Problema de la Coloración Mínima

Diego Castillo Reyes[a], **Marthon Leobardo Yañez Martinez**[a], **Aldo Escamilla Resendiz**[a] **y Muñoz González Eduardo**[a]

[a]*Investigadores en formación, ESCOM, IPN*

Dra. Miriam Pescador Rojas

Resumen—En este trabajo se presenta el problema de la coloración mínima, el cual es un problema NP-completo. Se propone una solución basada en algoritmos genéticos para encontrar la coloración mínima de un grafo.

Keywords—*Coloración Mínima, Genéticos, Algoritmos, Grafos*

1. Introducción

El problema de la coloración mínima es un problema NP-completo, el cual consiste en asignar un color a cada vértice de un grafo de tal manera que dos vértices adyacentes no tengan el mismo color. El objetivo es encontrar la coloración mínima, es decir, la menor cantidad de colores posibles para colorear el grafo. Este problema es de gran importancia en la teoría de grafos, ya que tiene aplicaciones en la asignación de horarios, asignación de frecuencias, asignación de canales, entre otros. En este trabajo se propone una solución basada en PSO, un algoritmo de optimización basado en la inteligencia de enjambre, para encontrar la coloración mínima de un grafo.

2. Antecedentes

En una pequeña ciudad de Rusia, Königsberg (Actualmente Kaliningrado, Rusia), existen siete puentes que conectan cuatro islas. Un matemático llamado Leonhard Euler (1707-1783) se preguntó si era posible recorrer todos los puentes una sola vez y regresar al punto de partida. Euler demostró en 1736 que no era posible en su publicación *Solutio problematis ad geometriam situs pertinentis* [1], y para ello utilizó un grafo para representar las islas y los puentes como se ve en la Figura 1.

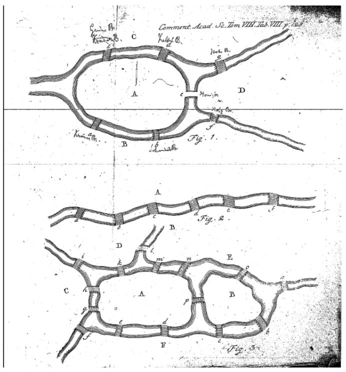

Figura 1. Grafo de Königsberg

Él demostró utilizando cada puente como una arista y cada isla como un vértice, que no era posible recorrer todos los puentes una sola vez y regresar al punto de partida. Siendo este el primer problema de teoría de grafos, el cual es un área de las matemáticas que estudia las relaciones entre los vértices y las aristas de un grafo. Un grafo es un conjunto de vértices y aristas, donde las aristas son pares no ordenados de vértices. Posteriormente, en 1852, Francis Guthrie planteó el problema de los cuatro colores, el cual consiste en colorear un mapa de tal manera que dos regiones adyacentes no tengan el mismo color. Este problema fue resuelto en 1976 por Kenneth Appel y Wolfgang Haken utilizando una computadora, demostrando que cuatro colores son suficientes para colorear cualquier mapa [2]. A partir de este problema, se han planteado diversos problemas de coloración, entre ellos el problema de la coloración mínima, el cual es un problema NP-completo.

3. Metodología

Para poder resolver el problema de la coloración mínima, se propone una solución basada en algoritmos genéticos. El algoritmo genético elegido fue el de PSO (Particle Swarm Optimization), el cual es un algoritmo de optimización basado en la inteligencia de enjambre. El algoritmo PSO se basa en el comportamiento social de las partículas, las cuales se mueven en un espacio de búsqueda en busca de la mejor solución. Las partículas tienen mejoras cognitivas y mejoras sociales, las cuales les permiten moverse en el espacio de búsqueda. En el caso del problema de la coloración mínima, las partículas representan una coloración de un grafo, y el objetivo es encontrar la coloración mínima. La representación de las partículas es un vector de enteros que representa los colores de los vértices del grafo. El algoritmo PSO se encarga de mover las partículas en el espacio de búsqueda, de tal manera que se encuentre la coloración mínima del grafo. Para evaluar la calidad de una coloración, se utiliza la función objetivo, la cual es el número de colores utilizados en la coloración.

4. Propuesta de solución

```python
import numpy as np
import networkx as nx
import random
import matplotlib.pyplot as plt

# Funcion de fitness
def fitness(solution, graph):
    conflicts = 0
    for edge in graph.edges:
        if solution[edge[0]] == solution[edge[1]]:
            conflicts += 1
    num_colors = len(set(solution))
    return num_colors + conflicts * 1000

# PSO parameters
num_particles = 30
num_vertices = 10 # Numero de nodos
max_iter = 100

# Probabilidad de conexion
p = 0.3

# Crear el grafo
graph = nx.erdos_renyi_graph(num_vertices, p)

# Asegurarse de que el grafo no sea totalmente conexo
while nx.is_connected(graph):
    graph = nx.erdos_renyi_graph(num_vertices, p)

# Dibujar el grafo
plt.figure(figsize=(8, 6))
nx.draw(graph, with_labels=True, node_color='skyblue', node_size=500, edge_color='gray')
plt.show()

# Inicializacion de particulas
particles = [np.random.randint(0, num_vertices, num_vertices) for _ in range(num_particles)]
velocities = [np.random.randint(-1, 2, num_vertices) for _ in range(num_particles)]
pbest_positions = particles.copy()
pbest_scores = [fitness(p, graph) for p in particles]
```

```python
gbest_position = pbest_positions[np.argmin(pbest_scores)]
gbest_score = min(pbest_scores)

# Parametros de PSO
w = 0.5  # Inercia
c1 = 1.0  # Constante cognitiva
c2 = 1.0  # Constante social

# PSO main loop
for iteration in range(max_iter):
    for i in range(num_particles):
        velocities[i] = (w * velocities[i]
                         + c1 * random.random() * (pbest_positions[i] - particles[i])
                         + c2 * random.random() * (gbest_position - particles[i]))

        # Actualiza la posicion de las particulas
        particles[i] = np.clip(particles[i] + velocities[i], 0, num_vertices - 1).astype(int)

        # Evaluar nueva posicion
        current_fitness = fitness(particles[i], graph)

        # Actualizar pbest
        if current_fitness < pbest_scores[i]:
            pbest_positions[i] = particles[i]
            pbest_scores[i] = current_fitness

        # Actualizar gbest
        if current_fitness < gbest_score:
            gbest_position = particles[i]
            gbest_score = current_fitness

    print(f"Iteracion {iteration + 1}/{max_iter}, Mejor Fitness: {gbest_score}")

# Resultado final
print("Mejor solucion encontrada:", gbest_position)
print("Numero de colores utilizados:", len(set(gbest_position)))

# Dibujar el grafo final coloreado
color_map = [f"C{color}" for color in gbest_position]  # Asigna un color a cada nodo segun la solucipn optima
plt.figure(figsize=(8, 6))
nx.draw(graph, with_labels=True, node_color=color_map, node_size=500, edge_color='gray')
plt.show()
```

Código 1. Algoritmo PSO para el problema de la coloración mínima

5. Resultados

6. Conclusiones

Referencias

[1] L. Euler, «Solutio problematis ad geometriam situs pertinentis,» *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, vol. 8, págs. 128-140, 1736.

[2] K. Appel y W. Haken, *Every Planar Map Is Four Colorable* (Contemporary Mathematics). Providence, RI: American Mathematical Society, 1976, vol. 98.