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 9, 2019

1 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 [2, 4] and class notes compiled by the author during his undergraduate and graduate graph theory classes at SJSU.

1.1 Simple Graphs

The problem of part consolidation is best served by a class of graphs called *simple graphs*:

Definition: Simple Graph

A *simple graph* is a mathematical object represented by a tuple $G = (V, E, \ldots)$ consisting of a non-empty and finite set of *vertices* (also called *nodes*) $V(G)$, a finite and possibly empty set of edges $E(G)$, and zero of more relations. Each edge is represented by a two-element subset of $V(G)$ called the *endpoints* of the edge:

$$E(G) \subseteq \mathcal{P}_2\left(V(G)\right)$$

Each relation has $V(G)$ or $E(G)$ as its domain and is used to associated vertices or edges with problem-specific attributes.

For the remainder of this work, the use of the term "graph" implies a "simple graph."

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:

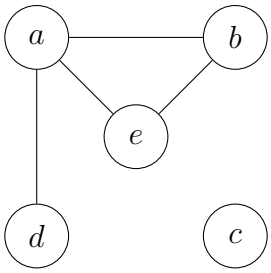
1. Every two vertices of a graph are the endpoints of at most one edge; there are no so-called *multiple* edges between two vertices.
2. The two endpoint vertices of an edge are always distinct; there are no so-called *loop* edges on a single vertex.
3. The two endpoint vertices are unordered, suggesting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

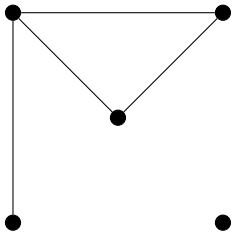
A part consolidation problem can be represented by a graph whose vertices are the functional requirements (FRs) of the design and whose edges indicate which endpoint FRs should never be combined into a single part.

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 1.1.

When referring to the edges in a graph, the following common notation will be used:

Notation: Edge





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

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

Figure 1: An Example Graph (labeled and unlabeled)

The edge $\{u, v\}$ is represented by the simple juxtaposition uv or vu.

Note that there is no requirement that every vertex in a graph be an endpoint to some edge:

Definition: Isolated Vertex

Let G be a graph and let $u \in V(G)$. To say that u is an *isolated* vertex means that it is not an endpoint for any edge in $E(G)$:

$$\forall\, e \in E(G), u \notin e$$

In the example graph of Figure 1.1, notice that vertex c is an isolated vertex.

When two vertices are the endpoints of the same edge the vertices are said to be *adjacent* or are called *neighbors*:

Definition: Adjacent Vertices

Let G be a graph and let $u, v \in V(G)$. To say that u and v are *adjacent* vertices, also called *neighbors*, means that they are the endpoints of some edge $e \in E(G)$:

$$\exists\, e \in E(G), e = uv$$

The edge e is said to *join* its two endpoint vertices u and v. Furthermore, the edge e is said to be *incident* to its endpoint vertices u and v.

In the example graph of Figure 1.1, notice that vertex a is adjacent to vertices b, e, and d; and vertex b is adjacent to vertex e.

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 1.1, notice that ab is adjacent to ad, ae, and be; and ae is adjacent to be.

1.2 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 1.1, 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.

1.3 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.

1.3.1 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 1.1: 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:

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

1.3.2 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.

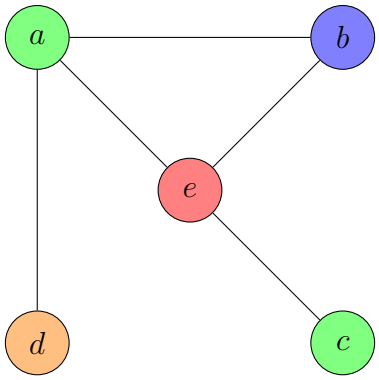
An example of a 4-coloring is shown in Figure 1.3.2.

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 1.3.2 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:

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



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

Figure 2: A Graph with a 4-coloring

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:

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

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

Proposition 2

Let G be a graph:

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

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 $k \in \mathbb{N}$:

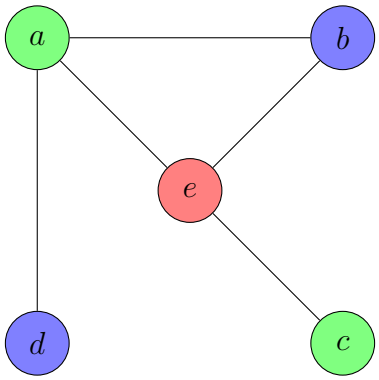
If $n \leq k$ then G is k-colorable.

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 1.3.2, 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 1.3.2. 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.



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

Figure 3: A Graph with a Chromatic 3-coloring

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:

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

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.

1.4 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.

1.4.1 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.

1.4.2 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 $E(H) \subseteq E(G)$.

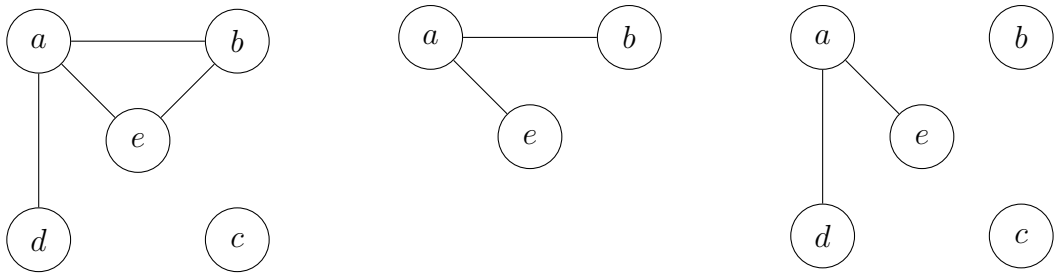
Thus, given a graph G and a subgraph H, there should be a sequence of zero or more vertex and/or edge removals to obtain H from G. Likewise, there should be a sequence of zero or more vertex and/or edge additions to obtain G from H. If H is a proper subgraph of G then H and G differ by at least one removed 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 1.4.2. 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.

1.4.3 Induced Subgraphs

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

Definition: Induced Subgraph



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

Figure 4: Subgraph Examples

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 1.4.2, 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 1.4.3 adds edge be so that H is now an induced subgraph of G.

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

Figure 5: Induced Subgraph Example

1.5 Mutators

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

1.5.1 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 1.5.1 shows an example of vertex removal: vertices c and e are removed, along with their incident edges ae and be.

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

Figure 6: Vertex Removal Example

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.

1.5.2 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 $G + uv$.

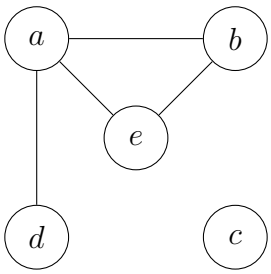
Figure 1.5.2 shows an example of edge addition: edge cd is added.

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

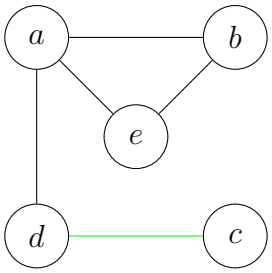
1.5.3 Edge Removal

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:

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



G

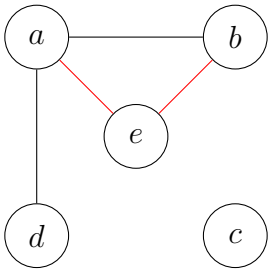


$G + cd$

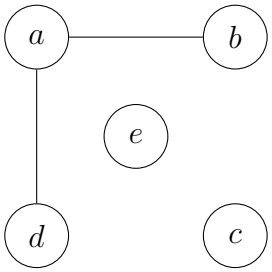
Figure 7: Edge Addition Example

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



G



$G - \{ae, be\}$

Figure 8: Edge Removal Example

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

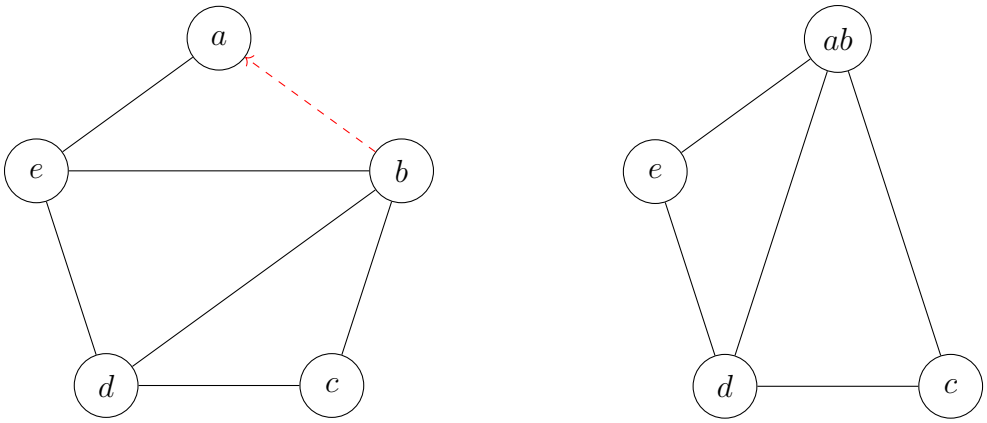
$$G - e = G - \{e\}$$

1.5.4 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 1.5.4 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.

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.



G $G \cdot ab$

Figure 9: Vertex Contraction Example

1.6 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.

1.6.1 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: $k = |W|$.

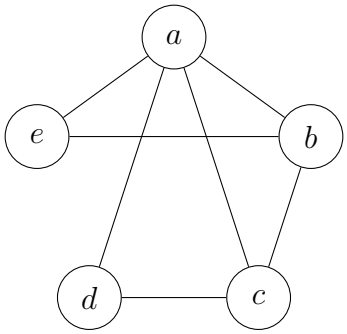
A *trivial* walk is a walk of zero length — i.e, a single vertex: $W = (u)$.

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 1.6.1. 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.



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

$|W_1| = |W_2| = 4$

Figure 10: Open and Closed Walks in a Graph

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

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 1.6.1.

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 1.6.1.

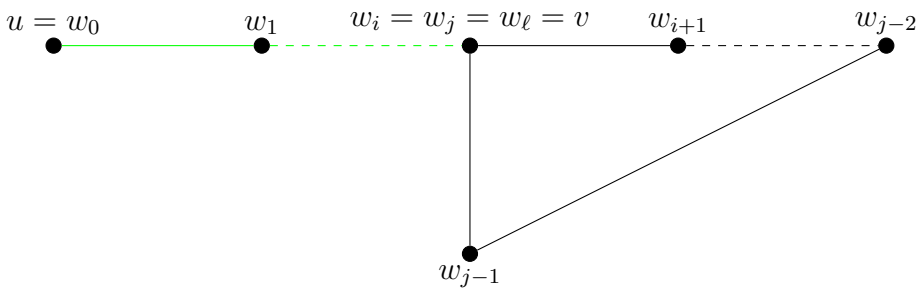


Figure 11: Repeated Vertex at End Case

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 1.6.1.

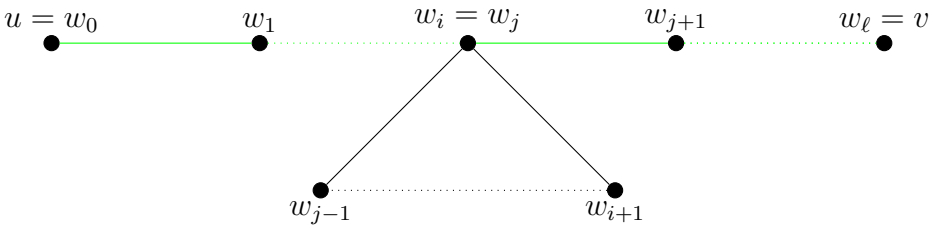


Figure 12: Repeated Vertex Inside Case

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. ∎

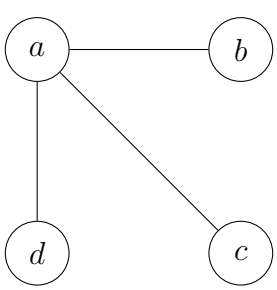
1.6.2 Connected

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

Definition: Connected Graph

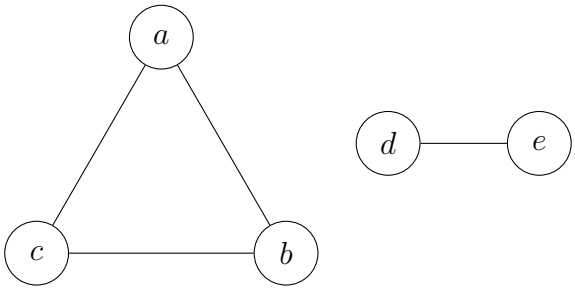
To say that a graph G is *connected* means that for all $u, v \in V(G)$ there exists a $u - v$ path. Otherwise, G is said to be *disconnected*.

Examples of connected and disconnected graphs are shown in figure 1.6.2.



(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 13: 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).

1.6.3 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. ∎

1.6.4 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 1.6.2. 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 [3]. 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.

1.7 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:

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

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:

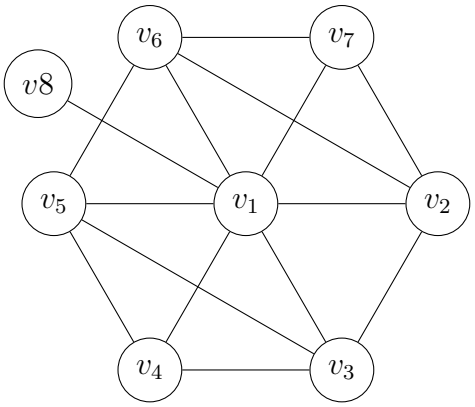
$$\sum_{v \in V(G)} \deg(v) = 2m$$

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

These concepts are demonstrated by the graph in Figure 1.7; note that the sum of the vertex degrees is 30, which is twice the number of edges in the graph.

1.8 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.



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	

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

Figure 14: Vertex Degrees

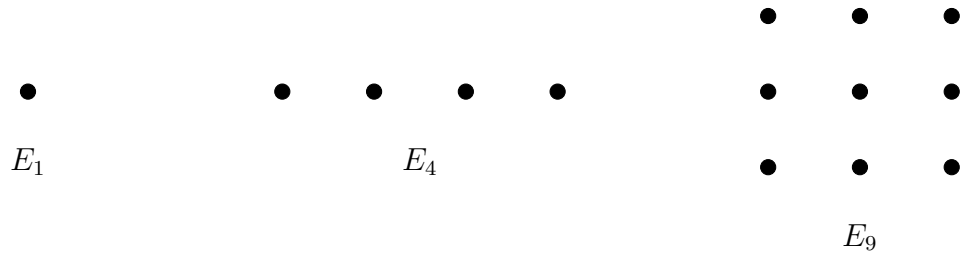


Figure 15: Empty Graphs

1.8.1 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 1.8.1.

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.

1.8.2 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 1.8.2.

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.

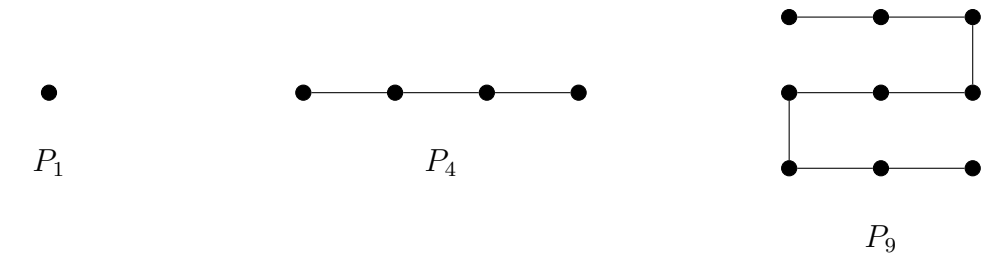


Figure 16: Path Graphs

1.8.3 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 1.8.3.

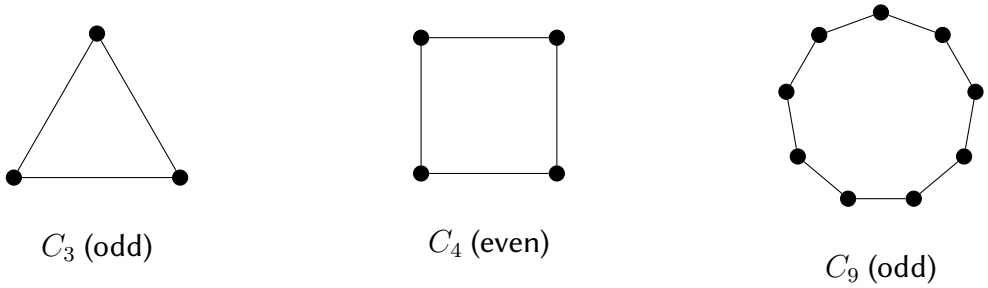


Figure 17: Cycle Graphs

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

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

1.8.4 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 1.8.4.

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.

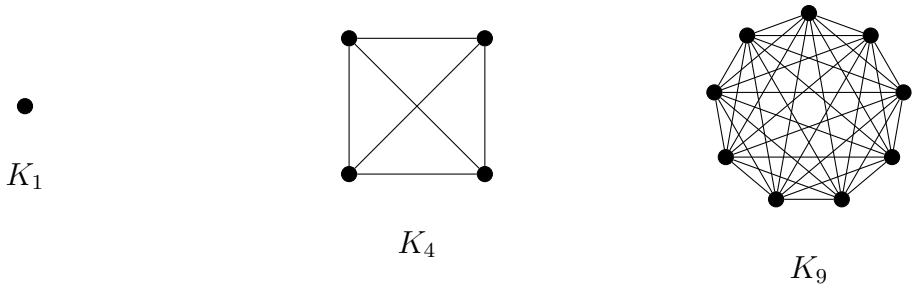


Figure 18: Complete Graphs

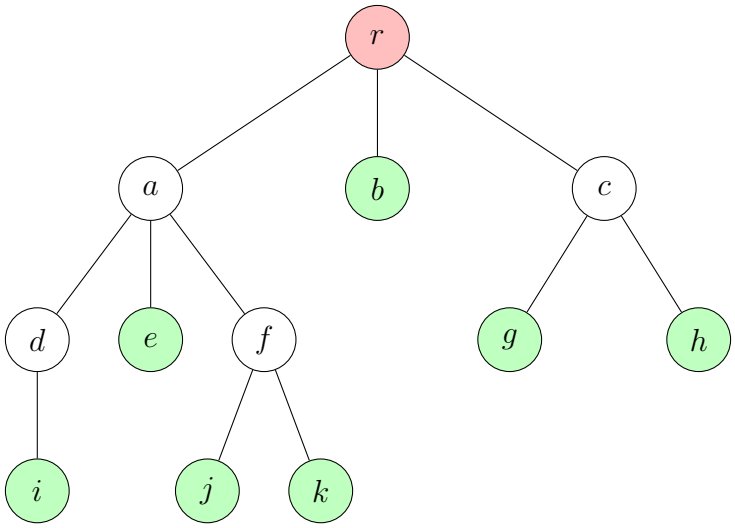


Figure 19: A Tree Organized from Root to Leaves

1.8.5 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 1.8.5. 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 "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 1.8.5, 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.

2 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.

2.1 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 2.1.

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.

2.2 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 2.3.1).

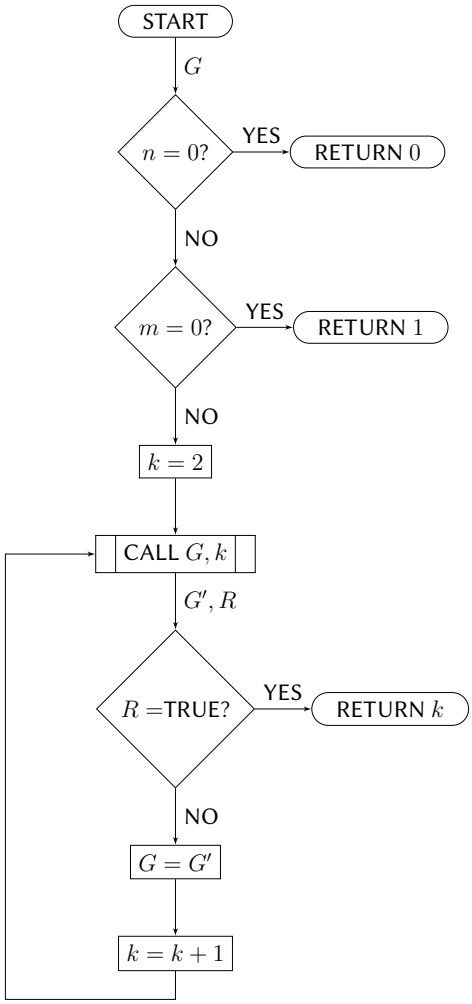


Figure 20: Algorithm Outer Loop

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 2.3.2).

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 2.3.2).

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 2.3.3).
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 2.3.4).
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 2.3.4).
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 2.2.

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.

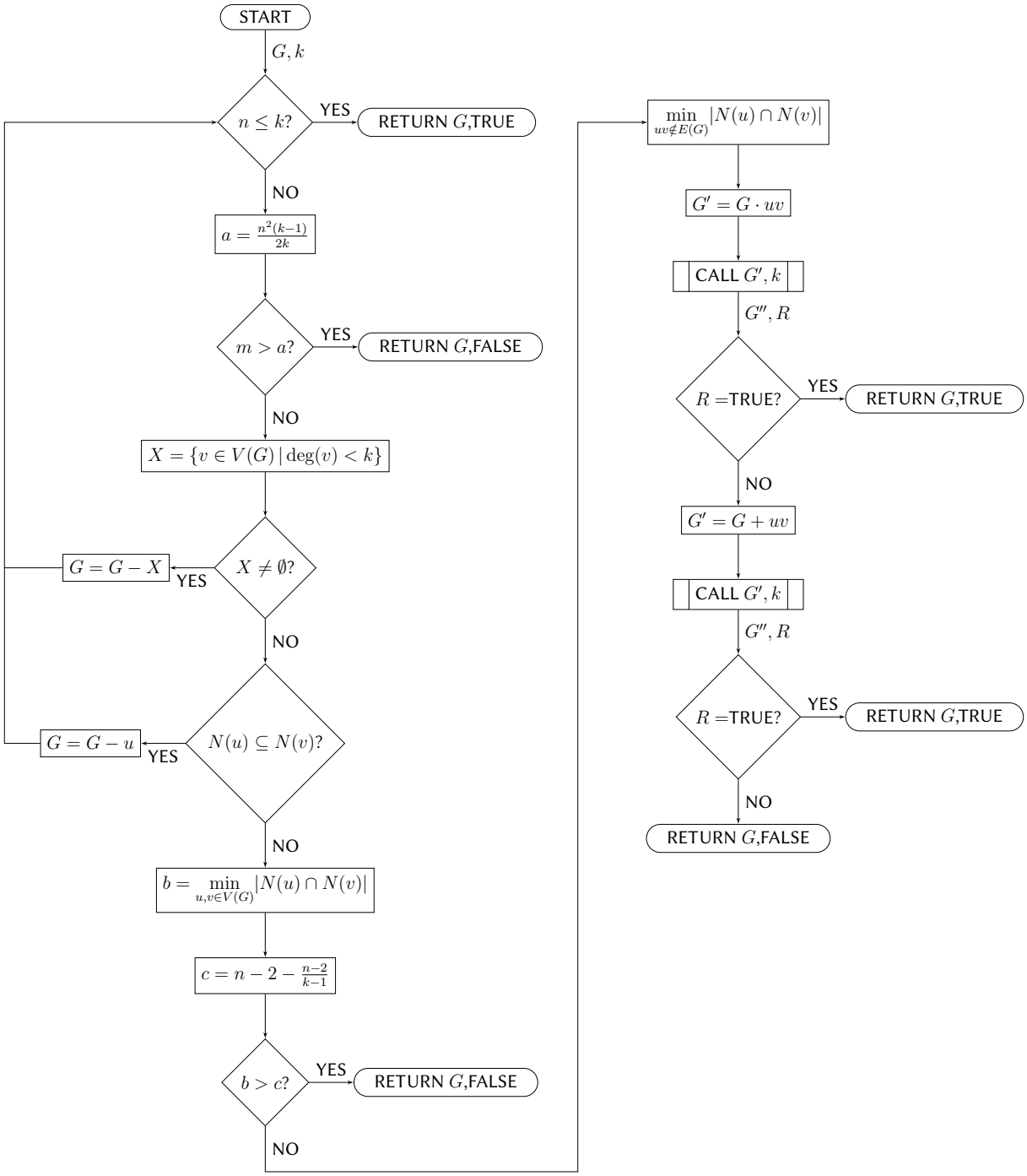


Figure 21: Algorithm Called Subroutine

2.3 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.

2.3.1 Maximum Edge Threshold

The maximum edge threshold test of steps 2 and 3 is supported by Theorem 2.3.1.

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 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 1.7, 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 2.3.1.

Corollary

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

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

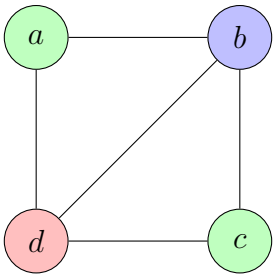
then G is not k-colorable.

Corollary 2.3.1 is demonstrated by Figure 2.3.1. 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 $k = 3$;

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



G

Figure 22: Corollary 2.3.1 Example

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 2.3.1 will always fail for a complete graph when $k < n$. Since $k, n > 0$:

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

2.3.2 Vertex Removal

The theorems that support vertex removal make use of Lemma 2.3.2.

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 G is k-colorable and let $c : V(G) \to C$ be such a coloring. This means that $|C| = k$ and for all $uw \in E(G)$ such that $v \notin uw$ it is the case that $c(u) = c_1$ and $c(w) = c_2$ and $c_1 \neq c_2$ for some $c_1, c_2 \in C$.

Now, consider the restricted function $c' = c|_{V(G-v)}$. It is still the case that $c'(u) = c_1$ and $c'(w) = c_2$ and $c_1 \neq c_2$, so c' is still proper with $|C| = k$.

Therefore $G - v$ is k-colorable. ∎

Lemma 2.3.2 is demonstrated in Figure 2.3.2. No matter which vertex is removed, the resulting subgraph is still properly colored using at most four colors.

Steps 4 and 5 remove vertices with degrees less than k. This is supported by Theorem 2.3.2.

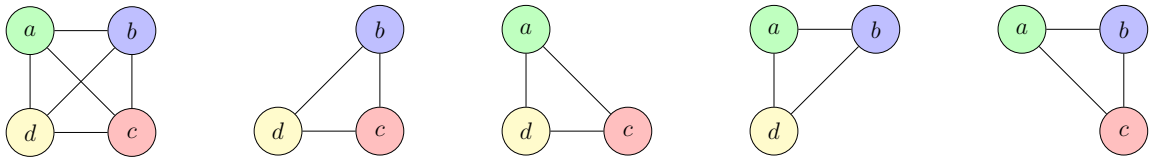


Figure 23: Lemma 2.3.2 Example

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.

Proof. Assume G is k-colorable. Therefore, by Lemma 2.3.2, $G - v$ is also k-colorable.

For the converse, assume that $G - v$ is k-colorable. By assumption, $\deg(v) < k$, meaning v has at most $k - 1$ neighbors, using at most $k - 1$ colors. Thus, there is always an additional color available for v. So extend $G - v$ to G and color v with one of the available $k - \deg(v)$ colors. The result is a proper coloring of G using at most k colors.

Therefore G is k-colorable. ∎

Theorem 2.3.2 is demonstrated in Figure 2.3.2 for $k = 4$ and $\deg(v) = 3$.

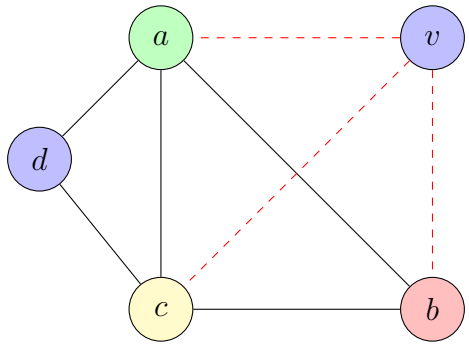


Figure 24: Theorem 2.3.2 Example

The called subroutine actually removes all such vertices at once, which is supported by the inductive proof in Corollary 2.3.2.

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 $|X|$)

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

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

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

Assume 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 2.3.2) 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 2.3.2, 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 2.3.2. 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 2.3.2.

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 G is k-colorable. Therefore, by Lemma 2.3.2, $G - u$ is also k-colorable.

For the converse, assume that $G - v$ is k-colorable. Since $N(u) \subseteq N(v)$ and (by definition) $u \notin N(u)$, it must be the case that $u \notin N(v)$ and hence $uv \notin E(G)$. Thus u and v are allowed to have the same color. Furthermore, since every vertex adjacent to u is also adjacent to v, none of these vertices can have the same color as v. So extend $G - u$ to G and color u with the same color as v. The result is a proper coloring of G using at most k colors.

Therefore G is k-colorable. ∎

Theorem 2.3.2 is demonstrated in Figure 2.3.2. 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.

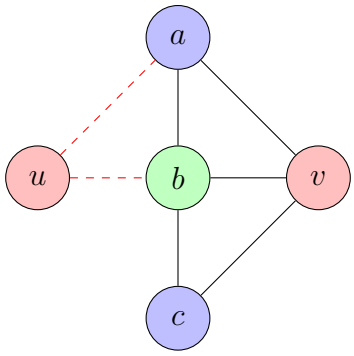


Figure 25: Theorem 2.3.2 Example

2.3.3 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 2.3.3 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 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 $w \in N(u)$. 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 2.3.3 is demonstrated in Figure 2.3.3. 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 $N(v)$.

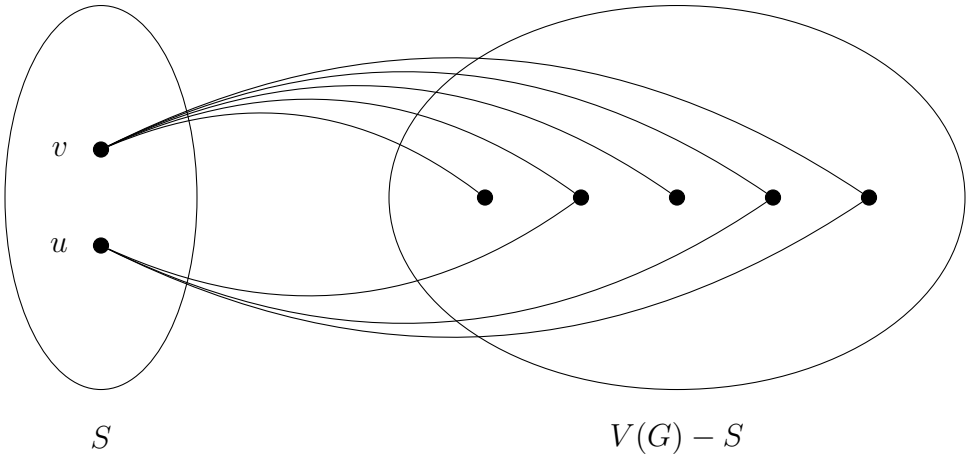


Figure 26: Lemma 2.3.3 Example

Theorem 2.3.3 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 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 $v \in A_1$.

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 2.3.3, $N(u) \subseteq N(v)$, 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 $v' \in A_i$ 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

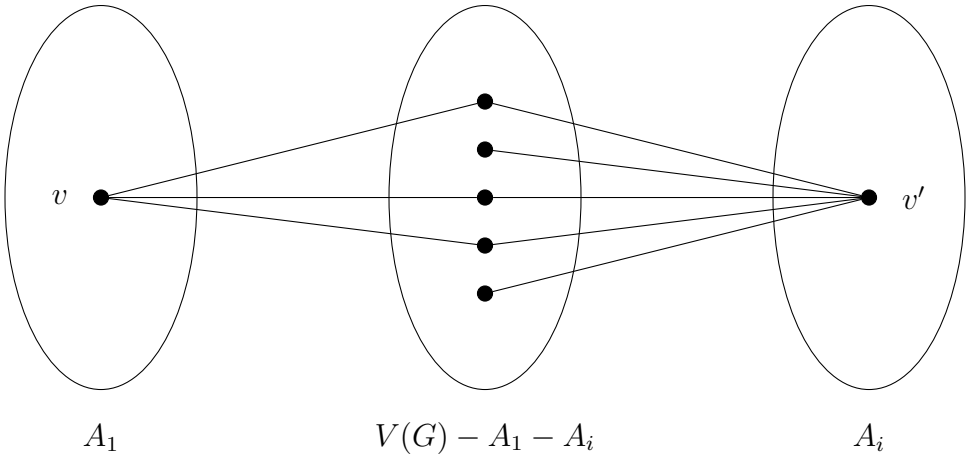


Figure 27: Case $a_i = 1$ Contradiction

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 2.3.3.

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:

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

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 2.3.3.

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 2.3.3 is demonstrated in Figure 2.3.3. The shown graph has $n = 5$, is 3-chromatic, and has:

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

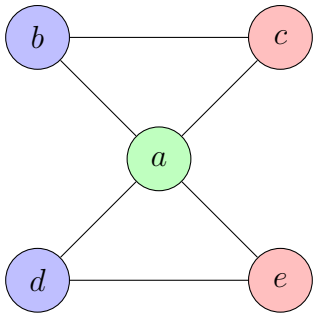
Testing for $k = 2$:

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

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 28: Corollary 2.3.3 Example

2.3.4 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 2.3.4.

Theorem

Let G be a graph of order $n >= 2$ 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 G is k-colorable. There are two possibilities:

Case 1: u and v have the same color.

For all $w \in N(u) \cup N(v)$, it must be the case that w is a different color than the color of u and v. Let v' be the contracted vertex, so that $N(v') = N(u) \cup N(v)$ and color v' with the same color as u and v. The result is a proper k-coloring of $G \cdot uv$.

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

Case 2: u and v have the different colors in c.

By adding edge uv, u and v become adjacent and thus must have different colors. Therefore, u and v can retain their same colors. The result is a proper k-coloring of $G + uv$.

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 v' be the contracted vertex with some assigned color. It must be the case that for all $w \in N(v')$, w has a different color than v'. Expand $G \cdot uv$ to G and color u and v with the same color as v'. The result is a proper k-coloring of G.

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

Remove edge uv. Since u and v are no longer adjacent, there are no requirements on their colors. Thus, they can retain their original colors. The result is a proper k-coloring of G.

Therefore G is k-colorable. ∎

References

[1] Sara Behdad, Praveen Kumare Gopalakrishnan, Sogol Jahanbekam, and Jeffery Cavallaro. 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] Gary Chartrand and Ping Zhang. *A First Course in Graph Theory*. Dover Publications, Mineola, New York, 2012.

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

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