

# Two Graph Vertex Partitioning Algorithms for Part Consolidation in Axiomatic Design

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

August 29, 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 two algorithms. This material is primarily taken from the textbooks used [1] 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  $G = (V, E, \dots)$  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 or more relations. Each edge is represented by a two-element subset of  $V(G)$  called the *endpoints* of the edge:

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

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

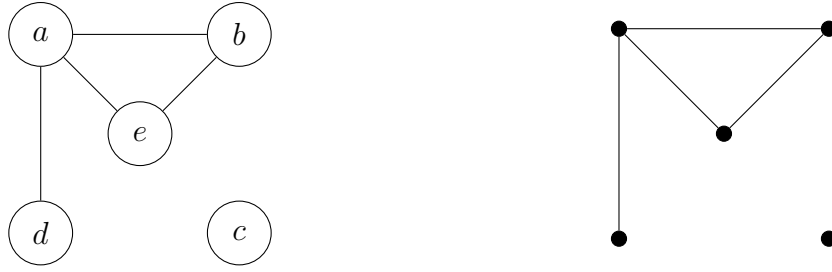
For the remainder of this work, the use of the term “graph” implies a “simple graph.”

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 *FRs* and whose edges discourage combining their endpoint *FRs* into a single part: in the case of the first algorithm, each edge is given a numerical score (weight) indicating the magnitude of the desire to not combine the endpoint *FRs* into a single part, and in the case of the second algorithm, each edge indicates that the 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.



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

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

Figure 1: An Example Graph (labeled and unlabeled)

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

### **Notation: Edge**

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  $E(G)$ :

$$\forall vw \in E(G), u \neq v \text{ and } u \neq w$$

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 let  $e, f \in E(G)$ . 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  $n(G)$ , 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  $m(G)$ , 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 two algorithms:

### **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  $\mathcal{R}$  is assumed and we say something like, “vertex  $v$  has attribute  $a$ ,” instead of the more formal, “vertex  $v$  has attribute  $\mathcal{R}(v)$ .”

The following sections describe the three relations used by the algorithms.

### **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:

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

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 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 Figure 1.1: the graph on the left is labeled and the graph on the right is unlabeled.

Since the labeling function  $\ell$  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  $v \in V(G)$  . . .” 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  $FR$  graphs that act as the inputs to the two algorithms are labeled graphs, where the labels represent the various functional requirements:

$$FR_1, FR_2, FR_3, \dots, FR_n$$

### 1.3.2 Edge Weight

Certain graph theory problems require that each edge be assigned a *weight*. An edge weight suggests a certain cost associated with the use of the edge, usually interpreted as traversing the edge from one endpoint to the other. This requires the addition of a surjective weight function to the graph tuple:

$$w : E(G) \rightarrow W$$

where  $W$  is a set of edge weights, usually integers or rational numbers. A special edge weight of  $\infty$  is used to indicate that an edge should never be used/traversed. Non-adjacent vertices can be viewed as being connected by edges with either zero or infinite weight, depending on the problem. In a drawn graph, edges are typically labeled with their weights, as shown in the example graph of Figure 1.3.2.

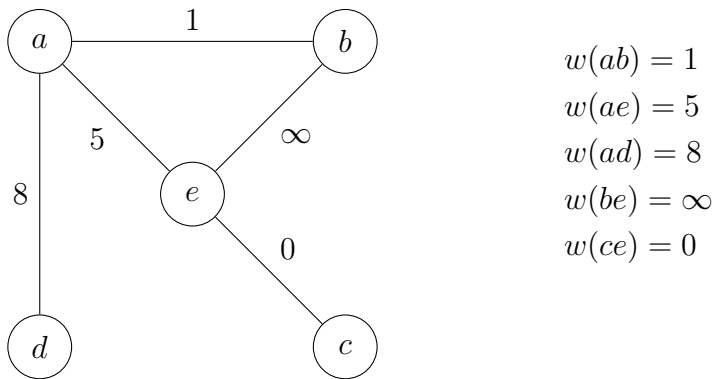


Figure 2: A Graph with Edge Weights

The first algorithm uses edge weights to represent the cost or penalty for consolidating an edge’s two  $FR$  endpoints into the same part. It is assumed that non-adjacent parts cannot be consolidated, and thus are considered to be joined by edges with infinite weight.

### 1.3.3 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:

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

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 1.3.3.

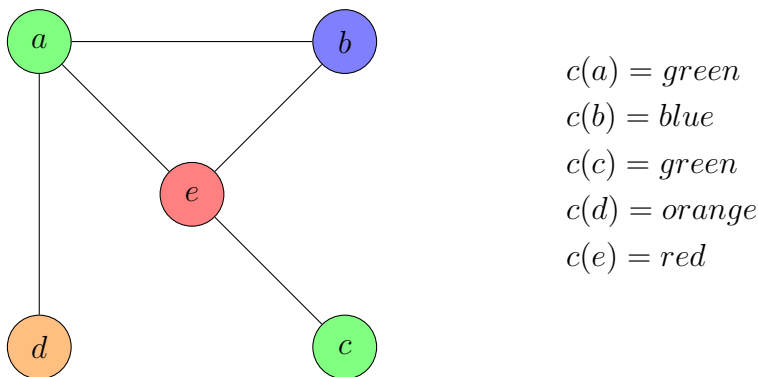


Figure 3: A Graph with a 4-coloring

Since there is no requirement that a coloring  $c$  be surjective, the codomain  $C$  may contain unused colors. For example, the codomain of the coloring shown in Figure 1.3.3 might be:

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

and hence  $c$  is surjective and  $G$  is 4-colorable. But we can always 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 following statement:

### **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 following conclusion:

### **Proposition 2**

Let  $G$  be a graph:

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

#### **1.3.4 Chromatic Coloring**

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 1.3.3, 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 1.3.4. 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.

#### **1.3.5 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:

#### **Definition: Independent Set**



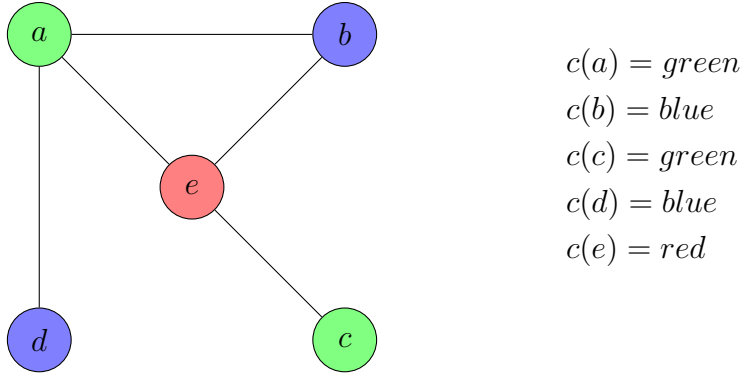


Figure 4: A Graph with a Chromatic 3-coloring

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 second algorithm is to find a chromatic coloring of an  $FR$  graph so that the resulting independent sets indicate how to consolidate the  $FR$ s into a minimum number of parts: one part per independent set.

## 1.4 Subgraphs

The basic strategy of the algorithms is to arrive at a solution by mutating an input graph into simpler graphs such that a solution is more easily determined. The algorithms make use of four particular mutators: vertex deletion, edge 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  $be$ .

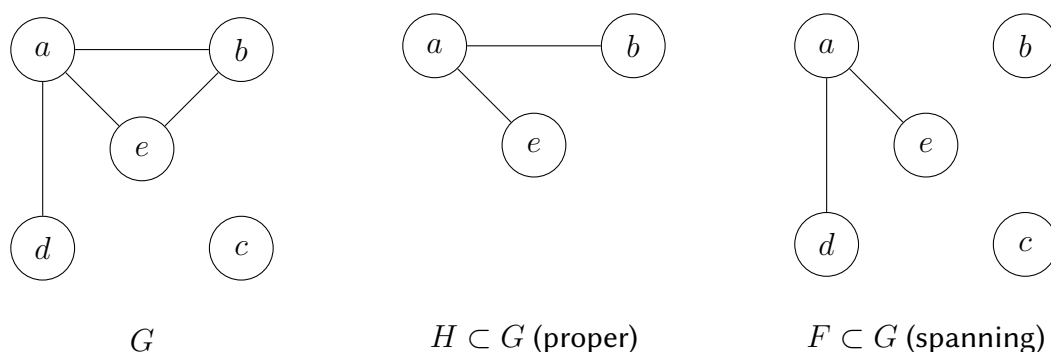


Figure 5: Subgraph Examples

### 1.4.3 Induced Subgraphs

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

#### Definition: Induced Subgraph

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

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

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

Note that when a vertex is removed to make an induced subgraph then all of that vertex's incident edges are also removed. However, for every pair of vertices included in an induced subgraph, if the vertices are the endpoints of a particular edge then that edge must also be included in the subgraph. In the examples of Figure 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  $G$ .

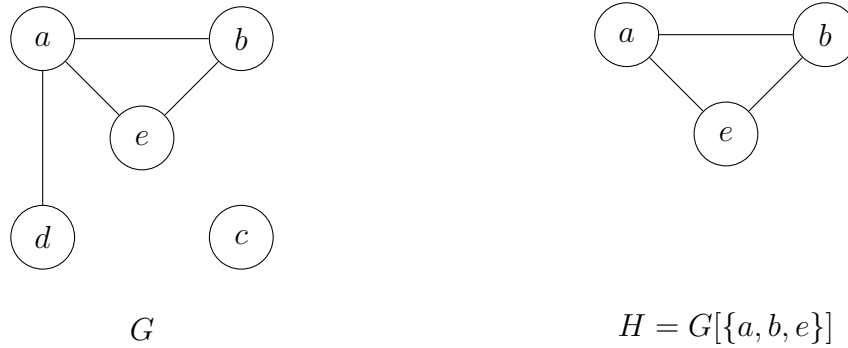


Figure 6: Induced Subgraph Example

## 1.5 Mutators

The following sections describe the graph mutators used by the algorithms.

### 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  $be$ .

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

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

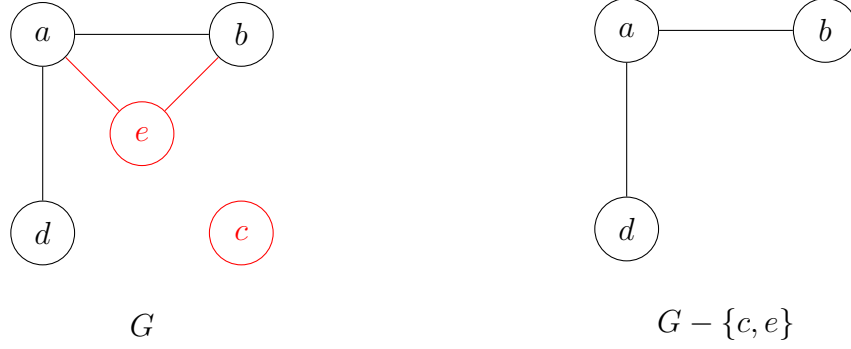


Figure 7: Vertex Removal Example

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

### 1.5.2 Edge Removal

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 (no edges).

Figure 1.5.2 shows an example of edge removal: edges  $ae$  and  $be$  are removed.

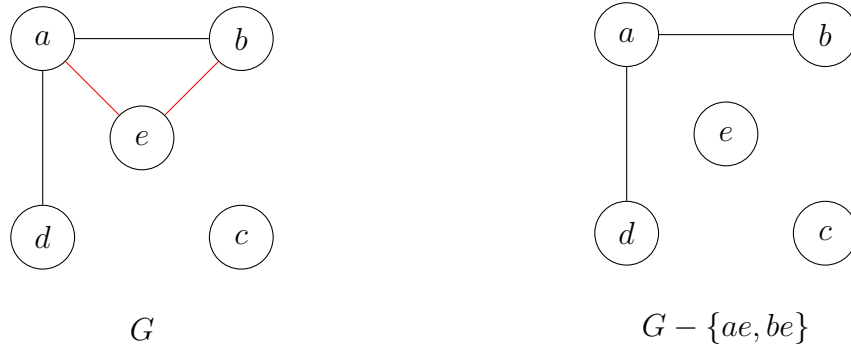


Figure 8: Edge Removal Example

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

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

The first algorithm uses edge removal to prevent two adjacent  $FRs$  from being consolidated into the same part.

### 1.5.3 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 1.5.3 shows an example of edge addition: edge  $cd$  is added.

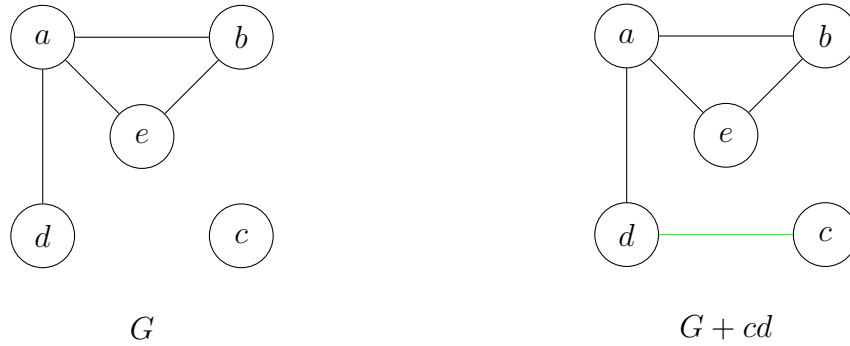


Figure 9: Edge Addition Example

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

### 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 (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 suppression 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  $e$ , so 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*. The first algorithm uses edge contraction to consolidate two adjacent  $FRs$  into the same part, where the contracted edge weight is interpreted as a penalty for doing so. Alternatively, if  $uv \notin E(G)$  then the operation is also referred to as *vertex identification*. The second algorithm uses vertex identification to consolidate two non-adjacent  $FRs$  into the same part.

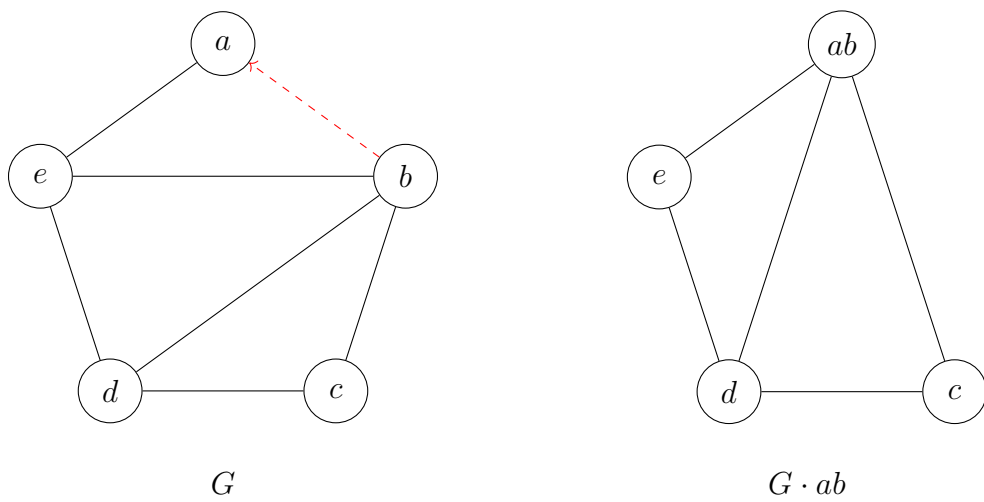


Figure 10: Vertex Contraction Example

## References

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