DETERMINING A GRAPH'S CHROMATIC NUMBER FOR PART CONSOLIDATION IN AXIOMATIC DESIGN

A Thesis

Presented to

The Faculty of the Department of Mathematics

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Jeffery A. Cavallaro

December 2019

The Designated Thesis Committee Approves the Thesis Titled

DETERMINING A GRAPH'S CHROMATIC NUMBER FOR PART CONSOLIDATION IN AXIOMATIC DESIGN

by

Jeffery A. Cavallaro

APPROVED FOR THE DEPARTMENT OF MATHEMATICS

SAN JOSÉ STATE UNIVERSITY

December 2019

Sogol Jahanbekam, Ph.D.	Department of Mathematics
Wasin So, Ph.D.	Department of Mathematics
Jordan Schettler, Ph.D.	Department of Mathematics

ABSTRACT

DETERMINING A GRAPH'S CHROMATIC NUMBER FOR PART CONSOLIDATION IN AXIOMATIC DESIGN

by Jeffery A. Cavallaro

Mechanical engineering design practices are increasingly moving towards a framework called *axiomatic design*, which starts with a set of independent *functional requirements* (FRs) for a manufactured product. A key tenet of axiomatic design is to decrease the *information content* of a design in order to increase the chance of manufacturing success. One important way to decrease information content is to fulfill multiple FRs by a single part: a process known as *part consolidation*. Thus, an important parameter when comparing two candidate designs is the minimum number of parts needed to satisfy all of the FRs. One possible method for determining the minimum number of parts is to represent the problem by a graph, where the vertices are the FRs and the edges represent the need to separate their endpoint FRs into separate parts. The answer then becomes the solution to a vertex coloring problem: finding the chromatic number of such a graph. Unfortunately, the chromatic number problem is known to be NP-hard. This research investigates a new algorithm that determines the chromatic number for a graph and compares the new algorithm's computer runtime performance to existing Zykov branch and bound algorithms using random graph analysis.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation–USA under grants #CMMI-1727190 and CMMI-1727743. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

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 [1]. 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 [1].

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 Chernobyl, 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. As a result, axiomatic design has been widely adopted by companies to promote efficiency and accuracy in the design process, resulting in more reliable products and reduced manufacturing costs [2].

The following sections provide an overview of axiomatic design as specified in detail by Professor Suh [1], [3], and summarized by Behdad, et al. [4], [5]. 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 to do 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. Therefore, 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 Axiomatic Design Framework

The *best* design among a set of candidates is the design that exactly satisfies a clearly defined set of needs and has the greatest probability of success. In a desire not to hinder the creative element needed for design, yet provide some methodology to distinguish bad

designs from good designs from better designs, the diagram in Fig. 1 establishes the overall framework for axiomatic design.

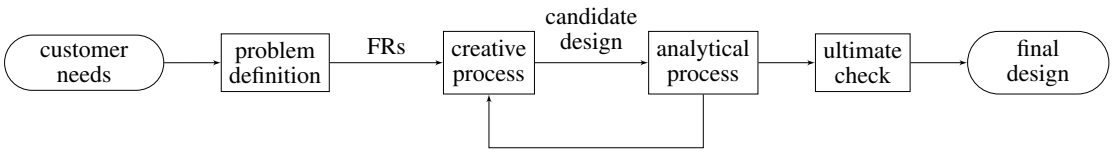


Fig. 1. The axiomatic design framework.

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 met 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 DPs 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 Fig. 2.

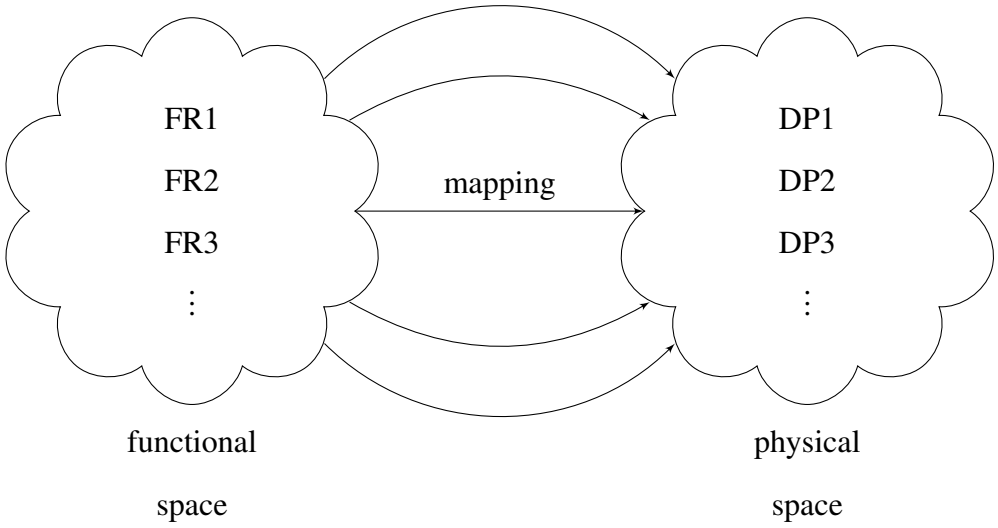


Fig. 2. Mapping FRs to DPs.

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 is 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 a FR related to a moving part in a design. The DP for this 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.

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

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 $m = n$. A full

discussion of the design matrix element values is beyond the scope of this research. Instead, the following two values are used:

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

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 Fig. 1 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 [1] imposes a restriction on the FR/DP mapping:

Axiom 1 (The Independence Axiom)**.** *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 Fig. 3. Uncoupled designs completely adhere to the independence axiom.

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

Fig. 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 Fig. 4. Although decoupled designs do not completely adhere to the independence axiom, they may be reasonable compromises in designs that address complex problems.

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

Fig. 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 Fig. 5. Coupled designs are in complete violation of the independence axiom and generally should be decoupled by reworking the FRs or by adding additional DPs.

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

Fig. 5. A coupled design.

Unfortunately, adding additional DPs runs counter to the second axiom: the information axiom [1].

Axiom 2 (The Information Axiom)**.** *The best design is a functionally uncoupled design that has the minimum information content.*

The amount of *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 its individual DPs.

From these two axioms come the following seven corollaries [1]:

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.*

The theorems that arise from these axioms and corollaries are used to prove that an optimal design results from a square design matrix. In other words, the number of FRs should be equal to the number of DPs. First, consider the case where there are more FRs than DPs. 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 [1].

Theorem 1. *When the number of DPs is less that the number of FRs, either a coupled design results or the FRs cannot be satisfied.*

A possible solution to this problem is given by Theorem 2 [1].

Theorem 2. *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 Fig. 6. Note that the addition of DP3 results in a decoupled design.

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

Fig. 6. Decoupling a design by adding DPs.

Next, consider the case where the number of FRs is less than the number of DPs. Assuming that the design is not coupled, this means that either a DP exists that does not address any FRs or multiple DPs exist that address a single FR and hence can be integrated into a single DP. Such a design is called a *redundant* design. This is addressed by Theorem 3 [1].

Theorem 3. *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 4 [1].

Theorem 4. *In an ideal design, the number of FRs is equal to the number of DPs.*

1.4 Part Consolidation

One particularly important design parameter 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 [1].

Decreasing the number of parts in a design while maintaining the independence of the FRs is consistent with Corollary 3 and lowers the information content of the design. In fact, Tang, et al. [6] describe how part consolidation reduces the weight and complexity of a final product while boosting reliability and reducing cost.

An informative example of part consolidation is the combination can/bottle opener shown in Fig. 7. The design of this handy utensil has two FRs, shown in Table 1. 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 (although that would be a popular trick around a campfire).

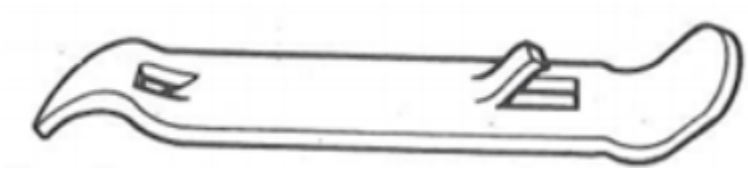


Fig. 7. A part consolidation example.

Table 1

Opener Functional Requirements

FR1	Open beverage cans
FR2	Open beverage bottles

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 in a FR/DP hierarchy. The designer is required to construct a graph whose vertices are the FRs and whose edges indicate that the endpoint FRs need to be realized by separate parts due to specific design constraints. How these edges are actually 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.

To be a viable tool, a computer program must be able to deliver an answer in a reasonable amount of time. Unfortunately, finding the chromatic number of a graph is known to be an NP-hard problem [7] and so the time required to find a solution grows exponentially with the number of FR vertices and edges in the graph. 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 undergraduate graph theory text by Chartrand and Zhang (2012) [8] and the graduate graph theory text by West (2001) [9].

2.1 Simple Graphs

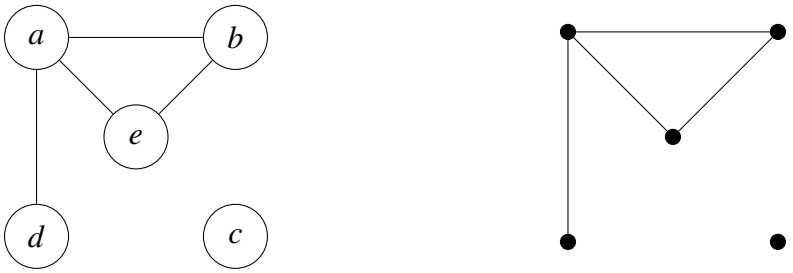
The problem of part consolidation is best served by a class of graphs called *simple graphs*. A *simple graph* is a mathematical object represented by an ordered pair $G = (V, E)$ consisting of a finite and non-empty set of *vertices* (also called *nodes*): $V(G)$, and a finite and possibly empty set of edges: $E(G)$. Each edge is represented by a two-element subset of $V(G)$ called the *endpoints* of the edge: $E(G) \subseteq \mathscr{P}_2(V(G))$. 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.

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 or vu.

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 Fig. 8.



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

$$E(G) = \{ab, ad, ae, be\}$$

Fig. 8. An example graph (labeled and unlabeled).

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 Fig. 8, vertex a is adjacent to vertices b, d, and e; however, it is not adjacent to vertex c.

As demonstrated by vertex c in Fig. 8, there is no requirement that every vertex in a graph be 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 exactly one endpoint. Note that two edges cannot share both of their endpoints — otherwise they would be multiple edges, which are not allowed in simple graphs. In the example graph of Fig. 8, edge ab is adjacent to edges ad and ae via common vertex a, and be via common vertex b.

2.2 Order and Size

Two of the most important characteristics of a graph are its *order* and its *size*. The *order* of a graph G, denoted by $n(G)$ or just n when G is unambiguous, is the number of vertices in G: $n = |V(G)|$. The *size* of a graph G, denoted by $m(G)$ or just m when G is unambiguous, is the number of edges in G: $m = |E(G)|$. In the example graph of Fig. 8: $n = 5$ and $m = 4$.

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

Theorem 5. *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:

- 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, a graph is called *non-trivial*.
- An *empty* graph is a graph containing no edges ($m = 0$).

Thus, both the null and trivial graphs are empty.

2.3 Graph Relations

In addition to its 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. Note that there are no particular limitations on the nature of such a relation — everything from a basic relation to a bijective function are possible. 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 a graph, we will not do so.

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 possible relation associated with a graph G is a bijective function $\ell : V(G) \to L$ that assigns to each vertex a unique identifying label. The codomain L is the set of available labels. When such a function is present, the graph is said to be a *labeled* graph and the vertices are considered to be distinct. Otherwise, a graph is said to be *unlabeled* and the vertices are considered to be identical (only the structure of the graph matters).

The vertices in a labeled graph are typically drawn 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 Fig. 8: 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.

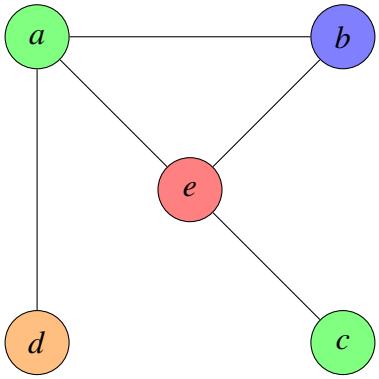
The design graphs that act as the inputs to the proposed algorithm are labeled graphs, where the labels represent the various functional requirements: $\mathrm{FR}_1, \mathrm{FR}_2, \mathrm{FR}_3, \ldots, \mathrm{FR}_n$.

2.3.2 Vertex Color

Other graph theory problems require that a graph's vertices 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 on a graph G is by using a *coloring* function $c : V(G) \to C$, 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 correspond to empty sets in the distribution.

A coloring $c : V(G) \to C$ on a graph G is called *proper* when no two adjacent vertices in G are assigned the same color: for all $u, v \in V(G)$, if $uv \in E(G)$ then $c(u) \neq c(v)$. Otherwise, c is called *improper*. A proper coloring 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 Fig. 9.



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

$$c(a) = \text{green}$$

$$c(b) = \text{blue}$$

$$c(c) = \text{green}$$

$$c(d) = \text{orange}$$

$$c(e) = \text{red}$$

Fig. 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 coloring shown in Fig. 9 is surjective, but we can add an unused color to C:

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

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 the statement in Proposition 1.

Proposition 1. *Let G be a graph. If G is k-colorable then 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 and let $r \in \mathbb{N}$. If G is k-colorable then G is $(k+r)$-colorable.*

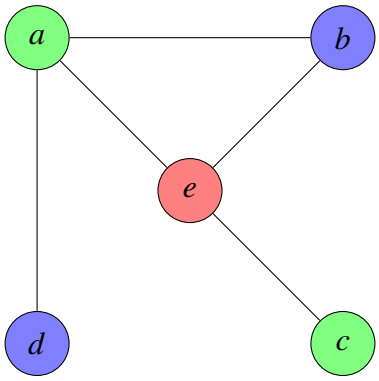
Furthermore, for a graph G of order n, if $n \leq k$ then we can conclude that G is k-colorable, since there are sufficient colors to assign each vertex its own unique color.

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. This minimum k 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 of G.

Returning to the example 4-coloring of Fig. 9, 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 Fig. 10. 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.



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

$$c(a) = \text{green}$$
$$c(b) = \text{blue}$$
$$c(c) = \text{green}$$
$$c(d) = \text{blue}$$
$$c(e) = \text{red}$$

Fig. 10. A Graph with a 3-chromatic coloring.

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

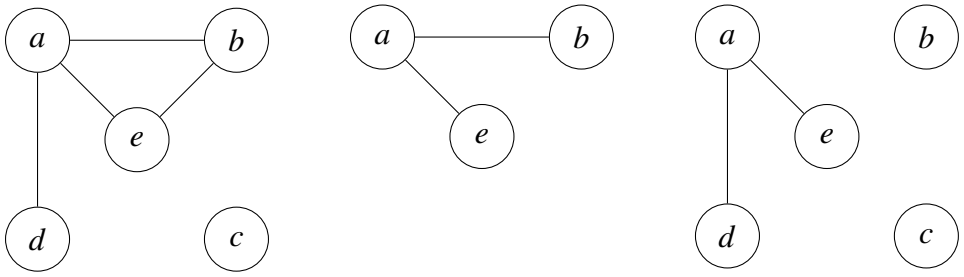
contraction. Before describing these mutators, it will be helpful to describe what is meant by graph equality and a *subgraph* of a graph.

To say that graph G is *equal* to graph H, denoted by $G = H$, means that the *exact same graph* is given two names: G and H. It is specifically *not* a comparison between two different graphs. Two different graphs that have the same structure, meaning there exists an adjacency-preserving bijection between the vertices of the two graphs, are referred to as being *isomorphic*, denoted by $G \cong H$, and are not considered to be equal. Of course, if $G = H$ then $G \cong H$; however, the converse is usually not true. In fact, $G = H$ if and only if $V(G) = V(H)$ and $E(G) = E(H)$.

To say that H is a *subgraph* of a graph G, denoted by $H \subseteq G$, means that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Thus, H can be achieved by removing zero or more vertices and/or edges from G, and G can be achieved by adding zero or more vertices and/or edges to H. Once again, H is not a different graph. If H is a different graph then one can say that it is isomorphic to a subgraph of G, but not a subgraph of G itself. By definition, $G \subseteq G$ and the null graph is a subgraph of every graph.

When G and H differ by at least one vertex or edge then H is called a *proper subgraph* of G, denoted by $H \subset G$. In fact, $H \subset G$ if and only if $H \subseteq G$ but $H \neq G$, meaning $V(H) \subset V(G)$ or $E(H) \subset E(G)$. When H and G differ by edges only: $V(H) = V(G)$ and $E(H) \subseteq E(G)$, then H is called a *spanning subgraph* of G.

The concept of subgraphs is demonstrated by graphs G, H, and F in Fig. 11. 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 $H \subset G$ (proper) $F \subset G$ (spanning)

Fig. 11. Subgraph examples.

An *induced* subgraph is a special type of subgraph. Let G be a graph and let $S \subseteq V(G)$. The subgraph of G *induced* by S, denoted by $G[S]$, is a subgraph H such that $V(H) = S$ and for every $u, v \in S$, if u and v are adjacent in G then they are also adjacent in H. Such a subgraph H is called an *induced subgraph* of G.

In the examples of Fig. 11, 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. Fig. 12 adds edge be so that H is now an induced subgraph of G.

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

Fig. 12. An induced subgraph.

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.

Fig. 13 shows an example of vertex removal: vertices c and e are removed, along with their incident edges ae and be.

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

Fig. 13. Vertex removal.

If S consists of a single vertex v then the alternate syntax $G - v$ is used instead of $G - \{v\}$.

The proposed algorithm uses vertex removal to simplify a graph that is assumed to be k-colorable into a smaller graph that is also k-colorable.

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 $G + uv$.

Fig. 14 shows an example of edge addition: edge cd is added.

G $G+cd$

Fig. 14. Edge addition.

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:

$$G - X = H(V(G), E(G) - X)$$

Thus, only edges are removed — 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.

Fig. 15 shows an example of edge removal: edges *ae* and *be* are removed.

G $\qquad\qquad$ $G-\{ae,be\}$

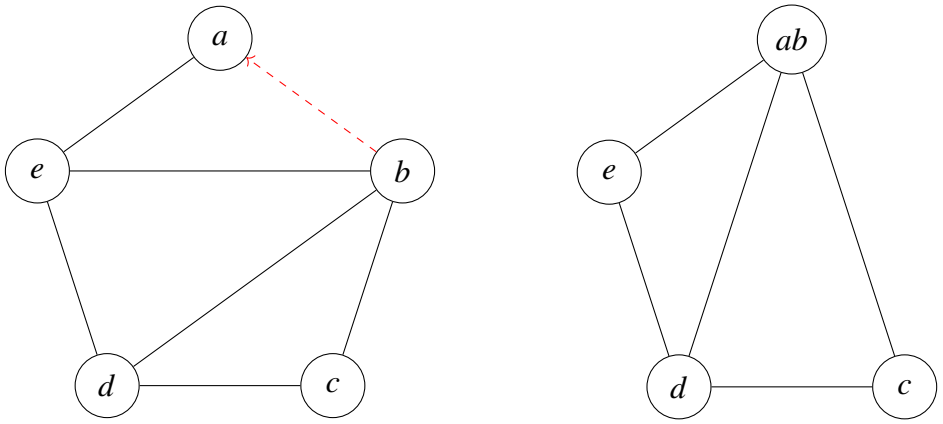
Fig. 15. Edge removal.

If X consists of a single edge e then the alternate syntax $G-e$ is used instead of $G-\{e\}$.

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.

Fig. 16 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 $G \cdot ab$

Fig. 16. Vertex contraction.

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 Independent Sets

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. For a graph G, an *independent* set $S \subseteq V(G)$, sometimes called a *stable* set, is a set of pairwise non-adjacent vertices in G: for all $u, v \in S$, $uv \notin E(G)$. By definition, the empty set is an independent set. The cardinality of the largest possible independent set for a graph G, denoted by $\alpha(G)$, is called the *independence number* for G.

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 non-empty 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.

A well-known algorithm for determining the chromatic number of a graph, introduced by Christofides (1971) [10], is based on finding the largest possible independent set S for a graph and then continually repeating the process on $G - S$ until $S = \emptyset$. The resulting partition represents a chromatic coloring of G. Part of this research compares the Christofides algorithm to the proposed algorithm.

2.7 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.

2.7.1 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*.

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 $v = w_k$:

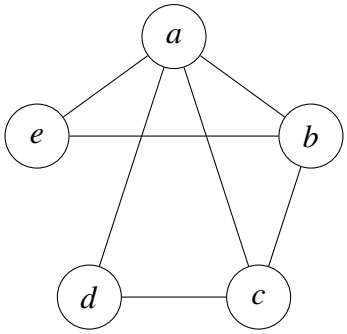
$$W = (u = w_0, w_1, \ldots, w_k = v)$$

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 Fig. 17. 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.



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

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

$|W_1| = |W_2| = 4$

Fig. 17. Open and closed walks.

Vertices and edges are allowed to be repeated during a walk. Certain special walks can be defined by restricting such repeats:

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 Fig. 17.

2.7.2 Paths

When discussing the connectedness of a graph, the main concern is the existence of paths between 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 6. *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 least 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 minimum 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 Fig. 18.

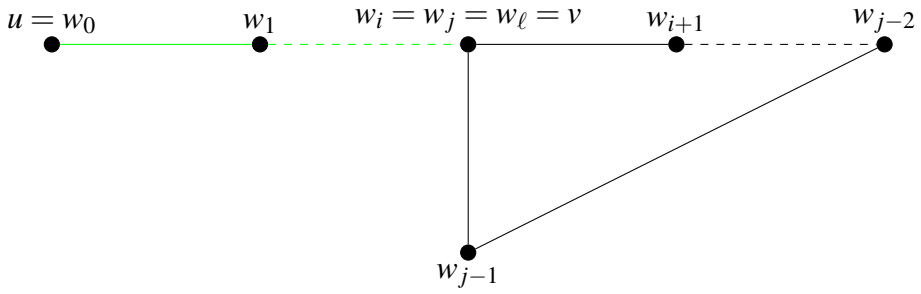


Fig. 18. Repeated vertex at end case.

Let $P' = (u = w_0, w_1, \ldots, w_i = v)$ be the walk shown in green in Fig. 18. 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 Fig. 19.

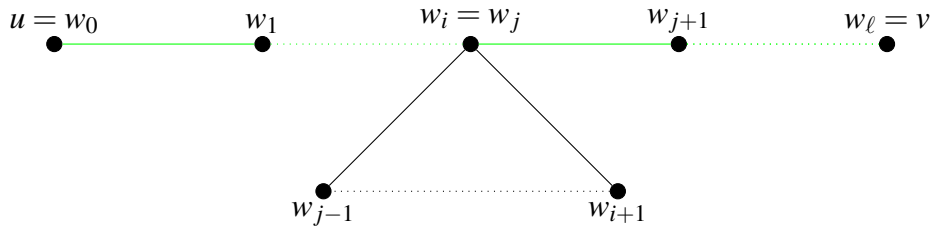


Fig. 19. Repeated vertex inside case.

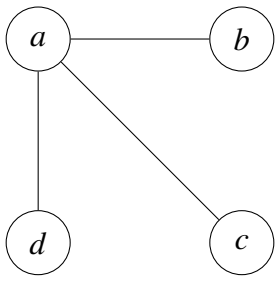
Let $P' = (u = w_0, w_1, \ldots, w_i, w_{j+1}, \ldots, w_\ell = v)$ be the walk shown in green in the Fig. 19. P' is a $u-v$ walk of length $\ell - (j-i) < \ell$ in G.

Both cases contradict the minimality of the length of P.

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

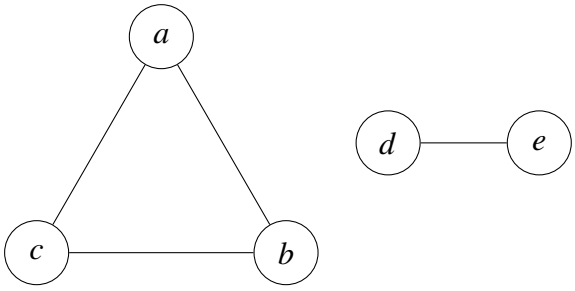
2.7.3 Connected

A *connected* graph G is a graph whose vertices are all connected: 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 20.



(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

Fig. 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).

2.7.4 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.

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 $\mathscr{G}$: for every $F \in \mathscr{G} - \{H\}$ it is the case that $H \not\subset F$. The number of distinct components in G is denoted by k(G), or just k if G is unambiguous. 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 7. *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. □

2.7.5 *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, as pointed out by Zykov (1949) [11], 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 Fig. 20. 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) [12]. 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. Given a graph G of order n and size m, this algorithm runs in $\max(n,m)$ steps.

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.

2.8 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. Let G be a graph and let $u \in V(G)$. If $v \in V(G)$ is adjacent to u then u and v are called *neighbors*. Note that for simple graphs, a vertex is never a neighbor of itself. The *neighborhood* of u, denoted by $N(u)$, is the set of all the neighbors of u in G:

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

The *degree* of u, denoted by $\deg_G(u)$ or just $\deg(u)$ if G is unambiguous, is then defined to be the cardinality of its neighborhood: $\deg(u) = |N(u)|$. Thus, 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:

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

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

Therefore, we can state the conclusion of Proposition 5:

Proposition 5. *Let G be a graph of order n. 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, as shown in Table 2.

Table 2

Classifying Vertex u in Graph G

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

Isolated vertices have degree 0; they are not adjacent to any other vertex in G. *Pendant* (also called *end* or *leaf*) vertices have degree 1; they are adjacent to exactly one other vertex in G. *Universal* vertices are adjacent to every other vertex in G. *Even* vertices are adjacent to an even number of vertices in G and *odd* vertices are adjacent to an odd number of vertices in G. Note that if G has a universal vertex then it cannot have an isolated vertex, and vice-versa.

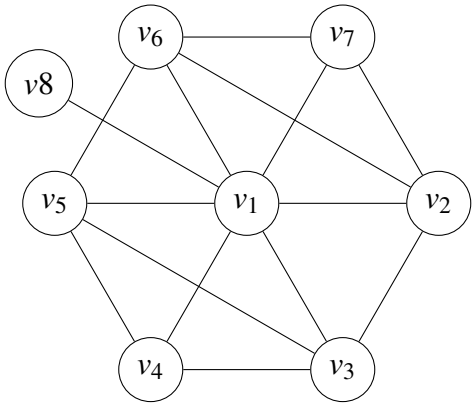
The degrees of the vertices in a graph and the number of edges in the graph are related by the so-called First Theorem of Graph Theory:

Theorem 8 (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 Fig. 21.



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

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	

Fig. 21. Vertex degrees and the first theorem of graph theory.

2.9 Special Graphs

The following sections described some special classes of graphs that are important to the execution of the proposed algorithm.

2.9.1 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 if and only if $n = 1$. Examples of empty graphs are shown in Fig. 22.

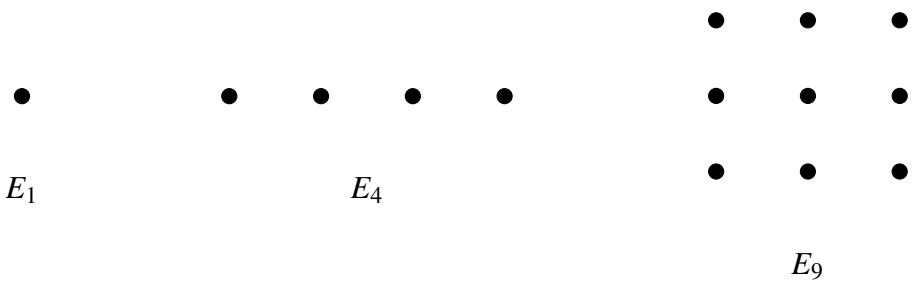


Fig. 22. Empty graphs.

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.

2.9.2 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 Fig. 23.

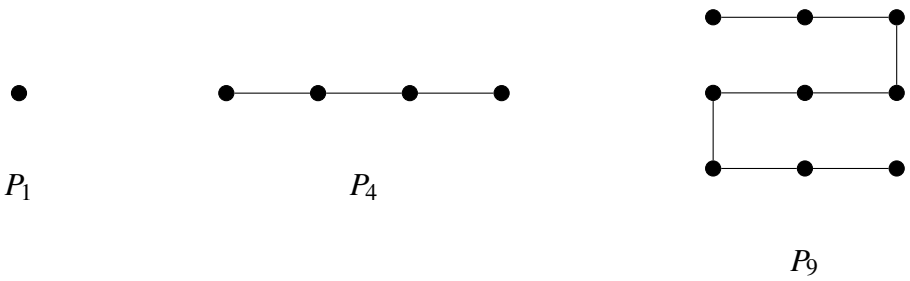


Fig. 23. Path graphs.

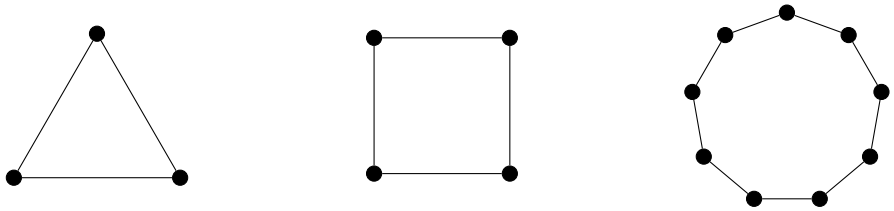
Note that $P_1 = E_1$ is 1-chromatic, whereas $P_{n>1}$ is 2-chromatic.

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

2.9.3 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 Fig. 24.



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

Fig. 24. Cycle graphs.

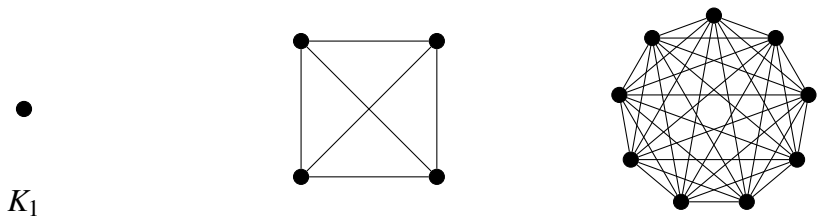
Note that even cycles are 2-chromatic; however, odd cycles are 3-chromatic.

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

2.9.4 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: $E(G) = \mathscr{P}_2(V(G))$. Thus, all of the vertices in a complete graph are universal.

Examples of complete graphs are shown in Fig. 25.



K_1 K_4 K_9

Fig. 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.

2.9.5 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. Thus, 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 Fig. 26. The root vertex r is shown in red and the leaf vertices b, e, g, h, i, j, k are shown in green.

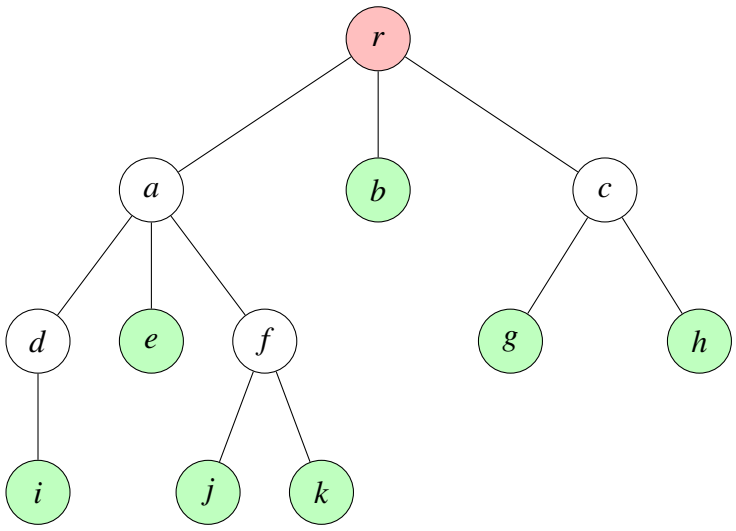


Fig. 26. A tree organized from root to leaves.

Trees are important because they can be used to track so-called “branch-and-bound” algorithms; each vertex represents a branch choice of the algorithm and thus a particular

state of the problem. All such states can be visited using a so-called *depth-first* walk. In the example in Fig. 26, such a depth-first walk would be:

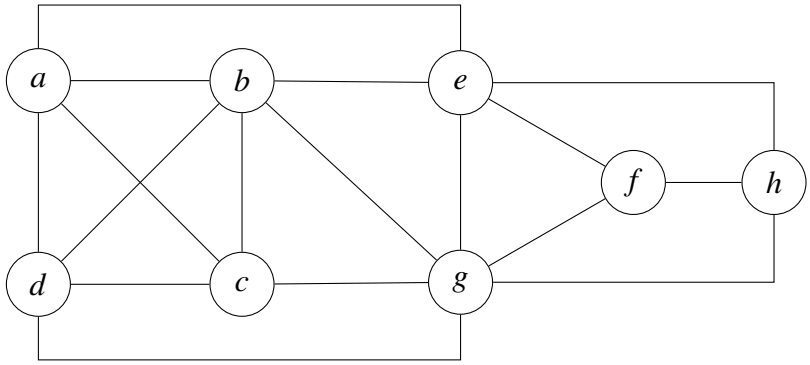
$$(r,a,d,i,d,a,e,a,f,j,f,k,f,a,r,b,r,c,g,c,h,c,r)$$

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 via a sequence of vertex contraction and edge addition choices, the tree is called a *Zykov* tree and the algorithm is called a *Zykov* algorithm [7]. Zykov algorithms are described in detail in the following sections. In fact, the proposed algorithm is a variation of a standard Zykov algorithm.

2.10 Cliques

A *clique* is a complete subgraph of a graph. A *maximal* clique is a clique that cannot be extended by an additional vertex. The *maximum* clique is the maximal clique of highest order. The *clique number* of a graph G, denoted by $\omega(G)$, is the order of the maximum clique in G. Consider the example in Fig. 27. G has many 2-cliques; however, none of them are maximal. G has several maximal 3-cliques — for example: $G[\{b,e,g\}]$. There is only one maximal 4-clique: $G[\{a,b,c,d\}]$ and no clique of higher order. Thus, $\omega(G) = 4$.



G

Fig. 27. A graph with cliques.

Since a k-clique of a graph G needs at least k colors in any proper coloring of G, the clique number of G provides a nice lower bound for the chromatic number of G. Unfortunately, the clique number problem is known to be NP-hard as well [13]. Thus, there are many attempts in the literature to find a good lower bound for the clique number of a graph G, usually denoted by $\omega'(G)$. If such a lower bound is known then the conclusion of Proposition 6 holds:

Proposition 6. *Let G be a graph with clique number lower bound* $\omega'(G)$*:*

$$\omega'(G) \leq \omega(G) \leq \chi(G)$$

Examples of such algorithms are discussed in detail in later sections, since they are typically used to "bound" Zykov branch-and-bound algorithms.

3 ALGORITHMS AND COMPUTATIONAL COMPLEXITY

The chromatic number problem is *inherently intractable*. Informally, this means that finding a solution for a given input graph, regardless of the means, can take a very, very long time in the worst cases. In fact, the chromatic number problem is a member of a class of problems referred to as being *NP-hard*, meaning that the time to find a solution increases exponentially with the number of vertices. Thus, trying to find an efficient but exact method of solution in all cases is a fool's errand; about the best that can be done is to perform better than existing well-known methods in most cases. [14]. In order to better understand why the proposed algorithm is inherently intractable and how to compare it with existing well-known chromatic number algorithms, this section provides an overview of the computational complexity of algorithms.

In today's modern computer age, the term *algorithm* is used without much thought to represent a well-defined and finite (however large) sequence of steps to accomplish some well-defined goal, usually within the context of a computer program. However, the programmable computer didn't exist until well into the 1950s. In fact, the formal study of algorithms didn't really begin until the early 20^{th} century. In order to understand why the chromatic number problem is so intractable and how to measure that intractability, it will be helpful to investigate this fairly recent history of algorithms.

The etymology for the word *algorithm* is the Latinized name *Algorithmi* for Muhammad ibn Musa al-Khwarizmi of algebra fame. In Europe, the term algorithmi became synonymous with any well-defined procedure used in mathematical problem solving, such as the division algorithm of the Babylonians, al-Khwarizmi's completing the square, the sieve of Eratosthenes for finding primes, and Euclid's algorithm for find the GCD of two numbers.

The modern understanding of algorithms has its roots in attempts by mathematicians of 1920s and 1930s to formalize mathematical systems. German mathematician David

Hilbert begins in 1920 by stating what he felt were requirements for a proper mathematical system based on some language or set of rules:

Completeness. Every proposition that is true can be proven so using the rules of the system.

Consistency. No contradictions arise when following the rules.

Decidability. There exists a method for deciding whether something is true or false.

It is probably true that Hilbert had in mind mathematical systems based on first-order logic, which was the modus operandi of mathematicians of the time. However, Austrian mathematician Kurt Gödel upset the apple cart in 1931 by showing that mathematical systems cannot be proved to be both complete and consistent. Thus, only the third requirement of decideability remained, what was commonly referred to as the *Entscheidunsproblem* [15].

In 1935, British mathematician and early pioneer of programmable computing Max Newman was giving a lecture on the Entscheidunsproblem at Cambridge. Newman put forth of the question of whether there exists a *mechanical* process that would meet the requirements of decideability. In the audience was a young Alan Turing who would take the challenge for a mechical process to heart. In 1936, Turing would submit his famous paper, "On Computable Numbers, with an Application to the Entscheidungsproblem [16]," which provided a definition of computable functions and a theoretical general-purpose machine to compute such functions: the A-machine, what we know today as the Turing machine. Turing showed that there are some problems that appear to be true; however, there is no way to design a process to prove that they are actually true - or that the original process even terminates with an answer (the halting problem). Most of these types of problems involve inherent contractions such as the Epimenides paradox: Epimenides the Cretan says that all Cretans are liars. [15]. Thus, Hilbert's third pillar was demolished. [15].

It should be noted that American mathematician Alan Church arrived at the same conclusions as Turing a bit earlier with his 1935 paper on lambda calculus. Unlike Newton and Leibniz, Turing and Church were willing collaborators, with Church eventually becoming Turing's PhD advisor at Princeton. Indeed, Turing adopted the use of the lambda calculus in his work [15].

The next step in the evolution of computability and algorithms occurred in the field of cryptography during WWII. Although cryptography had been important in WWI, the horrors of the new era of warfare instigated by Adolf Hitler brought the importance of codebreaking to the forefront. The story of the Polish code breakers and the efforts at Bletchley Park, in which Turing played a major role, are now legendary. The goal was to break the German Enigma code, and in this application the notion of the intractability of some problems becomes evident. [15].

The Enigma code was a letter-swapping code; however, the swap table changed with each letter. Clear messages were encoded using a typewriter-like device with the following characteristics:

1) Three slightly-different 26-position (one for each letter) rotors that selected the swap for the current letter in the clear text. As each character was added to the ciphertext, the rotors would rotate based on their gear mechanism to select the next swap, thus changing the swap table each time.
2) The ability to initialize the position of each rotor at the start of the message, which acted as a seed for the current cipher.
3) A plugboard where ten pairs of manual letter swaps could be specified.

Each of the twenty letters that made up the ten pairs on the plugboard were distinct. Furthermore, the ordering of the pairs was insignificant. Thus, the number of possible

configurations N of the plugboard is given by:

$$\begin{aligned} N &= \frac{\binom{26}{2}\binom{24}{2}\binom{22}{2}\binom{20}{2}\binom{18}{2}\binom{16}{2}\binom{14}{2}\binom{12}{2}\binom{10}{2}\binom{8}{2}}{10!} \\ &= \frac{26!24!22!20!18!16!14!12!10!8!}{2^{10}24!22!20!18!16!14!12!10!8!6!10!} \\ &= \frac{26!}{2^{10}6!10!} \end{aligned}$$

The German army used a set of five rotors, any three of which could be selected for a day's messages. Thus, the number of possible initial configurations for the army Enigma machine M is:

$$M = 5 \cdot 4 \cdot 3 \cdot 26^3 \cdot \frac{26!}{2^{10}6!10!} = 158.9 \times 10^{18}$$

To put this in perspective, an Intel i7 has a speed of about $50,000$ MIPS. Assuming that each one of Enigma's initial configurations could be tried in one instruction means that it would take about 100 years to try all possible combinations. Since it would actually take several thousand instructions to test each possible configuration, the real answer is upwards of 100,000 years — intractible indeed.

Although the German army used a set of five rotors, the German navy used a set eight rotors. An in 1942, both changed to a four rotor system. It is at this point that the art of crypography comes into play. The goal is to use techniques like letter frequency and candidate clear text segments to eliminate enough of the configuration so that the correct configuration can be found in a reasonable amount of time.

Literature Cited

[1] N. P. Suh, *The Principles of Design*. New York: Oxford University Press, 1990.

[2] R. A. Shirwaiker and G. E. Okudan, "Triz and axiomatic design: A review of case-studies and a proposed synergistic use," *Journal of Intelligent Manufacturing*, vol. 19, pp. 33–47, February 2008.

[3] N. P. Suh, "Axiomatic design theory for systems," *Research in Engineering Design*, vol. 10, pp. 189–209, 1998.

[4] S. Behdad, J. Cavallaro, P. K. Gopalakrishnan, and S. Jahanbekam, "A graph coloring technique for identifying the minimum number of parts for physical integration in product design," in *Proceedings of the ASME 2019 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, no. DETC2019-98251 in IDETC/CIE 2019, (Anaheim, CA), Aug 18–21 2019.

[5] S. Behdad, P. K. Gopalakrishnan, S. Jahanbekam, and H. Klein, "Graph partitioning technique to identify physically integrated design concepts," in *Proceedings of the ASME 2018 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, no. DETC2018-85646 in IDETC/CIE 2018, (Quebec, Canada), Aug 26–29 2018.

[6] Y. Tang, S. Yang, and Y. F. Zhao, "Sustainable design for additive manufacturing through functionality integration and part consolidation," in *Handbook of Sustainability in Additive Manufacturing* (S. S. Muthu and M. M. Savalani, eds.), vol. 1, pp. 101–144, Singapore: Springer, 2016.

[7] C. McDiarmid, "Determining the chromatic number of a graph," *SIAM Journal on Computing*, vol. 8, pp. 1–14, 02 1979.

[8] G. Chartrand and P. Zhang, *A First Course in Graph Theory*. Mineola, New York: Dover Publications, 2012.

[9] D. B. West, *Introduction to Graph Theory*. Upper Saddle River, NJ: Prentice Hall, 2^{nd} ed., 2001.

[10] N. Christofides, "An algorithm for the chromatic number of a graph," *The Computer Journal*, vol. 14, pp. 38–39, January 1971.

[11] A. A. Zykov, *On Some Properties of Linear Complexes*, vol. 7 of *American Mathematical Society Translations Series One*. American Mathematical Society, 1949 (translated 1952).

[12] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Communications of the ACM*, vol. 16, pp. 372–378, June 1973.

[13] D. G. Corneil and B. Graham, "An algorithm for determining the chromatic number of a graph," *SIAM Journal on Computing*, vol. 2, pp. 311–318, December 1973.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman and Company, 1979.

[15] D. Turing, *Prof: Alan Turing Decoded*. Stroud, Gloucestershire: The History Press, 2015.

[16] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. 42, no. 1, pp. 230–265, 1937.