a false return. Note that in the event of a false return, any modifications to the current state of G resulting from the recursive calls are not returned to the outer loop.

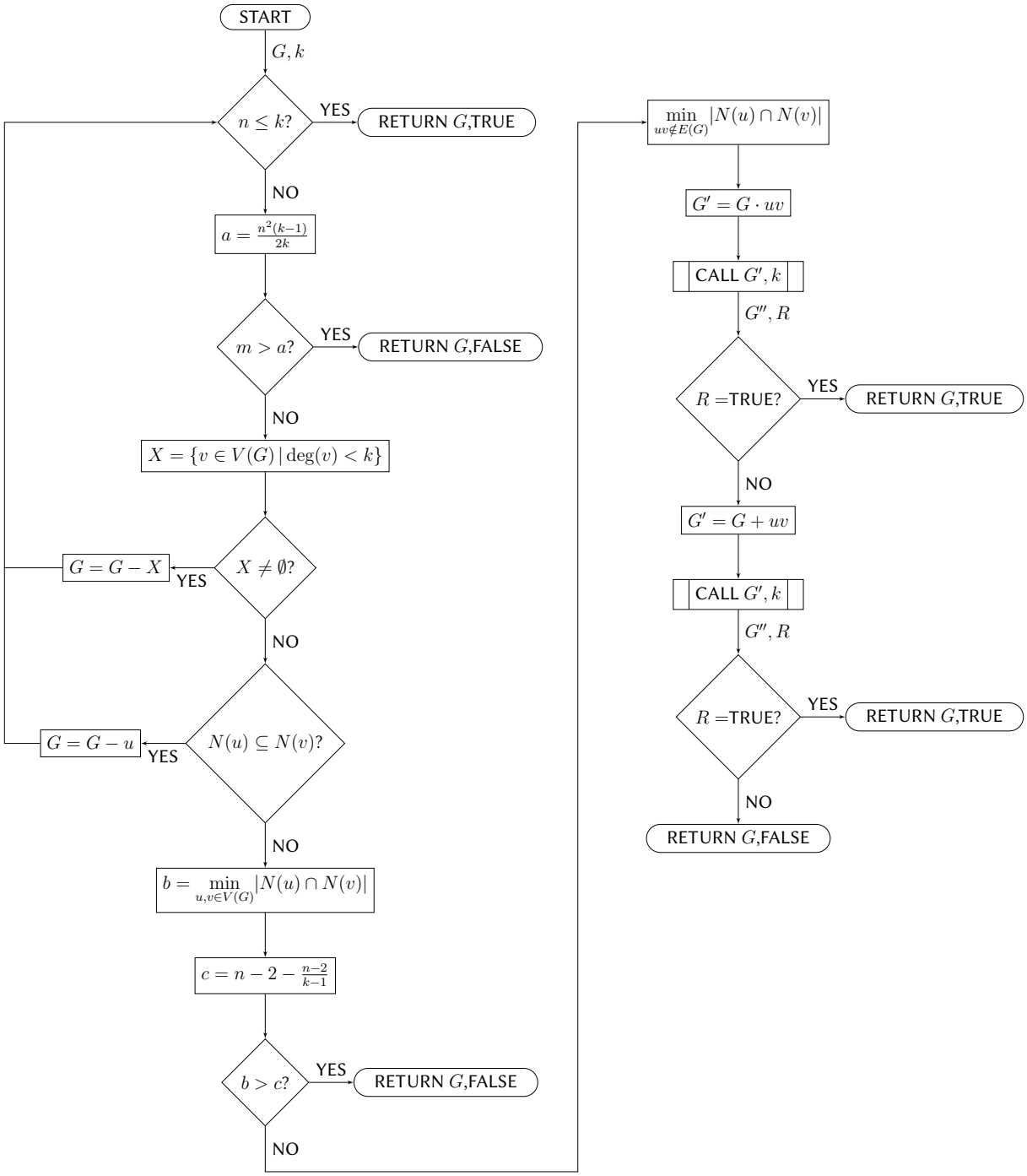


Figure 28: Algorithm Called Subroutine

4.3 Supporting Theorems

This section contains the theorems that support the steps in the called subroutine. Remember that the success check of step 1 is already supported by Proposition 3.

4.3.1 Maximum Edge Threshold

The maximum edge threshold test of steps 2 and 3 is supported by Theorem 4.3.1.

Theorem: Maximum Edge Threshold

Let G be a graph of order n and size m and let $k \in \mathbb{N}$. If G is k-colorable then:

$$m \leq \frac{n^2(k-1)}{2k}$$

Proof. Assume that G is k-colorable. This means that $V(G)$ can be distributed into k independent (some possibly empty) subsets. Call these subsets $A_1, \ldots A_k$ and let $a_i = |A_i|$. Thus, each $v \in A_i$ can be adjacent to at most $n - a_i$ other vertices in G, and hence the maximum number of edges incident to vertices in A_i is given by: $a_i(n - a_i) = na_i - a_i^2$. Now, using Theorem 2.7, the maximum number of edges in G is given by:

$$m \leq \frac{1}{2}\sum_{i=1}^{k}(na_i - a_i^2)$$

with the constraint:

$$\sum_{i=1}^{k} a_i = n$$

This problem can be solved using the Lagrange multiplier technique. We start by defining:

$$\begin{aligned} F(a_1, \ldots, a_k) &= f(a_1, \ldots, a_k) - \lambda g(a_1, \ldots, a_k) \\ &= \frac{1}{2}\sum_{i=1}^{k}(na_i - a_i^2) - \lambda \sum_{i=1}^{k} a_i \\ &= \sum_{i=1}^{k}\left(\frac{1}{2}na_i - \frac{1}{2}a_i^2 - \lambda a_i\right) \end{aligned}$$

Now, optimize by taking the gradient and setting the resulting vector equation equal to the zero vector:

$$\vec{\nabla} F = \sum_{i=1}^{k}(\frac{n}{2} - a_i - \lambda)\hat{a_i} = \vec{0}$$

This results in a system of k equations of the form:

$$\frac{n}{2} - a_i - \lambda = 0$$

And so:

$$a_i = \frac{n}{2} - \lambda$$

Plugging this result back into the contraint:

$$\sum_{i=1}^{k} a_i = \sum_{i=1}^{k} \left(\frac{n}{2} - \lambda\right) = k\left(\frac{n}{2} - \lambda\right) = n$$

Solving for λ yields:

$$\lambda = \frac{n}{2} - \frac{n}{k}$$

And finally, to get a_i in terms of n and k:

$$a_i = \frac{n}{2} - \left(\frac{n}{2} - \frac{n}{k}\right) = \frac{n}{k}$$

Therefore:

$$m \leq \frac{1}{2}\sum_{i=1}^{k}\left[n\left(\frac{n}{k}\right) - \left(\frac{n}{k}\right)^2\right] = \frac{k}{2}\left(\frac{n^2k - n^2}{k^2}\right) = \frac{n^2(k-1)}{2k}$$

■

The called subroutine actually uses the contrapositive of this result, as stated in Corollary 4.3.1.

Corollary

Let G be a graph of order n and size m and let $k \in \mathbb{N}$. If:

$$m > \frac{n^2(k-1)}{2k}$$

then G is not k-colorable.

Corollary 4.3.1 is demonstrated by Figure 4.3.1. The shown graph G has $n = 4$, $m = 5$, and $\chi(G) = 3$. Testing for $k = 2$:

$$a = \frac{4^2(2-1)}{2 \cdot 2} = 4$$

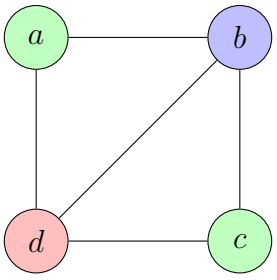
But $m = 5 > 4 = a$ and so we can conclude that G is not 2-colorable. However, testing for $k = 3$;

$$a = \frac{4^2(3-1)}{2 \cdot 3} = 5.3$$

So $m = 5 \not> 5.3 = a$ and thus G *may* be 3-colorable, since this test only provides a necessary and not a sufficient condition.

In fact, the the test of Corollary 4.3.1 will always fail for a complete graph when $k < n$. Since $k, n > 0$:

$$\frac{n(n-1)}{2} > \frac{n^2(k-1)}{2k}$$



G

Figure 29: Corollary 4.3.1 Example

$$kn(n-1) > n^2(k-1)$$
$$kn^2 - kn > kn^2 - n^2$$
$$kn < n^2$$
$$k < n$$

4.3.2 Vertex Removal

The theorems that support vertex removal make use of Lemma 4.3.2.

Lemma

Let G be a graph and let $v \in V(G)$. If G is k-colorable then $G - v$ is also k-colorable.

Proof. Assume that G is k-colorable. Let $c : V(G) \to C$ be such a coloring, and so $|C| = k$. Intuitively, removing v from G should not affect the proper coloring of the remaining vertices. Thus, we should be able to construct a proper coloring for $G - v$ based upon c. So consider the restricted function $c' = c|_{V(G-v)}$ and assume $uw \in E(G - v)$. Since c is proper:

$$c'(u) = c(u) \neq c(w) = c'(w)$$

Thus, c' is a proper coloring of $G - v$ using at most k colors.

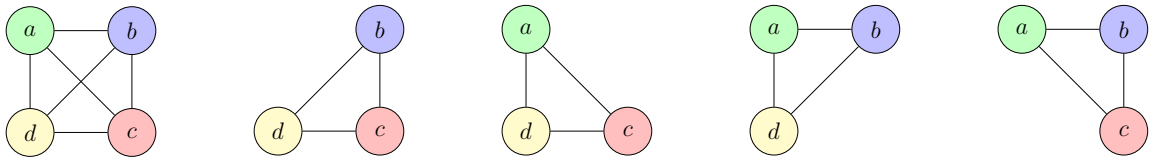
Therefore $G - v$ is k-colorable. ∎

Lemma 4.3.2 is demonstrated in Figure 4.3.2. No matter which vertex is removed, the resulting subgraph is still properly colored using at most four (in fact, three) colors.

Steps 4 and 5 remove vertices with degrees less than k. This is supported by Theorem 4.3.2.

Theorem

Let G be a graph and let $v \in V(G)$ such that $\deg(v) < k$ for some $k \in \mathbb{N}$. G is k-colorable iff $G - v$ is k-colorable.



G $G-a$ $G-b$ $G-c$ $G-d$

Figure 30: Lemma 4.3.2 Example

Proof. Assume that G is k-colorable. Therefore, by Lemma 4.3.2, $G-v$ is also k-colorable.

For the converse, assume that $G-v$ is k-colorable. Let $c : V(G-v) \to C$ be such a coloring, and so $|C| = k$. By assumption, $\deg(v) < k$, so v has at most $k-1$ neighbors in G, using at most $k-1$ colors. This means that there should be an additional color that can be assigned to v in G such that the coloring remains proper. So let $N(v) = \{v_1, \ldots, v_r\} \subseteq V(G-v)$ for some $r < k$, and let $c[N(v)] = \{c_1, \ldots, c_s\} \subset C$ for some $s \leq r < k$. Since $c[N(v)]$ is a proper subset of C, select $c_k \in C - c[N(v)]$ and define $c' : V(G) \to C$ as follows:

$$c'(u) = \begin{cases} c(u), & u \neq v \\ c_k, & u = v \end{cases}$$

Now, assume that $uw \in E(G)$ and consider the following two cases:

Case 1: $v \notin uw$

Since c is proper:

$$c'(u) = c(u) \neq c(w) = c'(w)$$

Case 2: $v \in uw$

Assume without loss of generality (AWLOG) that $u = v$. This means that $c'(v) = c_k$ and $c'(w) = c(w) \in c[N(v)]$. But $c_k \notin c[N(v)]$ and so $c'(v) \neq c'(w)$.

Thus, c' is a proper coloring of G using at most k colors.

Therefore G is k-colorable. ∎

Theorem 4.3.2 is demonstrated in Figure 4.3.2 for $k = 4$ and $\deg(v) = 3$.

The called subroutine actually removes all such vertices at once, which is supported by the inductive proof in Corollary 4.3.2.

Corollary

Let G be a graph of order n and let $X = \{v \in V(G) \,|\, \deg(v) < k\}$ for some $k \in \mathbb{N}$. G is k-colorable iff $G - X$ is k-colorable.

Proof. (by induction on $|X|$)

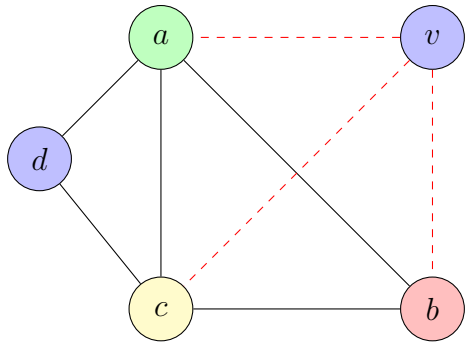


Figure 31: Theorem 4.3.2 Example

Base Case: Let $|X| = 0$.

But $G - X = G$ (trivial case).

Inductive Assumption: Let $|X| = r$.

Assume that G is k-colorable iff $G - X$ is k-colorable.

Inductive Step: Consider $|X| = r + 1$.

Since $|X| = r + 1 > 0$, there exists $v \in X$ such that $\deg(v) < k$. Let $Y = X - \{v\}$ and note that $|Y| = |X| - 1 = (r + 1) - 1 = r$. So, G is k-colorable iff $G - v$ is k-colorable (Theorem 4.3.2) iff $(G - v) - Y$ is k-colorable (inductive assumption).

Therefore, by the principle of induction, G is k-colorable iff $G - X$ is k-colorable. ∎

Returning to the example in Figure 4.3.2, note that $X = \{v, b\}$ is the set of all vertices with degree less than 4 and so both could be removed at once in accordance with Corollary 4.3.2. Furthermore, after these vertices are removed, the remaining vertices a, b, and c will all have degree 2. Since $2 < 4$, the remaining vertices are subsequently removed, leaving $n = 0 < 4$, indicating that the graph is indeed 4-colorable. This iterative collapsing of a graph is an ideal situation.

Step 6 removes vertices whose neighborhoods are subsets of other vertices. This is supported by Theorem 4.3.2.

Theorem

Let G be a graph and let $u, v \in V(G)$ such that $N(u) \subseteq N(v)$. G is k-colorable iff $G - u$ is k-colorable.

Proof. Assume that G is k-colorable. Therefore, by Lemma 4.3.2, $G - u$ is also k-colorable.

For the converse, assume that $G - u$ is k-colorable. Let $c : V(G - u) \to C$ be such a coloring, and so $|C| = k$. By definition, $u \notin N(u)$, and since, by assumption, $N(u) \subseteq N(v)$, it is also the case that $u \notin N(v)$. Hence, u is not adjacent to v in G. But everything that is adjacent to u in G is also adjacent to v in both $G - u$ and G. Since everything adjacent to v has a different color

than v, we should be able to assign u the same color as v in G in order to get a proper coloring of G.

Let $S = N(u) \cap N(v) = \{w_1, \dots w_r\} \subset V(G-u)$ for some $r = \deg(u) \leq \deg(v)$. For all $w \in S$, $w \in N(v)$ meaning $vw \in E(G-u)$. And, because c is proper, it must be the case that $c(w) \neq c(v)$. So define $c' : V(G) \to C$ as follows:

$$c'(w) = \begin{cases} c(w), & w \neq u \\ c(v), & w = u \end{cases}$$

Now, assume that $wz \in E(G)$ and consider the following two cases:

Case 1: $u \notin wz$

Since c is proper:

$$c'(w) = c(w) \neq c(z) = c'(z)$$

Case 2: $u \in wz$

Assume without loss of generality (AWLOG) that $u = w$. Since $vz \in E(G)$ and c is proper:

$$c'(u) = c(v) \neq c(z) = c'(z)$$

Thus, c' is a proper coloring of G using at most k colors.

Therefore G is k-colorable. ∎

Theorem 4.3.2 is demonstrated in Figure 4.3.2. Since $N(u) \subseteq N(v)$, G and $G-u$ are equivalently colorable. Futhermore, once u is removed, the degrees of vertices a and c will have degree 2. So if $k = 3$, those two vertices are subsequently removed by step 5. The remaining graph is of order 2, which then passes the success check of step 1 because $2 < 3$, so indeed the graph is 3-colorable. Once again, this iterative removal of vertices is very powerful.

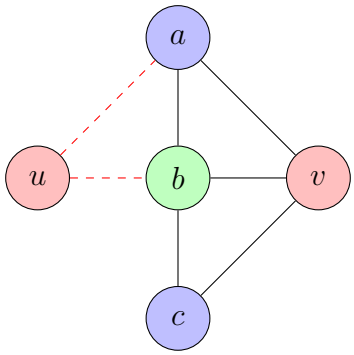


Figure 32: Theorem 4.3.2 Example

4.3.3 Minimum Shared Neighbor Upper Bound

Steps 7–9 establish an upper bound for the minimum shared neighbor count between any two vertices in a graph that is assumed to be k-colorable. This limit is dependent on the following facts that are guaranteed by previous steps:

1. $2 \leq k < n$
2. There are no $u, v \in V(G)$ such that $N(u) \subseteq N(v)$

The supporting theorem uses these facts along with Lemma 4.3.3 in its proof.

Lemma

Let G be a graph and let S be a non-empty independent subset of $V(G)$. If there exists a vertex $v \in S$ such that v is adjacent to all vertices in $V(G) - S$ (i.e., $N(v) = V(G) - S$) then for all vertices $u \in S$ it is the case that $N(u) \subseteq N(v)$.

Proof. Assume that such a v exists and then assume that $u \in S$. If $u = v$ then (trivially) $N(v) = N(v)$, so assume $u \neq v$. Furthermore, since $u, v \in S$ and S is independent (by assumption), it must be the case that u and v are not neighbors.

Case 1: $N(u) = \emptyset$.

Therefore, by definition, $N(u) = \emptyset \subseteq N(v)$.

Case 2: $N(u) \neq \emptyset$.

Assume that $w \in N(u)$. This means that w is adjacent to u and hence $w \notin S$, since S is an independent set. So $w \in V(G) - S$ and thus, by assumption, v is adjacent to w and we can conclude that $w \in N(v)$. Therefore $N(u) \subseteq N(v)$.

Therefore, for all $u \in S$, $N(u) \subseteq N(v)$. ∎

Lemma 4.3.3 is demonstrated in Figure 4.3.3. Note that since $v \in S$ is adjacent to every vertex in $V(G) - S$, vertex $u \in S$ can't help but be adjacent to some subset of $N(v)$.

Theorem 4.3.3 establishes the desired upper bound.

Theorem

Let G be a graph of order n and size m such that there are no $u, v \in V(G)$ where $N(u) \subseteq N(v)$, and let $k \in \mathbb{N}$ such that $2 \leq k < n$. If G is k-colorable then there exists two vertices $w, z \in V(G)$ such that:

$$|N(w) \cap N(z)| \leq n - 2 - \frac{n-2}{k-1}$$

Proof. Assume that G is k-colorable. This means that $V(G)$ can be distributed into k independent (some possibly empty) subsets $A_1, \ldots, A_k$ such that $a_i = |A_i|$ and $a_1 \geq a_2 \geq \cdots \geq a_k$. Since $n > k$, by the pigeonhole principle, it must be the case that $a_1 \geq 2$. Assume that $v \in A_1$.

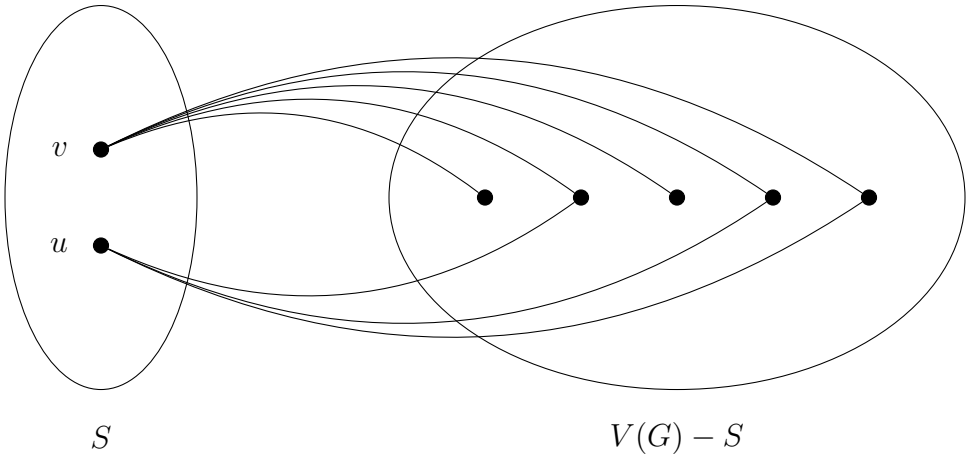


Figure 33: Lemma 4.3.3 Example

First, assume by way of contradiction (ABC) that v is adjacent to all other vertices in $V(G) - A_1$. Since $a_1 \geq 2$, there exists $u \in A_1$ such that $u \neq v$ and u is not adjacent to v. Thus, by Lemma 4.3.3, $N(u) \subseteq N(v)$, contradicting the assumption. Note that this contradiction also eliminates the degenerate case where $A_1 = V(G)$; however, this case does not occur here because the graph would be an empty graph and would have been eliminated by previous steps. Therefore, there exists some $v' \in V(G) - A_1$ such that v is not adjacent to v'. Assume that $v' \in A_i$ for some i such that $1 < i \leq k$:

Case 1: $a_i = 1$

By the pigeonhole principle:

$$a_1 \geq \left\lceil \frac{n-1}{k-1} \right\rceil \geq \frac{n-1}{k-1}$$

Now, assume by way of contradiction (ABC) that v' is adjacent to all vertices in $V(G) - A_1 - A_i$ and assume $u \in N(v)$. Then it must be the case that $u \in V(G) - A_1 - A_i$, and so u is adjacent to v', and thus $u \in N(v')$. Therefore $N(v) \subseteq N(v')$, which contradicts the assumption. This situation is demonstrated by Figure 4.3.3.

So there must exist some $u \in V(G) - A_1 - A_i$ such that u is not adjacent to v'. This results in the upper bound:

$$|N(v) \cap N(v')| \leq n - |\{u, v'\}| - a_1 \leq n - 2 - \frac{n-1}{k-1}$$

Note that since $v \in A_1$, it is already counted in a_1. Comparing this bound to the desired bound:

$$\left(n - 2 - \frac{n-2}{k-1}\right) - \left(n - 2 - \frac{n-1}{k-1}\right) = \frac{(n-1) - (n-2)}{k-1} = \frac{1}{k-1} > 0$$

for $k \geq 2$. Thus the new bound is tighter and so:

$$|N(v) \cap N(v')| \leq n - 2 - \frac{n-1}{k-1} \leq n - 2 - \frac{n-2}{k-1}$$

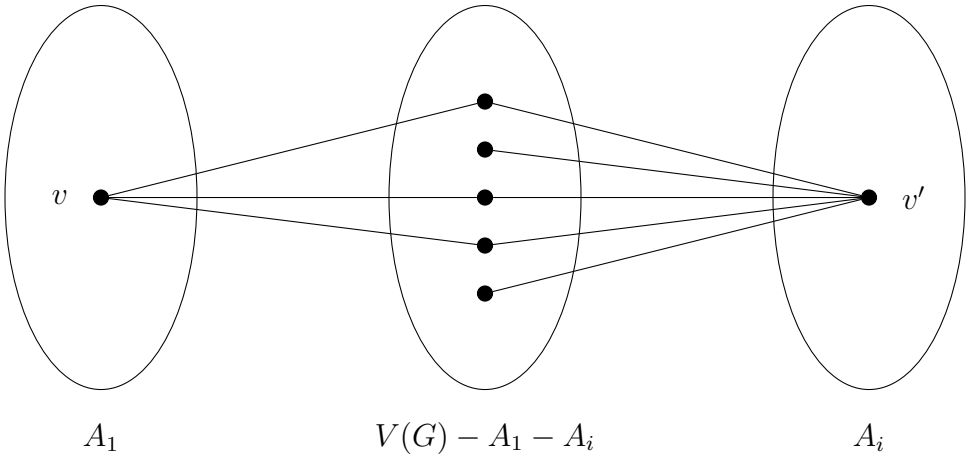


Figure 34: Case $a_i = 1$ Contradiction

Case 2: $a_i = 2$

By the pigeonhole principle:

$$a_1 \geq \left\lceil \frac{n-2}{k-1} \right\rceil \geq \frac{n-2}{k-1}$$

This results in the upper bound:

$$|N(v) \cap N(v')| \leq n - a_i - a_1 \leq n - 2 - \frac{n-2}{k-1}$$

Case 3: $a_i \geq 3$

By the pigeonhole principle:

$$a_1 \geq \left\lceil \frac{n-3}{k-1} \right\rceil \geq \frac{n-3}{k-1}$$

This results in the upper bound:

$$|N(v) \cap N(v')| \leq n - a_i - a_1 \leq n - 3 - \frac{n-3}{k-1}$$

Comparing this to the desired bound:

$$\left(n - 2 - \frac{n-2}{k-1}\right) - \left(n - 3 - \frac{n-3}{k-1}\right) = 1 - \frac{(n-3) - (n-2)}{k-1} = 1 - \frac{1}{k-1} > 0$$

for $k \geq 2$. Thus the new bound is tighter and so:

$$|N(v) \cap N(v')| \leq n - 3 - \frac{n-3}{k-1} \leq n - 2 - \frac{n-2}{k-1}$$

Therefore, there exists $w, v \in V(G)$ such that:

$$|N(w) \cap N(z)| \leq n - 2 - \frac{n-2}{k-1}$$

■

The called subroutine actually uses the contrapositive of this result, as stated in Corollary 4.3.3.

Corollary

Let G be a graph of order n and size m such that there are no $u, v \in V(G)$ where $N(u) \subseteq N(v)$, and let $k \in \mathbb{N}$ such that $2 \leq k < n$. If for all $w, z \in V(G)$ it is the case that:

$$|N(w) \cap N(z)| > n - 2 - \frac{n-2}{k-1}$$

then G is not k-colorable.

Corollary 4.3.3 is demonstrated in Figure 4.3.3. The shown graph has $n = 5$, is 3-chromatic, and has:

$$\min_{u,v \in V(G)} |N(u) \cap N(v)| = 1$$

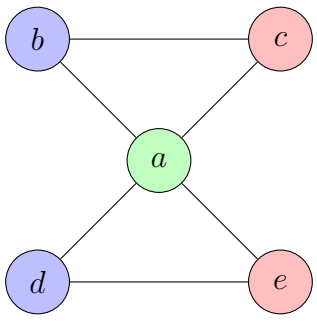
Testing for $k = 2$:

$$5 - 2 - \frac{5-2}{2-1} = 0$$

But $1 > 0$ and so we can conclude that G is not 2-colorable. However, testing for $k = 3$:

$$5 - 2 - \frac{5-2}{3-1} = \frac{3}{2}$$

So $1 \not> \frac{3}{2}$ and thus G *may* be 3-colorable, since this test only provides a necessary and not a sufficient condition.



G

Figure 35: Corollary 4.3.3 Example

4.3.4 Recursive Steps

If nothing more can be done in the preceding steps then steps 10–13 revert to the exhaustive method. Step 10 selects two non-adjacent vertices with the smallest number of shared neighbors. Such a pair must exist. Otherwise, the current state of G is complete, which would have been eliminated by step 3. The first recursive call (step 11) assumes that the two selected vertices have the same color, so they are contracted. The second recursive call (step 12) assumes that the two selected vertices have different colors, so they are joined by an added edge. Each call starts a new branch of the Zykov tree corresponding to the current value of k. If either call returns true then it can be concluded that the input graph was indeed k-colorable. Otherwise, it can be concluded that the input graph is not k-colorable and the called subroutine returns the state of G prior to the recursive calls to the outer loop.

These steps are supported by Theorem 4.3.4.

Theorem

Let G be a graph of order $n \geq 2$ and let $u, v \in G$ such that u and v are not adjacent. G is k-colorable iff $G \cdot uv$ or $G + uv$ is k-colorable.

Proof. Assume that G is k-colorable. Let $c : V(G) \to C$ be such a coloring, and so $|C| = k$. There are two possibilities, corresponding to the two recursive choices:

Case 1: u and v have the same color: $c(u) = c(v)$.

Let $c_1 = c(u) = c(v) \in C$. For all $w \in N(u) \cup N(v)$, since w is adjacent to u and v and because c is proper, it must be the case that w is a different color than the color of u and v: $c(w) \neq c_1$. Let v' be the contracted vertex, so that $N(v') = N(u) \cup N(v)$ and assign color c_1 to v'. So define $c' : V(G \cdot uv) \to C$ as follows:

$$c'(w) = \begin{cases} c(w), & w \neq v' \\ c_1, & w = v' \end{cases}$$

Now, assume that $wz \in E(G \cdot uv)$ and consider the following two cases:

Case a: $v' \notin wz$

Since c is proper:

$$c'(w) = c(w) \neq c(z) = c'(z)$$

Case b: $v' \in wz$

Assume without loss of generality that $v' = w$. Since $z \in N(v')$:

$$c'(v') = c_1 \neq c(z) = c'(z)$$

Thus, c' is a proper coloring of $G \cdot uv$ using at most k colors.

Therefore $G \cdot uv$ is k-colorable.

Case 2: u and v have the different colors: $c(u) \neq c(v)$.

By adding edge uv, u and v become adjacent and thus must have different colors. Thus, u and v can retain their same colors. So define $c' : V(G + uv) \to C$ as follows:

$$c'(w) = c(w)$$

Now, assume $wz \in E(G)$. Since c is proper:

$$c'(w) = c(w) \neq c(z) = c'(z)$$

Thus, c' is a proper coloring of $G + uv$ using at most k colors.

Therefore $G + uv$ is k-colorable.

Therefore $G \cdot uv$ or $G + uv$ is k-colorable.

For the converse, assume $G \cdot uv$ or $G + uv$ is k-colorable. Again, there are two cases:

Case 1: $G \cdot uv$ is k-colorable.

Let $c : V(G \cdot uv) \to C$ be such a coloring, and so $|C| = k$. Let v' be the contracted vertex, and so $N(v') = N(u) \cup N(v)$. Let $c(v') = c_1 \in C$. For all $w \in N(v')$ in $G \cdot uv$, since w and v' are adjacent, they must have different colors: $c(w) \neq c_1$.

Consider u and v in G. Since they are not adjacent, they can be assigned the same color. So, defined $c' : V(G) \to C$ as follows:

$$c'(w) = \begin{cases} c(w), & w \neq u, v \\ c_1, & w = u \\ c_1, & w = v \end{cases}$$

Now, assumed $wz \in V(G)$ and consider the following cases:

Case a: $uv = wz$

This is not possible since, by assumption, u is not adjacent to v.

Case b: $u \in wz$ and $v \notin wz$

Assume without loss of generality (AWLOG) that $u = w$. Since $z \in N(u)$ in G, it must be the case that $z \in N(v')$ in $G \cdot uv$. And so:

$$c'(u) = c_1 \neq c(w) = c'(w)$$

Case c: $u \notin wz$ and $v \in wz$

Assume without loss of generality (AWLOG) that $v = w$. Since $z \in N(v)$ in G, it must be the case that $z \in N(v')$ in $G \cdot uv$. And so:

$$c'(v) = c_1 \neq c(w) = c'(w)$$

Case d: $u, v \notin wz$

Since c is proper:

$$c'(w) = c(w) \neq c(z) = c'(z)$$

Thus, c' is a proper coloring of G using at most k colors.

Case 2: $G + uv$ is k-colorable.

Let $c : V(G + uv) \to C$ be such a coloring, and so $|C| = k$. Since u and v are adjacent in $G + uv$, they must have different colors. Once uv is removed in G, u and v are no longer adjacent and so there are no requirements on their colors. Thus, they can retain their original colors from $G + uv$. So define $c' : V(G) \to C$ as follows:

$$c'(w) = c(w)$$

Now assume $wz \in E(G)$. Since c is proper:

$$c'(w) = c(w) \neq c(z) = c'(z)$$

Thus, c' is a proper coloring of G using at most k colors.

Therefore G is k-colorable. ∎

5 An Example

In this section, the proposed algorithm will be applied to the example graph G ($n = 8$ and $m = 12$) shown in Figure 5 in order to determine its chromatic number. The steps of the algorithm are then traversed in reverse order to determine a chromatic coloring.

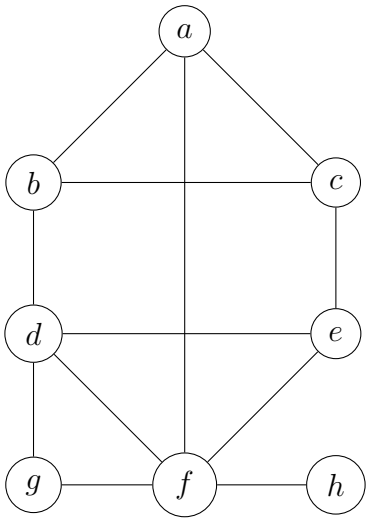


Figure 36: Example Graph

The algorithm steps are as follows. Outer loop steps are marked by "O#" and called subroutine steps are marked by "I#.". Recursive subroutine calls add a call level: "I#–#."

1. (O1) Since $n = 8 > 0$, G is not the null graph, so continue.
2. (O2) Since $n = 8 > 1$, G is not an empty graph, so continue.
3. (O3) Initialize k to 2.
4. (O4) Call the subroutine with G and $k = 2$.
5. (I1) $n = 8 \not\leq k = 2$, so continue.
6. (I2) Calculate the maximum edge threshold for $n = 8$ and $k = 2$:

$$a = \frac{n^2(k-1)}{2k} = \frac{8^2(2-1)}{2 \cdot 2} = \frac{64}{4} = 16$$

7. (I3) Since $m = 12 \not> 16 = a$, continue.
8. (I4) Since $\deg(h) = 1 < 2$, set $X = \{h\}$.
9. (I5) Since $X \neq \emptyset$, replace G with $G - X$. The result is shown in Figure 9. Now $n = 7$ and $m = 11$.
10. (I1) $n = 7 \not\leq k = 2$, so continue.

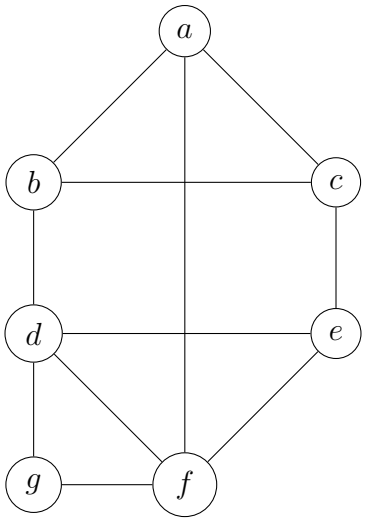


Figure 37: $G - \{h\}$

11. (I2) Calculate the maximum edge threshold for $n = 7$ and $k = 2$:

$$a = \frac{n^2(k-1)}{2k} = \frac{7^2(2-1)}{2 \cdot 2} = \frac{49}{4} \approx 12.3$$

12. (I3) Since $m = 12 \not> 12.3 = a$, continue.
13. (I4) Since $\delta(G) = 2 \geq 2 = k$, set $X = \emptyset$.
14. (I5) Since $X = \emptyset$, continue.
15. (I6) Note that $N(g) = \{d, f\}$ and $N(e) = \{c, d, f\}$. Since $N(g) \subseteq N(e)$, replace G with $G - g$. The result is shown in Figure 15. Now $n = 6$ and $m = 9$.

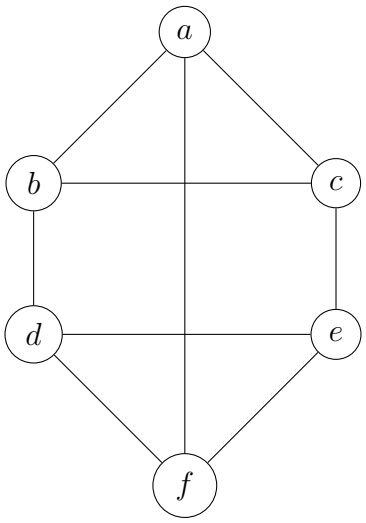


Figure 38: $G - g$

16. (I1) $n = 6 \not\leq k = 2$, so continue.
17. (I2) Calculate the maximum edge threshold for $n = 6$ and $k = 2$:

$$a = \frac{n^2(k-1)}{2k} = \frac{6^2(2-1)}{2 \cdot 2} = \frac{36}{4} = 9$$

18. (I3) Since $m = 9 \not> 9 = a$, continue.
19. (I4) Since $\delta(G) = 2 \geq 2 = k$, set $X = \emptyset$.
20. (I5) Since $X = \emptyset$, continue.
21. (I6) Since there is no $N(u) \subseteq N(v)$, continue.
22. (I7) Note that due to the symmetry of G every two vertices share exactly two neighbors:

$$b = \min_{u,v \in V(G)} |N(u) \cap N(v)]| = 2$$

23. (I8) Calculate the upper bound for minimum number of common neighbors for $n = 6$ and $k = 2$:

$$c = n - 2 - \frac{n-2}{k-1} = 6 - 2 - \frac{6-2}{2-1} = 4 - 4 = 0$$

24. (I9) Since $b = 2 > 0 = c$, conclude that G is not 2-colorable. Return false and the current state of G to the outer loop.
25. (O5) Since the called subroutine returned false, G is not 2-colorable, so continue.
26. (O6) Replace G with the graph returned by the previous call (Figure 15).
27. (O7) Increment k to 3.
28. (O4) Call the subroutine with the new G and $k = 3$.
29. (I1) $n = 6 \not\leq k = 3$, so continue.
30. (I2) Calculate the maximum edge threshold for $n = 6$ and $k = 3$:

$$a = \frac{n^2(k-1)}{2k} = \frac{6^2(3-1)}{2 \cdot 3} = \frac{72}{6} = 12$$

31. (I3) Since $m = 9 \not> 11 = a$, continue.
32. (I4) Since $\delta(G) = 3 \geq 3 = k$, set $X = \emptyset$.
33. (I5) Since $X = \emptyset$, continue.
34. (I7) Note that due to the symmetry of G every two vertices share exactly two neighbors:

$$b = \min_{u,v \in V(G)} |N(u) \cap N(v)]| = 2$$

35. (I8) Calculate the upper bound for minimum number of common neighbors for $n = 6$ and $k = 3$:
$$c = n - 2 - \frac{n-2}{k-1} = 6 - 2 - \frac{6-2}{3-1} = 4 - 2 = 2$$

36. (I9) Since $b = 2 \not> 2 = c$, continue.

37. (I10) Since every two vertices share exactly two neighbors, select any two non-adjacent vertices: a and e.

38. (I11) Recursively call the subroutine with $G \cdot ae$ and $k = 3$. The result is shown in Figure 38. Now $n = 5$ and $m = 7$.

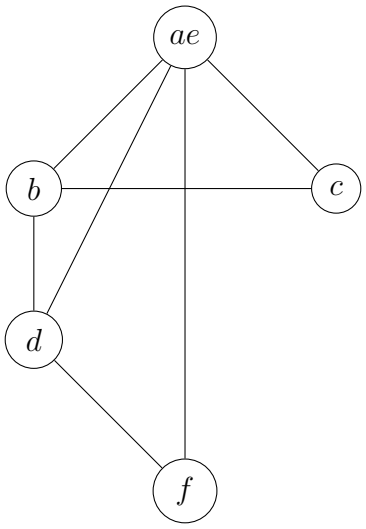


Figure 39: $G \cdot ae$

39. (I1-1) $n = 5 \not\leq k = 3$, so continue.

40. (I1-2) Calculate the maximum edge threshold for $n = 5$ and $k = 3$:
$$a = \frac{n^2(k-1)}{2k} = \frac{5^2(3-1)}{2 \cdot 3} = \frac{50}{6} \approx 8.3$$

41. (I1-3) Since $m = 7 \not> 8.3 = a$, continue.

42. (I1-4) Since $\deg(c) = \deg(f) = 2 < 3 = k$, set $X = \{c, f\}$.

43. (I1-5) Since $X \neq \emptyset$, replace G with $G - X$. The result is shown in Figure 43. Now $n = 3$ and $m = 3$.

44. (I1-1) $n = 3 \leq k = 3$, so conclude that G is 3-colorable and return true.

45. (I11) The recursive call returned true, so conclude that G is 3-colorable and return true.

46. (O5) The called subroutine returned true, so conclude that G is 3-chromatic and return $\chi(G) = 3$.

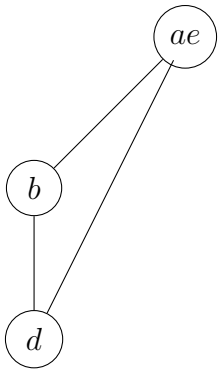


Figure 40: $G - \{c, f\}$

So the algorithm determines that the example G of Figure 5 is 3-chromatic. In order to determine a 3-chromatic coloring for G, first construct a set C of three colors:

$$C = \{\text{green}, \text{blue}, \text{red}\}$$

Next, follow the algorithm's modification steps in reverse order and color the corresponding vertices accordingly. The steps are as follows:

1. First, color the vertices that are present in the final simplified graph. Assign each vertex its own color, except assign contracted vertices the same color. This is shown in Figure 1.

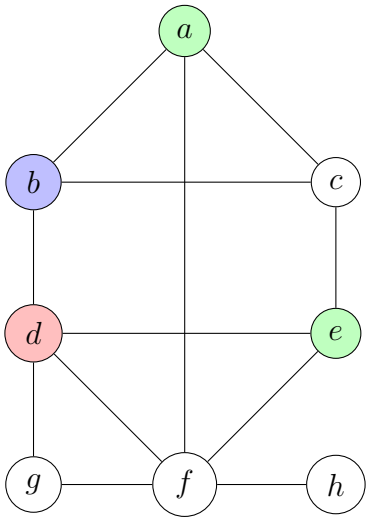


Figure 41: Initial Coloring

2. The last modification was the removal of vertices c and f. Note that color selection needs to honor the vertices already colored. This is shown in Figure 2.

3. The contracted vertices have already been colored, so the next simplification to address is the removal of g due to the neighborhood subset test. Since $N(g) \subseteq N(e)$, color g and e the same color. Since e has already been assign a color, use it for g. The result is shown in Figure 3.

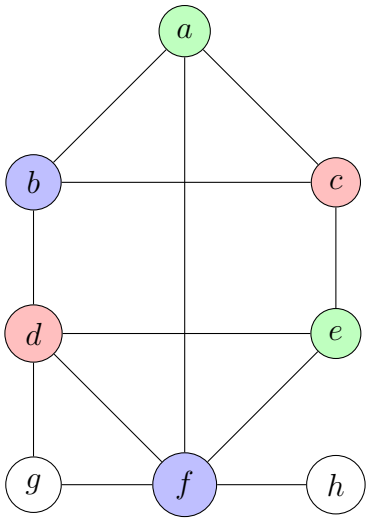


Figure 42: Coloring c and f

4. Finally, color the first removed vertex h with any appropriate color. The final result is shown in Figure 4.

At first glance, one might wonder what advantage this method of coloring has over greedy coloring. The difference is that greedy coloring only has knowledge about what has already been colored, whereas this method uses assumptions that are made during the simplifications and recursive calls on the entire graph. Thus, the greedy algorithm is heuristic, but this method selects a particular workable coloring path through the modified Zykov tree. In this way, this method is not so much different from the exhaustive algorithm.

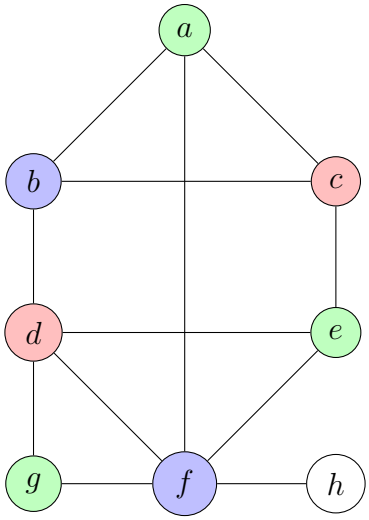


Figure 43: Coloring g

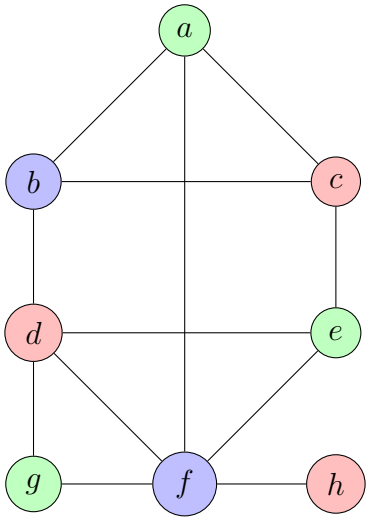


Figure 44: Final Chromatic Coloring

6 Toaster Design Case Study

This section contains a simplistic case study of how the proposed algorithm could be used to compare the functional requirement (FR) graphs resulting from two slightly different designs of the basic kitchen toaster shown in Figure 6.

Figure 45: Example Toaster

The functional requirements (FRs) are shown in Table 6. It is assumed that the design is either uncoupled or decoupled and hence the independence of the FRs is as strong as possible.

The graph G_1 for the first candidate design with $n = 7$ and $m = 13$ is shown in Figure 6.

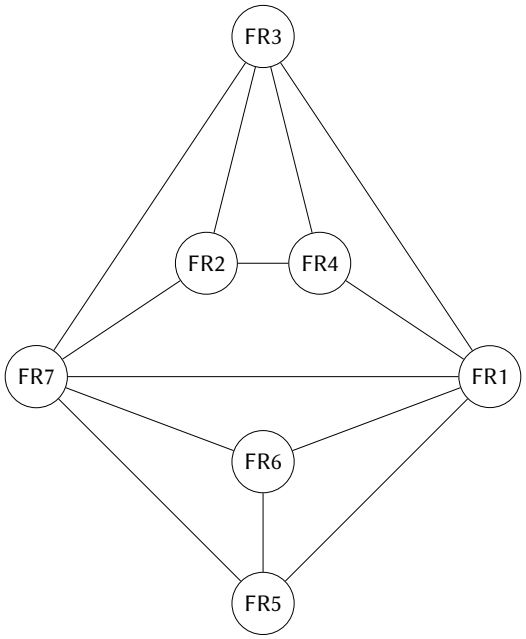
For $k = 2$, the maximum edge threshold test fails:

$$\frac{n^2(k-1)}{2k} = \frac{7^2(2-1)}{2 \cdot 2} = \frac{49}{4} \approx 12.3 < 13 = m$$

so G_1 is not 2-colorable.

FR1	Body contains all parts
FR2	Can be safely moved while hot
FR3	Can hold two slices of bread
FR4	Heats each slice of bread on both sides
FR5	Toasting is manually started
FR6	Toasting is automatically or can be manually stopped
FR7	Heat level can be controlled

Table 1: Toaster Functional Requirements



G_1

Figure 46: First Candidate Design

For $k = 3$, the maximum edge threshold test passes:

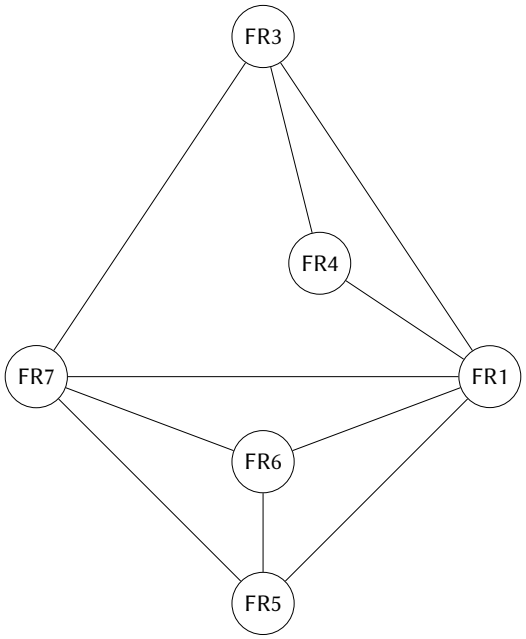
$$\frac{n^2(k-1)}{2k} = \frac{7^2(3-1)}{2 \cdot 3} = \frac{98}{6} \approx 16.3 > 13 = m$$

There are no vertices with degrees less than $k = 3$; however, note that $N(\text{FR2}) \subseteq N(\text{FR1})$, so remove vertex FR2. The result is shown in Figure 6.

Next, since $\deg(\text{FR4}) = 2 < 3 = k$, remove FR4. As a result of removing FR4, $\deg(\text{FR3}) = 2 < 3 = k$, so remove FR3 as well. The result is show in Figure 6.

At this point, $G_1 = K_4$, so it is guaranteed to fail the maximum edge threshold check for $k = 3$ and thus not be 3-colorable. At $k = 4$ we have $n = 4 \leq 4 = k$. Thus, we can conclude that G_1 is 4-chromatic. An example chromatic coloring with:

$$C = \{\text{green}, \text{blue}, \text{red}, \text{yellow}\}$$



$G_1 - \text{FR2}$

Figure 47: Design 1—Remove FR2

is show in Figure 6.

Notice in the design that FR5 (start toasting) and FR6 (stop toasting) have been forced into separate parts, conceivably to accommodate the separate "cancel" button shown in Figure 6. But what if the designer decides to eliminate the cancel button and allow manual cancellation via the lever? Thus, FR5 and FR6 no longer need to be separated, so the edge between their vertices can be eliminated. The result is shown in Figure 6.

Rerun the algorithm on G_2 for $n = 7$ and $m = 12$. For $k = 2$, the maximum edge threshold test passes:

$$\frac{n^2(k-1)}{2k} = \frac{7^2(2-1)}{2 \cdot 2} = \frac{49}{4} \approx 12.3 > 12 = m$$

Note that $N(\text{FR6}) \subseteq N(\text{FR5})$, so remove FR6. The result is shown in Figure 6.

Now, with $n = 6$ and $m = 10$, G_2 fails the maximum edge threshold test for $k = 2$:

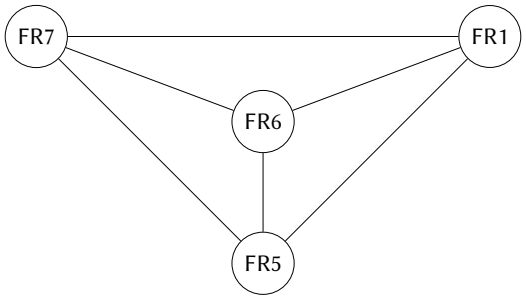
$$\frac{n^2(k-1)}{2k} = \frac{6^2(2-1)}{2 \cdot 2} = \frac{36}{4} = 9 < 10 = m$$

so G_2 is not 2-colorable.

For $k = 3$, note that $\deg(\text{FR5}) = 2 < 3 = k$, so remove FR5. The result is shown in Figure 6.

Now, with $n = 5$ and $m = 8$, G_2 passes the maximum edge threshold test for $k = 3$:

$$\frac{n^2(k-1)}{2k} = \frac{5^2(3-1)}{2 \cdot 3} = \frac{50}{6} \approx 8.3 > 8 = m$$



$G_1 - \text{FR4} - \text{FR3}$

Figure 48: Design 1—Remove FR4 and FR3

Since $N(\text{FR2}) \subseteq N(\text{FR1})$, remove FR2. The result is shown in Figure 6.

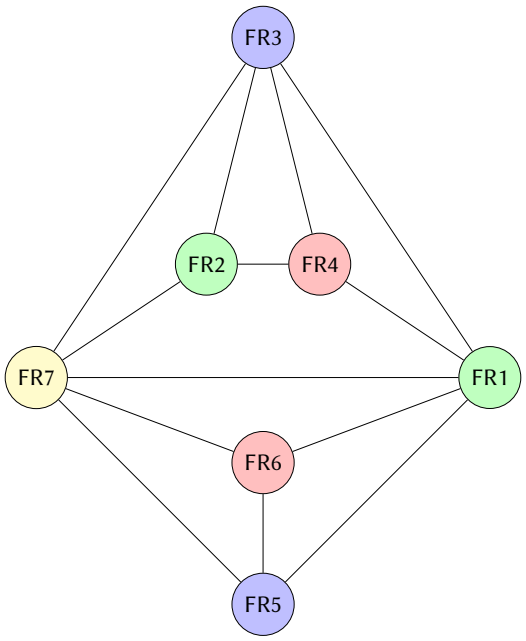
Finally, $\deg(\text{FR4}) = \deg(\text{FR7}) = 2 < 3 = k$, so remove FR4 and FR7. The result is shown in Figure 6.

This final state of G_2 has $n = 2 \leq 3 = k$, so we can conclude that G_2 is 3-chromatic. An example chromatic coloring with:

$$C = \{\text{green}, \text{blue}, \text{red}\}$$

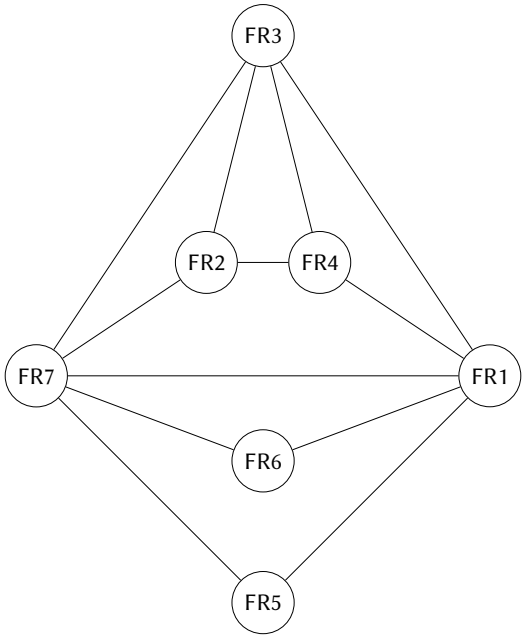
is show in Figure 6.

This process gives the designer the feedback that the second design requires only three parts instead of four, and thus has less information content and hence a higher chance of success than the first design. It will be up to the designer to weigh this result against other aspects of the design.



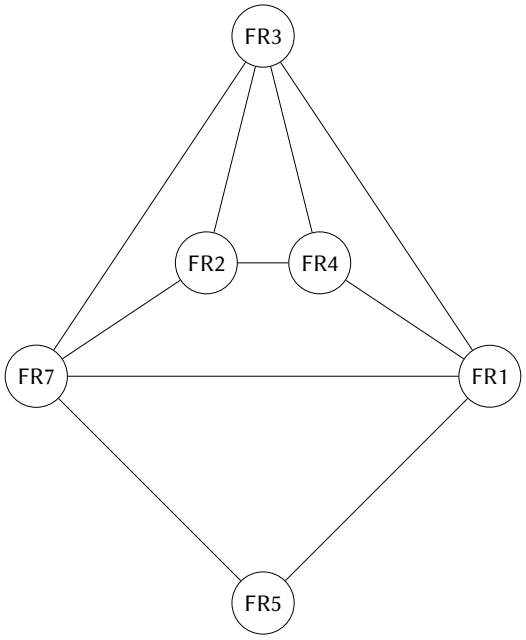
G_1

Figure 49: Design 1—Chromatic Coloring



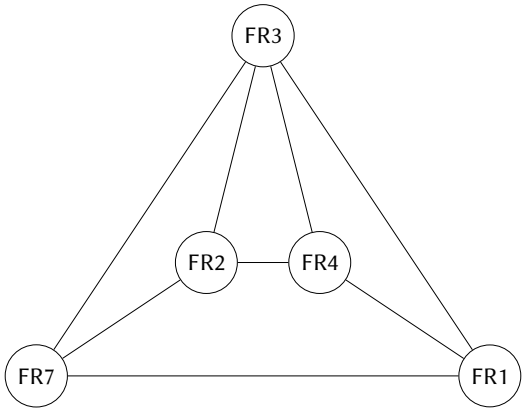
G_2

Figure 50: Second Candidate Design



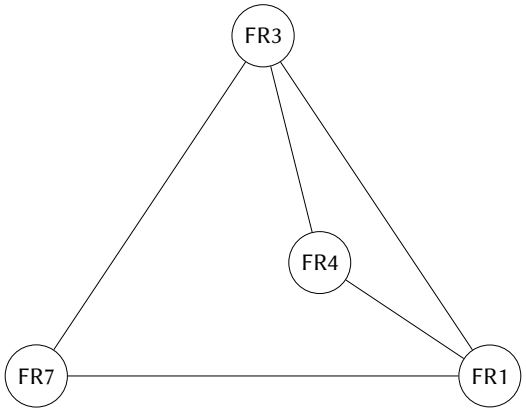
$G_2 - \text{FR6}$

Figure 51: Design 2—Remove FR6



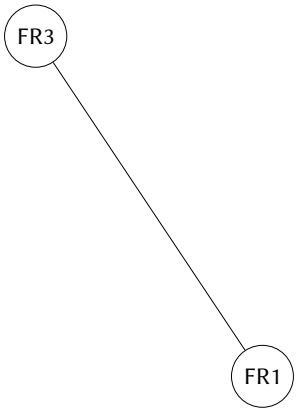
$G_2 - \text{FR5}$

Figure 52: Design 2—Remove FR5



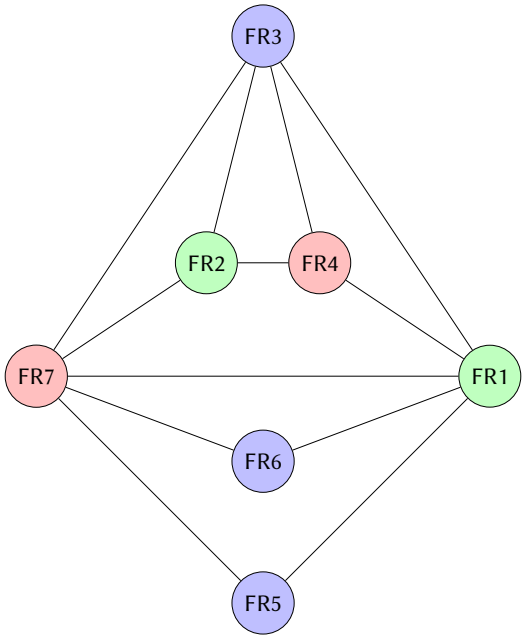
$G_2 -$ FR2

Figure 53: Design 2—Remove FR2



$G_2 -$ FR4,FR7

Figure 54: Design 2—Remove FR4 and FR7



G_2

Figure 55: Design 2—Chromatic Coloring

References

[1] Sara Behdad, Jeffery Cavallaro, Praveen Kumare Gopalakrishnan, and Sogol Jahanbekam. A graph coloring technique for identifying the minimum number of parts for physical integration in product design. *Proceedings of the ASME 2019 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, (DETC2018-98251), 2019.

[2] Sara Behdad, Praveen Kumare Gopalakrishnan, Sogol Jahanbekam, and Helen Klein. Graph partitioning technique to identify physically integrated design concepts. Proceedings of the ASME 2018 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2018", Quebec, Canada, Aug 26–29 2018.

[3] Gary Chartrand and Ping Zhang. *A First Course in Graph Theory*. Dover Publications, Mineola, New York, 2012.

[4] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[5] Nam P. Suh. *The Principles of Design*. Oxford University Press, New York, 1990.

[6] Nam P. Suh. Axiomatic design theory for systems. *Research in Engineering Design*, 10:189–209, 1998.

[7] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 2^{nd} edition, 2001.