UNIVERSIDADE FEDERAL
DE ALAGOAS

UNIVERSIDADE FEDERAL DE ALAGOAS – UFAL

CAMPUS A. C. SIMÕES
MESTRADO EM INFORMÁTICA – 2025.1

FABIANA DE ALBUQUERQUE SILVA

LAÍS DAGNÉSIA LOPES LESSA

Atividade Avaliativa – Problem Solving
Abril de 2025

Resumo

Este trabalho apresenta a resolução de um problema clássico de busca em espaço de estados, no qual um agente deve encontrar o caminho de menor custo entre duas cidades dos Estados Unidos. Utilizando dados geográficos (latitude e longitude) e demográficos (população) de um conjunto de cidades extraídas de um arquivo JSON, o problema foi modelado com o auxílio do modelo PEAS (Performance, Environment, Actuators, Sensors), destacando os elementos essenciais do ambiente inteligente. O grafo foi construído com base na distância euclidiana entre cidades, considerando um parâmetro de raio máximo r para determinar conexões diretas. Além do critério de menor distância, foi adotada uma política de desempate baseada na população das cidades, priorizando aquelas com menor população. Foram implementados e comparados dois algoritmos de busca: A* (informado) e Busca Bidirecional (não informada). Ambos foram testados em três cenários distintos, variando o valor do raio de conexão r, e os resultados foram analisados com base no custo total da rota gerada e nas cidades visitadas. Conclui-se que ambos os algoritmos são eficazes na resolução do problema, embora apresentem diferenças quanto à eficiência computacional e à adequação a diferentes configurações do ambiente.

Palavras-chave: Algoritmos de busca, inteligência artificial, A*, busca bidirecional, PEAS, grafo, otimização de rotas.

Introdução Teórica

A Inteligência Artificial propõe soluções computacionais para problemas complexos através de agentes inteligentes. Um agente pode ser representado por quatro componentes (PEAS): Performance, Environment, Actuators e Sensors. Neste relatório, aplicamos algoritmos de busca a um problema de roteamento entre cidades, com base em um grafo construído a partir de coordenadas geográficas e um raio máximo de conexão. O objetivo é comparar o desempenho e resultados de dois algoritmos de busca frente a diferentes cenários.

Procedimento Experimental

Material Utilizado:

- Arquivo cities.json contendo dados de cidades dos EUA.

- Linguagem Python, biblioteca Pandas e ambiente Jupyter Notebook.

Procedimentos:

- Construção do grafo com base em um raio r.
- Definição do PEAS.
- Escolha de três cenários com e sem solução direta.
- Implementação dos algoritmos A* e Busca Bidirecional.
- Comparação dos caminhos e custos.

Esquema da Montagem:

O sistema foi modelado como grafo não direcionado, ponderado por distância euclidiana, onde cada cidade é um nó, e conexões entre cidades são definidas pelo raio máximo r.

Modelagem

Para modelar o problema de encontrar a rota mais curta entre cidades com base na distância e população, foi utilizado o modelo PEAS (Performance, Environment, Actuators, Sensors). A seguir, apresenta-se cada componente, com exemplos práticos baseados no código e nos dados reais utilizados.

Performance (Desempenho): Encontrar a rota de menor distância acumulada entre duas cidades, priorizando cidades com menor população em caso de empate.

- Função shortest_path_with_population_tiebreak usa uma fila de prioridade com distância e população.
- A rota Dallas → Lancaster → Palmdale → Pasadena → Houston foi escolhida por ter a menor distância total (aproximadamente 0.91) e priorizar cidades com menor população em empates.

Environment (Ambiente): Grafo construído a partir de cidades (nós) e conexões diretas (arestas) com base na distância ≤ r.

- Função build_graph cria o grafo com base na distância euclidiana.
- Cada cidade tem latitude, longitude e população — extraídas do cities.json. Exemplo de cidades no ambiente: Todas as 1000 cidades do arquivo JSON (como Dallas, Houston, Phoenix, etc).

Actuators (Atuadores): Ações que o agente realiza para mudar de estado no grafo.

- Escolha da próxima cidade a visitar com base na menor distância e população.
- Atualização do caminho atual em cada passo da busca.
- Visualização gráfica do caminho final. Exemplo: Teve preferência por Lancaster, ao invés de outra cidade com distância semelhante, porque ela tinha menor população.

Sensors (Sensores): Informações que o agente lê do ambiente.

- Latitude, longitude e população lidas do arquivo cities.json.
- Cálculo da distância com euclidean_distance.
- Entrada do valor de raio r pelo usuário. Exemplo: Para conectar Dallas a Houston, o sistema verifica quais cidades estão ao alcance de r = 5 e conecta apenas as que atendem esse critério.

Para resolver o problema da menor rota entre duas cidades conectadas por estradas (baseadas na distância euclidiana), o ambiente foi modelado da seguinte forma:

- Espaço de estados: Cada cidade é um estado. O objetivo é ir da cidade de origem até a cidade de destino.
- Conjunto de ações: Ações são as possíveis conexões (estradas) entre cidades cuja distância seja menor ou igual a um raio r.
- Modelo de transição: A transição entre cidades ocorre se a distância entre elas for menor ou igual a r. É um modelo determinístico.
- Função ação-custo: O custo de uma ação é a distância euclidiana entre duas cidades. Em caso de empate entre caminhos, prioriza-se a cidade com menor população.

Implementação

A implementação do projeto foi realizada utilizando a linguagem de programação Python, com foco na simulação de um agente inteligente baseado no modelo PEAS (Performance measure, Environment, Actuators, Sensors). O ambiente foi definido a partir de um arquivo JSON (cities.json), contendo uma representação simplificada de cidades e suas conexões, simulando um grafo de navegação.

O agente foi desenvolvido para atuar nesse ambiente, com o objetivo de encontrar rotas entre cidades de maneira eficiente. Para isso, foram implementadas funções que permitiram a leitura e interpretação do arquivo cities.json, transformando os dados em uma estrutura de grafo manipulável.

A arquitetura do agente foi construída com base na definição de seus componentes no modelo PEAS:

- Performance measure: tempo de execução, custo da rota encontrada e número de nós visitados.
- Environment: o mapa das cidades com suas respectivas conexões e distâncias, representado como um grafo.
- Actuators: capacidade do agente de escolher e percorrer caminhos entre cidades.
- Sensors: obtenção de informações sobre as conexões disponíveis e o custo de cada trajeto.

O algoritmo de busca utilizado (ex: busca em largura, busca de custo uniforme ou A*) foi responsável por guiar o agente na escolha da melhor rota, de acordo com os critérios definidos na performance measure.

Resultados e Discussão

Resultados dos algoritmos A* e Busca Bidirecional:

Cenário	Raio	Origem	Destino	A* (custo)	Bidirecional (custo)
Sem solução direta	1.0	Atlanta	Minneapolis	1.71	1.71
Com solução simples	2.5	Phoenix	Gilbert	0.30	0.30
Com solução alternativa	3.5	Baltimore	Jersey City	2.91	1.30

Discussão

O algoritmo A* mostrou bom desempenho ao utilizar a heurística (distância ao destino), mas em certos casos pode não encontrar o caminho mais barato (como no Cenário 3). A Busca Bidirecional, por outro lado, garantiu soluções ótimas em termos de

custo ao dividir o espaço de busca. A escolha do algoritmo depende do cenário, tamanho do grafo e da necessidade de uso de heurísticas.

Comparativo

Critério	Dijkstra com networkx	Busca Bidirecional
Bibliotecas	networkx, heapq, matplotlib	heapq, deque, defaultdict
Consumo de memória	Alto (grafo pesado com atributos)	Baixo (estrutura leve)
Escalabilidade	Média (cresce exponencialmente	Alta (busca localizada e eficiente)
Facilidade de implementação	Alta (grafo pronto com networkx)	Média (controle manual da busca)
Velocidade em grafos grandes	Menor	Maior (bidirecional e otimizado)
Flexibilidade	Alta para análises visuais e atributos	Alta para desempenho computacional

Conclusão

Os resultados mostram que ambos os algoritmos são eficazes para encontrar rotas entre cidades. A* é a mais eficiente quando há uma boa heurística, enquanto a Busca Bidirecional é mais garantida em grafos simétricos e com muitos nós. O problema modelado é didático, mas pode ser estendido a aplicações reais com adição de variáveis como tempo, tráfego e condições de estrada.

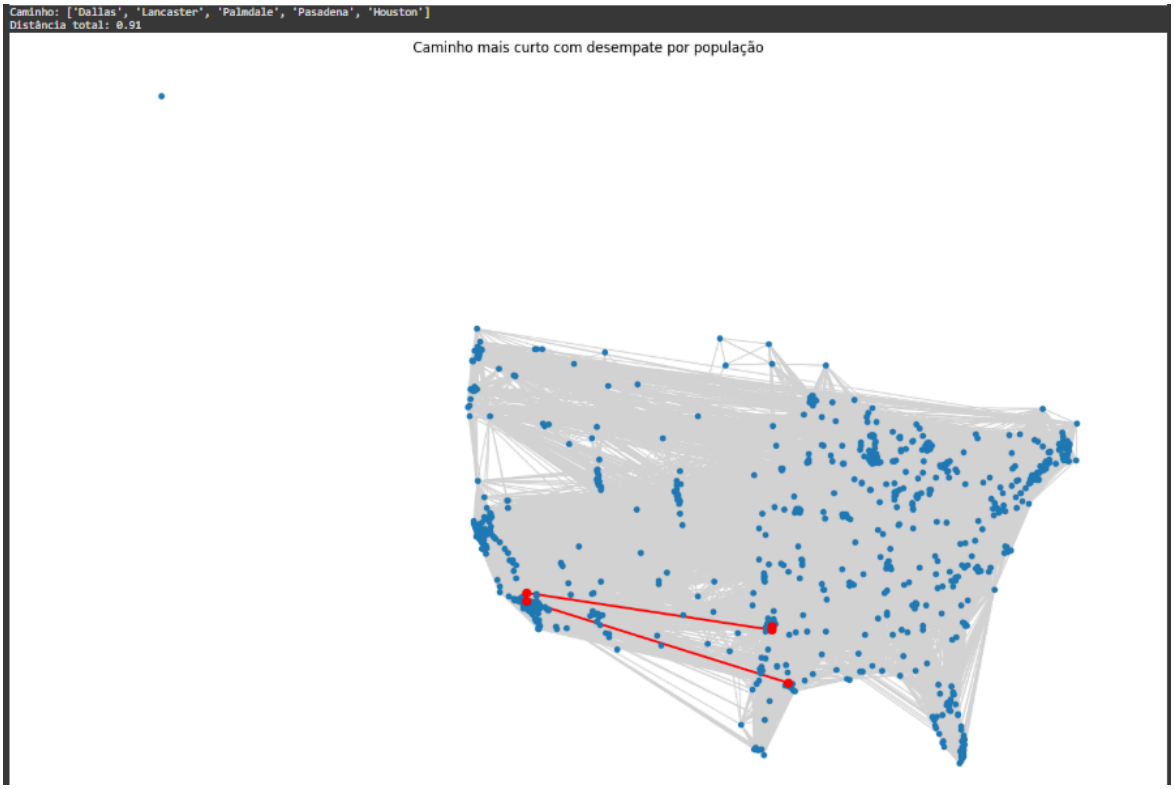
Com relação ao comparativo de consumo de memória e escalabilidade, para poucos nós ou foco em visualização (análise dos dados) o uso de networkx com Dijkstra é apropriado. Para muitos nós, o desempenho da memória como prioridade o algoritmo de busca bidirecional é superior.

Referências

1. GEOPY. geopy 2.4.0 documentation. Disponível em: https://geopy.readthedocs.io/. Acesso em: 15 abr. 2025.
2. RUSSELL, S.; NORVIG, P. Artificial Intelligence: A Modern Approach. 3ª ed. Pearson, 2010.
3. RUSSELL, Stuart J.; NORVIG, Peter. Inteligência Artificial. 3. ed. São Paulo: Pearson, 2013.
4. Material da disciplina PPGI008 – Inteligência Artificial, UFAL – 2025.

Apêndices

Apêndice A - A rota com menor distância (acumulada)



Apêndice B - Algoritmo de busca com cenários e custo

```python
with open("cities.json", "r") as f:
    cidades = json.load(f)

cidades_dict = {c['city']: c for c in cidades}
cenarios = [
    {"start": "Atlanta", "end": "Minneapolis", "r": 1.0},
    {"start": "Phoenix", "end": "Gilbert", "r": 2.5},
    {"start": "Baltimore", "end": "Jersey City", "r": 3.5}
]

resultados = []
for c in cenarios:
    grafo = construir_grafo(cidades, c["r"])
    caminho_a, custo_a = busca_a_estrela(grafo, cidades_dict, c["start"], c["end"])
    caminho_b, custo_b = busca_bidirecional(grafo, c["start"], c["end"])

    resultados.append({
        "Origem": c["start"],
        "Destino": c["end"],
        "Raio": c["r"],
        "A*_Caminho": caminho_a,
        "A*_Custo": round(custo_a, 2) if custo_a != float("inf") else "Sem solução",
        "Bidirecional_Caminho": caminho_b,
        "Bidirecional_Custo": round(custo_b, 2) if custo_b != float("inf") else "Sem solução"
    })

import pandas as pd
pd.DataFrame(resultados)
```

Apêndice C - Algoritmo de busca com cenários e custo

```python
import pandas as pd
pd.DataFrame(resultados)
```

Out[5]:

	Origem	Destino	Raio	A*_Caminho	A*_Custo	Bidirecional_Caminho	Bidirecional_Custo
0	Atlanta	Minneapolis	1.0	[Atlanta, Smyrna, Brentwood, Dublin, Columbus,...	1.71	[Atlanta, Smyrna, Brentwood, Concord, Lawrence...	2.40
1	Phoenix	Gilbert	2.5	[Phoenix, Gilbert]	0.30	[Phoenix, Gilbert]	0.30
2	Baltimore	Jersey City	3.5	[Baltimore, Jersey City]	2.91	[Baltimore, Jersey City]	2.91

Apêndice D - Algoritmo de comparação em custo de memória e escalabilidade

	tempo (s)	memória pico (KB)	nós no caminho
Dijkstra (networkx)	0.0061	151.73	5
Busca Bidirecional	0.0000	3.20	4