Relatório de Experimentos - Inteligência Artificial

Antonio Joabe Alves Morais, Iarley Natã Lopes Souza
& João Pedro Soares Matias

January 2025

1 Introdução

Iremos comentar a respeito dos dados gerados pelas execuções dos experimentos descritos no documento de entrega do trabalho.

2 Implementação

2.1 Linguagem Utilizada

Utilizamos Python em toda a implementação.

2.2 Bibliotecas

Na verdade, importamos módulos contendo estruturas de dados que utilizamos, como Queue; PriorityQueue; Type - Callable; Random - Shuffle; Random - Randint; Math.

2.3 Rodando Algoritmos

Para executar um algoritmo específico com um nó inicial e um nó objetivo definidos, utilize o comando:

```
python test.py
```

Siga o fluxo do código conforme solicitado.
Alternativamente, para executar todos os algoritmos com uma entrada fornecida pelo usuário, compile o arquivo principal com o seguinte comando:

```
python main.py
```

Em seguida, execute o experimento 0, acompanhando o fluxo do código conforme indicado.

3 Experimentação

Esta seção discute os resultados obtidos dos experimentos requeridos pelo documento de descrição do trabalho. Estes se encontram na pasta `experiments_output` e são:

```
experiment_0.txt
experiment_1.txt
experiment_2.txt
experiment_3.txt
experiment_4.txt
experiment_5.txt
```

3.1 Parte 1

3.1.1 Análise dos Algoritmos: BFS, DFS e UCS

Os resultados obtidos no *Experiment 1* evidenciaram as diferenças entre os três algoritmos abordados: Busca em Largura (BFS), Busca em Profundidade (DFS) e Busca de Custo Uniforme (UCS).

3.1.2 Busca em Largura (BFS)

A BFS mostrou-se eficiente na busca pelo caminho de menor custo em grafos não ponderados, especialmente ao utilizar a função de custo C_1 e até a C_2. No entanto, sua eficiência diminuiu em grafos com arestas ponderadas, pois não considera os custos das arestas ao expandir nós. Nesse contexto, o algoritmo acabou por explorar caminhos subótimos, aumentando o custo total.

Resultados de BFS (Exemplos principais):

- **Iteração 1:**

  Função de custo C_1: custo do caminho = 200, nós gerados = 1230, nós visitados = 317.

  Função de custo C_2: custo do caminho = 70, nós gerados = 340, nós visitados = 85.

- **Iteração 4:**

  Função de custo C_3: custo do caminho = 430, nós gerados = 170, nós visitados = 43.

  Função de custo C_4: custo do caminho = 199, nós gerados = 1452, nós visitados = 373.

3.1.3 Busca de Custo Uniforme (UCS)

A UCS, por outro lado, utiliza uma fila de prioridade para garantir que sempre expanda o nó com o menor custo acumulado. Isso assegura que ele encontre o caminho de menor custo, independentemente do peso das arestas. No pior caso, seu desempenho em termos de nós visitados e custo total foi equivalente ao da BFS, mas sua capacidade de otimização o torna superior em grafos ponderados, sendo o algoritmo mais adequado para nosso ambiente de teste em relação aos dois.

Resultados de UCS (Exemplos principais):

- **Iteração 1:**

  Função de custo C_1: custo do caminho = 200, nós gerados = 1423, nós visitados = 367.

  Função de custo C_2: custo do caminho = 205, nós gerados = 1013, nós visitados = 263.

- **Iteração 4:**

  Função de custo C_3: custo do caminho = 200, nós gerados = 1606, nós visitados = 416.

  Função de custo C_4: custo do caminho = 195, nós gerados = 1891, nós visitados = 488.

3.1.4 Busca em Profundidade (DFS)

Embora a DFS não garanta o caminho de menor custo, destacou-se por gerar e visitar menos nós do que os outros dois algoritmos. com exceções de buscas com nós relativamente perto mas em níveis diferentes. Isso a torna eficiente em termos de exploração de nós, mas menos eficaz em contextos onde o custo do caminho é crítico.

Resultados de DFS (Exemplos principais):

- **Iteração 1:**

  Função de custo C_1: custo do caminho = 200, nós gerados = 81, nós visitados = 20.

  Função de custo C_2: custo do caminho = 205, nós gerados = 81, nós visitados = 20.

- **Iteração 4:**

  Função de custo C_3: custo do caminho = 433, nós gerados = 170, nós visitados = 43.

  Função de custo C_4: custo do caminho = 430, nós gerados = 170, nós visitados = 43.

3.1.5 Analogia

Uma analogia para ilustrar os três algoritmos seria considerar motoristas de aplicativos durante horários de pico:

- **BFS**: Um motorista que conhece bem a cidade, mas segue rotas desatualizadas, podendo enfrentar ruas bloqueadas ou engarrafamentos inesperados.
- **DFS**: Um motorista recém-habilitado que segue rigorosamente o GPS, explorando menos ruas, mas suscetível a ficar preso no trânsito em horários de pico.
- **UCS**: Um motorista experiente, que conhece todos os atalhos e sempre encontra o caminho mais eficiente, chegando ao destino com o menor custo de tempo.

3.1.6 Conclusão

A análise demonstra que o UCS é a melhor escolha para situações em que o custo das arestas varia, garantindo a solução mais eficiente em termos de custo. Outrora, se o custo não é o requisito principal, DFS pode se destacar neste ambiente por gerar e visitar menos nós, enquanto a BFS se equivaleria ao UCS em um contexto de arestas não ponderadas com um custo computacional menor devido a heap inútil no UCS.

3.2 Parte 2

3.2.1 Análise dos Algoritmos: UCS e A*

Os resultados do *Experiment 2* destacaram a eficiência dos algoritmos UCS (Uniform-Cost Search) e A* em diferentes cenários com custo de caminho variável e heurísticas distintas.

3.2.2 Busca de Custo Uniforme (UCS)

O UCS demonstrou ser robusto e confiável, encontrando sempre o caminho de menor custo, independentemente da função de custo utilizada. No entanto, ele gerou e visitou mais nós em comparação com o A* em vários cenários.

Resultados de UCS (Exemplos principais):

- **Iteração 1:**

  Função de custo C_1: custo do caminho = 340, nós gerados = 2596, nós visitados = 669.

  Função de custo C_4: custo do caminho = 293, nós gerados = 3106, nós visitados = 796.

- **Iteração 4:**

  Função de custo C_2: custo do caminho = 430, nós gerados = 2672, nós visitados = 686.

Nos casos analisados, é evidente que o UCS é eficaz em contextos onde o custo do caminho é crítico e não pode ser comprometido.

3.2.3 Busca Heurística A*

A Busca A* destacou-se por combinar o custo acumulado e uma função heurística (Euclidiana ou Manhattan), o que resultou em uma redução significativa de nós gerados e visitados em relação ao UCS

Resultados de A* (Exemplos principais):

- **Iteração 1:**

  Heurística Euclidiana (H_1) com C_1: custo do caminho = 340, nós gerados = 2503, nós visitados = 645.

  Heurística Manhattan (H_2) com C_4: custo do caminho = 293, nós gerados = 2696, nós visitados = 690.

- **Iteração 4:**

  Heurística Euclidiana (H_1) com C_3: custo do caminho = 355, nós gerados = 3064, nós visitados = 785.

O uso da heurística permitiu ao A* equilibrar custo e exploração de forma mais eficiente, especialmente em grafos onde a estrutura permitia estimativas precisas.

3.2.4 Conclusão

A* foi consistentemente mais eficiente em termos de nós gerados e visitados, particularmente quando utilizou a heurística Euclidiana. UCS, embora mais conservadora, garantiu resultados ótimos em todos os casos.

UCS é mais adequado quando a precisão no custo do caminho é prioridade. A* é ideal para grafos grandes, onde a eficiência em termos de recursos computacionais é crucial, e uma boa heurística está disponível.

Ambos os algoritmos são altamente eficazes, mas o A* tem uma ligeira vantagem em termos de eficiência computacional, desde que a função heurística seja bem definida e apropriada para o grafo em questão.

3.3 Parte 3

3.3.1 Análise dos algoritmos GBFS x A*

Os resultados obtidos do *Experiment 3* destacaram a eficiência dos algoritmos GBFS e A* em diferentes cenários com custo de caminho variável e heurísticas distintas.

3.3.2 Greedy Best-First Search (GBFS)

O Greedy Best-First Search (GBFS) é um algoritmo de busca que se baseia exclusivamente na heurística para determinar a expansão dos nós, priorizando aqueles que parecem estar mais próximos da meta. Quando utilizada uma heurística precisa, como a euclidiana, o GBFS mostrou-se eficiente, apresentando desempenho equivalente ao A* em termos de custo com as funções C_1 e C_2, gerando muito menos nós. Isso se deve à sua abordagem focada na velocidade, ignorando o custo acumulado e focando na expansão rápida de nós promissores, o que reduz o número total de nós gerados e visitados. Mesmo com a heurística de Manhattan, a diferença no seu comportamento não foi tão notória, para variações de custo mínimas como em C_1 e C_2 o custo total foi equivalente a A*, já em condições de custos variados, ele não provou ser capaz de encontrar um caminho ótimo, com ambas as heurísticas.

Resultados de GBFS (Exemplos principais):

- **Iteração 1:**

  Heurística Euclidiana (H_1) com C_1: custo do caminho = 180, nós gerados = 72, nós visitados = 19.

  Heurística Manhattan (H_2) com C_4: custo do caminho = 180, nós gerados = 72, nós visitados = 19.

- **Iteração 4:**

  Heurística Euclidiana (H_1) com C_1: custo do caminho = 350, nós gerados = 141, nós visitados = 36.

  Heurística Manhattan (H_2) com C_4: custo do caminho = 342, nós gerados = 141, nós visitados = 36.

3.3.3 Algoritmo A*

Por outro lado, o Algoritmo A* combina a heurística com o custo acumulado para encontrar o caminho de menor custo total até a meta, o que o torna especialmente eficaz em cenários com custos variados, como nas funções C_3 e C_4. A capacidade do A* de balancear a heurística com o custo real garante que ele encontre a solução mais eficiente, ainda que gere mais nós do que o GBFS. Com a heurística de Manhattan, o A* apresentou uma leve vantagem na quantidade de nós gerados, reforçando a adequação desta heurística para ambientes com movimentação restrita a direções ortogonais, que é o ambiente que estamos trabalhando. Apesar de sua maior complexidade computacional, o A* garante a obtenção do caminho de menor custo independente da função utilizada.

Resultados de A* (Exemplos principais):

- **Iteração 1:**

Heurística Euclidiana (H_1) com C_1: custo do caminho = 180, nós gerados = 1.039, nós visitados = 269.

Heurística Manhattan (H_2) com C_4: custo do caminho = 180, nós gerados = 1.252, nós visitados = 269.

- **Iteração 4:**

  Heurística Euclidiana (H_1) com C_1: custo do caminho = 350, nós gerados = 2.972, nós visitados = 765.

  Heurística Manhattan (H_2) com C_4: custo do caminho = 305, nós gerados = 3.539, nós visitados = 905.

3.3.4 Conclusão

Falando primeiramente das heurísticas, tanto euclidiana quanto manhattan mantiveram os resultados equivalentes quando utilizadas pelo GBFS, a diferença foi mais notória quando utilizadas pelo A*, onde manhattan obteve uma singela vantagem na geração de nós, graças novamente à ortogonalidade da heurística. Em questão de desempenho, o experimento seguiu o que se esperava, para arestas não ponderadas ou com pouca variação como C_1 e C_2, GBFS se demonstrou superior, já para arestas de custos variados como C_3 e C_4, A* levou vantagem por garantir caminhos com custos menores, coisa que GBFS não consegue garantir. Vale destacar que a diferença de nós gerados foi gritante com vantagem para GBFS, mesmo que não tenha a certeza de encontrar caminhos ótimos, ele é mais viável quando se busca soluções rápidas, principalmente em cenários de custo uniforme, nessas condições ele, além de garantir o caminho mínimo, gera e, consequentemente, visita menos nós, além de ter um custo computacional bem inferior. Já o A*, por mais que gere mais nós e seja mais lento, é a escolha preferida em situações onde a minimização do custo total é crítica.

3.4 Parte 4

3.4.1 Busca em Largura e Profundidade com Randomização

Este experimento teve como objetivo observar o comportamento dos algoritmos de Busca em Largura (BFS) e Busca em Profundidade (DFS) ao introduzir randomização na ordem de geração de vizinhos. Esse procedimento foi realizado através de um embaralhamento aleatório dos operadores de vizinhança, permitindo variabilidade nas execuções dos algoritmos.

3.4.2 Busca em Largura com Randomização

Os resultados indicaram que a BFS, mesmo com a randomização, continuou garantindo o menor custo de caminho entre os pares de coordenadas inicial e final. Apesar disso, o impacto da randomização levou a um aumento no número de nós gerados e visitados, quando comparado à execução tradicional, devido à expansão de nós em ordens distintas.

Exemplo de resultados BFS

- **Iteração 1:**

  Custo do caminho (C_1): 390, Nós gerados: 3241, Nós visitados: 833.

  Custo do caminho (C_4): 378, Nós gerados: 3241, Nós visitados: 833.

- **Iteração 5:**

  Custo do caminho (C_1): 390, Nós gerados: 3221, Nós visitados: 828.

  Custo do caminho (C_4): 383, Nós gerados: 3221, Nós visitados: 828.

3.4.3 Busca em Profundidade com Randomização

A DFS apresentou uma variabilidade significativa nos custos de caminho e no número de nós gerados e visitados devido à natureza exploratória e dependente da ordem dos nós gerados. Em diversos casos, os caminhos encontrados não foram os mais ótimos, refletindo a limitação da DFS em priorizar a eficiência do custo total.

Exemplo de resultados DFS

- **Iteração 1:**

  Custo do caminho (C_1): 1930, Nós gerados: 939, Nós visitados: 247.

  Custo do caminho (C_4): 1935, Nós gerados: 939, Nós visitados: 247.

- **Iteração 5:**

  Custo do caminho (C_1): 2550, Nós gerados: 1096, Nós visitados: 286.

  Custo do caminho (C_4): 2565, Nós gerados: 1096, Nós visitados: 286.

3.4.4 Conclusão

A randomização permitiu observar a influência da ordem de geração de vizinhos no comportamento dos algoritmos BFS e DFS. A BFS manteve-se eficiente em termos de custo, mas teve um aumento no número de nós gerados e visitados devido ao embaralhamento. Por outro lado, a DFS apresentou maior variabilidade nos resultados, com custos mais elevados e menos consistência na otimização do caminho encontrado.

Portanto, a BFS continua sendo a escolha preferida para situações que demandam soluções de custo mínimo, enquanto a DFS pode ser considerada em casos onde a eficiência de nós gerados seja mais importante do que a otimalidade do caminho.

3.5 Parte 5

3.5.1 Análise do algoritmo A* com uma parada a mais

Os resultados obtidos do *Experiment 5* destacaram a eficiência do algoritmo A* com uma parada a mais em diferentes cenários, com custo de caminho variável e heurísticas distintas.

3.5.2 Algoritmo A*

No contexto do *Experiment 5*, o A* mostrou-se capaz de encontrar caminhos otimizados que passam por pelo menos uma parada intermediária, mantendo um bom equilíbrio entre o custo total do caminho e a heurística aplicada. Comparando os nós gerados no *Experiment 0* (sem parada) com os do *Experiment 5* (com parada), observou-se que o custo adicional do caminho, apesar de maior, não adquiriu uma diferença tão obstante, destacando a eficiência do A* em minimizar custos mesmo com a exigência de uma parada extra.

A heurística de Manhattan, devido à sua adaptação a ambientes com movimentação ortogonal, proporcionou vantagens significativas. Ela resultou em caminhos de custo semelhante ao obtido com a heurística Euclidiana, mas com uma redução notável no número de nós gerados e visitados. Isso se deve à fórmula de avaliação do A*, $f(n) = g(n) + h(n)$, que permite uma combinação eficaz do custo acumulado ($g(n)$) e a estimativa restante ($h(n)$),tornando a busca mais eficiente em ambientes estruturados.

Exemplos Principais:

- **Cenário 1 (Start: (9, 7), Goal: (12, 22), Pharmacies: (7, 20), (17, 4), (11, 23), (5, 25)):**

  Heurística H_1 e Função de custo C_1: custo do caminho = 200, nós gerados = 2487, nós visitados = 633.

  Heurística H_2 e Função de custo C_3: custo do caminho = 202, nós gerados = 2956, nós visitados = 752.

- **Cenário 2 (Start: (18, 30), Goal: (13, 15), Pharmacies: (22, 28), (5, 19), (21, 4), (7, 6)):**

  Heurística H_1 e Função de custo C_4: custo do caminho = 249, nós gerados = 4895, nós visitados = 1255.

  Heurística H_2 e Função de custo C_2: custo do caminho = 345, nós gerados = 4332, nós visitados = 1107.

3.5.3 Conclusão

O experimento demonstrou a capacidade do A* de lidar com problemas de minimização de custo que exigem uma parada intermediária. Mesmo com um custo computacional alto e número elevado de nós, o A* cumpriu sua função de encontrar caminhos ótimos com precisão e eficiência. Com heurísticas como

a Euclidiana e, especialmente, a de Manhattan, o algoritmo se mostrou uma solução robusta para aplicações práticas que demandam a inclusão de pontos intermediários em rotas.