LAB PROGRAM-6

Write a program to implement Simulated Annealing algorithm:

N-queens problem

import mlrose\_hiive as mlrose

import numpy as np

def queens\_max(position):

    n = len(position)

    attacks = 0

    for i in range(n):

        for j in range(i + 1, n):

            # Check if queens attack each other

            if position[i] == position[j] or abs(position[i] - position[j]) == j - i:

                attacks += 1

    # The fitness is the total number of pairs of queens minus the number of attacks

    return (n \* (n - 1) // 2) - attacks

# Define the custom fitness function

objective = mlrose.CustomFitness(queens\_max)

# Set up the optimization problem

problem = mlrose.DiscreteOpt(length=8, fitness\_fn=objective, maximize=True, max\_val=8)

T = mlrose.ExpDecay()

# Define the initial position

initial\_position = np.array([4, 6, 1, 5, 2, 0, 3, 7])

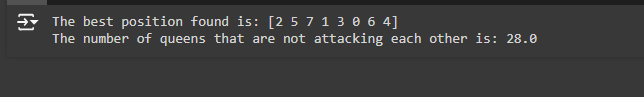
# Run the simulated annealing algorithm

best\_state, best\_fitness, \_ = mlrose.simulated\_annealing(problem, schedule=T, max\_attempts=500, max\_iters=5000, init\_state=initial\_position)

print('The best position found is:', best\_state)

print('The number of queens that are not attacking each other is:', best\_fitness)

OUTPUT:



Another Application of SA- Job Scheduling Problem

import mlrose\_hiive as mlrose

import numpy as np

# Define the processing times for each job

job\_times = [2, 14, 4, 16, 6, 5, 3, 12]  # Processing times for each job

num\_machines = 3  # Number of machines available

# Define a fitness function to calculate the makespan (total time taken by the slowest machine)

def job\_scheduling\_fitness(state):

    machine\_times = [0] \* num\_machines  # Initialize machine times

    for i, job in enumerate(state):

        machine\_times[job] += job\_times[i]  # Add job time to the assigned machine

    makespan = max(machine\_times)  # Makespan is the max time of all machines

    return -makespan  # We negate because we want to minimize the makespan

# Define a custom fitness function

objective = mlrose.CustomFitness(job\_scheduling\_fitness)

# Define the optimization problem

problem = mlrose.DiscreteOpt(length=len(job\_times), fitness\_fn=objective, maximize=True, max\_val=num\_machines)

# Define the simulated annealing schedule

schedule = mlrose.ExpDecay()  # Exponential decay schedule for simulated annealing

# Initial state: assign each job to a random machine

initial\_state = np.random.randint(0, num\_machines, size=len(job\_times))

# Perform the simulated annealing algorithm

best\_state = mlrose.simulated\_annealing(

    problem=problem,

    schedule=schedule,

    max\_attempts=100,

    max\_iters=1000,

    init\_state=initial\_state

)

# Since best\_state is an object with both the best fitness and state, extract them

best\_position = best\_state[0]

best\_fitness = best\_state[1]

print("Best job-to-machine assignment:", best\_position)

print("Minimum makespan:", -best\_fitness)

# Display machine assignments

machine\_assignments = [[] for \_ in range(num\_machines)]

for job, machine in enumerate(best\_position):

    machine\_assignments[machine].append((job, job\_times[job]))

for i, jobs in enumerate(machine\_assignments):

    print(f"Machine {i+1} jobs:", jobs, "Total time:", sum(job[1] for job in jobs))

