Shuvam Rajbanshi

1BM22CS275

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

	foriinrange(n):
		forjinrange(i+1,n):
			#Checkifqueensattackeachother
			ifposition[i]==position[j]orabs(position[i]-
position[j]) == j - i:
				attacks+=1

	#Thefitnessisthetotalnumberofpairsofqueensminusthe
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:

```
The best position found is: [2 5 7 1 3 0 6 4]
The number of queens that are not attacking each other is: 28.0
```

```
# Define the processing times for each job
job_times=[2,14,4,16,6,5,3,12] #Processingtimesforeach
job
num_machines=3 #Numberofmachinesavailable

# 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 #Initializemachinetimes

	fori,jobinenumerate(state):
		machine_times[job]+=job_times[i] #Addjobtimetothe
assigned machine

	makespan=max(machine_times) #Makespanisthemaxtimeofall
machines
	return-makespan #Wenegatebecausewewanttominimizethe
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() #Exponentialdecayschedulefor
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,"Totaltime:",sum(job[1]for
job in jobs))
```

```
Best job-to-machine assignment: [1 2 0 1 2 0 1 0]
Minimum makespan: 21.0
Machine 1 jobs: [(2, 4), (5, 5), (7, 12)] Total time: 21
Machine 2 jobs: [(0, 2), (3, 16), (6, 3)] Total time: 21
Machine 3 jobs: [(1, 14), (4, 6)] Total time: 20
```