Atividade Prática 2 - Algoritmos de Busca

- Fazer um comparativo entre os algoritmos Simulated Annealing e Hill Climbing para as bases P01 a P07;
- Desenvolver a função de aptidão knapsack no Mlrose;
- Apresentar a melhor solução encontrada por cada algoritmo e comparar com a melhor solução global disponível para a base de dados;
- Enviar os arquivos *.ipynb e uma versão pdf do código fonte.

Responsável: Marcos Angelo Cemim

```python
from urllib.request import urlopen
import numpy as np
import six
import sys
sys.modules['sklearn.externals.six'] = six
import mlrose
import time
import warnings
warnings.filterwarnings("ignore")
```

```python
#Iterates over 7 bases
for _ in range(1,8):

    base = f'p0{_}'
    # Assign values of current base to variables
    c = int(urlopen(f'https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/{base}_c.txt').read().decode('utf-8').split()
    w = [int(x) for x in urlopen(f'https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/{base}_w.txt').read().decode('ut
    p = [int(x) for x in urlopen(f'https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/{base}_p.txt').read().decode('ut
    s = [int(x) for x in urlopen(f'https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/{base}_s.txt').read().decode('ut

    # Print to check mistakes
    # print(f'{"*"*15} Base: {base} {"*"*15}')
    # print(f'Capacity: {c}')
    # print(f'Weight: {w}')
    # print(f'Profit: {p}')
    # print(f'Optimal Selection: {s}')

    # Define fitness function (total profit = solution_array * profit_array) . If total weight > capacity, penalizes returnin
    def fn_fitness(solution):
        if sum(np.multiply(solution, w).tolist()) <= c:
            return sum(np.multiply(solution, p).tolist())
        else:
            return 1

    # Assign fitness function to mlrose format
    fitness = mlrose.CustomFitness(fn_fitness)

    # Define problem
    problema = mlrose.DiscreteOpt(length = len(s), fitness_fn = fitness,
                                  maximize = True, max_val = 2)

    # Run "Hill Climb" algorithm
    start_time_hc = time.time()
    best_fit_hc = 0
    len_curve_hc = 0
    while best_fit_hc < sum(np.multiply(s, p).tolist()):
        solution_hc, best_fit_hc, curve_hc = mlrose.hill_climb(problema, restarts=10, curve=True)
        len_curve_hc += len(curve_hc)
    end_time_hc = time.time()

    # Run "Simulated Annealing" algorithm
    start_time_sa = time.time()
    best_fit_sa = 0
    len_curve_sa = 0
    while best_fit_sa < sum(np.multiply(s, p).tolist()):
        solution_sa, best_fit_sa, curve_sa = mlrose.simulated_annealing(problema, max_attempts=10, curve=True)
        len_curve_sa += len(curve_sa)
    end_time_sa = time.time()

    # Results
    print(f' Base P0{_} '.center(98, '*'))
    print(f"{'Algorithm':20s} | {'Solutions Tried':15s} | {'Fitness Value':13s} | {'Optimal Fitness':15s} | {'Time (ms)':9s}
    print(f"{'-'*20} | {'-'*15} | {'-'*13} | {'-'*15} | {'-'*9} | {'-'*10}")
    print(f"{'Hill Climb':20s} | {len_curve_hc:15d} | {best_fit_hc:13.0f} | {sum(np.multiply(s, p).tolist()):15.0f} | {1000 *
    print(f"{'Simulated Annealing':20s} | {len_curve_sa:15d} | {best_fit_sa:13.0f} | {sum(np.multiply(s, p).tolist()):15.0f}
```

```
    print()
```

```
******************************************** Base P01 ********************************************
Algorithm            | Solutions Tried | Fitness Value | Optimal Fitness | Time (ms) | Array Size
-------------------- | --------------- | ------------- | --------------- | --------- | ----------
Hill Climb           |              11 |           309 |             309 |    1.9975 |         10
Simulated Annealing  |             275 |           309 |             309 |    4.1316 |         10

******************************************** Base P02 ********************************************
Algorithm            | Solutions Tried | Fitness Value | Optimal Fitness | Time (ms) | Array Size
-------------------- | --------------- | ------------- | --------------- | --------- | ----------
Hill Climb           |               6 |            51 |              51 |    0.0000 |          5
Simulated Annealing  |              52 |            51 |              51 |    0.9990 |          5

******************************************** Base P03 ********************************************
Algorithm            | Solutions Tried | Fitness Value | Optimal Fitness | Time (ms) | Array Size
-------------------- | --------------- | ------------- | --------------- | --------- | ----------
Hill Climb           |               7 |           150 |             150 |    1.0014 |          6
Simulated Annealing  |              84 |           150 |             150 |    0.9973 |          6

******************************************** Base P04 ********************************************
Algorithm            | Solutions Tried | Fitness Value | Optimal Fitness | Time (ms) | Array Size
-------------------- | --------------- | ------------- | --------------- | --------- | ----------
Hill Climb           |              17 |           107 |             107 |    1.9982 |          7
Simulated Annealing  |             649 |           107 |             107 |    8.5533 |          7

******************************************** Base P05 ********************************************
Algorithm            | Solutions Tried | Fitness Value | Optimal Fitness | Time (ms) | Array Size
-------------------- | --------------- | ------------- | --------------- | --------- | ----------
Hill Climb           |              60 |           900 |             900 |    5.0151 |          8
Simulated Annealing  |             273 |           900 |             900 |    4.0448 |          8

******************************************** Base P06 ********************************************
Algorithm            | Solutions Tried | Fitness Value | Optimal Fitness | Time (ms) | Array Size
-------------------- | --------------- | ------------- | --------------- | --------- | ----------
Hill Climb           |              20 |          1735 |            1735 |    3.1841 |          7
Simulated Annealing  |             205 |          1735 |            1735 |    2.5451 |          7

******************************************** Base P07 ********************************************
Algorithm            | Solutions Tried | Fitness Value | Optimal Fitness | Time (ms) | Array Size
-------------------- | --------------- | ------------- | --------------- | --------- | ----------
Hill Climb           |            1761 |          1458 |            1458 |  561.2063 |         15
Simulated Annealing  |           14519 |          1458 |            1458 |  196.5473 |         15
```

Conclusion:

De forma geral, o algoritmo Hill Climb leva vantagem com relação ao número de iterações até encontrar a solução ótima, porém perde quando medimos o tempo necessário para atingir tal ponto. Em arrays maiores, essa diferença fica ainda maior. Esse tempo pode (e deve) ser ajustado em função dos parâmetros dos modelos (restarts / max_attempts) para cada aplicação.