Relatório do Laboratório 3: Otimização com Métodos de Busca Local

Isabelle Ferreira de Oliveira
CT-213 - Engenharia da Computação 2020
Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

***Resumo*—Esse relatório documenta a implementação dos seguintes algoritmos de otimização baseados em busca local: Descida do Gradiente, Hill Climbing e Simulated Annealing. Esses métodos foram testados em uma regressão linear para obter parâmetros físicos relativos ao movimento de uma bola. Por fim, os resultados das três implementações foram comparados.**

***Index Terms*—Algoritmos de Otimização, busca local, Descida do Gradiente, Hill Climbing, Simulated Annealing, regressão linear**

I. Introdução

Otimização consiste em encontrar o mínimo (ou máximo) de uma função, ou seja, encontrar o conjunto de parâmetros que levem essa função ao seu mínimo (ou máximo). Também pode ser visto como encontrar a melhor solução dentre todas as soluções viáveis. Nesses problemas de otimização também é possível haver restrições acerca dos parâmetros que serão analisados.

Quando não se é possível encontrar essa solução de forma analítica, podem ser utilizados métodos numéricos. Dentre esses, estão algoritmos da classe de Metaheurísticas, que testam um número alto de soluções, conduzindo seus testes de forma a convergirem em alguma solução otimizada.

Um exemplo famoso de solução analítica é o Método dos Mínimos Quadrados (MMQ). Já exemplos de algoritmos da classe de Metaheurísticas são o Descida do Gradiente, o Hill Climbing e o Simulated Annealing.

Os pseudo-códigos desses algoritmos de Metaheurísticas podem ser vistos nas subseções a seguir. Em seguida, será apresentado como esses algoritmos foram implementados no contexto do laboratório.

A. Descida do Gradiente

```
def gradient_descent(dJ, theta, alpha):
  while not check_stopping_condition():
    theta = theta - alpha * dJ(theta)
  return theta
```

B. Hill Climbing

```
def hill_climbing(J, theta):
  while not check_stopping_condition():
    best = None # J(None) = inf
    for neighbor in neighbors(theta):
      if J(neighbor) < J(best):
        best = neighbor
    if J(best) > J(theta):
      return theta
    theta = best
  return theta
```

C. Simulated Annealing

```
def simulated_annealing(J, theta):
  while not check_stopping_condition():
    T = temperature_schedule(i)
    if T < 0.0:
      return theta
    neighbor = random_neighbor(theta)
    deltaE = J(neighbor) - J(theta)
    if deltaE < 0:
      theta = neighbor
    else:
      r = random_uniform(0.0, 1.0) # Draws
          random number w/ uniform dist.
      if r <= exp(-deltaE / T):
        theta = neighbor
  return theta
```

II. Implementação dos algoritmos

Na parte relativa a implementação dos algoritmos de busca, era necessário preencher os códigos das funções *dijkstra*, *greedy* e *a_star* da classe *PathPlanner* do código base fornecido [1].

A análise de vários pontos dos algoritmos descritos acima terão uma breve descrição em alto nível da sua implementação a seguir.

Como os algoritmos foram implementados em Python, as filas de prioridade foram importadas da biblioteca *heapq*, sendo utilizado as funções *heappush* e *heappop* para adicionar e retirar um elemento da fila, respectivamente.

Já as funções heurísticas dos algoritmos Greedy e A* foram calculadas a partir da função *distance_to*, que calcula a distância euclidiana entre um nó específico e o nó objetivo. Essa função já foi fornecida pelo código básico.

Na etapa de percorrer os sucessores de um determinado nó para se calcular os custos relativos aquele caminho analisado, utilizou-se a função *get_successors* para se obter esses sucessores. Para cada nó analisado, então, seu nó sucessor foi

adicionado à fila de prioridade, sendo o custo dependente se o movimento se deu pela diagonal ou pela lateral do nó do *grid*. Dessa forma, passos laterais tiveram custos menores que passos diagonais, na proporção de $1{:}\sqrt{2}$. Uma representação de um nó específico do grafo e seus 8 vizinhos foi apresentado na Figura **??**.

Quando um determinado nó era, por fim, retirado da fila de prioridade, o custo mínimo do início do trajeto até aquele ponto já estaria definido. Nesse instante, então, o atributo *closed* daquele nó foi setado como *True*. Ao nó com custo definido de caminho do início até ele for o nó objetivo, a busca encerrou-se. No algoritmo Greedy, excepcionalmente, o custo já estava definido desde a primeira vez que o nó em específico era visitado.

III. Resultados e Conclusões

Os resultados das trajetórias de otimização obtidos após a execução das implementações dos algoritmos descritos acima foram apresentados nas Figuras 1, 2 e 3 para Descida do Gradiente, Hill Climbing e Simulated Annealing, respectivamente, e a sobreposição dessas trajetórias foi apresentada na Figura 4 para melhor comparação visual.

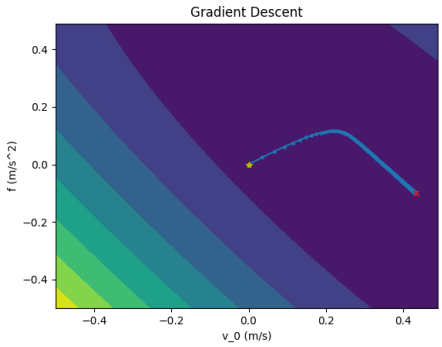

Figura 1. Trajetória de otimização usando Descida do Gradiente.

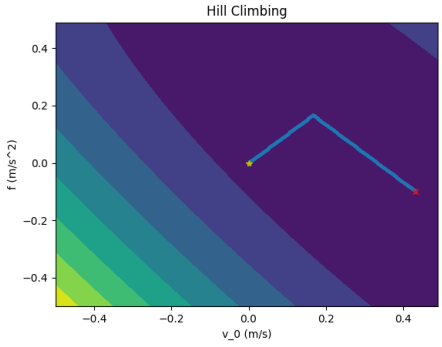

Figura 2. Trajetória de otimização usando Hill Climbing.

Aplicando os valores encontrados para os parâmetros v_0 e f, o gráfico de velocidade da bola por tempo foi apresentado na Figura 5. Esses valores numéricos de v_0 e f podem ser vistos na Tabela I para cada método estudado nesse laboratório, além

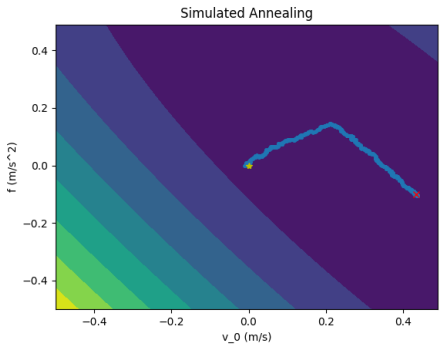

Figura 3. Trajetória de otimização usando Simulated Annealing.

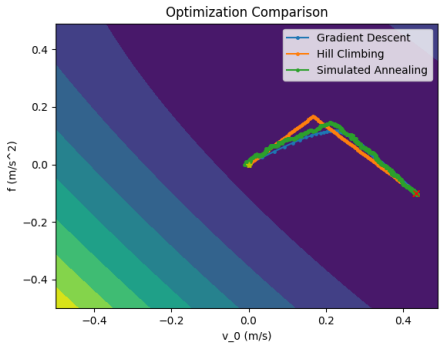

Figura 4. Comparação de trajetórias de otimização usando Descida do Gradiente, Hill Climbing e Simulated Annealing.

do resultado fornecido inicialmente para o método de Mínimos Quadrados.

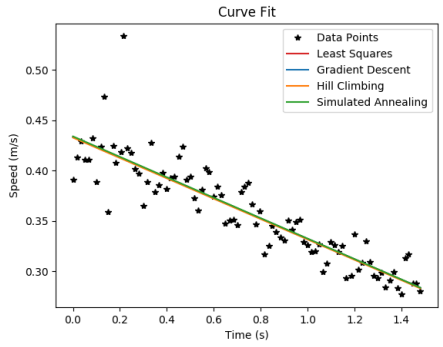

Figura 5. Comparação das regressões lineares das otimizações usando Descida do Gradiente, Hill Climbing e Simulated Annealing.

Para os três casos, os algoritmos foram parados pelo critério de quantidade de iterações. Foram executadas, então, 1000 iterações para Descida do Gradiente e Hill Climbing, e 5000 para Simulated Annealing.

Tendo em vista o que foi apresentado, pode-se notar, por fim, que esses algoritmos realmente se demonstraram eficazes em encontrar parâmetros otimizados para uma determinada função de custo e um ponto inicial de partida.

Tabela I
COMPARAÇÃO DAS SOLUÇÕES ENCONTRADAS PARA OS PARÂMETROS FÍSICOS DA BOLA PARA OS ALGORITMOS IMPLEMENTADOS NO LABORATÓRIO.

Algoritmo	**Solução**	
	v_0	f
Mínimos Quadrados	0.43337277	-0.10102096
Descida do Gradiente	0.4333707	-0.10101849
Hill Climbing	0.43274935	-0.10099495
Simulated Annealing	0.43397656	-0.10134529

REFERÊNCIAS

[1] M. Maximo, "Roteiro: Laboratório 3 - Otimização com Métodos de Busca Local". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.