```
import random
import numpy as np
import matplotlib.pyplot as plt

# Graph represented as an adjacency list
graph = {
    0: [1, 2],
    1: [0, 2, 3],
    2: [0, 1, 3],
    3: [1, 2, 4],
    4: [3]
}

num_vertices = len(graph)

# Parameters
n = 20  # Number of cuckoos (solutions)
iterations = 100  # Number of iterations
pa = 0.25  # Discovery probability (chance to replace a solution)
max_colors = 4  # Maximum number of colors to use in the coloring

# Generate a random coloring (solution)
def generate_random_coloring():
    return [random.randint(0, max_colors - 1) for _ in
range(num_vertices)]

# Fitness function: count the number of color conflicts (edges where
two adjacent vertices have the same color)
def fitness(coloring):
    conflicts = 0
    for u in range(num_vertices):
        for v in graph[u]:
            if coloring[u] == coloring[v]:  # Conflict: two adjacent
vertices have the same color
                conflicts += 1
    return conflicts

# Swap mutation: randomly pick two vertices and change their colors
def swap_mutation(coloring):
    new_coloring = coloring[:]
    i, j = random.sample(range(num_vertices), 2)
    new_coloring[i] = random.randint(0, max_colors - 1)  # Reassign
color to vertex i
    new_coloring[j] = random.randint(0, max_colors - 1)  # Reassign
color to vertex j
    return new_coloring

# Cuckoo Search Algorithm
def cuckoo_search():
    # Initialize the population (host nests)
    nests = [generate_random_coloring() for _ in range(n)]
```

```
    fitness_values = [fitness(nest) for nest in nests]

    best_nest = nests[np.argmin(fitness_values)]
    best_fitness = min(fitness_values)

    for iteration in range(iterations):
        new_nests = []
        for i in range(n):
            # Generate a new solution using swap mutation
            new_nest = swap_mutation(nests[i])
            new_fitness = fitness(new_nest)

            # If the new nest has fewer conflicts, replace the old nest
            if new_fitness < fitness_values[i]:
                nests[i] = new_nest
                fitness_values[i] = new_fitness

        # Find the best solution in the population
        best_nest_idx = np.argmin(fitness_values)
        if fitness_values[best_nest_idx] < best_fitness:
            best_fitness = fitness_values[best_nest_idx]
            best_nest = nests[best_nest_idx]

        # Replace some of the worst nests with random colorings based
on discovery probability
        for i in range(n):
            if random.random() < pa:
                nests[i] = generate_random_coloring()
                fitness_values[i] = fitness(nests[i])

        # Optional: Print progress
        if iteration % 10 == 0:
            print(f"Iteration {iteration}, Best Conflicts:
{best_fitness}")

    return best_nest, best_fitness

# Run the Cuckoo Search Algorithm
best_coloring, best_conflicts = cuckoo_search()

# Display the result
print(f"Best Coloring: {best_coloring}")
print(f"Best Fitness (Conflicts): {best_conflicts}")

# Visualize the result (coloring of the graph)
plt.figure(figsize=(8, 6))
colors = ['r', 'g', 'b', 'y']  # Color palette

# Plot vertices with corresponding colors
for i, vertex in enumerate(graph):
```

```
    plt.scatter(i, 0, c=colors[best_coloring[i]], s=100, label=f"Vertex
{i}")

# Draw edges
for u in range(num_vertices):
    for v in graph[u]:
        if u < v:
            plt.plot([u, v], [0, 0], 'k-', lw=1)

plt.title(f"Best Coloring with {best_conflicts} conflicts")
plt.xlabel("Vertex")
plt.ylabel("Color")
plt.show()
```

Output:

```
Iteration 0, Best Conflicts: 0
Iteration 10, Best Conflicts: 0
Iteration 20, Best Conflicts: 0
Iteration 30, Best Conflicts: 0
Iteration 40, Best Conflicts: 0
Iteration 50, Best Conflicts: 0
Iteration 60, Best Conflicts: 0
Iteration 70, Best Conflicts: 0
Iteration 80, Best Conflicts: 0
Iteration 90, Best Conflicts: 0
Best Coloring: [0, 1, 3, 2, 1]
Best Fitness (Conflicts): 0
```