Comparação de Buscas Informadas - Uma Análise Comparativa dos Algoritmos A-estrela e Busca Gulosa

Getúlio Santos Mendes, João Gustavo Silva Guimarães, João Pedro Freitas de Paula Dias

11 de dezembro de 2024

1 Problematica

Esse trabalho tem como objetivo a implementação e comparação dos algoritmos de busca em grafos: A-estrela (A*) e Busca Gulosa. A partir da posição **U** no labirínto e chegar até a posição **E** passando por um caminho construído sob as regras de caminhamento de duas buscas propostas, neste caso **A*** e **Busca Gulosa**. O grafo que representa o labirinto foi modelado da seguinte forma

Figura 1: Labirinto

2 Descrição dos Algoritmos Implementados

Com base no conceito apresentado por Russell e Norvig (2021), "As buscas são uma maneira de seguir opções, deixando outras opções em aberto para mais tarde, caso a escolha inicial não leve a uma solução", os algoritmos A* e Busca Gulosa são classificados como algoritmos de busca informada, pois utilizam uma heurística para guiar a decisão sobre qual caminho seguir. A heurística fornece informações adicionais sobre a proximidade do objetivo, permitindo que esses algoritmos explorem o espaço de nós de maneira melhor direcionada.

Embora a Busca Gulosa não garanta a obtenção de uma solução ótima, ela é capaz de encontrar uma solução viável ao priorizar os caminhos que parecem mais promissores com base na função heurística **h(n)**, ou seja, os caminho que possuem menor custo. Por outro lado, o **A***, ao equilibrar o custo real acumulado **g(n)** e a estimativa heurística **h(n)**, assegura a obtenção de uma solução ótima desde que a heurística seja admissível. Ambos algoritmos seguem o princípio de explorar as opções mais promissoras primeiro, mas o A* é mais robusto devido à sua consideração do custo total do caminho.

2.1 Busca Gulosa

A Busca Gulosa tem base em sempre ter a melhor escolha local, com base na problemática abordada, o algoritimo guloso sempre vai optar pelo menor custo heuristico **h(n)** possível. Sendo assim, esse metodo garante uma solução viável, porém não garante a otimalidade da solução. Para melho entendimento da implementação, tem-se o pseudo código a seguir:

Algorithm 1 GBF (Busca Gulosa)

```
1: procedure GBF(Graph, Start, Goal)
2:     OpenList ← {}
3:     Add (ManhattanHeuristic(Start, Goal), Start) to OpenList
4:     CameFrom ← empty map
5:     Visited ← empty set
6:     while OpenList is not empty do
7:         Remove the element with lowest heuristic from OpenList and set as Current
8:         if Current = Goal then
9:             Path ← empty list
10:            while Current is in CameFrom do
11:                Add Current to Path
12:                Current ← CameFrom[Current]
13:            end while
14:            Add Start to Path
15:            return Path (reversed)
16:        end if
17:        if Current is in Visited then
18:            continue
19:        end if
20:        Add Current to Visited
21:        for each Neighbor in Graph[Current] do
22:            if Neighbor is not in Visited then
23:                CameFrom[Neighbor] ← Current
24:                Add (ManhattanHeuristic(Neighbor, Goal), Neighbor) to OpenList
25:            end if
26:        end for
27:    end while
28:    return None
29: end procedure
```

2.2 A*

A Busca A* é um algoritmo de busca informada que combina o custo acumulado g(n) com o custo heurístico estimado h(n) para priorizar a exploração de caminhos promissores. Diferentemente da Busca Gulosa, o A* busca balancear eficiência e otimalidade, garantindo a solução mais curta sempre que a heurística utilizada for admissível e consistente. Dessa forma, o algoritmo é amplamente aplicado em problemas que exigem soluções ótimas em grafos. Para melhor entendimento da implementação, tem-se o pseudocódigo a seguir:

Algorithm 2 A* (Busca A Estrela)

```
1: procedure A*(Graph, Start, Goal)
2:     OpenList ← empty priority queue
3:     Add (0 + Heuristic(Start, Goal), Start, 0) to OpenList
4:     CameFrom ← empty map
5:     GScore ← map with all nodes initialized to ∞
6:     GScore[Start] ← 0
7:     Visited ← empty set
8:     while OpenList is not empty do
9:         Remove the element with the lowest priority from OpenList and set as Current
10:        if Current = Goal then
11:            Path ← empty list
12:            while Current is in CameFrom do
13:                Add Current to Path
14:                Current ← CameFrom[Current]
15:            end while
16:            Add Start to Path
17:            return Path (reversed)
18:        end if
19:        if Current is in Visited then
20:            continue
21:        end if
22:        Add Current to Visited
23:        for each Neighbor in Graph[Current] do
24:            TentativeGScore ← CurrentCost + Cost(Current, Neighbor)
25:            if TentativeGScore < GScore[Neighbor] then
26:                CameFrom[Neighbor] ← Current
27:                GScore[Neighbor] ← TentativeGScore
28:                FScore ← TentativeGScore + Heuristic(Neighbor, Goal)
29:                Add (FScore, Neighbor, TentativeGScore) to OpenList
30:            end if
31:        end for
32:    end while
33:    return None
34: end procedure
```

3 Metodologia

Para medir a performance foi-se realizado a média de 100 execuções do problema e calculado o desvio padrão para a melhor análise dos resultados. Para medir a utilização de memória foi-se utilizado a bliblioteca *tracemalloc* do python e para as medidas de desempenho a bliblioteca *time*.

Para a medidas foi-se usado um computador com um i5-13420H.

4 Resultados das Medições de Desempenho

Para obter as métricas de tempo de execução e consumo de memória, as bibliotecas time e tracemalloc do Python foram utilizadas. Deste modo foi percebido que os algoritmos obtiveram tempo menor que o de 1 segundo para serem executados, mostrando maior diferença em relação à quantidade de memória armazenada.

Sendo assim, o **Busca Gulosa** Realizou o caminho $U \rightarrow V \rightarrow Q \rightarrow L \rightarrow M \rightarrow N \rightarrow I \rightarrow H \rightarrow C \rightarrow D \rightarrow E, tempodeexecuomdio$: (0.000044 ± 0.000012) segundos, com uma memória inicial alocada de $0.09KB$ e um pico de $1.38KB$.

Equanto o **A*** trilhou $U \rightarrow V \rightarrow Q \rightarrow L \rightarrow M \rightarrow N \rightarrow I \rightarrow H \rightarrow C \rightarrow D \rightarrow E, tempodeexecuomdio$: (0.000072± 0.000009) segundos com uma memória utilizada média de $0.10KB$ e apresentando um pico de $2.18KB$.

Tempo de execução: Os tempos de execução foram próximos, com o A* mais lento, em geral, o tempo de execução dos algoritmos nesse problema e nesse caminho favorece a busca gulosa por encontrar o caminho optimo sem uma mair complexidade de calculos que o A* possui. Isso não necessariamente quer dizer que a busca gulosa será mais performático que o A* em outras situações, pois pode encontrar um caminho maior e ter que executar mais vezes que o A*.

Completude: À respeito da completude é importante ponderar que ambos os algoritmos são capazes de encontrar a solução para o problema de busca proposto que é encontrar um caminho entre o ponto inicial e final. Sendo assim, amblos são completos.

Optimalidade: A Busca Gulosa não garante encontrar o caminho mais curto, mesmo tendo encontrado-o neste caso. Como ela ignora o custo acumulado g(n)g(n), pode optar por caminhos com um bom valor heurístico, mas que levam a soluções subótimas. Já o A* sempre será optimo se a heuristica utlizada for apropriada para o problema e o grafo não possua arestas negativas.

Consumo de Memória: Dadas as métricas acima compreende-se a Busca Golosa como o algoritmo que consumiu menos memória no problema proposto. A justificativa para o A* ser mais custoso em memória, se traduz na sua necessidade em armazenar mais nós simultaneamente que a Busca Golosa. Enquanto a Busca golosa precisa armazenar menos nós pois seu foco é na melhor escolha local, no A* todos os nós da fila serão tentativas.

5 Referências

CORMEN, T. H. et al. Introduction to Algorithms, third edition. [s.l.] MIT Press, 2009. Acessado em 20 de Março de 2023.

Russel, Stuart, and Peter Norvig. Inteligência Artificial. 3rd ed., Elsevier Editora Ltda., 2013.