Tecnológico de Monterrey

TC3023

Inteligencia Computacional

Proyecto Final

Autor: *Matrícula:*
Donaldo Alfredo Garrido Islas *A01275416*

Dr. Gabriel González Sahagún
Fecha: 28 de noviembre de 2022

Índice

1. Introducción

Los algoritmos genéticos son un método computacional de selección natural que permite resolver problemas de optimización que pueden ser limitados o no poseer límites [1]. Otra forma de entender los algoritmos genéticos es como un proceso en el que sobrevive el que más se ajusta. Los algoritmos genéticos copian el proceso de evolución, en el que los elementos más fuertes se vuelven más fuertes, mientras que los más débiles se descartan [2].

En este caso, se emplea un algoritmo genético programado a mano para resolver el problema del coloreado de grafos (mejor conocido como *graph coloring*).

2. Descripción del problema y artículos relacionados

En su artículo, Celia Glass presenta al *Graph Coloring* como uno de los problemas de optimización combinatoria más estudiados y nos describe que el problema se trata de encontrar el número mínimo de colores tal que dos nodos adyacentes (conectados) no tengan asignado el mismo color, dado un grafo en específico. Además presenta un algoritmo, en el que no profundizaremos aquí, pero del que se obtienen resultados interesantes llamado *Galinier and Hao's Algorithm* [3].

Por otro lado, Musa Hindi puntualiza que el *Graph Coloring Problem* es un problema del tipo NP y nos presenta un nuevo acercamiento al problema, usando algoritmos genéticos (como en el caso de este trabajo) junto con algunas otras estrategias de inteligencia computacional, destacando el hecho de que usa más de un método de selección y mutación dependiendo del estado en los pesos de la mejor solución. Este enfoque adaptativo ayuda a que se encuentre el mínimo global más rápida y eficazmente que con métodos convencionales [4].

Dado lo anterior, nos planteamos resolver el *Graph Coloring Problem* con las estrategias conocidas para los algoritmos genéticos.

3. Descripción de la representación de los individuos

Lo primero que se representó fueron las conexiones aleatorias entre nodos, por ejemplo, para un grafo con $n = 3$ nodos, se podía crear una matriz de la siguiente forma:

$$\mathcal{N} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Lo que equivale a las siguientes conexiones de nodo:

$$1 \Longleftrightarrow 2, \qquad 2 \Longleftrightarrow 3$$

Nótese que las matrices han de ser simétricas puesto que las conexiones de nodos son adireccionales.

Por otro lado, los individuos dentro del algoritmo como tal, se representan a través de vectores de números que contienen el color de cada uno de los nodos. Por ejemplo, si tenemos $n = 10$ nodos y $c = 5$ colores, el vector sería de la forma:

$$\mathbf{v} = [4, 3, 5, 5, 1, 1, 1, 2, 3, 2]$$

Indicando, por ejemplo que el nodo 3 tiene asignado el color 5.

4. Descripción del Código

El código, que se puede ver en el Apéndice A, comienza definiendo el número de nodos y las conexiones aleatorias entre ellos. Posteriormente, se grafica el grafo acorde a las relaciones antes obtenidas. Para esto estas gráficas se usa la librería *networks* de python, además de *plotly*. Lo necesario se importa con:

```
import networkx as nx
import plotly.graph_objs as go
```

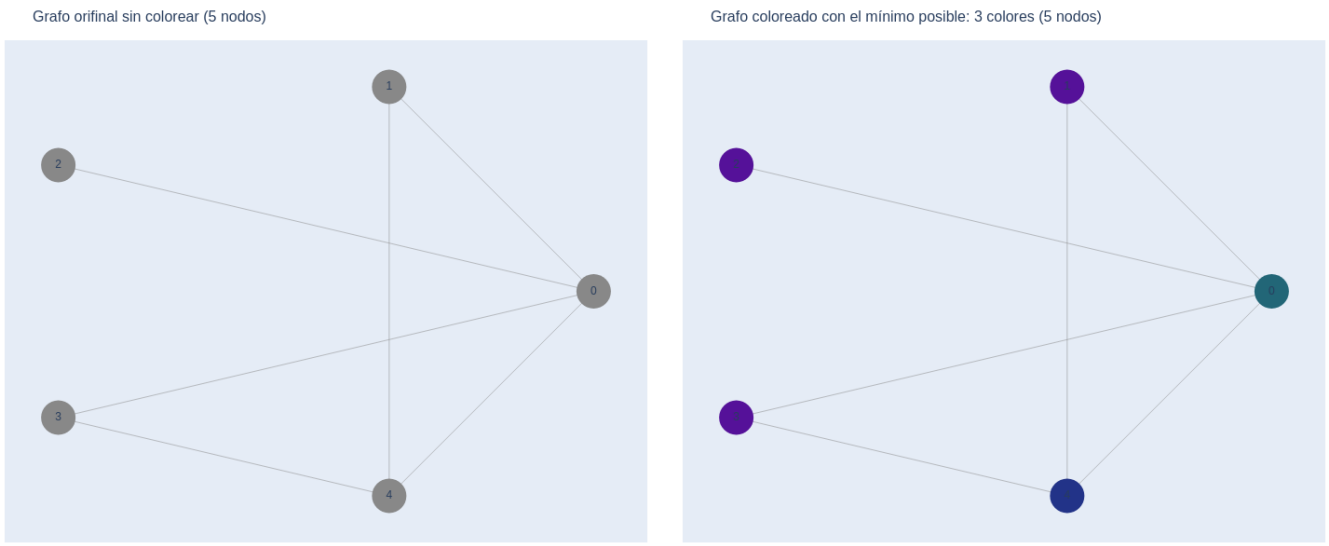
Pasado esto, se definen las funciones que se usarán en el código principal, como la creación de individuos, la función de pesos (fitness), la función de cruce, de mutación y finalmente la función de selección, para la que se escogió una selección de rueda de ruleta.

Ya en el código principal, se crea la población de 200 individuos, y se calculan pesos, posteriormente, se ejecuta el algoritmo en el que se mutan los padres y se calculan las mejores soluciones para el número k de colores dado. Una vez se encuentra una solución con fitness $= 0$ se disminuye el número de colores a $k - 1$, así sucesivamente.

5. Resultados y discusión

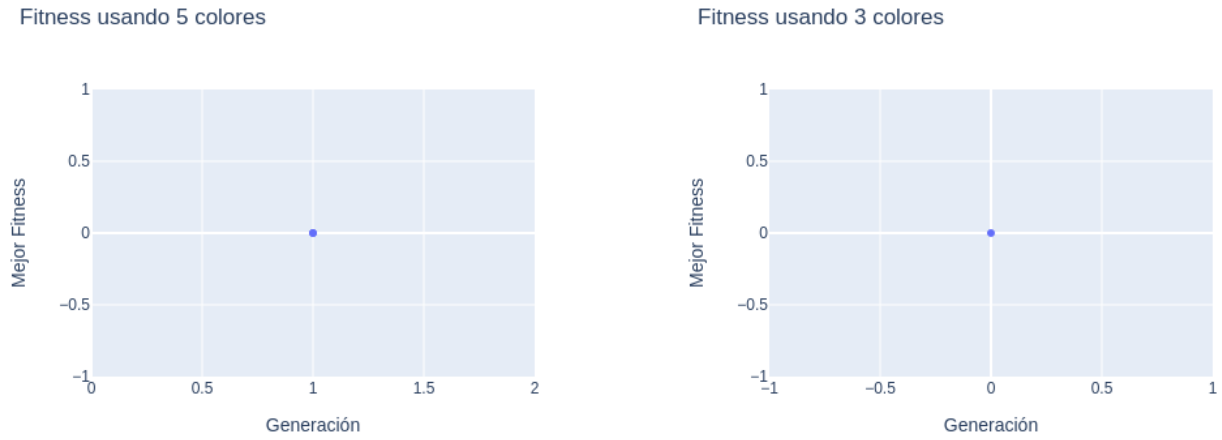
A continuación se presentan los grafos sin colorear y coloreados para $n = 5, 15, 40$ número de nodos. Además de algunas de las funciones de la evolución del fitness, acorde a una disminución en el k usado.

5.1. 5 nodos



(a) (b)

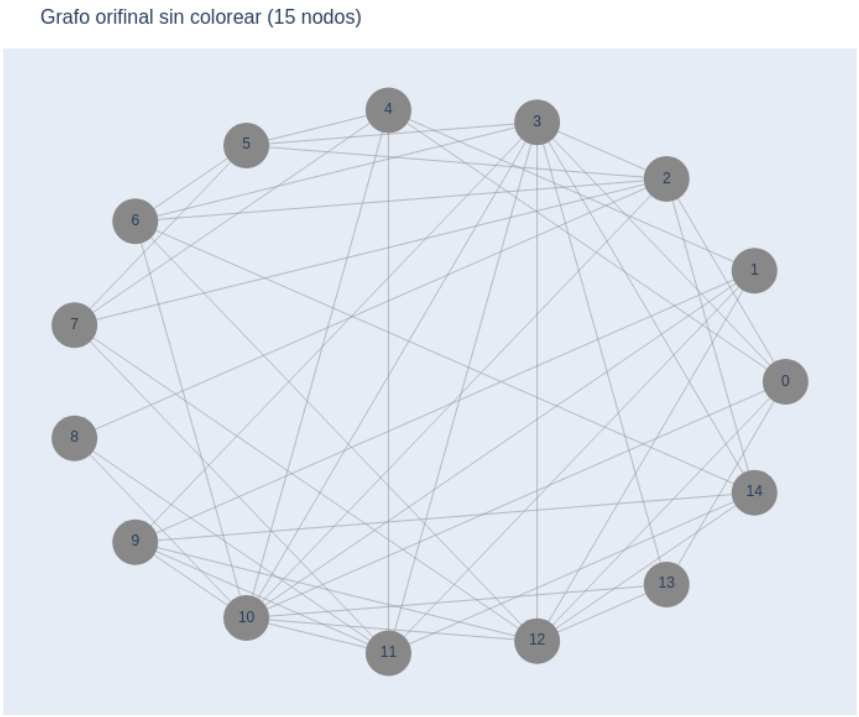
Figura 1: Diagrama de un grafo con 5 nodos y aristas arbitrarias. (a) Antes del coloreado. (b) Después de resolver el problema y el coloreado



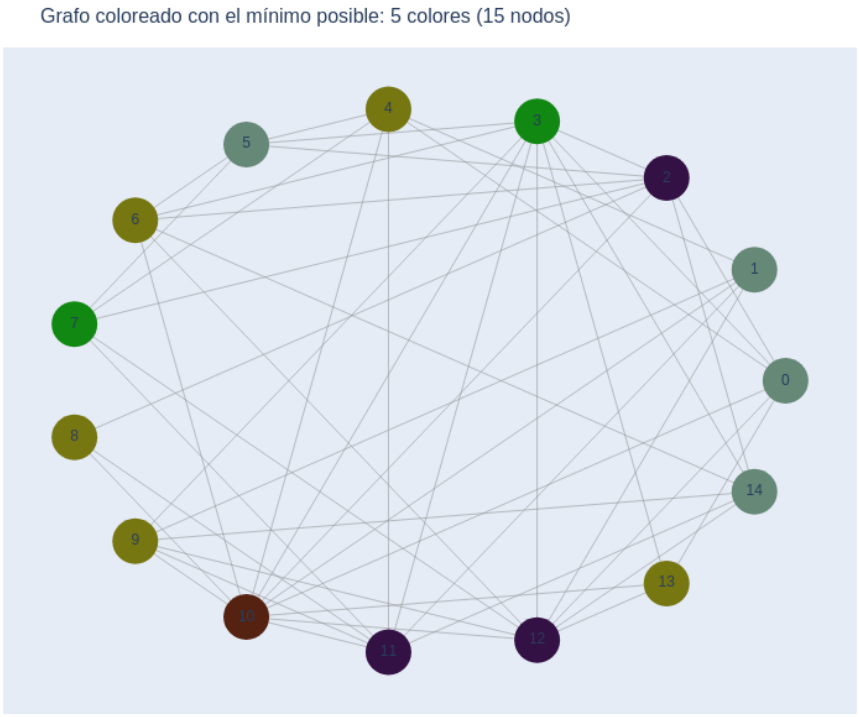
(a) (b)

Figura 2: Gráfica de la evolución del Fitness para 5 nodos con respecto a la generación de hijos para un número de colores dado: (a) 5 colores, (b) 3 colores.

5.2. 15 nodos

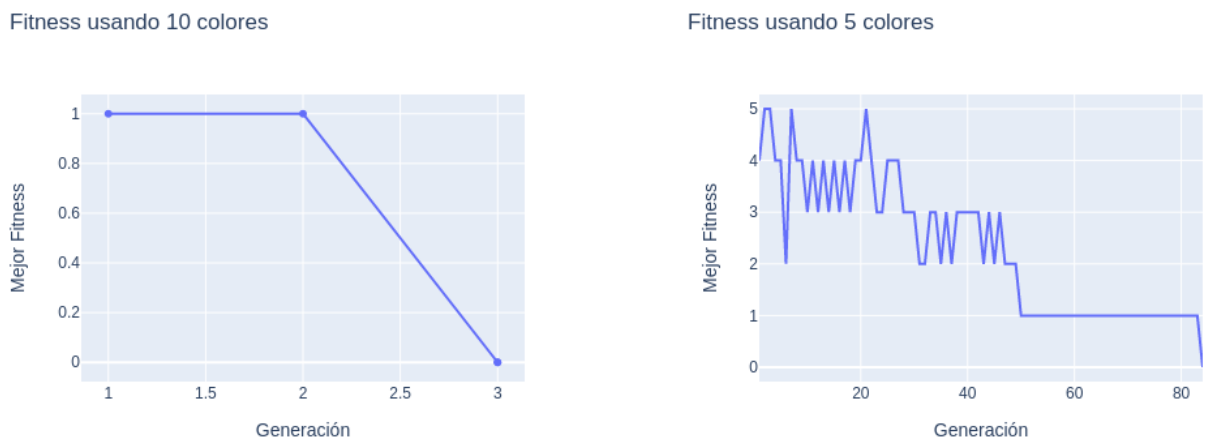


(a)



(b)

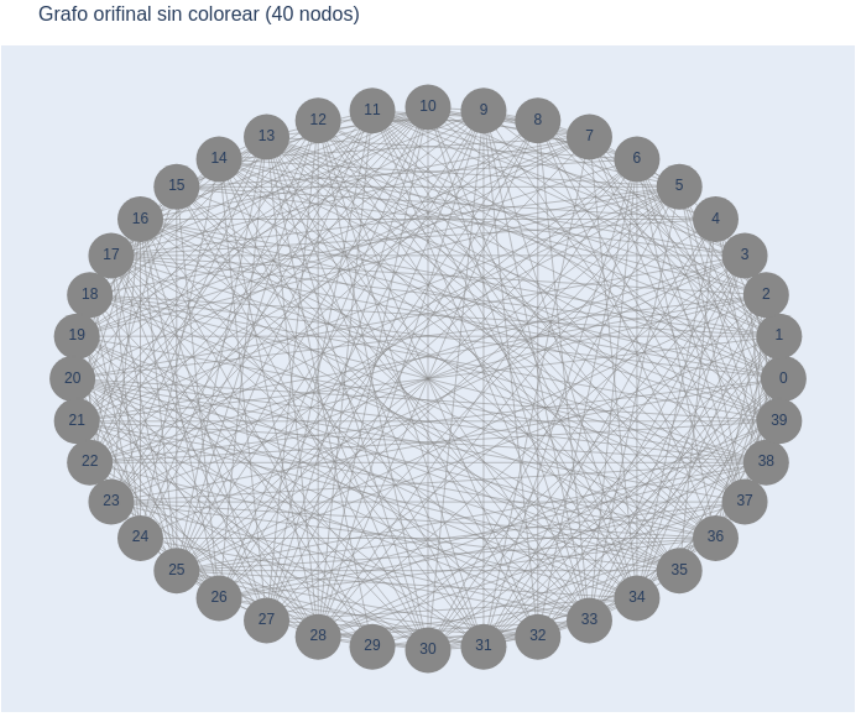
Figura 3: Diagrama de un grafo con 15 nodos y aristas arbitrarias. (a) Antes del coloreado. (b) Después de resolver el problema y el coloreado



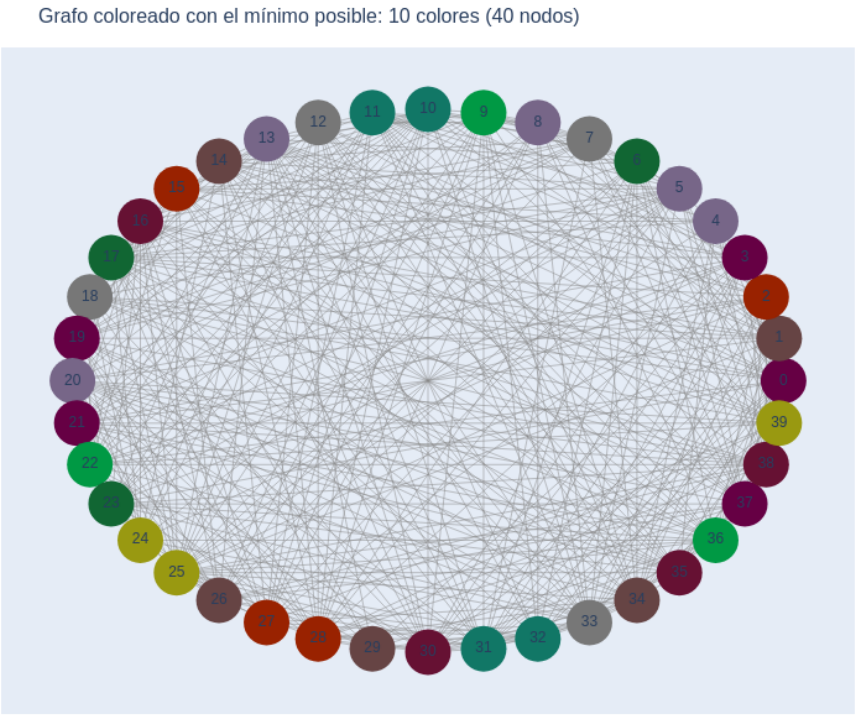
(a) (b)

Figura 4: Gráfica de la evolución del Fitness para 15 nodos con respecto a la generación de hijos para un número de colores dado: (a) 10 colores, (b) 5 colores.

5.3. 40 nodos

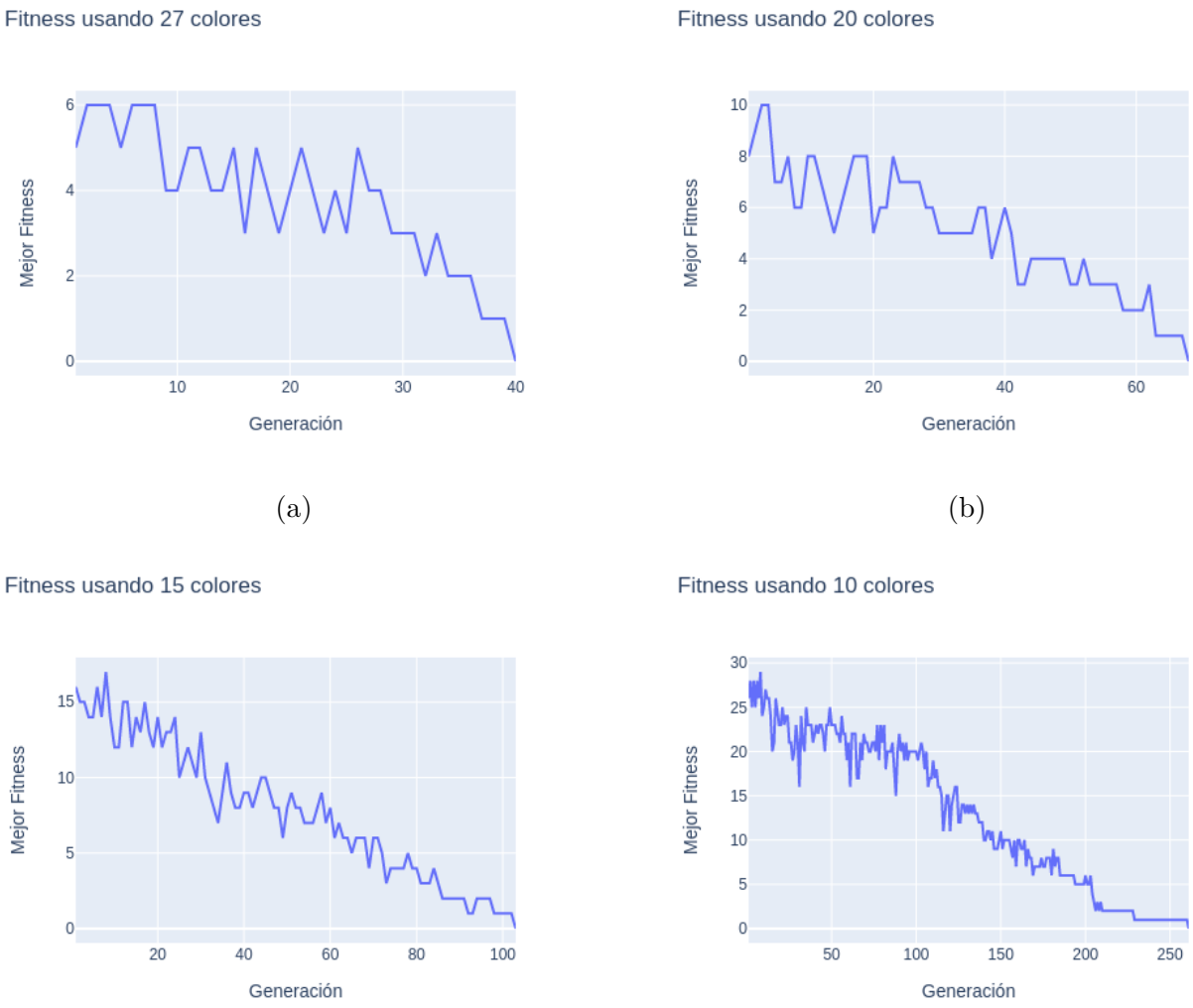


(a)



(b)

Figura 5: Diagrama de un grafo con 40 nodos y aristas arbitrarias. (a) Antes del coloreado. (b) Después de resolver el problema y el coloreado



(a) (b)

(c) (d)

Figura 6: Grafica de la evolución del Fitness para 15 nodos con respecto a la generación de hijos para un número de colores dado: (a) 10 colores, (b) 5 colores.

6. Conclusiones

En este trabajo, se puso en práctica una vez más los algoritmos genéticos para un problema que puede ser muy complejo como lo es el *graph coloring*. Una de las dificultades más importantes que tuve fue la mutación ya que en ocasiones el código se podía ciclar de manera inesperada, por lo que tuve que introducir un criterio de cambio en la probabilidad de mutación. Además, fue complicada estipular la representación de los individuos de manera que fuera práctico su uso en el algoritmo.

Por otro lado, creo que fue interesante crear y escribir el propio código en vez de usar alguna librería de python (como *deap*, por ejemplo), ya que el entendimiento del algoritmo y sus bases quedan mucho más claras, así como las partes que lo constituyen.

En complejidad es un proyecto muy rico, además de visualmente atractivo y su solución puede

ser interesante para introducir a estudiantes en el campo de los algoritmos genéticos.

Referencias

[1] What is the genetic algorithm? URL `https://la.mathworks.com/help/gads/what-is-the-genetic-algorithm.html`.

[2] Murray-Smith, D. J. 6 - experimental modelling: system identification, parameter estimation and model optimisation techniques. In Murray-Smith, D. J. (ed.) *Modelling and Simulation of Integrated Systems in Engineering*, 165–214 (Woodhead Publishing, 2012). URL `https://www.sciencedirect.com/science/article/pii/B9780857090782500063`.

[3] Glass, C. & Prugel-Bennett, A. Genetic algorithm for graph coloring: Exploration of galinier and hao's algorithm. *J. Comb. Optim.* **7**, 229–236 (2003).

[4] Hindi, M. & Yampolskiy, R. V. Genetic algorithm applied to the graph coloring problem. In *Midwest Artificial Intelligence and Cognitive Science Conference* (2012).

7. Apéndice A: Código

El código puede ser visualizado en google collab como jupyter notebook en `https://colab.research.google.com/drive/1vZXMMxt8trACOa3qByT39RUaDkwEuEJg?usp=sharing`.

```
# -*- coding: utf-8 -*-
"""A01275416_PF.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1vZXMMxt8trACOa3qByT39RUaDkwEuEJg

# Graph Coloring Problem - Proyecto Final
## Inteligencia computacional
### A01275416 - Donaldo Alfredo Garrido Islas


El Problema de Coloreado de Gráficos (Graph Coloring Problem) consiste en asignar colores a ciertos elementos de un grafo sujeto a ciertas restricciones. En este código se usa un algoritmo genético para resolver el problema y encontrar el número mínimo de colores requeridos para colorear un grafo.

La coloración de grafos es una asignación de etiquetas, tradicionalmente llamadas "colores", a los vértices de un gráfo sujeto a la condición de que no se asigne la misma etiqueta/color a dos vértices incidentes con un borde. El menor número de colores necesarios para colorear un grafo G se conoce como su número cromático. Una coloración que utiliza como máximo n colores se denomina n-coloración. Un grafo al que se le puede asignar una n-coloración es n-coloreable.

```

```
El algoritmo inicia con  con un tope máximo de colores *k*, cuande se encuentra
    una coloración válida con esta cantidad de colores, se decrementa la
   cantida de colores a *k-1*. El proceso es repetido hasta que no es posible
   encontrar una coloración válida para el número dado de colores.
"""

# Instalamos librería para visualización de grafos
!pip install networkx

# Importamos las librerías necesarias
import numpy as np
import pandas as pd
import random

import plotly.graph_objects as go
import plotly.express as px

from array import *

# Creamos los grafos

n = 20
grafos = []

for ii in range(n):
  vertices = []
  for jj in range(n):
    vertices.append(random.randint(0,1))
  grafos.append(vertices)

# Hacemos simétrica la matriz de conexiones
for ii in range(n):
  for jj in range(ii, n):
    grafos[ii][jj] = grafos[jj][ii]

for ii in range(n):
  grafos[ii][ii] = 0

for grafo in grafos:
  print(grafo)

"""## Diagrama de los grafos"""

node_list = list(range(n))
from_list = []
to_list = []

for ii in range(n):
  for jj in range(ii,n):
    if grafos[ii][jj] == 1:
      from_list.append(ii)
      to_list.append(jj)

# Importamos la librería para los grafos
import networkx as nx
import plotly.graph_objs as go
```

```
G = nx.Graph()

lst_color = []
for i in range(len(node_list)):
  G.add_node(node_list[i])
  lst_color.append('#888')


for i in range(len(from_list)):
  G.add_edges_from([(from_list[i], to_list[i])])

# Plantilla para grafo circulares (mejor visualización)
pos = nx.circular_layout(G)
for m, p in pos.items():
    G.nodes[m]['pos'] = p

# Creamos la traza de las aristas
edge_trace = go.Scatter(
    x=[],
    y=[],
    line=dict(width=0.5, color='#888'),
    hoverinfo='none',
    mode='lines')
for edge in G.edges():
    x0, y0 = G.nodes[edge[0]]['pos']
    x1, y1 = G.nodes[edge[1]]['pos']
    edge_trace['x'] += tuple([x0, x1, None])
    edge_trace['y'] += tuple([y0, y1, None])

# Creamos la traza de los nodos
node_trace = go.Scatter(
    x=[],
    y=[],
    text=[],
    mode='markers+text',
    hoverinfo='text',
    # Aquí va el color de los nodos
    marker=dict(
        showscale=False,
        reversescale=False,
        color=lst_color,
        size=37,
        colorbar=dict(
            thickness=1,
            title='Node Connections',
            xanchor='left',
            titleside='right'
        ),
        line=dict(width=0)))
for node in G.nodes():
    x, y = G.nodes[node]['pos']
    node_trace['x'] += tuple([x])
    node_trace['y'] += tuple([y])

for node, adjacencies in enumerate(G.adjacency()):
    node_trace['marker']['color'] += tuple([len(adjacencies[1])])
```

```
    node_info = adjacencies[0]
    node_trace['text'] += tuple([node_info])

# Graficamos el grafo sin colorear
title = "Grafo orifinal sin colorear ({} nodos)".format(n)
fig = go.Figure(data=[edge_trace, node_trace],
            layout=go.Layout(
            title=title, width=700, height=600,
            titlefont=dict(size=16),
            showlegend=False,
            hovermode='closest',
            margin=dict(b=21, l=5, r=5, t=40),
            xaxis=dict(showgrid=False, zeroline=False,
                       showticklabels=False, mirror=True),
            yaxis=dict(showgrid=False, zeroline=False,
showticklabels=False, mirror=True)))
fig.show()

"""## Algoritmo Genético para recolver el porblema"""

max_colors = 1

for ii in range(n):
  if sum(grafos[ii]) > max_colors:
    max_colors = sum(grafos[ii])+1
max_colors

# Definición de funciones

# Función para la creación de individuos
def individuals_creation_Func(n_colores, n):
  indiv = []

  for ii in range(n):
    indiv.append(random.randint(1,n_colores))

  return indiv


# Función para los pesos
def fitness_Func(grafo, individuo, n):
  fitness = 0

  for ii in range(n):
    for jj in range(ii, n):
      if individuo[ii] == individuo[jj] and grafo[ii][jj]==1:
        fitness += 1

  return fitness


# Función para el cruce
def cruce_Func(padre_1, padre_2, n):
  hijo_1 = []
  hijo_2 = []
```

```
  posicion_cruce = random.randint(2,n-2) # Revisar, creo que se puede con 1, n-1

  for ii in range(posicion_cruce+1):
    hijo_1.append(padre_1[ii])
    hijo_2.append(padre_2[ii])

  for jj in range(posicion_cruce+1, n):
    hijo_1.append(padre_2[jj])
    hijo_2.append(padre_1[jj])

  return hijo_1, hijo_2


# Función para la mutación
def mutacion_Func(p, individuo, n_colores):
  n_random = random.uniform(0, 1)
  if n_random <= p:
    posicion = random.randint(0, n-1)
    individuo[posicion] = random.randint(1, n_colores)
  return individuo

# Función para la selección
# Se usa una selección de rueda de ruleta
def seleccion_Func(poblacion, grafo, n):
  fitness_tot = 0
  fitness_cum = []
  fitness_cum_sum = 0

  for pop in poblacion:
    fitness_tot += 1/(1+fitness_Func(grafo, pop, n))

  for ii in range(len(poblacion)):
    fitness_cum_sum += 1 / (1+fitness_Func(grafo, poblacion[ii], n))/fitness_tot
    fitness_cum.append(fitness_cum_sum)

  new_poblacion = []

  for ii in range(len(poblacion)):
    ruleta = random.uniform(0, 1)
    for jj in range(len(poblacion)):
      if ruleta <= fitness_cum[jj]:
        new_poblacion.append(poblacion[jj])
        break
  return new_poblacion

num_colors = max_colors
generacion_hist = np.array([])
fit_hist = np.array([])

best_color_number = 0

cut_point = 200

hall_Fame = []
```

```
236
237
238 while best_color_number==0 and num_colors>0:
239   pop_size = 200
240   generacion = 0
241   poblacion = []
242
243
244   # Creamos la población
245   for ii in range(pop_size):
246     individual = individuals_creation_Func(num_colors, n)
247     poblacion.append(individual)
248
249   mejor_fitness = fitness_Func(grafos, poblacion[0], n)
250   #print(mejor_fitness)
251   fittest_individual = poblacion[0]
252
253   #print(poblacion)
254
255   gen = 0
256
257   while mejor_fitness != 0 and gen != 10000:
258     gen += 1
259     # Aplicamos la selección
260     poblacion = seleccion_Func(poblacion, grafos, n)
261     random.shuffle(poblacion)
262
263     new_pop = []
264
265     for ii in range(0, pop_size-1, 2):
266       hijo1, hijo2 = cruce_Func(poblacion[ii], poblacion[ii+1], n)
267
268       new_pop.append(hijo1)
269       new_pop.append(hijo2)
270
271     for ind in new_pop:
272       if gen < cut_point:
273         p = 0.5
274       else:
275         p = 0.25
276       ind = mutacion_Func(p, ind, num_colors)
277
278     poblacion = new_pop
279     mejor_fitness = fitness_Func(grafos, poblacion[0], n)
280     fittest_individual = poblacion[0]
281
282     for ind in poblacion:
283       if fitness_Func(grafos, ind, n) < mejor_fitness:
284         mejor_fitness = fitness_Func(grafos, ind, n)
285         fittest_individual = ind
286
287     enunc_Control = 'Generación: {},   Mejor Fitness: {},   Individuo: {}'
288     if gen % 10 == 0:
289
290       print(enunc_Control.format(gen, mejor_fitness, fittest_individual))
291
```

```
292
293     generacion_hist = np.append(generacion_hist, gen)
294     fit_hist = np.append(fit_hist, mejor_fitness)
295
296     if gen >= 1200:
297       break
298
299   enunc_no_Colors = 'Usando {} colores'
300   enunc_Cromatico = 'El número cromático de este grafo es {}'
301
302   print(enunc_no_Colors.format(num_colors))
303   print(enunc_Control.format(gen, mejor_fitness, fittest_individual))
304   print('\n')
305
306   hall_Fame.append(fittest_individual)
307
308   if mejor_fitness != 0:
309     best_color_number = 1
310
311     cromatic_num = num_colors+1
312     print(enunc_Cromatico.format(cromatic_num))
313   else:
314     generacion_hist = np.append(generacion_hist, gen)
315     fit_hist = np.append(fit_hist, mejor_fitness)
316
317
318
319     fig = go.Figure(data=go.Scatter(x=generacion_hist, y=fit_hist))
320     fig.update_layout(title='Fitness usando {} colores'.format(num_colors),
321                       xaxis_title='Generación',
322                       yaxis_title='Mejor Fitness',
323                       width=500, height=400)
324     fig.show()
325
326     print('\n \n')
327     generacion_hist = np.array([])
328     fit_hist = np.array([])
329     num_colors -= 1
330
331 # Colores finales
332 final_colors = hall_Fame[-2]
333
334 enunc_Mejor_Ind = 'El mejor individuo fue {} \n por lo que se procede a asignar
        sus colores a los grafos'
335 print(enunc_Mejor_Ind.format(final_colors))
336
337 # Creación aleatoria de los colores
338 col = []
339 for ii in range(cromatic_num):
340   col.append('#{}{}{}'.format(random.randint(0,9), random.randint(0,9), random.
      randint(0,9)))
341
342
343 colores = []
344 for ii in range(len(final_colors)):
345   colores.append(col[final_colors[ii]-1])
```

```
346
347 node_trace = go.Scatter(
348     x=[],
349     y=[],
350     text=[],
351     mode='markers+text',
352     hoverinfo='text',
353     # Aquí va el color de los nodos
354     marker=dict(
355         showscale=False,
356         reversescale=False,
357         color=colores,
358         size=37,
359         colorbar=dict(
360             thickness=1,
361             title='Node Connections',
362             xanchor='left',
363             titleside='right'
364         ),
365         line=dict(width=0)))
366 for node in G.nodes():
367     x, y = G.nodes[node]['pos']
368     node_trace['x'] += tuple([x])
369     node_trace['y'] += tuple([y])
370
371 for node, adjacencies in enumerate(G.adjacency()):
372     node_trace['marker']['color'] += tuple([len(adjacencies[1])])
373     node_info = adjacencies[0]
374     node_trace['text'] += tuple([node_info])
375
376 # Graficamos finalmente el grafo con el mínimo de colores posible
377 title = "Grafo coloreado con el mínimo posible: {} colores ({} nodos)".format(
    cromatic_num,n)
378 fig = go.Figure(data=[edge_trace, node_trace],
379              layout=go.Layout(
380              title=title, width=700, height=600,
381              titlefont=dict(size=16),
382              showlegend=False,
383              hovermode='closest',
384              margin=dict(b=21, l=5, r=5, t=40),
385              xaxis=dict(showgrid=False, zeroline=False,
386                         showticklabels=False, mirror=True),
387              yaxis=dict(showgrid=False, zeroline=False,
388 showticklabels=False, mirror=True)))
389 fig.show()
```

Listing 1: Código en Python para resolver el problea de coloreado de grafos con algoritmos genéticos.