Knapsack-SimulatedAnnealing

November 17, 2023

[ ]:
```
import six
import sys
sys.modules['sklearn.externals.six'] = six
import mlrose
import numpy as np


def solve_Knapsack(capacity, weights, values, optimal_solution):
    # Calculating and checking weight percentage
    peso = sum(weights)
    w_pct = capacity/peso
    print('Weight percentage:', w_pct)

    # Converting the optimal solution type
    optimal_solution = np.array(optimal_solution)

    # Checking if the optimal solution is truly optimal
    print("Optimal solution's weight: ", sum(optimal_solution*weights))

    # Evaluating fitness of the optimal solution
    fitness = mlrose.Knapsack(weights, values, w_pct)
    print("Optimal solution's fitness:", fitness.evaluate(optimal_solution))

    # Defining fitness for the input data
    fitness_1 = mlrose.Knapsack(weights = weights, values = values,␣
 ↪max_weight_pct=w_pct)

    # Defining the optimization problem to be solved
    problem = mlrose.DiscreteOpt(length = len(weights), fitness_fn=fitness_1,␣
 ↪maximize=True)

    # Defining a temperature parameter
    T = mlrose.GeomDecay()

    # Defining best state and fitness for the optimized data
    best_state, best_fitness = mlrose.simulated_annealing(problem,␣
 ↪max_attempts=10, curve=False, random_state=1, schedule=T)
```

```
    # Checking results
    print("My model's knapsack weight:", sum(best_state*weights))
    print("My model's fitness:",best_fitness)
    print("My model's best state:",best_state)

print('Knapsack P01')
capacity1 = 165
weights1 = [23, 31, 29, 44, 53, 38, 63, 85, 89, 82]
values1= [92,57,49,68,60,43,67,84,87,72]
optimal_solution1 = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0]

knapsack1 = (capacity1, weights1, values1, optimal_solution1)
solve_Knapsack(*knapsack1)
```

```
Knapsack P01
Weight percentage: 0.30726256983240224
Optimal solution's weight:  165
Optimal solution's fitness: 309
My model's knapsack weight: 158
My model's fitness: 187.0
My model's best state: [0 1 0 0 0 1 0 0 1 0]
```

[ ]:
```
print('Knapsack P02')
capacity2 = 26
weights2 = [12,7,11,8,9]
values2= [24,13,23,15,16]
optimal_solution2 = [0,1,1,1,0]

knapsack2 = (capacity2, weights2, values2, optimal_solution2)
solve_Knapsack(*knapsack2)
```

```
Knapsack P02
Weight percentage: 0.5531914893617021
Optimal solution's weight:  26
Optimal solution's fitness: 51
My model's knapsack weight: 21
My model's fitness: 40.0
My model's best state: [1 0 0 0 1]
```

[ ]:
```
print('Knapsack P03')
capacity3 = 190
weights3 = [56,59,80,64,75,17]
values3 = [50,50,64,46,50,5]
optimal_solution3 = [1,1,0,0,1,0]

knapsack3 = (capacity3, weights3, values3, optimal_solution3)
```

```
solve_Knapsack(*knapsack3)
```

```
Knapsack P03
Weight percentage: 0.5413105413105413
Optimal solution's weight:  190
Optimal solution's fitness: 150
My model's knapsack weight: 140
My model's fitness: 101.0
My model's best state: [0 1 0 1 0 1]
```

[ ]:
```
print('Knapsack P04')
capacity4 = 50
weights4 = [31,10,20,19,4,3,6]
values4 = [70,20,39,37,7,5,10]
optimal_solution4 = [1,0,0,1,0,0,0]

knapsack4 = (capacity4, weights4, values4, optimal_solution4)
solve_Knapsack(*knapsack4)
```

```
Knapsack P04
Weight percentage: 0.5376344086021505
Optimal solution's weight:  50
Optimal solution's fitness: 107
My model's knapsack weight: 43
My model's fitness: 81.0
My model's best state: [0 1 1 0 1 1 1]
```

[ ]:
```
print('Knapsack P05')
capacity5 = 104
weights5 = [25,35,45,5,25,3,2,2]
values5 = [350,400,450,20,70,8,5,5]
optimal_solution5 = [1,0,1,1,1,0,1,1]

knapsack5 = (capacity5, weights5, values5, optimal_solution5)
solve_Knapsack(*knapsack5)
```

```
Knapsack P05
Weight percentage: 0.7323943661971831
Optimal solution's weight:  104
Optimal solution's fitness: 900
My model's knapsack weight: 92
My model's fitness: 838.0
My model's best state: [1 1 0 0 1 1 1 1]
```

[ ]:
```
print('Knapsack P06')
capacity6 = 170
weights6 = [41,50,49,59,55,57,60]
```

```
values6 = [442,525,511,593,546,564,617]
optimal_solution6 = [0,1,0,1,0,0,1]

knapsack6 = (capacity6, weights6, values6, optimal_solution6)
solve_Knapsack(*knapsack6)
```

```
Knapsack P06
Weight percentage: 0.4582210242587601
Optimal solution's weight:  169
Optimal solution's fitness: 1735
My model's knapsack weight: 165
My model's fitness: 1688.0
My model's best state: [0 1 0 0 1 0 1]
```

```
/home/hub/Documents/Disciplina IA/Atividade Prática 1 -
Knapsack/venv_mlrose/lib/python3.11/site-packages/mlrose/algorithms.py:311:
RuntimeWarning: overflow encountered in exp
  prob = np.exp(delta_e/temp)
```

[ ]:
```
print('Knapsack P07')
capacity7 = 750
weights7 = [70,73,77,80,82,87,90,94,98,106,110,113,115,118,120]
values7 = [135,139,149,150,156,163,173,184,192,201,210,214,221,229,240]
optimal_solution7 = [1,0,1,0,1,0,1,1,1,0,0,0,0,1,1]

knapsack7 = (capacity7, weights7, values7, optimal_solution7)
solve_Knapsack(*knapsack7)
```

```
Knapsack P07
Weight percentage: 0.5233775296580601
Optimal solution's weight:  749
Optimal solution's fitness: 1458
My model's knapsack weight: 745
My model's fitness: 1448.0
My model's best state: [1 1 0 0 1 0 1 1 1 0 0 0 0 1 1]
```

Considerando os resultados obtidos através dos algoritmos de Hill Climb e Simulated Annealing, torna-se possível uma comparação entre a saída fornecida por cada algoritmo para cada problema:

Knapsack P01
Hill Climb's fitness: 309.0
Simulated Annealing's fitness: 187.0

Knapsack P02
Hill Climb's fitness: 51.0
Simulated Annealing's fitness: 40.0

Knapsack P03
Hill Climb's fitness: 150.0
Simulated Annealing's fitness: 101.0

Knapsack P04
Hill Climb's fitness: 107.0
Simulated Annealing's fitness: 81.0

Knapsack P05
Hill Climb's fitness: 900.0
Simulated Annealing's fitness: 838.0

Knapsack P06
Hill Climb's fitness: 1735.0
Simulated Annealing's fitness: 1688.0

Knapsack P07
Hill Climb's fitness: 1454.0
Simulated Annealing's fitness: 1448.0

Conclui-se através dos valores mostrados acima que o algoritmo Hill Climb apresentou um fitness melhor que o algoritmo Simulated Annealing.