Escola de Engenharia
Universidade do Minho

Inteligência Artificial 2024/25

Trabalho Prático

Resolução de Problemas - Algoritmos de procura

Grupo 7

Pedro Seabra Vieira
a104352

DELTA = 0

Pedro Filipe Maneta Pinto
a104176

DELTA = 0

Marco António Fernandes Brito
a104187

DELTA = 0

Tomás Henrique Alves Melo
a104529

DELTA = 0

Índice

1 Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular de Inteligência Artificial da Licenciatura em Engenharia Informática da Universidade do Minho, com o objetivo de abordar a resolução de problemas relacionados com a distribuição eficiente de recursos em situações de catástrofes naturais. O projeto centrou-se no desenvolvimento e implementação de algoritmos de procura, tanto informada como não informada, permitindo otimizar as operações de entrega de suprimentos em cenários marcados por restrições e variabilidade das condições.

Com base na formulação do problema, foi criado um modelo que representa as zonas afetadas e as respetivas rotas através de grafos, incorporando as limitações e prioridades inerentes ao contexto. Ao longo deste documento, será apresentada uma visão detalhada do processo de conceção, implementação e avaliação das estratégias de procura adotadas, destacando as soluções encontradas para os desafios específicos do problema.

2 Formulação do Problema

Os algoritmos implementados neste trabalho permitiram a obtenção de rotas de entrega eficientes, considerando a distribuição de suprimentos em zonas afetadas por catástrofes naturais. Este problema pode ser representado como um problema de procura em grafo, onde os nodos representam as zonas de entrega e a base central de operações. Tendo em vista a maximização da assistência às áreas mais necessitadas, dentro de um prazo limitado e com recursos restritos, o objetivo principal é minimizar o tempo total de operação e a utilização de recursos, enquanto se garante a cobertura das áreas prioritárias.

Uma vez que o problema ocorre num ambiente determinístico e observável, no qual o agente "conhece" o estado em que se encontra e as soluções são sequências de ações, podemos classificá-lo como um problema de estado único. A formulação do problema inclui as seguintes informações:

- **Estado inicial**: A base central de operações é a cidade do Porto, onde todos os recursos estão inicialmente armazenados e prontos para serem distribuídos.
- **Estado objetivo**: Zonas afetadas pelas catástrofes que requerem entrega de suprimentos.
- **Operadores**: Deslocação entre zonas, ou seja, o percurso entre quaisquer zonas ligadas por arestas do grafo, respeitando as restrições dos veículos (capacidade, combustível, etc.).
- **Solução**: A solução ideal é a sequência de zonas visitadas que minimize os custos totais (tempo, recursos, combustível) e maximize a assistência, começando na base central e cobrindo todas as zonas prioritárias, efetuando todas as entregas possíveis e necessários.
- **Custo da solução**: Soma dos custos de deslocação, que pode incluir distância percorrida, consumo de combustível e tempo gasto para alcançar e atender cada zona de entre.

3 Grafo

3.1 Estrutura do grafo

O grafo é implementado através da classe Graph, que utiliza:

- Nodos (m_nodes): Representam zonas, que podem ser pontos de entrega ou a base central. Representam cidades como Lisboa, Porto, Braga, Coimbra, entre outras.

- Arestas (m_graph): Representam as conexões entre as zonas, associadas a pesos que indicam o custo de deslocação (por exemplo, distância, tempo ou consumo de recursos).
- Heurísticas (m_h): Valores estimados para cada nodo, utilizados em algoritmos de procura informada, como A* e Busca Gulosa.

Nós temos dois tipos de grafos o *Zonas.json* que apenas apresentamos todos os distritos de Portugal mais Açores e Madeira e o *ZonasBig.json* onde apresenta todos os municípios de Portugal. Apenas mostramos o grafo das cidades

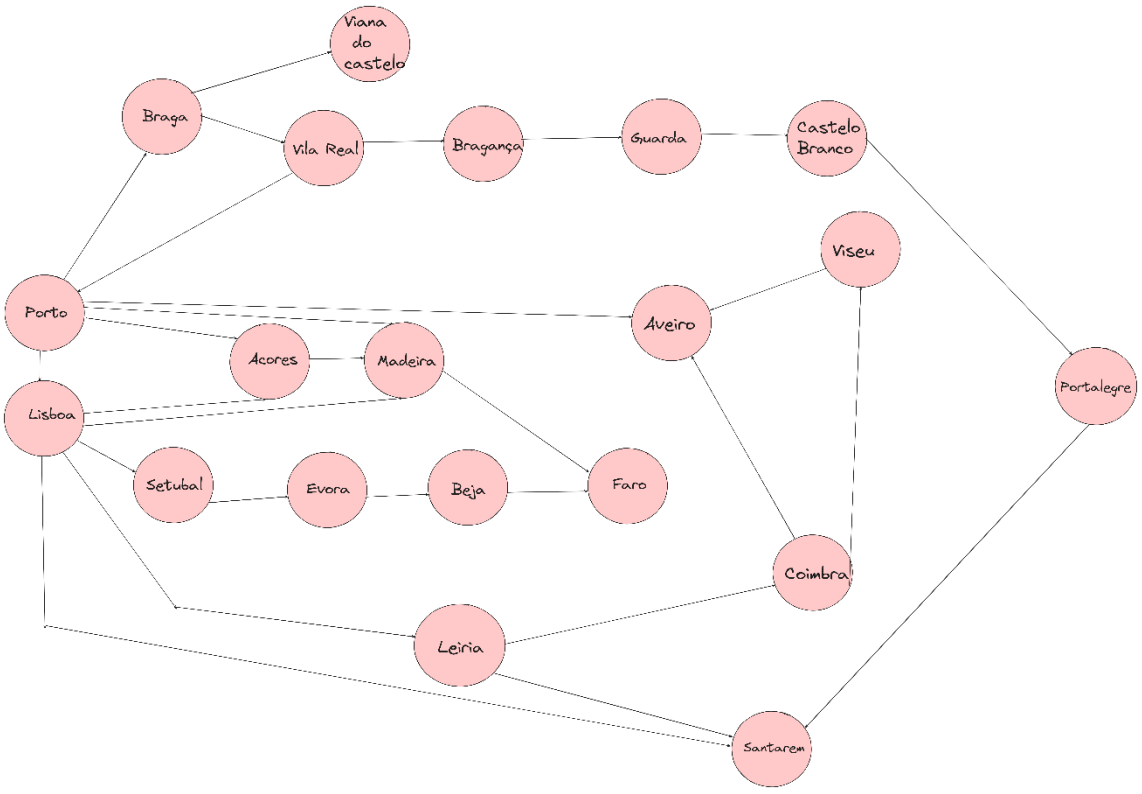


Figura 1- Exemplo de um grafo de algumas cidades

4 Funcionalidades Implementadas

No trabalho desenvolvido, implementámos diversas funcionalidades para resolver o problema da distribuição de suprimentos utilizando grafos e algoritmos de procura. Começámos pela criação e representação do grafo, onde foi implementada uma classe Graph que permite representar as cidades como nodos e as conexões entre elas como arestas. Esta classe inclui funcionalidades para adicionar nodos e arestas, representando os custos associados a cada conexão (como distância, tempo ou consumo de combustível), imprimir as conexões do grafo e visualizar graficamente o mesmo utilizando as bibliotecas networkx e matplotlib.

Como era pedido, implementámos a simulação de condições variáveis, permitindo alterar o estado do grafo em tempo real. É possível simular bloqueios de rotas devido a desastres ou condições climáticas adversas e ajustar os custos das conexões em função de mudanças nas condições de acessibilidade.

Adicionalmente, foi desenvolvido um menu interativo para facilitar a interação com o sistema. Este menu permite adicionar novas zonas e conexões, executar diferentes algoritmos de procura. Por fim, gerámos relatórios detalhados que incluem uma análise comparativa entre os algoritmos implementados, destacando a eficiência e os resultados obtidos.

4.1 Menu

O menu inicial do programa permite configurar a distribuição de entregas de forma simples e eficiente. O utilizador escolhe o número de entregas a realizar (1 a 3) e o algoritmo de busca desejado, com opções como A*, BFS, DFS, Greedy, ou a comparação de todos os algoritmos. É apresentada uma lista de zonas disponíveis, incluindo informações como nome, população, gravidade (atualmente None), clima e impacto climático, permitindo uma análise informada. O utilizador seleciona as zonas afetadas inserindo os números correspondentes.

```
Quantas entregas queres fazer? (1 a 3): 2
Você escolheu fazer 2 entregas.

Escolha o algoritmo de busca para as rotas:
1. A*
2. BFS
3. DFS
4. Greedy
5. Comparar Todos
Digite o número correspondente ao algoritmo desejado: 5
Algoritmo selecionado: Comparar Todos

Zonas disponíveis:
1. Lisboa (População: 2275000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
2. Porto (População: 1722000, Gravidade: None, Clima: chuva (Impacto: 0.7))
3. Setubal (População: 829000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
4. Braga (População: 848000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
5. Aveiro (População: 713000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
6. Faro (População: 451000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
7. Coimbra (População: 429000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
8. Leiria (População: 470000, Gravidade: None, Clima: chuva (Impacto: 0.7))
9. Madeira (População: 251000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
10. Acores (População: 137856, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
11. Beja (População: 152000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
12. Braganca (População: 136000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
13. CasteloBranco (População: 197000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
14. Evora (População: 166000, Gravidade: None, Clima: tempestade (Impacto: 0.6))
15. Guarda (População: 143000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
16. Portalegre (População: 118000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
17. Santarem (População: 445000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
18. VianaDoCastelo (População: 252000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
19. VilaReal (População: 193000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
20. Viseu (População: 378000, Gravidade: None, Clima: ensolarado (Impacto: 1.0))
Escolha o número das zonas afetadas separadas por vírgula (restante 2 zonas):
```

4.2 Exemplo ficheiro Zonas.json

A partir do ficheiro JSON, podemos extrair diversas informações sobre a cidade de Lisboa. Este é apenas um exemplo de uma de muitas localidades que fizemos. Podemos extrair deste exemplo que Lisboa aceita três tipos de transporte: aéreo, terrestre e aquático, sendo que está conectada a várias zonas. As conexões incluem Setúbal, Santarém e Leiria, todas acessíveis por transporte terrestre; o Porto, acessível por transporte aéreo; a Madeira, acessível tanto por transporte aéreo como aquático; e os Açores, acessíveis por transporte aéreo.

Estas informações são fundamentais para o planeamento e otimização da distribuição de suprimentos, permitindo identificar os meios de transporte disponíveis, os destinos conectados e a população que poderá influenciar a prioridade de atendimento. Além disso, as coordenadas geográficas podem ser utilizadas para calcular distâncias e auxiliar algoritmos de procura informada, como o A*, garantindo uma gestão eficiente e direcionada dos recursos.


```
{
  "nome": "Lisboa",
  "necessidade": null,
  "gravidade": null,
  "isBase": false,
  "x": 38.7169,
  "y": -9.1390,
  "restricoes": ["aereo", "terrestre", "aquatico"],
  "populacao": 2275000,
  "janela_tempo_critica": null,
  "conexoes": [
    {"destino": "Setubal", "tipo": "terrestre"},
    {"destino": "Santarem", "tipo": "terrestre"},
    {"destino": "Leiria", "tipo": "terrestre"},
    {"destino": "Porto", "tipo": "aereo"},
    {"destino": "Madeira", "tipo": "aereo"},
    {"destino": "Madeira", "tipo": "aquatico"},
    {"destino": "Acores", "tipo": "aereo"}
  ]
},
```


Estrutura geral

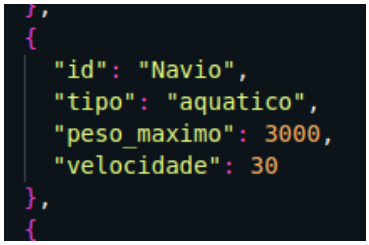
Para uma melhor compreensão do json os tópicos como: necessidade refere-se a uma necessidade específica associada à localização. O campo isBase indica se esta é a localização inicial. As coordenadas da localização são representadas pelos campos x e y. As restrições correspondem a uma lista dos tipos de transporte permitidos ou necessários para aceder este local. A janela de tempo crítica é um campo que define o tempo máximo permitido para uma necessidade crítica. Por fim, as conexões são uma lista que descreve as ligações com outras localidades, especificando o nome dos destinos e o tipo de transporte usado para conectar cada localização.

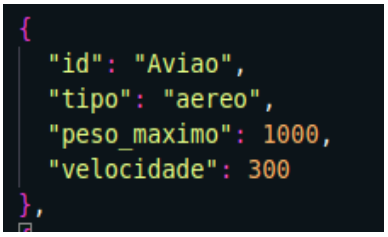
4.3 Exemplo ficheiro Veículos.json

Os ficheiros fornecem informações sobre os veículos disponíveis para o transporte de suprimentos.

- A **Moto** é um veículo terrestre com capacidade para 150 kg e velocidade de 100 km/h.
- O **Navio** é um veículo aquático com capacidade para 3000 kg e velocidade de 30 km/h.
- O **Avião** é um veículo aéreo com capacidade para 1000 kg e velocidade de 300 km/h.

Essas características influenciam o planeamento das rotas, com base na velocidade, capacidade de carga e tipo de transporte exigido por cada zona. O Avião é ideal para entregas rápidas, o Navio para grandes volumes em áreas aquáticas, e a Moto para entregas ágeis em zonas terrestres. Esses dados ajudam a otimizar a distribuição de recursos com base nas condições e restrições de cada zona.





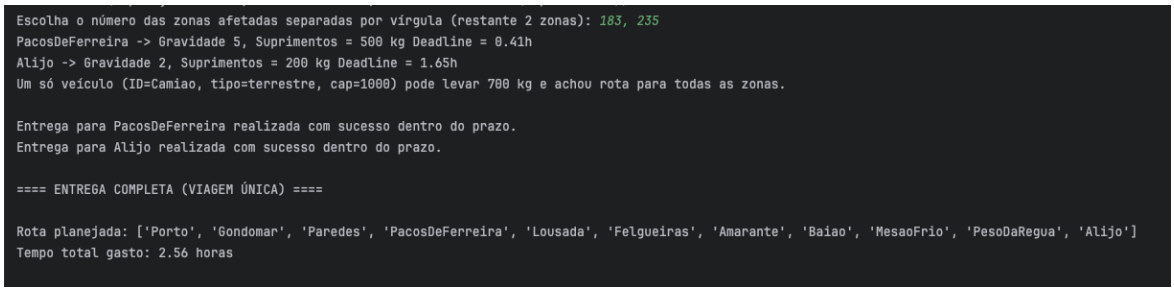
5 Estratégias de Procura Não Informada

5.1 Procura em Profundidade (DFS)

A pesquisa em profundidade baseia-se na estratégia de explorar os nodos mais profundos do grafo antes de retroceder. As suas principais vantagens incluem o uso reduzido de memória e uma maior probabilidade de encontrar caminhos viáveis em problemas com várias soluções. No entanto, esta abordagem pode ser menos eficiente em grafos muito profundos com poucas soluções e não garante que a solução encontrada seja a melhor possível.

No entanto, no nosso problema de distribuição de suprimentos em cenários de catástrofes, é crucial encontrar um percurso eficiente rapidamente para garantir a entrega de recursos nas zonas prioritárias dentro do prazo. Por essa razão, a estratégia de pesquisa em profundidade dificilmente conseguirá determinar uma solução ótima de forma eficiente. Contudo, na nossa implementação, o algoritmo de pesquisa em profundidade foi adaptado para evitar que zonas já visitadas sejam exploradas novamente, prevenindo ciclos e tornando o processo de procura mais eficaz.

Exemplo a utilizarmos o DFS:

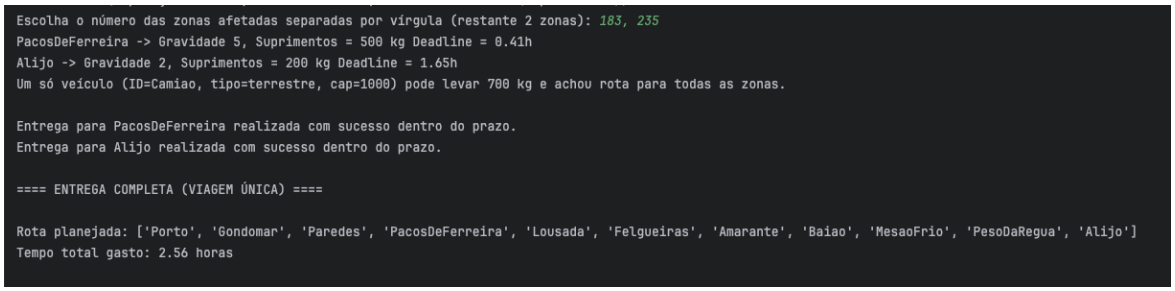


5.2 Procura em Largura (BFS)

A pesquisa em largura expande primeiro os nodos de menor profundidade no grafo. A principal vantagem desta abordagem é que garante a solução ótima quando todas as arestas têm o mesmo custo. No entanto, apresenta desvantagens como o elevado tempo de pesquisa, já que tende a explorar um grande número de nodos desnecessários antes de encontrar um caminho válido, e o elevado consumo de memória, devido à necessidade de armazenar todos os nodos de um nível antes de prosseguir para o próximo.

No contexto do nosso problema, o grafo apresenta um tamanho considerável, tornando este algoritmo pouco eficiente para a aplicação, já que o elevado número de nodos implica um aumento significativo no tempo de execução e no consumo de memória.

Exemplo a utilizarmos o BFS:



6 Estratégias de Procura Informada

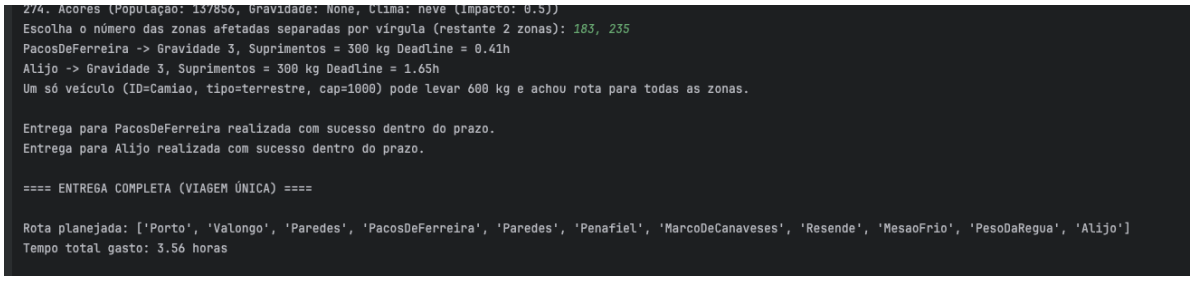
6.1 Procura com A*

O algoritmo A* implementado no nosso código é uma técnica de procura informada que combina o custo acumulado até ao nodo atual g(n) com uma estimativa do custo restante para alcançar o objetivo h(n). Este método garante que a solução encontrada seja ótima, desde que a heurística utilizada seja admissível (não superestima o custo real).

O algoritmo começa no nodo inicial (Porto) com custo acumulado (g) igual a 0 e calcula o custo total estimado ($f = g + h$). Em cada iteração, seleciona o nodo com menor f(n), explora os seus vizinhos, atualiza g e h, e adiciona-os à lista de exploração se o caminho for mais eficiente. O processo termina ao alcançar o objetivo ou esgotar os caminhos, reconstruindo então o caminho ótimo.

No nosso problema, o algoritmo A* é eficiente, pois combina o menor custo acumulado com uma estimativa informada para encontrar soluções ótimas rapidamente. Além disso, a heurística pode ser ajustada para incluir fatores como distância, gravidade das zonas ou populações, tornando-o altamente adaptável ao contexto.

Exemplo a utilizarmos o A*:



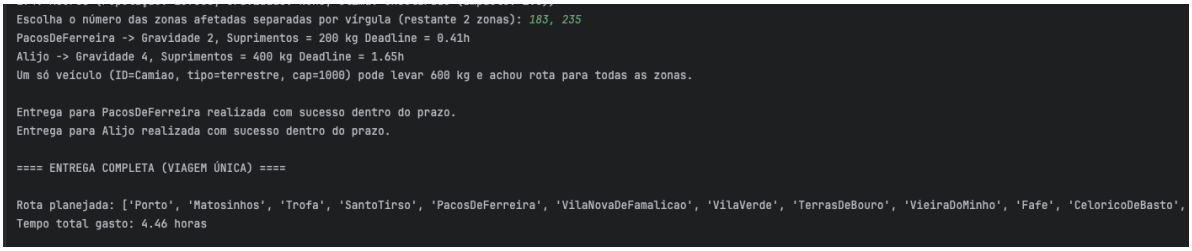
6.2 Procura Greedy

O algoritmo Greedy implementado no nosso código é uma técnica de procura informada que utiliza apenas a estimativa do custo restante para alcançar o objetivo h(n) como critério de decisão. Este método não considera o custo acumulado g(n), focando-se exclusivamente na heurística para orientar a busca, o que pode levar a soluções rápidas, mas não necessariamente ótimas.

O algoritmo começa no nodo inicial (Porto), calculando a heurística (h) para todos os nodos e selecionando, em cada iteração, o nodo com o menor h(n). Explora os vizinhos, atualiza as heurísticas e adiciona os nodos não explorados à lista de exploração. O processo termina ao alcançar o objetivo ou esgotar os caminhos, reconstruindo o percurso a partir dos nodos visitados.

No nosso problema, o algoritmo Greedy é eficiente em termos de rapidez, uma vez que se foca na heurística para guiar a procura, permitindo encontrar soluções em menos tempo. No entanto, como não considera o custo acumulado, pode não garantir a solução ótima. A sua heurística pode ser ajustada para considerar fatores como distância, gravidade das zonas ou populações, tornando-o útil para situações onde a rapidez é mais importante que a otimização absoluta.

Exemplo a utilizarmos o Greedy:



7 Conclusão

Concluindo, este projeto permitiu-nos aprofundar e aplicar na prática as técnicas de formulação de problemas e a sua resolução através da implementação de algoritmos de procura ensinados em aula, adaptando-os a um contexto de distribuição de suprimentos em cenários de catástrofes naturais. Durante o desenvolvimento, enfrentámos algumas dificuldades, como a necessidade de ajustar as heurísticas para refletir corretamente a gravidade das zonas afetadas, já que inicialmente algumas áreas prioritárias não estavam a ser atendidas de forma eficiente. Para resolver este problema, refinámos as heurísticas utilizadas, integrando fatores como população e risco operacional. Apesar dos desafios, acreditamos que conseguimos atingir todos os objetivos propostos e estamos bastante satisfeitos com os resultados alcançados.