Laboratório 3
Otimização com Métodos de Busca Local

CT-213
Prof. Marcos Ricardo Omena de Albuquerque Maximo

Carlos Matheus Barros da Silva
carlosmatheusbs@gmail.com

Março - 2019

Introdução

Foram estudados e implementados três métodos de otimização que possuem algoritmos baseados em busca local. Esses métodos foram Gradient Descent, Hill Climbing e Simulated Anneling.

Os métodos foram testados em um problema proposto no roteiro dessa atividade[]. Esse problema utilizou regressão linear para ober parâmetros físicos relativos ao movimento de uma bola.

Como o problema tratado possui solução analítica, o experimento é a titulo educacional, já que, dessa forma, pode-se obter facilmente sua solução pelo Método dos Mínimos Quadrados (MMQ).

De maneira geral, o Laboratório foi desenvolvido com sucesso. Em cada uma das sessões seguintes encontram-se o respectivo desenvolvimento de um dos métodos estudado.

Gradient Descent

Explicação 1. *De maneira simplificada possvel reduzir o funcionamento de todas as classes dos estados implementadas funo execute e funo checktransition Em execute sero setados os parmetros no agente para que ele execute a ao do estado corretamente enquanto que no checktransition ser verificado se determinada condio para mudana de estado atendida e se for o estado mudar A construo dos estados foi feita de acordo com o Esquema 1 Todos as classes que representam os estados da mquina de estados finitos foram completadas com sucesso e seus cdigos esto definidos do Cdigo 1 ao Cdigo 4 Os cdigos Cdigo 5 e Cdigo 6 se referem a funes auxiliares criadas e usada nas classes dos estados As imagens mostradas da Figura 1 Figura 5 denotam o comportamento do rob bem como suas transies de estados.*

```
1       algorithm = "gradient_descent"
2
3       theta = theta0
4       history = [theta0]
5       i = 0
6       while not check_stopping_condition(
        max_iterations, i, cost_function,
        epsilon, theta):
7           theta = theta - alpha *
        gradient_function(theta)
8           history.append(theta)
9           i += 1
10
11      create_history_file(history, algorithm)
12
13      return theta, history
```

Código 1: Interior da função *Gradient Descent*.

```
1 def create_history_file(history, algorithm):
2     f = open(algorithm + ".txt", "w+")
3     for theta in history:
4         f.write(str(theta)+"\n")
```

Código 2: Código para criação do arquivo com o hitórico de *theta*.

```
1 def check_stopping_condition(max_iterations,
      i, cost_function, epsilon, theta):
2     return i > max_iterations or
      cost_function(theta) < epsilon
```

Código 3: Código para checagem da condição de parada.

```
[0. 0.]
[0.03584969 0.02468302]
[0.06628493 0.04489148]
[0.09217881 0.06135615]
[0.11426295 0.07468929]
[0.13315044 0.08540385]
[0.14935503 0.09392958]
[0.16330723 0.10062648]
[0.17536785 0.10579616]
[0.18583924 0.10969127]
[ 0.43337044 -0.10101818]
[ 0.43337047 -0.10101822]
[ 0.43337049 -0.10101825]
[ 0.43337052 -0.10101828]
[ 0.43337055 -0.10101831]
[ 0.43337057 -0.10101834]
[ 0.4333706  -0.10101837]
[ 0.43337062 -0.1010184 ]
[ 0.43337065 -0.10101843]
[ 0.43337067 -0.10101846]
[ 0.4333707  -0.10101849]
```

Código 4: Histórico de *theta* para o algoritmo *Gradient Descent*

Discussão 1. O resultado foi obtido dessa forma devido ao fato de que a uma das primeiras coisas que o programa pai faz é chamar *fork()* de modo que logo em seguida seu filho chama *fork()* também. Então ficaram os 3 processos rodando paralelamente, de modo que cada um desses 3 começaram a fazer o que estava descrito no enunciado do problema.

Dessa forma, como pôde ser visto na saída do problema, os três processos estavam executando paralelamente. Como cada um desses 3 processos *printava* "simultaneamente" foi visto que o Neto foi *printado* ante do filho, o que é algo possível, já que como todos estão rodando mais ou menos na mesma posição o que chamar a função *printf* primeiro seria *printado* antes.

Os três processos ficaram em seus ciclos *printando* de acordo com o previsto, até que eles morreram. O pai, como era previsto, morreu primeiro, já que ele havia começado a *printar* antes. Logo em seguida, o filho e o neto também terminaram.

Problema 2

Enunciado 1. *Altere o programa 1 para que primeiro sejam exibidos primeiramente apenas os*

prints do neto, depois so os do filho e por ultimo so os do pai.

Resposta 1. O Problema 2 foi resolvido com sucesso. Seu código pode ser conferido no Código **??**. Seu funcionamento pode ser observado pela saída representada pelo Código **??**. As respostas às perguntas referentes a esse problema podem ser vistas na Discussão 2.

Discussão 2. A alteração necessária para a mudança requisitada foi bem simples e direta, já que o que foi pedido foi que rodasse o neto antes do filho, bastava fazer o filho esperar terminar a execução do neto para então executar o que ele ia fazer, e como foi pedido também que o filho rodasse antes do pai, bastava que o pai, também, esperasse o termino da execução do filho para que então executasse o que ia executar.

Portanto foi necessário apenas acrescentar duas linhas com a função *wait()*, a Linha 34 e a Linha 40.