**Rješavanje problema bojanja grafa evolucijskim algoritmom**

# Uvod

Postoji mnogo NP-teških problema pretraživanja u kojima rješenje nije moguće pronaći iscrpnim pretraživanjem u realnom vremenu. Iz tog se razloga koriste heuristički algoritmi koji uz različite pretpostavke o problemu pronalaze rješenja u prihvatljivom vremenu, smanjujući prostor pretraživanja. Upravo zbog smanjivanja prostora pretraživanja rješenje pronađeno takvim algoritmima nije uvijek optimalno.

Jedan NP-težak problem je određivanje kromatskog broja grafa, a jedna heuristička skupina algoritama su evolucijski algoritmi. Ovaj rad pobliže opisuje način rada evolucijskih algoritama, problem bojanja grafa, te primjenu i analizu evolucijskih algoritama na određivanje kromatskog broja grafa.

# Kromatski broj grafa

Kromatski broj grafa, χ(G), predstavlja minimalan broj boja s kojima se mogu obojati vrhovi grafa, tako da niti jedna dva susjedna vrha (vrhovi spojeni bridom) nisu obojana istom bojom. Kromatski broj koristi se u rješavanju mnogih problema; problema raspoređivanja, dodjele registara u procesoru, rješavanja Sudoku, dodjele radio frekvencije, sparivanje uzoraka...

## Problem

Pronalazak kromatskog broja grafa je NP-težak problem, što znači da je vremenska složenost njegovog pronalaska eksponencijalna. Zbog toga nije praktično koristiti se algoritmima grube sile na grafovima s većim brojem vrhova. Jednostavan algoritam isrcpnog pretreaživanja bi prošao sve kombinacije k boja i n vrhova → O(k^n). Implementacija pohlepnog algoritma pretraživanja izgledala bi ovako:

TODO → napiši sam pseudokod obje fje

def greedy\_chromatic\_number(graph):

"""Returns the chromatic number of a graph using a greedy algorithm."""

# Initialize all vertices to be uncolored

colors = {}

for v in graph:

colors[v] = None

# Assign colors to vertices in order of highest degree

for v in sorted(graph, key=lambda v: len(graph[v]), reverse=True):

used\_colors = set(colors[n] for n in graph[v])

available\_colors = set(range(len(graph)))

available\_colors.difference\_update(used\_colors)

colors[v] = min(available\_colors)

# Return the maximum color used

return max(colors.values()) + 1

Jedan iterativni pristup pronalaska kromatskog broja je pokušati obojati graf s k boja. Ukoliko uspijemo pronaći takvo rješenje, pokušavamo s k-1 bojom. Postupak nastavljamo sve dok ne uspijemo pronaći k-n bojanje, ili ne pronađemo ispravno bojanje za k=1. Svaki graf s n vrhova možemo obojati s n boja. Uvjet na gornju ogradu kromatskog broja možemo postrožiti.

## Gornja ograda na kromatski broj

Prema Brookovom teoremu, kromatski broj svakog grafa je maksimalno Δ + 1, gdje je Δ maksimalan stupanj grafa. Maksimalan stupanj grafa je najveći broj bridova koji su incidentni s bilo kojim pojedinačnim vrhom u grafu.

## Donja ograda na kromatski broj

Donja ograda na kromatski broj je 1. Ovo se postiže samo za grafove u kojima niti jedna dva vrha nisu povezana.

# Evolucijski algoritmi

# Analiza literature

# Praktični dio

# Rezultati praktičnog dijela

# Zaključak