Relatório do Projeto Final - Sistema de Busca Informada para Resolução do Problema de Menor Caminho.

Celso Vinícius S. Fernandes[1]

[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Divinópolis – MG – Brasil

[2]Bacharelado em Engenharia de Computação

celso.23@aluno.cefetmg.br

Resumo: *Este trabalho apresenta o desenvolvimento e análise de diferentes algoritmos de busca aplicados ao problema clássico de planejamento de rotas, utilizando Python. A aplicação permite aos usuários explorar o comportamento de um algoritmo de busca informada (A*), com diferentes heurísticasde e dois algoritmos de busca não informada (Busca em Largura - BFS e Busca em Profundidade - DFS). O cenário utilizado é o mapa simplificado de cidades da Romênia, onde o objetivo é encontrar o menor caminho entre duas cidades, considerando distâncias reais e estimativas heurísticas. A implementação mede o desempenho dos algoritmos em termos de tempo de execução e número de nós explorados. O trabalho também discute o impacto das heurísticas no desempenho do algoritmo A*.*

1. Introdução

Este trabalho desenvolve e compara a eficiência de diferentes algoritmos de busca aplicados ao problema de planejamento de rotas em um grafo, utilizando a linguagem de programação Python. O algoritmo de destaque é o `A*`, que faz uso de diferentes heurísticas para otimizar a busca pelo caminho mais curto entre dois pontos. Além do `A*`, dois algoritmos de busca não informada, Busca em Largura (`BFS`) e Busca em Profundidade (`DFS`), são implementados como critério de comparação. O cenário utilizado é o mapa de cidades da Romênia, onde o objetivo é encontrar o menor caminho entre duas cidades.

As heurísticas implementadas para o algoritmo `A*` incluem:

- **hDLR**: uma heurística baseada na distância em linha reta (*straight-line distance*) de cada cidade até o destino final, Bucareste. Esses valores são pré-definidos.
- **Euclidiana**: calcula a distância euclidiana entre as coordenadas geográficas (latitude e longitude) de duas cidades, fornecendo uma estimativa contínua da distância.
- **Manhattan**: calcula a soma das distâncias absolutas das diferenças entre as coordenadas geográficas, usando o conceito de distância de Manhattan.

O `A*` é avaliado com essas três heurísticas, e o impacto de cada uma no desempenho é analisado. Para comparação, os algoritmos `BFS` e `DFS` são implementados, explorando os nós sem o uso de heurísticas e sem qualquer tipo de otimização na seleção do próximo nó a ser explorado.

Considerações importantes para a execução deste trabalho incluem:

1. A implementação abrange o algoritmo de busca informada `A*`, com as três heurísticas mencionadas, e os algoritmos de busca não informada `BFS` e `DFS`.
2. Cada algoritmo é testado 21 vezes em um cenário fixo que utiliza o mapa rodoviário simplificado da Romênia, onde as distâncias entre as cidades são baseadas na literatura.
3. O ponto de partida é `Arad`, e o ponto de chegada é `Bucareste`, e os algoritmos devem ser capazes de explorar o grafo de cidades para encontrar o menor caminho entre essas duas cidades.
4. O desempenho dos algoritmos é medido em termos de tempo de execução, número de nós explorados e completude.

Neste trabalho, espera-se observar o papel das heurísticas na otimização do `A*` em termos de eficiência computacional e número de nós explorados, bem como comparar a eficácia desse algoritmo com métodos de busca tradicionais.

2. Metodologia

Este projeto foca na implementação e comparação de diferentes algoritmos de busca aplicados ao problema de planejamento de rotas, utilizando a linguagem de programação Python. O principal objetivo é encontrar o menor caminho entre duas cidades no mapa da Romênia, e o algoritmo de destaque é o `A*`, que faz uso de diferentes heurísticas para otimizar a busca. Além disso, dois algoritmos de busca não informada, `BFS` e `DFS`, são implementados como critério de comparação.

O mapa rodoviário utilizado para este projeto, que representa parte da Romênia, foi retirado da literatura de Inteligência Artificial [Russell and Norvig 2010], onde ele é apresentado como parte do clássico problema de planejamento de rotas. A Figura 1 ilustra este mapa, no qual as diferentes cidades estão conectadas por estradas com distâncias específicas, servindo como base para os algoritmos de busca. Este problema foi amplamente utilizado para testar algoritmos de busca informada e não informada neste projeto.

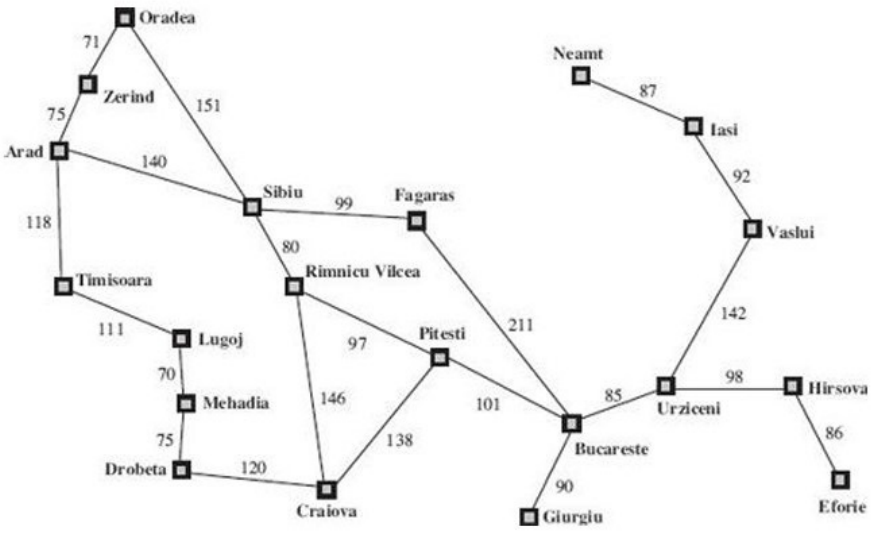

Figura 1. Mapa rodoviário simplificado de parte da Romênia [Russell and Norvig 2010].

2.1. Seleção de Tecnologias

Para o desenvolvimento deste projeto, utilizou-se a linguagem Python e as bibliotecas `NetworkX` e `Matplotlib`. A primeira foi escolhida pela simplicidade e por fornecer funcionalidades adequadas para modelagem de grafos, enquanto a segunda foi usada para a visualização dos resultados, exibindo os caminhos encontrados pelos algoritmos. Todo o desenvolvimento foi realizado no ambiente Visual Studio Code.

2.2. Métricas de Desempenho

Para avaliar o desempenho dos algoritmos, utilizamos as seguintes métricas:

- **Tempo de Execução:** calculado em milissegundos com o uso da função `time.perf_counter()`, permitindo medir o tempo total necessário para que cada algoritmo encontre uma solução.
- **Nós Explorados:** número total de nós visitados durante a execução de cada algoritmo, representando a quantidade de nós avaliados até encontrar a solução.

2.3. Algoritmos de Busca

A implementação principal do projeto consiste no algoritmo de busca informada `A*`, que é avaliado com três heurísticas diferentes: distância em linha reta (`hDLR`), distância Euclidiana e distância Manhattan. Além do `A*`, os algoritmos de busca não informada `BFS` e `DFS` foram implementados para fins comparativos, com o objetivo de medir a eficiência e a eficácia em relação à busca informada.

- **A***: O `A*` utiliza a soma do custo acumulado (`g_cost`) e uma estimativa do custo restante para o destino (`h_cost`) para guiar a busca. A seguir está o pseudocódigo geral do `A*`:

```
AStar(grafo, inicio, destino, heuristica):
    inicializar fila de prioridade
    inicializar custo_acumulado[inicio] = 0

    enquanto a fila de prioridade não estiver vazia:
        no_atual = remover nó com menor f_custo da fila
        se no_atual == destino:
            retornar caminho até o destino

        para cada vizinho de no_atual:
            custo_novo = custo_acumulado[no_atual] +
    distancia_para_vizinho
            se vizinho não foi visitado ou custo_novo for menor:
                custo_acumulado[vizinho] = custo_novo
                prioridade = custo_novo + heuristica(vizinho,
    destino)
                adicionar vizinho à fila de prioridade com
    prioridade

    retornar "caminho não encontrado"
```

Pseudocódigo 1. A*

- **BFS** (Busca em Largura): Algoritmo de busca não informada que utiliza uma fila (`deque`) para explorar os nós por níveis. Garante encontrar o menor caminho em termos de número de arestas, mas tende a explorar muitos nós desnecessários.

- **DFS** (Busca em Profundidade): Outro algoritmo de busca não informada, que explora o grafo o mais profundo possível antes de retroceder. Utiliza uma pilha para rastrear o caminho. Embora seja útil em algumas situações, pode explorar caminhos não ótimos.

2.4. Heurísticas Implementadas para o A*

As heurísticas são centrais no desempenho do A*, guiando o algoritmo a escolher o nó mais promissor a cada passo. As três heurísticas implementadas foram:

- **hDLR**: A heurística hDLR utiliza valores pré-calculados para a distância em linha reta entre cada cidade e o destino final. Esses valores são armazenados em um dicionário, no qual a chave é o nome da cidade e o valor é a distância em linha reta até o destino final, ou seja, a heurística simplesmente retorna o valor associado à cidade atual, sem realizar nenhum cálculo adicional.
  Nessa lógica, no pseudocódigo abaixo, a função recebe como entrada a cidade atual e busca no dicionário o valor pré-calculado da distância. Se a cidade não estiver no dicionário, retorna um valor infinito como fallback.

```
def heuristica_hDLR(cidade_atual, destino, _):
    return hDLR.get(cidade_atual, float('inf'))
```

Pseudocódigo 2. Pseudocódigo da heurística hDLR

- **Euclidiana**: A heurística Euclidiana calcula a distância em linha reta entre duas cidades com base em suas coordenadas (latitude e longitude). O cálculo é feito utilizando o teorema de Pitágoras. Para duas cidades $A(x_1, y_1)$ e $B(x_2, y_2)$, a fórmula para calcular a distância é:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

  Aqui, x_1 e y_1 representam as coordenadas da cidade atual, e x_2 e y_2 as coordenadas do destino. Esse cálculo mede a menor distância "em linha reta"entre as cidades.
  Desse modo, no pseudocódigo à seguir, a função obtém as coordenadas da cidade atual e do destino, calcula a diferença entre as coordenadas x e y, eleva essas diferenças ao quadrado, soma os resultados e, finalmente, calcula a raiz quadrada da soma para obter a distância euclidiana.

```
def heuristica_euclidiana(cidade_atual, destino, coordenadas):
    x1, y1 = coordenadas[cidade_atual]
    x2, y2 = coordenadas[destino]
    return ((x2 - x1)**2 + (y2 - y1)**2) ** 0.5
```

Pseudocódigo 3. Pseudocódigo da heurística Euclidiana

- **Manhattan**: A heurística Manhattan estima a distância entre duas cidades assumindo que o movimento é restrito a direções ortogonais, ou seja, apenas movimentações ao longo dos eixos x e y. A distância é calculada somando as diferenças absolutas entre as coordenadas x e y das duas cidades. Para duas cidades $A(x_1, y_1)$ e $B(x_2, y_2)$, a fórmula para calcular a distância Manhattan é:

$$d = |x_2 - x_1| + |y_2 - y_1|$$

Neste caso, $|x_2 - x_1|$ representa a diferença absoluta nas coordenadas x e $|y_2 - y_1|$ representa a diferença absoluta nas coordenadas y. A soma dessas duas diferenças fornece a distância "em grade"entre as cidades.
Nesse contexto, no pseudocódigo adiante, a função obtém as coordenadas da cidade atual e do destino, calcula as diferenças absolutas entre x e y, e soma esses valores para determinar a distância Manhattan.

```
def heuristica_manhattan(cidade_atual, destino, coordenadas):
    x1, y1 = coordenadas[cidade_atual]
    x2, y2 = coordenadas[destino]
    return abs(x2 - x1) + abs(y2 - y1)
```

Pseudocódigo 4. Pseudocódigo da heurística Manhattan

Cada algoritmo foi testado no grafo das cidades da Romênia, onde `Arad` é o ponto de partida e `Bucareste` o destino. As heurísticas foram aplicadas ao `A*` para avaliar seu impacto no tempo de execução e no número de nós explorados. Os algoritmos `BFS` e `DFS` foram executados como critérios de comparação para medir a eficiência relativa da busca informada. Todos os tempos e números de nós explorados foram registrados, e a análise dos resultados será discutida na próxima seção.

3. Resultados e Discussão

3.1. Modelo e Reprodução

A fim de garantir a reprodutibilidade dos resultados obtidos neste estudo, todo o código-fonte utilizado, incluindo as implementações dos algoritmos `A*`, `BFS` e `DFS`, está disponível em um repositório GitHub público. O diretório `src/` contém todos os scripts desenvolvidos em Python, permitindo que outros pesquisadores ou interessados reproduzam e validem a implementação. O link do repositório Git será disponibilizado na seção de Referências[Celso 2024] deste artigo.

3.2. Experimentação e Análise

Nesta seção, são apresentados os resultados da comparação dos algoritmos de busca implementados (`A*`, `BFS`, `DFS`) no cenário fixo do mapa rodoviário simplificado da Romênia, onde o objetivo é encontrar o menor caminho entre as cidades de `Arad` e `Bucareste`. As três variantes do algoritmo `A*` foram testadas com diferentes heurísticas: distância em linha reta (`hDLR`), Euclidiana e Manhattan. As métricas de avaliação incluem o tempo de execução, o número de nós explorados, bem como a eficiência geral do algoritmo em cada caso.

Algoritmo	Melhor Tempo (ms)	Pior Tempo (ms)	Média de Tempo (ms)	Desvio Padrão (ms)	Nº Nós explorados	Distância Percorrida
A* hDLR	0,34	0,50	0,41	0,05	6	418
A* Euclidiana	0,24	0,47	0,44	0,07	9	418
A* Manhattan	0,24	0,39	0,38	0,01	9	418
BFS	0,13	0,16	0,14	0,01	8	441
DFS	0,11	0,13	0,11	0,01	6	608

Tabela 1. Resultados de desempenho dos algoritmos de busca em termos de tempo de execução, desvio padrão e número de nós explorados.

Os resultados apresentados na Tabela 1 avaliam o desempenho dos algoritmos de busca em termos de tempo de execução, desvio padrão, número de nós explorados

e a distância percorrida pelos algoritmos. Cada algoritmo foi executado 21 vezes para garantir a robustez dos resultados. Como mostrado na tabela, o A* foi executado com três diferentes heurísticas, enquanto BFS e DFS foram utilizados como base de comparação para métodos de busca não informada.

3.3. Análise dos Resultados

- **Algoritmo A***: O A* com a heurística Manhattan apresentou a melhor média de tempo (0,38 ms), sendo mais rápido que a heurística Euclidiana (0,44 ms) e ligeiramente mais eficiente do que a heurística hDLR (0,41 ms). No entanto, o A* com Manhattan e Euclidiana explorou mais nós (9) do que o A* com hDLR (6), o que pode indicar que, apesar de explorar mais nós, essas heurísticas forneceram uma avaliação mais precisa do caminho ótimo. Todos os métodos A* encontraram o caminho de menor distância (418 km), refletindo a alta qualidade dos caminhos gerados com a ajuda das heurísticas.
- **Busca em Largura (BFS)**: O BFS teve um desempenho intermediário com um tempo médio de 0,14 ms e explorou 8 nós. No entanto, o caminho encontrado pelo BFS não foi o mais curto, com uma distância total de 441 km, indicando que, apesar de sua completude, o BFS não é capaz de garantir um caminho ótimo. O desvio padrão foi extremamente baixo (0,01 ms), o que reflete uma execução bastante consistente. Isso também destaca uma limitação do BFS: ao explorar nós sem uma heurística, ele verifica caminhos desnecessários, o que pode resultar em soluções subótimas.
- **Busca em Profundidade (DFS)**: O DFS teve o tempo médio mais rápido (0,11 ms) e explorou 6 nós, igualando-se ao A* hDLR no número de nós explorados. No entanto, o DFS gerou um caminho significativamente mais longo (608 km), mostrando que, enquanto ele explora poucos nós e é rápido, ele não garante um caminho eficiente em termos de distância percorrida. Isso reforça a ideia de que o DFS, embora eficiente em termos de tempo, pode ser inadequado para cenários onde a optimalidade do caminho é um requisito.

3.4. Discussão Geral

Os resultados revelam que, embora o DFS tenha o menor tempo de execução, ele sacrifica a qualidade do caminho. O caminho gerado pelo DFS percorre 608 km, significativamente mais longo do que os 418 km percorridos pelos algoritmos A*, e mesmo do que os 441 km do BFS. O A*, utilizando heurísticas, foi o único algoritmo capaz de encontrar o caminho mais curto (418 km), confirmando sua superioridade em termos de solução ótima para problemas de planejamento de rotas.

A heurística Manhattan, em particular, conseguiu encontrar um bom equilíbrio entre a exploração de nós e o tempo de execução, enquanto a hDLR, embora eficiente em termos de nós explorados, foi ligeiramente mais lenta em média. O BFS, por sua vez, explorou mais nós do que o DFS, o que resultou em um caminho mais curto que o DFS, mas ainda assim subótimo.

Esses resultados reforçam que o A*, com heurísticas bem escolhidas, é o algoritmo mais recomendado para problemas de planejamento de rotas onde a qualidade do caminho é crucial, enquanto algoritmos não informados, como o DFS e BFS, são mais adequados

para situações onde o tempo de execução é prioritário, mas a optimalidade do caminho pode ser sacrificada.

4. Conclusão

A combinação entre a contextualização do problema de planejamento de rotas no mapa da Romênia e a metodologia aplicada neste estudo forneceu uma base razoável para explorar e avaliar a eficiência dos algoritmos de busca. O `A*`, como esperado, apresentou resultados consistentes ao encontrar o caminho mais curto com o auxílio de heurísticas bem selecionadas, dado o cenário testado. Em contraste, os algoritmos `BFS` e `DFS` foram úteis como pontos de comparação, revelando diferentes compromissos entre tempo de execução e precisão. Dessa forma, o estudo permitiu uma análise relevante da hipótese central, sugerindo que algoritmos de busca informada, como o `A*`, podem ser mais eficazes na obtenção de soluções ótimas em problemas de planejamento de rotas, desde que a heurística escolhida seja adequada ao cenário.

Entretanto, o estudo possui algumas limitações que devem ser ressaltadas. A análise foi realizada em um cenário fixo e relativamente simples, o que pode não refletir o comportamento dos algoritmos em ambientes mais dinâmicos ou complexos. Além disso, o foco na comparação entre diferentes heurísticas para o `A*` poderia ser ampliado para incluir outros algoritmos de busca informada ou híbrida, potencialmente mais eficazes em cenários maiores e mais desafiadores. Assim, para trabalhos futuros, seria interessante expandir o estudo para ambientes com maior variabilidade, como mapas dinâmicos ou com múltiplos pontos de destino.

5. Compilação e Execução

5.1. Escolha dos Algoritmos

O programa está configurado para executar todos os algoritmos de busca automaticamente. Ao rodar o script principal `main.py`, são executadas cinco variações: o algoritmo `A*` com três diferentes heurísticas (`Euclidiana`, `Manhattan` e `hDLR`), além das buscas não informadas `BFS` e `DFS`. Não é necessário que o usuário faça escolhas manuais de heurísticas ou algoritmos, pois o script já está preparado para executar todos sequencialmente.

A função `main()` exibe os resultados de cada algoritmo, incluindo o caminho encontrado, a distância total, o número de nós explorados e o tempo de execução em milissegundos. Além disso, o grafo é visualizado graficamente, destacando o caminho percorrido por cada algoritmo.

5.2. Execução

Para executar o programa, siga os passos abaixo:

- Abra o terminal no diretório onde os arquivos do projeto estão localizados.
- Certifique-se de que as bibliotecas `networkx` e `matplotlib` estão instaladas. Se não estiverem instaladas, você pode instalar essas bibliotecas manualmente utilizando o seguinte comando:

```
pip install networkx matplotlib
```

- Em seguida, execute o programa com o comando:

```
python main.py
```

  Isso irá executar o arquivo `main.py`, que automaticamente rodará todos os algoritmos de busca e exibirá os resultados no terminal e as visualizações gráficas na tela.

Referências

Celso, V. (2024). Sistema de busca informada para resolução de problemas de planejamento. `https://github.com/celzin/IA-Trabalho-Final`. Acesso em: 06 ago. 2024.

Russell, S. and Norvig, P. (2010). *Inteligência Artificial: Uma Abordagem Moderna*. Prentice Hall, Upper Saddle River, NJ, USA, 3 edition.