Universidade Federal de Viçosa

Campus Florestal

Meta-Heurísticas CCF 480

Relatório de Trabalho Prático 1

Samuel Sena - 3494

04/09/2021

Neste trabalho foi realizada uma implementação dos algoritmos de busca Hill-Climbing (HC) e Iterated Local Search (ILS) para minimização na linguagem de programação Python3. Foram realizadas 30 execuções de cada algoritmo para cada uma das funções objetivos com seus respectivos limites para X e Y. Os resultados de cada grupo de execução foram armazenados separadamente e posteriormente foram calculados os valores mínimo e máximo, média e desvio padrão, além da plotagem de um boxplot.

## Implementação

Para a implementação dos algoritmos propostos, eu utilizei os slides disponibilizados pelo Pvanet para servir de base de conceito e também realizei pesquisas pela Web para entender melhor como realizar a implementação em código fonte. Todos os sites utilizados foram referenciados ao final deste relatório.

Uma das primeiras funções implementadas e que é utilizada pelos dois algoritmos, consiste na função “entre_limites”, que retorna True para caso o ponto esteja dentro dos limites e False caso contrário. A função recebe como parâmetro duas variáveis: um vetor que representa um ponto (x,y) e uma matriz 2x2 que cada linha representa os limites de uma variável (x e y) e cada coluna representa o limite inferior e superior de cada variável.

```python
def entre_limites(ponto, limites):
    #  Para cada linha, isto é, x e y, verificamos os intervalos.
    for d in range(len(limites)):
        # verificando se está fora dos limites
        if ponto[d] < limites[d, 0] or ponto[d] > limites[d, 1]:
            return False
    return True
```

A função “hc” implementa o algoritmo Hill-Climbing, ela recebe como parâmetro uma função objetivo (para ser utilizada no processo de minimização), uma matriz com os limites das variáveis, a quantidade de iterações desejadas e um número ponto flutuante que representa o tamanho do passo no processo de perturbação do candidato a nova solução.

Inicialmente, na inicialização, as variáveis “ls” e “li” recebem os limites superior e inferior respectivamente, cada uma delas então armazena um vetor de duas posições contendo os limites de x e y, em seguida utilizando um conceito definido nos slides (para a geração de um ponto aleatório dentro dos limites), um ponto inicial aleatório é definido e salvo na variável “solucao”. Mais adiante, na modificação, o processo de execução iterativa é iniciado executando a quantidade de vezes passada por parâmetro. Um candidato a nova solução é criado a partir de uma perturbação aleatória uniforme da solução atual e caso o valor da função objetivo deste candidato apresente um resultado melhor (isto é, menor que a atual), a solução candidata ocupa o espaço da solução atual. É importante ressaltar que durante o processo de geração da solução candidata, é verificado se os limites para x e y são respeitados. Ao final, uma matriz contendo os pontos x e y e o valor da função objetiva dos pontos é retornada.

```
def hc(objetivo, limites, iteracoes, Tam_passo):
    # Inicialização
    ls = limites[:,1] #Limite Superior
    li = limites[:,0] #Limite Inferior
    #De acordo com o slide, isso garante que ponto inicial aleatorio esteja dentro do int
    solucao = (ls - li) * rand(len(limites)) + li #V = (Ls - Li) * ri +Li
    # Calculando valor da solucao inicial
    valor_solucao = objetivo(solucao)
    # Modificação
    for i in range(iteracoes):
        candidato = None
        while candidato is None or not entre_limites(candidato, limites): #verificando se
            candidato = solucao + randn(len(limites)) * Tam_passo #Tamanho do passo (Pert
        # Calculando valor de candidato
        valor_candidato = objetivo(candidato)
        # Verificando se candidato é melhor
        if valor_candidato <= valor_solucao:
            # Armazenando nova solução melhor
            solucao, valor_solucao = candidato, valor_candidato
            #print("-> Iteracao: ", i, " ->X:", solucao[0], " ->Y: ", solucao[1], "-> Val
    return [solucao, valor_solucao]
```

A função “ils” implementa o algoritmo Iterated Local Search, ela recebe como parâmetro uma função objetivo (para ser utilizada no processo de minimização), uma matriz com os limites das variáveis, a quantidade de iterações desejadas, um número ponto flutuante de tamanho do passo (para ser utilizado na busca local feita com o Hill-Climbing), a quantidade de reinícios desejada e um ponto flutuante que representa o tamanho da perturbação no algoritmo ILS.

Inicialmente, assim como no Hill-Climbing são declaradas as variáveis “ls” e “li” representando o limite das variáveis e em seguida é definido o ponto de partida na variável “inicial”. Antes do processo de pesquisa em bacias vizinhas, é procurado um mínimo local em uma busca local com o Hill-Climbing partindo do ponto inicial aleatório gerado. Adiante, um laço é iniciado executando a quantidade de reinícios desejado e para cada iteração um novo ponto inicial aleatório uniforme é definido, ao mesmo tempo que é verificado se os limites são respeitados. Para cada ponto aleatório gerado, uma busca local utilizando o algoritmo Hill-Climbing é executada e, ao final, caso a solução encontrada seja melhor (menor) que a solução melhor atual, ela então é armazenada como solução melhor. Ao final, uma matriz contendo os pontos x e y e o valor da função objetiva dos pontos é retornada.

```
def ils(objetivo, limites, iteracoes, Tam_passoHC, reinicios, Tam_P):
    # definindo ponto de partida inicial
    li = limites[:,0]
    ls = limites[:,1]
    #De acordo com o slide, isso garante que ponto inicial aleatorio esteja dentro do intervalo l
    inicial = (ls - li) * rand(len(limites)) + li #gerando valor inicial aleatoriamente
    #valor_inicial = objetivo(inicial)
    #encontrando melhor local para ponto de partida inicial utilizando HC
    melhor , valor_melhor = hillclimbing(objetivo, limites, iteracoes, Tam_passoHC, inicial)
    # reinicios
    for n in range(reinicios):
        # gerando um ponto inicial de cada reinicio como uma versão perturbada do último melhor
        ponto_inicial = None
        while ponto_inicial is None or not entre_limites(ponto_inicial, limites): #verificando se
            ponto_inicial = melhor + randn(len(limites)) * Tam_P #Pertubando
        #Utilizando hillclimbing para busca local em ILS
        solucao, valor = hillclimbing(objetivo, limites, iteracoes, Tam_passoHC, ponto_inicial)
        # Verificando se solução encontrada é melhor que melhor atual
        if valor < valor_melhor:
            melhor, valor_melhor = solucao, valor
            #print("-> Reinicio: ",n,"-> Melhor: X =  ", melhor[0], " , Y = ", melhor[1],"-> Valo
    return [melhor, valor_melhor]
```

Os limites foram armazenados em arrays diferentes para cada letra de cada exercício, dessa forma foram realizadas chamadas das funções dos algoritmos HC e ILS para cada par (função Objetivo, Limite):

```
# definindo limites
limites1a = asarray([[-1.5, 4.],
                     [-3., 4.]])

limites1b = asarray([[-1., 0.],
                     [-2., -1.]])

limites2c = asarray([[-512., 512.],
                     [-512., 512.]])

limites2d = asarray([[511., 512.],
                     [404., 405.]])
```

As funções objetivo foram implementadas em duas funções com os nomes de “objetivo1” e “objetivo2”, sendo que cada uma recebe como parâmetro um vetor contendo os valores de x e y e retorna o valor da função no ponto.

```
def objetivo1(v):
    x, y = v
    return (sin(x+y) + pow(x-y, 2) - 1.5*x + 2.5*y + 1)


def objetivo2(v):
    x, y = v
    return (-(y+47)*sin(sqrt(abs((x/2) + (y+47)))) - x*sin(sqrt(abs(x-(y+47)))))
```

Para a realização dos cálculos estatísticos, cada resultado de cada execução foi armazenado em uma lista independente em cada uma das 30 iterações. Após o término do laço iterativo, com o uso das funções mean(), max(), min(), std() e boxplot() os resultados estatísticos foram encontrados e são impressos na saída padrão e em um arquivo de texto “Resultados.txt” e os boxplots salvos em png na pasta “plots”.

```
for i in range(30):
    _ , valor = ils(objetivo1, limites1a, iteracoes, Tam_passoHC, reinicios, Tam_P)
    Resultados1A_ILS.append(valor)
    _ , valor = melhor, valor = hc(objetivo1, limites1a, iteracoes, Tam_passoHC)
    Resultados1A_HC.append(valor)
    _ , valor = ils(objetivo1, limites1b, iteracoes, Tam_passoHC, reinicios, Tam_P)
    Resultados1B_ILS.append(valor)
    _ , valor = melhor, valor = hc(objetivo1, limites1b, iteracoes, Tam_passoHC)
    Resultados1B_HC.append(valor)
    _ , valor = ils(objetivo2, limites2c, iteracoes, Tam_passoHC, reinicios, Tam_P)
    Resultados2C_ILS.append(valor)
    _ , valor = melhor, valor = hc(objetivo2, limites2c, iteracoes, Tam_passoHC)
    Resultados2C_HC.append(valor)
    _ , valor = ils(objetivo2, limites2d, iteracoes, Tam_passoHC, reinicios, Tam_P)
    Resultados2D_ILS.append(valor)
    _ , valor = melhor, valor = hc(objetivo2, limites2d, iteracoes, Tam_passoHC)
    Resultados2D_HC.append(valor)
```

Vale ressaltar que as função numpy.random.randn gera amostras a partir da distribuição normal, enquanto numpy.random.rand gera a partir de uma distribuição uniforme (no intervalo [0,1)), ambas foram utilizadas durante a implementação dos algoritmos.

As configurações escolhidas para realizar a execução dos dois algoritmos foram:

- Quantidade de iterações em HC: 1000
- Tamanho do passo (perturbação) em HC: 0.05
- Tamanho da perturbação em ILS: 0.5
- Quantidade de reinícios em ILS: 50

## Resultados

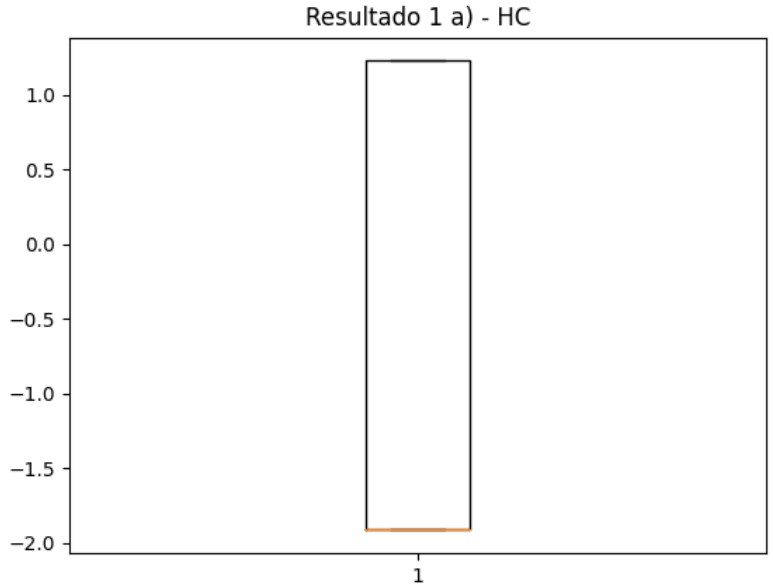
1) Minimizar $f(x,y) = sin(x + y) + sin(x - y)^2 - 1.5x + 2.5y + 1$

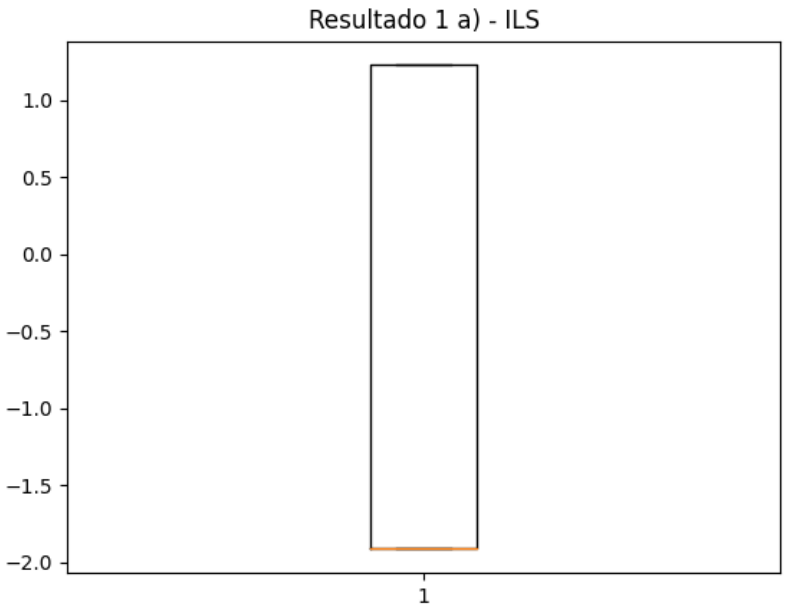
a) Para $-1.5 \leq x \leq 4$ e $-3 \leq y \leq 4$ :

Para esses intervalos de x e y foram encontradas as seguintes estatísticas para cada algoritmo:

| Algoritmo | Mínimo | Máximo | Média | Desvio-Padrão |
|---|---|---|---|---|
| HC | -1.9132229029665 | 1.22838588757041 | -0.4471406042876 | 1.56730272153633 |
| ILS | -1.9132229522357 | 1.22836990221301 | -0.7613055299338 | 1.51391592749463 |

Boxplots:



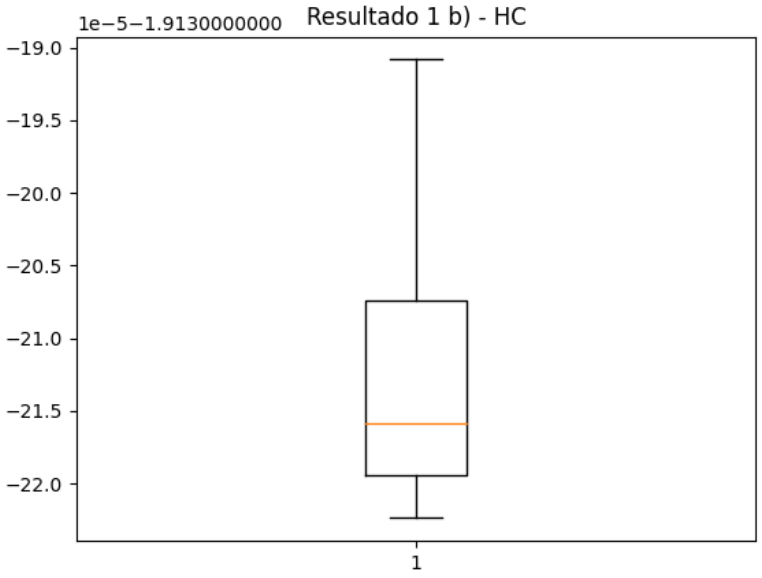



b) Para $-1 \leq x \leq 0$ e $-2 \leq y \leq 1$ :

Para esses intervalos de x e y foram encontradas as seguintes estatísticas para cada algoritmo:

| Algoritmo | Mínimo | Máximo | Média | Desvio-Padrão |
|---|---|---|---|---|
| HC | -1.9132223680248 | -1.9131908309585 | -1.9132128369735 | 8.60365019666e-6 |
| ILS | -1.9132229514177 | -1.9132226323117 | -1.9132228594964 | 8.79348032775e-8 |

Boxplots:



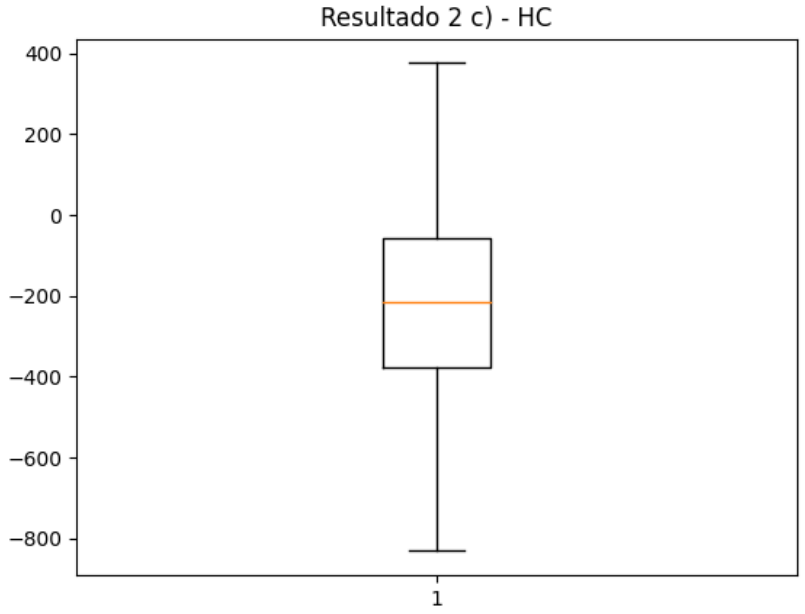

2) Minimizar $f(x,y) = -(y+47)sin\sqrt{(|\frac{x}{2}+(y+47)|)} - xsin\sqrt{(|x-(y+47)|)}$

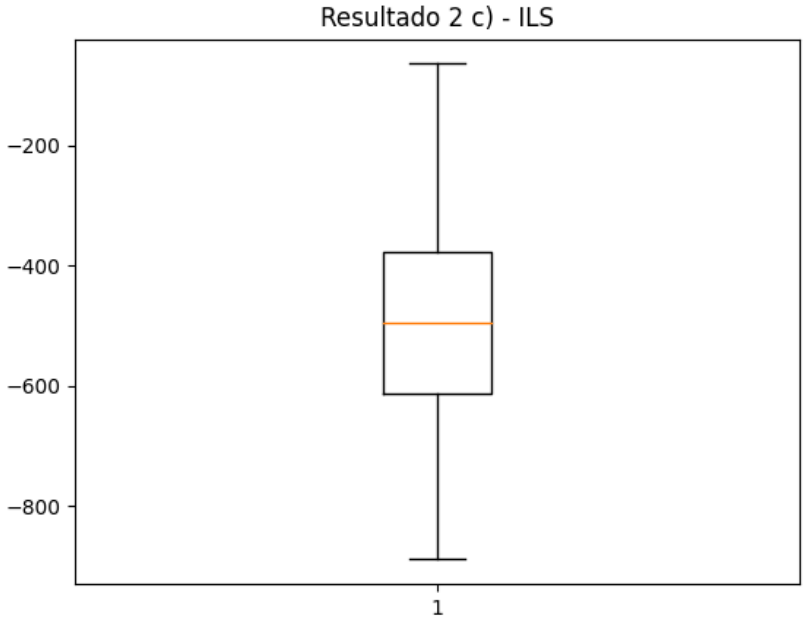
c) Para $-512 \leq x, y \leq 512$ :

Para esses intervalos de x e y foram encontradas as seguintes estatísticas para cada algoritmo:

| Algoritmo | Mínimo | Máximo | Média | Desvio-Padrão |
|---|---|---|---|---|
| HC | -831.23969251234 | 376.223481003286 | -202.43467755016 | 268.472472762431 |
| ILS | -888.94912520791 | -63.872552746297 | -475.01541503428 | 199.541654755927 |

Boxplots:



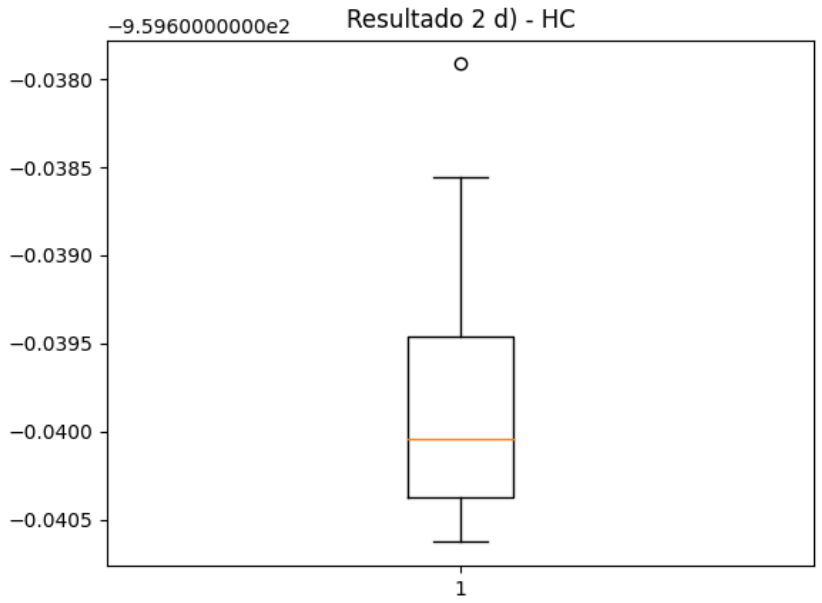



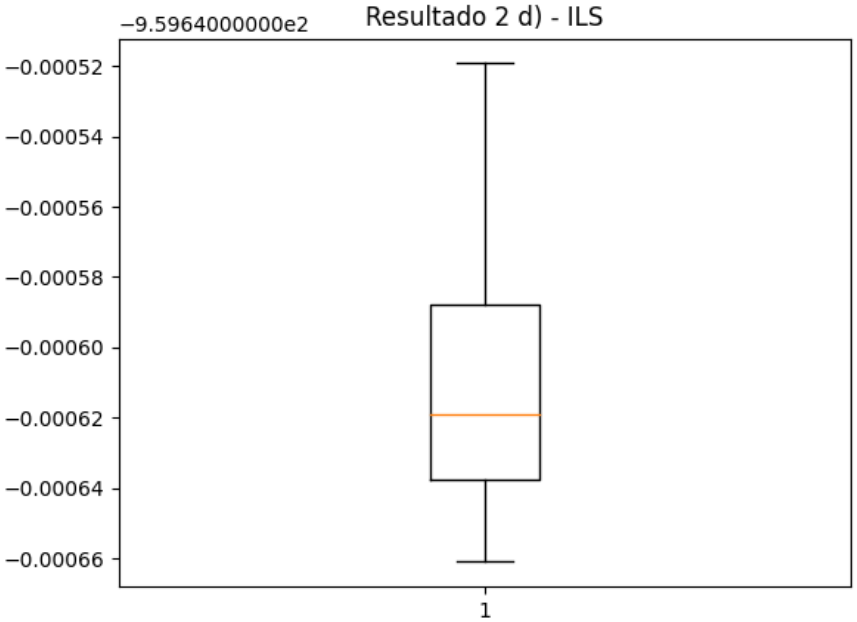
d) Para $511 \leq x \leq 512$ e $404 \leq y \leq 405$ :

Para esses intervalos de x e y foram encontradas as seguintes estatísticas para cada algoritmo:

| Algoritmo | Mínimo | Máximo | Média | Desvio-Padrão |
|---|---|---|---|---|
| HC | -959.64062765709 | -959.63791351343 | -959.63983202578 | 0.00068492299310 |
| ILS | -959.64066082173 | -959.64051931244 | -959.64060590492 | 4.05219895621e-5 |

Boxplots:



## Conclusão

A partir dos resultados e tendo em mente a visualização tridimensional dos gráficos das funções objetivo disponibilizados na descrição do trabalho, é possível afirmar que para o exercício 1 (em ambas as letras) o HC e o ILS tiveram resultados próximos quanto a máximos, mínimos e desvio padrão, apresentando uma pequena diferença apenas na média dos valores encontrados. Esse comportamento pode se justificar devido ao gráfico da função objetivo possuir apenas duas bacias, dessa forma assim como o ILS, o HC também gerará bons resultados. Já no exercício 2, o gráfico tridimensional apresenta diversas bacias e esse pode ser o motivo pelo qual os resultados da letra C apresentaram resultados diferentes significativamente em todas as quatro estatísticas em cada algoritmo. Na letra D os resultados permaneceram relativamente próximos e isso pode ser devido ao valor ser um -959 ser um mínimo predominante entre as suas bacias vizinhas.

## Referências


BROWNLEE, Jason. **Stochastic Hill Climbing in Python from Scratch**. Disponível em: https://machinelearningmastery.com/stochastic-hill-climbing-in-python-from-scratch/. Acesso em: 31 ago. 2021.

BROWNLEE, Jason. **Iterated Local Search From Scratch in Python**. Disponível em: https://machinelearningmastery.com/iterated-local-search-from-scratch-in-python/. Acesso em: 31 ago. 2021.