Graph Coloring using Integer Programming

Derryl Sayo

```
import numpy as np
import cvxpy as cp
import networkx as nx
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Graph Coloring - Problem Statement

Graph coloring is the classifcation of problems where we assign color labels to elements of a graph under certain constraints (Wikipedia contributors 2024a).

A common variation is **vertex coloring** in which we assign colors to each vertex in the graph with no adjacent vertices being the same color.

Some Graph Terminology

- Let V be the set of vertices/nodes.
- Let $E \subset \{(x, y) \mid x, y \in V, x \neq y\}$ be the set of edges. An edge exists between two distinct vertices and the set of edges can include any number of the possible pairs of vertices. We can represent these as tuples of vertices $(x, y) \in E$.
- Then, let $G = (V, E)$ be a graph. A graph can be represented (ie. drawn) using only the set of edges/vertices.
- $|A|$ is the cardinality of set A which is the number of elements in the set.

Representation as an Integer Programming Problem

We formulate the vertex coloring problem with the following problem statement

> How can we color all vertices of a given graph using the minimum number of colors such that no two adjacent vertices are the same color?

Decision Variables

Consider a set of i graph nodes/vertices and a set of j colors.

Similar to other assignment problems using binary integer programming, let

$$x_{ij} = \begin{cases} 1 & \text{vertex } i \text{ is assigned to color } j \\ 0 & \text{otherwise} \end{cases}$$

However, this alone is not enough to represent all choices made in the optimization problem statement. We introduce a second variable

$$w_j = \begin{cases} 1 & \text{color } j \text{ is used in the graph coloring} \\ 0 & \text{otherwise} \end{cases}$$

to keep track of how many colors are used in the solution.

On the selection of the upper bound of j:

The allowed indices that the vertices can take is $i = 0, \ldots, |V| - 1$ but how large should we make j? If we want to see if a certain coloring is feasible, we could limit j to some target value but for the purposes of this demonstration and a reasonable limit is to set let $j = 0, \ldots, |V| - 1$ as well. In short, we are stating that the worst-case that the algorithm will be able to spit out is if each vertex is colored a separate color. This is a reasonable assumption when you consider graph coloring solutions to the assignment problem (eg. exam assignment); the easiest way to make all exams have no conflict with each other is to simply schedule each on seaparate days.

Objective Function

Our goal is to minimize the number of colors used in the graph coloring:

$$\min \sum_j w_j$$

Since $w_j = 1$ if and only if color j is used, this will find the minimum possible number of colors to satisfy all constraints.

Constraints

For the vertex coloring problem, we want to satisfy the following constraints:

- Every vertex has exactly 1 color assigned:

$$\sum_j x_{ij} = 1,\ i = 0, \dots, |V| - 1$$

- For every set of edges, **at most one of the vertices has the color** j. We look at each vertex in an edge and make sure that for each j, the count is ≤ 1:

$$x_{uj} + x_{vj} \leq 1,\ \forall (u, v) \in E,\ j = 0, \dots, |V| - 1$$

- We need a way to increment the w_j count when a vertex is colored with color j (ie. $x_{ij} = 1$). The intuition is as follows. Let's say that we have assigned vertex x_i with a color j already. Then by definition, $x_{ij} = 1$. However, since color j is used then we would need to set $w_j = 1$ as well. So, for every node and "color counter" w:

$$x_{ij} \leq w_j,\ i = 0, \dots, |V| - 1,\ j = 0, \dots, |V| - 1$$

- Decision variables cannot be negative:

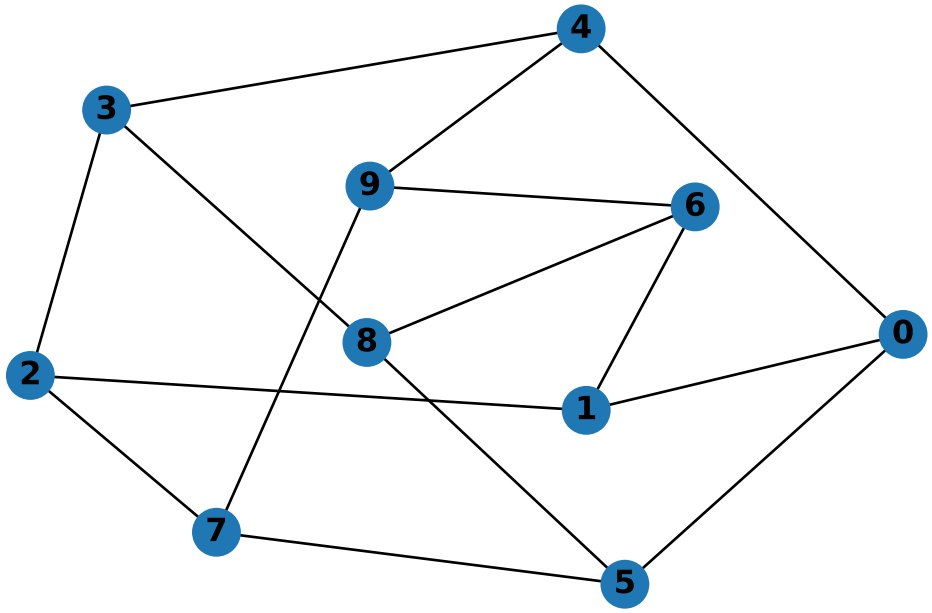
$$x_{ij} \geq 0,\ w_j \geq 0,\ \forall i, j$$

Solutions using Python

Let's implement these constraints using Python. We will use NetworkX (Hagberg, Swart, and Schult 2008) to represent and visualize graphs and CVXPY (Diamond and Boyd 2016).

First, let's create a simple graph. NetworkX has a built-in function to create a simple Petersen graph (see Wikipedia contributors 2024b) which will serve as a good starting point.

```
G = nx.petersen_graph()
nx.draw(G, with_labels=True, font_weight='bold')
```

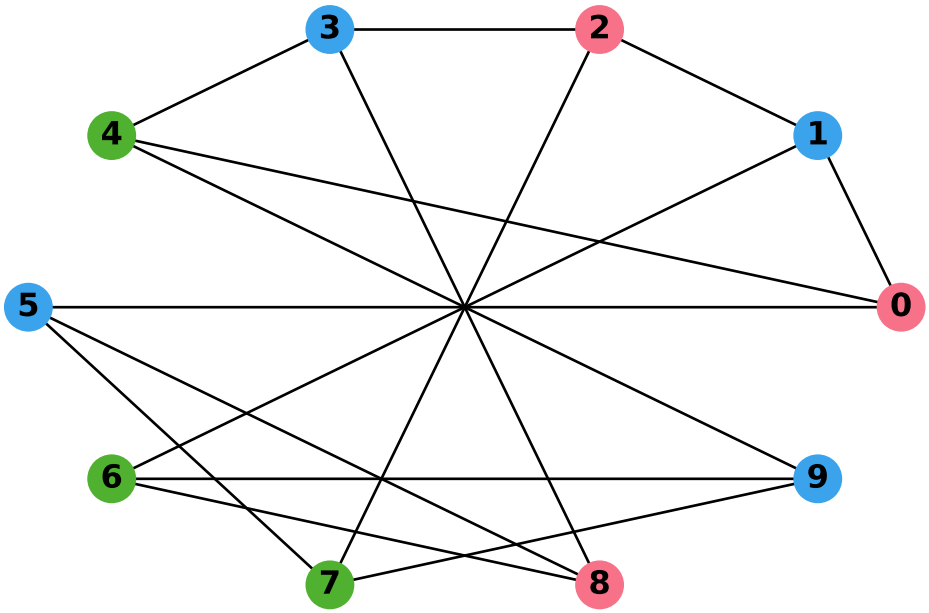


Now we can implement the problem defined above using CVXPY:

```
def draw_graph(G, color_assignment):
  """
  Draw the graph coloring for input graph G
  """
  # Filter out all columns of all-zeros
  color_assignment = color_assignment[:, color_assignment.any(0)]

  # Get the number of colors
  num_colors = np.shape(color_assignment)[1]
```

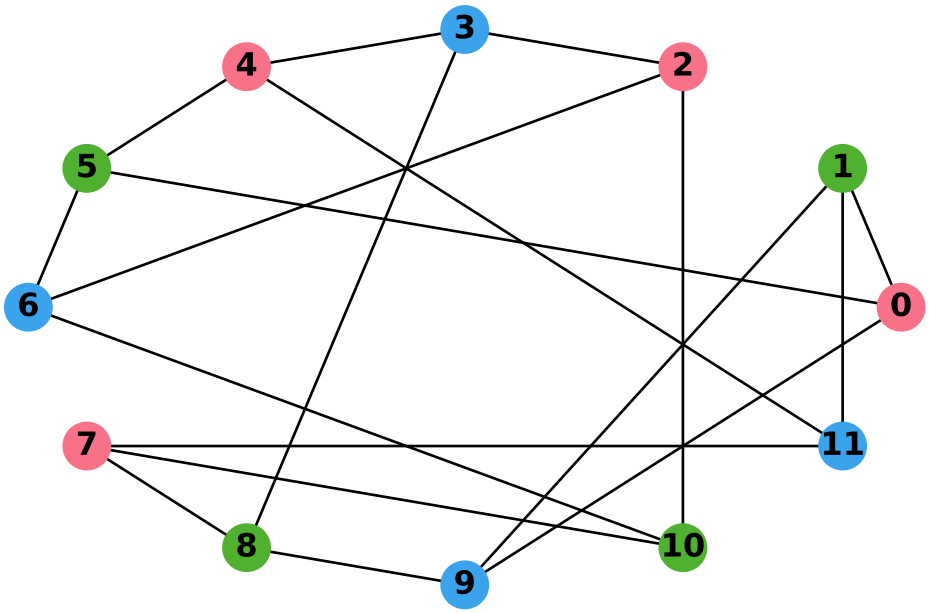
```
  # Generate a color palette using seaborn
  palette = sns.color_palette("husl", num_colors)
  color_map = []

  # `node` is the index of the node by default
  for node in G:
    color_idx = np.argmax(color_assignment[node,:])
    color_map.append(palette[color_idx])


  nx.draw_circular(G, node_color=color_map, with_labels=True, font_weight='bold')

def graph_coloring(G):
  """
  Solve the vertex coloring graph for an input graph G.

  Params:
    - G: NetworkX graph instance

  Returns:
    -
  """
  # Extract graph data from G
  vertices = G.nodes
  edges = G.edges
  num_vertices = len(vertices)
  num_colors = len(vertices)

  # Define decision variables
  X = cp.Variable((num_vertices, num_colors), integer=True)
  W = cp.Variable(num_colors, integer=True)

  # Define objective function
  objective = cp.Minimize(cp.sum(W))

  # Define constraints
  constraints = []

  # Non-negativity of decision variables
  constraints = constraints + [X >= 0]
  constraints = constraints + [W >= 0]
```

```
  # Every vertex must be colored
  constraints = constraints + [cp.sum(X[i,:]) == 1 for i in range(num_vertices)]

  # For every edge, at most 1 vertex has color j
  for edge in edges:
    vertex_1 = edge[0]
    vertex_2 = edge[1]

    for j in range(num_colors):
      constraints = constraints + [X[vertex_1, j] + X[vertex_2, j] <= 1]

  # If a color is taken, increment w_j
  constraints = constraints + [X[i,j] <= W[j] for i in range(num_vertices) for j in range(nu

  problem = cp.Problem(objective, constraints)
  problem.solve()

  # color_assignment = X.value

  return X.value

color_assignment = graph_coloring(G)
draw_graph(G, color_assignment)
```

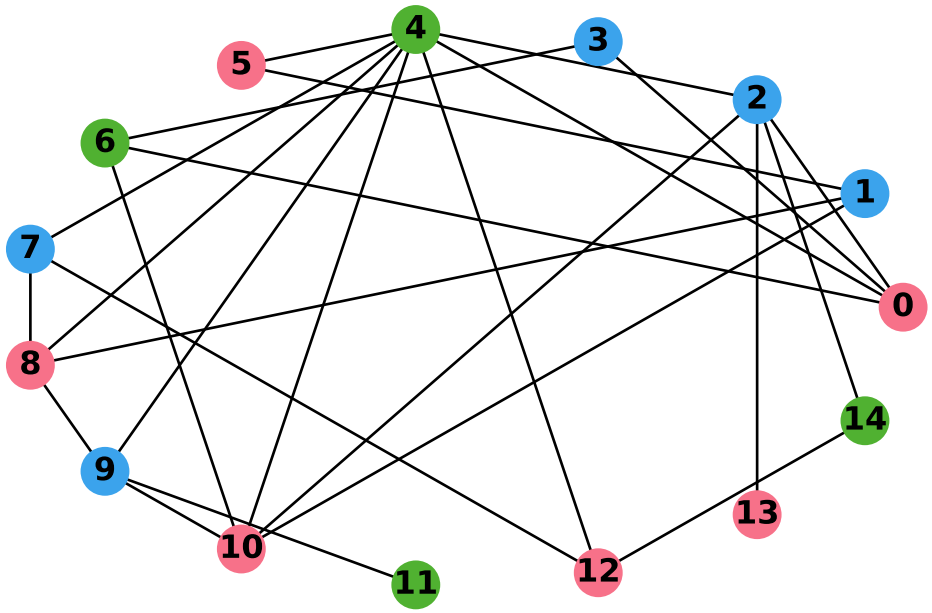


We can try it on some other graphs too:

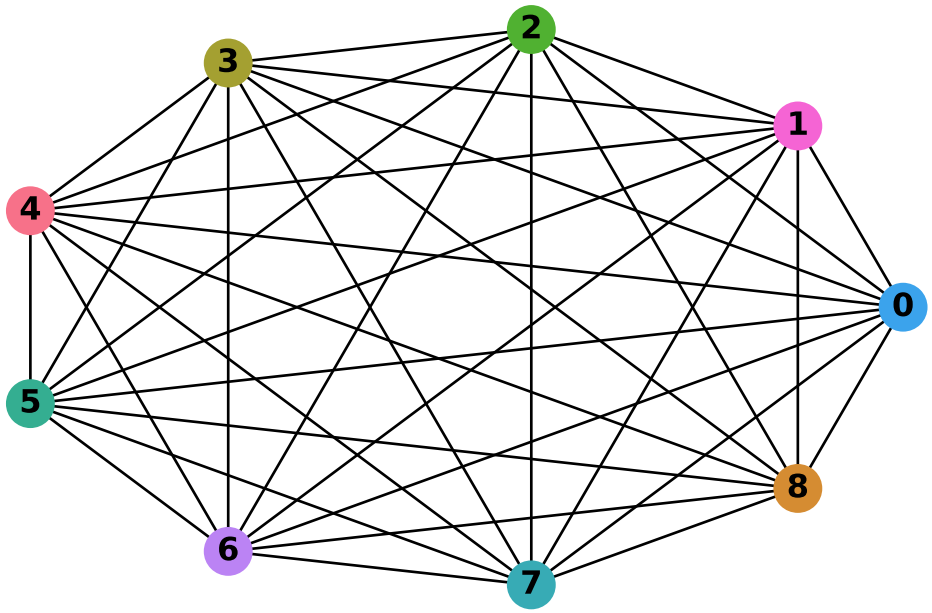
```
G2 = nx.random_regular_graph(3, 12)
G2 = nx.convert_node_labels_to_integers(G2, first_label=0)

color_assignment = graph_coloring(G2)
draw_graph(G2, color_assignment)
```




```
G3 = nx.gnm_random_graph(15, 25)
G3 = nx.convert_node_labels_to_integers(G3, first_label=0)

color_assignment = graph_coloring(G3)
draw_graph(G3, color_assignment)
```




```
G4 = nx.complete_graph(9)
G4 = nx.convert_node_labels_to_integers(G4, first_label=0)

color_assignment = graph_coloring(G4)
draw_graph(G4, color_assignment)
```



Applications

Vertex graph coloring provides a convenient way to represent and solve many problems involving the assignment of resources under conflict constraints.

As a result, this formulation of the graph coloring problem can be used for various resource scheduling/assignment problems in real life such as radio frequency allocation, job allocation, and team building problems (Thadani, Bagora, and Sharma 2022).

Optimizing Cooking Steps

I love cooking and I was inspired by Alex's video on how fast you can feasibly make eggs benedict. Although the challenge is long over by now, such a scenario can be represented and "solved" using a graph coloring approach.

Let G be our graph and let each vertex $v \in V$ represent some task required in cooking. Then, let each edge $(x, y) \in E$ represent some scheduling conflict between tasks x and y. This can be because they require the same cooking utensil, they are blocked until the completion of the other is complete, etc.

We will apply these methods to a steak frites dinner with a bruschetta appetizer below.

Bruschetta

0. Chop tomatoes
1. Chop basil
2. Mix tomatoes, basil, olive oil, garlic, balsamic vinegar
3. Slice bread
4. Top bread with bruschetta and serve

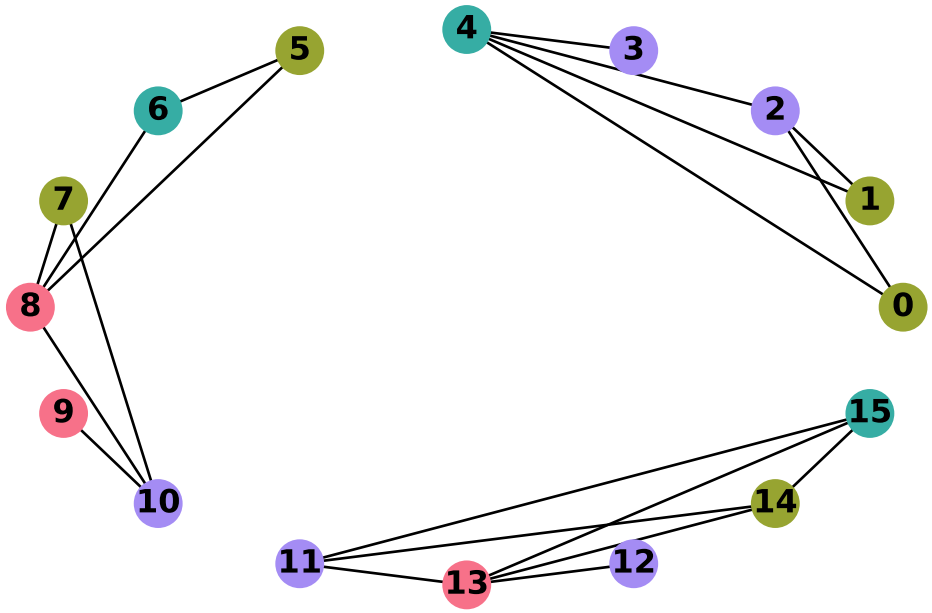
Grilled Ribeyes and Corn

5. Salt and dry brine steak
6. Prepare steak with fresh pepper and pat dry
7. Preheat grill
8. Grill steak
9. Prepare corn
10. Grill corn

Baked Fries

11. Cut potatoes into thick cut fries
12. Boil water with baking soda and salt
13. Boil fries until tender
14. Let fries cool and toss in spices and olive oil
15. Bake in oven, turning occasionally until crispy

```
# Graph representation of tasks list
G = nx.Graph()
G.add_nodes_from(range(11))
G.add_edges_from([
    # Bruschetta
    (0, 2), (1, 2), (0, 4), (1, 4), (2, 4), (3, 4),

    # Steak and corn
    (5, 6), (5, 8), (6, 8), (7, 8), (7, 10), (8, 10), (9, 10),

    # Baked fries
    (11, 13), (12, 13), (11, 14), (13, 14), (11, 15), (13, 15), (14, 15)
])

color_assignment = graph_coloring(G)
draw_graph(G, color_assignment)
```



The graphical interpretation can be done as follows. Consider each subgraph separately and read the steps in order. All steps with the same color can be done at the same time before moving onto the next color/steps.

For example, we see that we should chop the tomatoes and basil at the same time then mix the bruschett and slice the bread together before serving. Then, we should dry brine the steak and preheat the grill at the same time, then prepare the steak, then grill and prepare corn, then grill the corn. Finally, we should cut the potatoes and start boiling the water before par boiling the potatoes, seasoning, and then baking.

Conclusion

Graph coloring can be used to represent a variety of assignment/scheduling problems with a simple integer programming formulation. As a result, many problems involving some assignment under conflicting relationships can be solved for the minimum non-conflicting solution by a very intuitive visualization using graph networks.

References

Diamond, Steven, and Stephen Boyd. 2016. "CVXPY: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research* 17 (83): 1–5.

Hagberg, Aric, Pieter J. Swart, and Daniel A. Schult. 2008. "Exploring Network Structure, Dynamics, and Function Using NetworkX," January. https://www.osti.gov/biblio/960616.

Thadani, Satish, Seema Bagora, and Anand Sharma. 2022. "Applications of Graph Coloring in Various Fields." *Materials Today: Proceedings* 66: 3498–3501. https://doi.org/https://doi.org/10.1016/j.matpr.2022.06.392.

Wikipedia contributors. 2024a. "Graph Coloring — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=Graph_coloring&oldid=1195834521.

———. 2024b. "Petersen Graph — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=Petersen_graph&oldid=1209188117.