Universidade do Minho
Escola de Engenharia

Inteligência Artificial

Trabalho Prático

Resolução de Problemas - Algoritmos de procura

Ano letivo 2024/2025

Grupo 11:
Afonso Gregório de Sousa (A104262)
Filipa Oliveira da Silva (A104167)
Maria Cleto Rocha (A104441)
Mário Rafael Figueiredo da Silva (A104182)

Avaliação por Pares

PG104262 Afonso Sousa = 0
PG104182 Mário Silva = 0
PG104167 Filipa Silva = 0
PG104441 Maria Rocha = 0

1. Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Inteligência Artificial e tem como objetivo principal desenvolver e implementar algoritmos de procura que permitam otimizar a distribuição de alimentos em zonas afetadas por uma catástrofe natural. Catástrofes naturais impõem diversos desafios significativos na logística de ajuda humanitária, onde a rápida e eficiente alocação de recursos pode salvar vidas. Neste cenário, o uso de algoritmos de procura permite explorar soluções que otimizem a distribuição de alimentos, mesmo em condições de acesso adversas e recursos limitados. Para atingir este objetivo, foram implementados diversos algoritmos de procura, informada e não informada, como o BFS (Procura em largura), DFS (Procura em profundidade), A*, Greedy (Procura gulosa) e Custo Uniforme, que estruturam o problema com o auxílio de um grafo e priorizam as áreas mais necessitadas, maximizando assim o número de pessoas assistidas dentro de um tempo limitado.

2. Descrição do Problema

O problema apresentado decorre num cenário de emergência humanitária provocado por uma catástrofe natural, onde é necessário distribuir alimentos e mantimentos de forma eficiente às zonas mais afetadas. Estas zonas variam consoante a gravidade, desde áreas densamente povoadas até locais isolados e de difícil acesso. Adicionalmente, a operação é condicionada por restrições como alterações meteorológicas, a capacidade limitada dos veículos de transporte e as condições adversas do terreno (bloqueio de rotas).

Este contexto exige a priorização das áreas mais críticas, de forma a maximizar o número de pessoas assistidas dentro do tempo disponível, enquanto se minimiza o desperdício de recursos. Para tal, o problema envolve decisões complexas de logística, incluindo a seleção das rotas mais eficazes, a gestão de veículos com características distintas (capacidade de carga, autonomia, entre outros) e a adaptação a condições dinâmicas, como bloqueios de estradas ou alterações no clima. Assim, a utilização de algoritmos de procura em Inteligência Artificial torna-se fundamental para modelar e resolver este problema de forma eficiente.

3. Formulação do Problema

Este problema pode ser representado como um problema de procura com elementos estáticos e dinâmicos, visto que o mesmo decorre num cenário onde as condições iniciais são conhecidas e o ambiente é determinístico. A sua formulação inclui os seguintes aspetos:

- **Representação do estado:** O problema é estruturado com a utilização de um grafo orientado, onde cada vértice representa uma zona a ser assistida e cada aresta corresponde a uma rota viável e segura entre zonas. As arestas são associadas a custos, que neste caso indica a distância percorrida, em quilómetros (km), entre duas zonas.
- **Estado inicial:** O estado inicial consiste nos pontos de partida dos veículos de transporte, que representam os locais onde os alimentos e suprimentos estão armazenados para serem distribuídos.
- **Estado objetivo:** O problema é considerado resolvido quando todas as zonas prioritárias recebem os alimentos necessários dentro dos prazos estipulados, garantindo de tal forma que as restrições dos veículos (capacidade e autonomia) e as condições do terreno (acessibilidade e meteorologia) sejam respeitadas.
- **Ações disponíveis:** As ações correspondem ao deslocamento de um veículo entre duas zonas conectadas no grafo. Estas ações estão condicionadas por fatores como:
    - Acessibilidade da rota: Algumas rotas podem estar bloqueadas ou serem inadequadas para determinados tipos de veículos, como camiões.
    - Capacidade do veículo: Cada veículo tem uma capacidade máxima (peso e volume) que não pode ser ultrapassada.
    - Autonomia: Os veículos possuem limites de combustível ou bateria, que influencia a sua mobilidade.
- **Solução esperada:** Uma solução válida consiste numa sequência ordenada de ações que permita realizar as entregas prioritárias nas zonas afetadas, minimizando o desperdício de recursos, respeitando os prazos estipulados e otimizando a eficiência global da operação.
- **Custo da solução:** O custo total de uma solução é medido através de métricas como:
    - O tempo total necessário da viagem para realizar as entregas.

4. Funcionamento do Programa

4.1. Criação do Grafo

O grafo respetivo foi estruturado como um conjunto de localidades representadas por nós e rotas representadas por arestas que simboliza a relação entre as mesmas, onde:

- Nós (localidades): representam as zonas que necessitam de mantimentos. Cada localidade possui atributos como:
  - População residente;
  - Nível de urgência (prioridade);
- Arestas (rotas): conectam localidades adjacentes e contêm informações como:
  - Distância entre localidades;
  - Tipo de pavimento;
  - Restrições de transporte;
  - Estado da rota (pode-se encontrar bloqueada ou inacessível).

Na implementação da classe '*Grafo.py'*, os métodos '*add_node'* e *'add_edge'* foram usados para adicionar nós (localidades) e arestas (rotas). Por exemplo, o nó 'Braga' foi adicionado com atributos como 'população=300' e 'urgência=5' e a rota entre 'Braga' e 'Guimarães' foi criada com uma distância de 50km e com um pavimento em asfalto.

O grafo, Figura 1, é visualizado graficamente através da funcionalidade no ficheiro 'visualização.py', que utiliza *NetworkX* e *Matplotlib.*

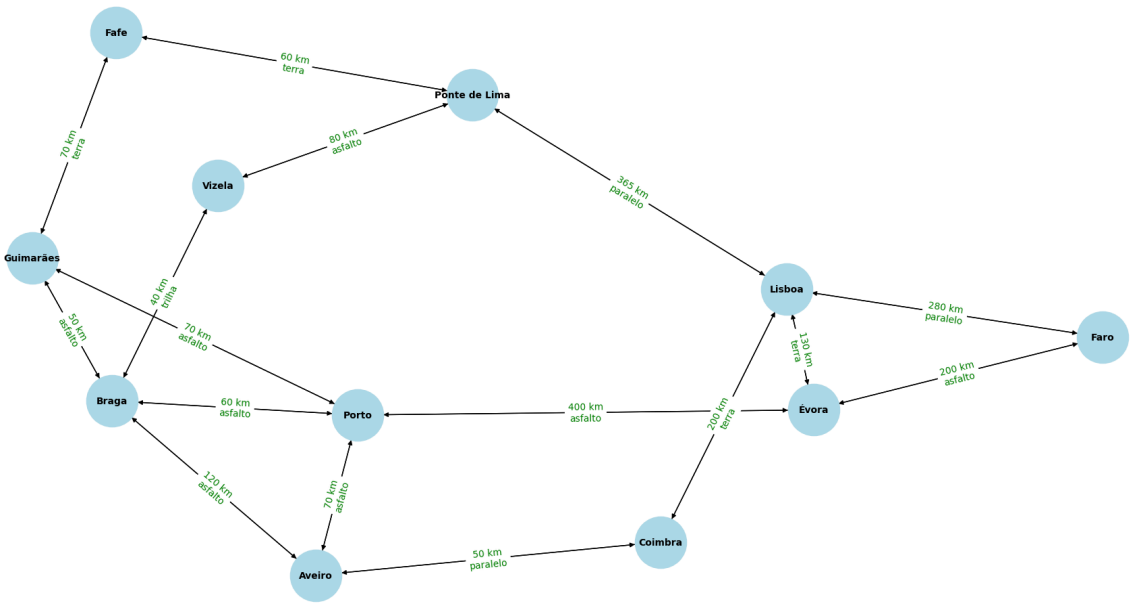

Figura 1 - Grafo Implementado

4.2. Menu

O nosso programa inclui um menu interativo, implementado no ficheiro ‘main.py’, que permite ao utilizador explorar e testar as funcionalidades do sistema. O mesmo tem como funcionalidades:

- **Visualizar Grafo:** O utilizador pode visualizar a estrutura do grafo atual com informações detalhadas sobre localidades e rotas.
- **Escolher o Algoritmo de Procura:** Disponível a seleção entre 5 algoritmos: BFS, DFS, Custo Uniforme, A*, e Greedy Search.
- **Executar o Algoritmo:**
    - O utilizador define o nó inicial (localidade de partida) e o objetivo (localidade de destino);
    - Os algoritmos calculam o melhor caminho e o custo total, com base nas condições do grafo e nos veículos disponíveis.

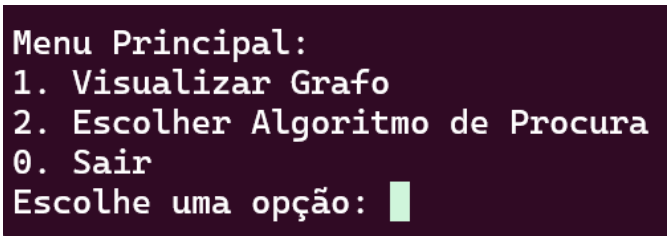

Figura 2 - Menu do Programa

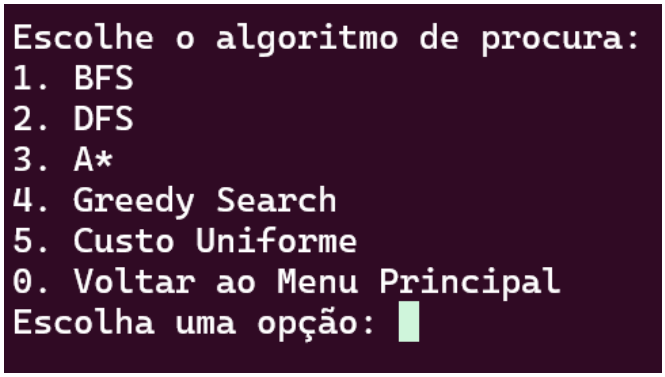

Figura 3 - Opções de Procura

4.3. Deteção de Obstáculos e Restrições

A funcionalidade de deteção de obstáculos e gestão de restrições foi implementada para tornar o programa mais realista. Os bloqueios de rotas, introduzidos por eventos aleatórios como tempestades, exigem que os algoritmos se adaptem em tempo real, considerando rotas alternativas. Adicionalmente, cada tipo de transporte tem limitações específicas, como camiões que não podem acessar trilhas ou drones que não podem operar em tempestades. Estas restrições foram cuidadosamente integradas para refletir os desafios logísticos de um cenário de catástrofe. Assim sendo, temos presentes:

- **Simulação de Eventos:** A função *'simulate_events'*, implementada no ficheiro 'main.py', introduz eventos aleatórios que afetam o estado do grafo:

    - Bloqueio de Rotas: Algumas rotas são marcadas como bloqueadas devido a tempestades ou condições adversas;
    - Alteração na Urgência: O nível de urgência de localidades é alterado para simular mudanças nas prioridades.

    Estes eventos tornam o problema mais dinâmico e desafiam os algoritmos a encontrar soluções alternativas.

- **Gestão de Restrições:** Cada tipo de transporte (camião, drone, helicóptero) possui restrições específicas para determinadas rotas:
    - Determinadas rotas por terra podem ser inacessíveis para camiões;
    - Tempestades podem bloquear rotas aéreas para os drones e helicópteros.

    As restrições são verificadas através do método *'can_access_route'* na classe 'Transporte'.

5. Estratégias de Procura

As estratégias de procura implementadas no trabalho incluem tanto algoritmos não informados como algoritmos informados. Estas estratégias foram selecionadas para explorar diferentes abordagens na resolução do problema de distribuição de mantimentos em cenários de catástrofe.

5.1. Procura em Largura (BFS)

O algoritmo de procura em largura explora todos os nós de um nível antes de passar para o próximo nível, garantindo de tal forma que encontra o caminho com o menor número de passos (em termos de número de arestas) para o objetivo.

Vantagens:

- Garante encontrar o caminho mais curto (em número de passos), se existir;
- Simples de implementar e de compreender.

Desvantagens:

- Ineficiente em termos de espaço, pois mantém em memória todos os nós de um nível;
- Não considera custos associados às arestas, como a distância ou o tempo.

No nosso trabalho, a BFS foi implementada na função *'bfs(graph, start, transport)'*, onde:

- Explora os vizinhos do nó atual utilizando uma fila (FIFO);
- Ignora rotas bloqueadas ou inacessíveis ao transporte selecionado.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^d)$
- Espaço: $O(b^d)$
- Ótima: Sim, se o custo de cada passo for 1

Onde '*b'* é o número máximo de sucessores de um nó na árvore de procura e '*d'* a profundidade da melhor solução.

5.2. Procura em Profundidade (DFS)

O algoritmo de procura em profundidade explora ao máximo um caminho antes de retroceder e tentar outras alternativas. Utiliza uma pilha (LIFO) para gerir os nós a explorar.

Vantagens:

- Utiliza menos memória do que a BFS, pois não mantém todos os nós de um nível em memória;
- Adequado para grafos muito profundos, se o caminho objetivo estiver próximo do início de uma ramificação.

Desvantagens:

- Pode entrar em loops infinitos se o grafo contiver ciclos (no nosso caso, evitado com o conjunto de nós visitados);
- Não garante o caminho mais curto.

No nosso trabalho, a DFS foi implementada na função '*dfs(graph, start, transport)'*, onde:

- Explora recursivamente os vizinhos do nó atual;
- Ignora vizinhos já visitados e rotas bloqueadas.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^m)$
- Espaço: $O(bm)$
- Ótima: Não (em princípio devolve a 1ª solução que encontra)

Onde '*b'* é o número máximo de sucessores de um nó na árvore de procura e '*m'* a máxima profundidade do espaço de estados.

5.3. Procura do Custo Uniforme

A procura do custo uniforme explora os nós de acordo com o custo acumulado desde o início, garantindo dessa forma que o caminho com o menor custo seja encontrado (considerando distâncias ou outro critério).

Vantagens:

- Garante encontrar o caminho com menor custo (ótimo), se os custos forem não negativos;
- Considera os custos das arestas, ao contrário da BFS.

Desvantagens:

- Pode ser computacionalmente dispendioso para grafos muito grandes;
- Requer mais memória do que a DFS.

No nosso trabalho, o Custo Uniforme foi implementado na função '*uniform_cost_search(graph, start, transport)'*, onde:

- Utiliza uma fila de prioridade para explorar os nós com menor custo acumulado.
- Ignora rotas bloqueadas ou inacessíveis.

As propriedades da Procura de Custo Uniforme são as seguintes:

- Tempo: $O(b^{C*/\varepsilon})$
- Espaço: $O(b^{C*/\varepsilon})$
- Ótima: Sim

Onde *'b'* é o número máximo de sucessores de um nó na árvore de procura, '*C*'* é o custo da solução ótima e *'ε'* é o menor custo de uma aresta não nula no grafo.

5.4. Procura A*

O algoritmo A* combina o custo acumulado desde o início com uma estimativa heurística do custo até o objetivo (procura gulosa com a procura do custo uniforme), sendo assim, uma procura informada. Utiliza esta função para a escolha do nó a ser explorado de seguida:

$f(n) = g(n) + h(n)$, onde:

- $g(n)$ = custo estimado
- $h(n)$ = custo estimado para chegar ao destino (heurística)

Vantagens:

- Garante encontrar o caminho com um menor custo (ótimo) se a heurística for admissível;
- É geralmente mais eficiente do que a procura do Custo Uniforme, pois considera a estimativa heurística.

Desvantagens:

- Depende da qualidade da heurística. Uma heurística má pode degradar o desempenho;
- Requer mais memória do que DFS e Custo Uniforme.

No nosso trabalho, o algoritmo A* foi implementado na função '*a_star(graph, start, heuristic, transport)'*, onde:

- Utiliza uma fila de prioridade para explorar os nós com menor custo total estimado (custo acumulado + heurística);
- A heurística baseia-se na urgência das localidades, priorizando as mais críticas.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^m)$ (com uma boa função heurística pode diminuir significativamente)
- Espaço: $O(b^m)$
- Ótima:

Onde '*b'* é o número máximo de sucessores de um modo da árvore de procura e '*m'* a máxima profundidade do espaço de estados.

5.5. Procura Gulosa (Greedy)

A procura gulosa utiliza apenas a heurística para decidir qual nó explorar a seguir, ignorando o custo acumulado. Foca-se no nó que parece mais promissor com base na heurística, sem considerar os custos já gastos para chegar até aquele ponto. O objetivo é encontrar rapidamente uma solução que parece ser promissora, mas sem garantir a otimização total do caminho.

Vantagens:

- Simples de implementar e muitas vezes rápida;
- Pode funcionar bem em problemas onde a heurística é altamente confiável.

Desvantagens:

- Não garante encontrar o caminho com o menor custo ou mesmo uma solução que seja válida;
- Pode ficar preso em caminhos que parecem promissores, mas na verdade não levam ao objetivo.

No nosso trabalho, a procura gulosa foi implementada na função '*greedy_search(graph, start, transport, heuristic)'*, onde:

- A função utiliza uma fila de prioridade para explorar os nós com base apenas na heurística, ou seja, a função não leva em consideração o custo acumulado.
- A heurística é calculada com base na urgência das localidades, priorizando as zonas mais críticas.
- Para cada nó, a função avalia o custo estimado com base na heurística e adiciona os vizinhos ao conjunto de nós a explorar, ignorando o custo já percorrido.

Esta estratégia de procura apresenta as seguintes propriedades:

- Tempo: $O(b^m)$ (com uma boa função heurística pode diminuir significativamente)
- Espaço: $O(b^m)$
- Ótima: Não, porque não encontra sempre a solução ótima.

Onde 'b' é o número máximo de sucessores de um modo da árvore de procura e 'm' a máxima profundidade do espaço de estados.

5.6. Heurística

A heurística utilizada neste projeto é um componente essencial em alguns algoritmos de procura informada, como o A* e o Greedy Search. Ela é usada para ajudar na escolha de qual caminho seguir, priorizando as zonas mais críticas com base na urgência da localidade. O objetivo da heurística é estimar o custo restante para alcançar o objetivo, tornando a busca mais eficiente e focada nas soluções mais promissoras.

A heurística é uma função de estimativa que fornece uma avaliação do custo restante entre o estado atual e o objetivo. No contexto do nosso projeto, a heurística é baseada na urgência das localidades, um fator que reflete a necessidade imediata de ajuda. A heurística para um nó é calculada de acordo com a seguinte fórmula:

$$Heurística = \frac{1}{Urgência + 1}$$

Esta função garante que as localidades com maior urgência tenham uma heurística maior, priorizando a sua visita e entrega de mantimentos. É importante ressaltar que a urgência das localidades é limitada e varia de 1 a 10, sendo que o valor 1 representa a menor urgência e o valor 10 a maior urgência.

Vantagens:

- Permite uma exploração mais direcionada do espaço de estados, priorizando as localidades mais urgentes;
- Reduz o tempo de processamento, já que evita explorar caminhos irrelevantes ou de baixo valor;
- Melhora a qualidade da solução, atendendo as localidades mais necessitadas com mais rapidez.

Desvantagens:

- A heurística pode ser imprecisa em alguns casos, especialmente se a urgência não refletir de maneira adequada a real necessidade de entrega;
- Pode não funcionar bem em ambientes dinâmicos onde as condições de urgência mudam rapidamente, o que pode tornar a estimativa inadequada durante a execução do algoritmo.

A heurística é aplicada nos algoritmos A* e Greedy Search, que são algoritmos de procura informada. Ambos os algoritmos utilizam a heurística para tomar decisões sobre qual caminho seguir, considerando a urgência das localidades como um critério de prioridade:

- A*: No A*, a heurística é combinada com o custo real acumulado (g(n)) para calcular o custo estimado total (f(n)) de alcançar o objetivo. O algoritmo procura minimizar a soma de g(n) + h(n), onde h(n) é a heurística, e g(n) é o custo real do caminho até o nó.
- Greedy Search: O Greedy utiliza a heurística para selecionar o próximo nó a ser explorado com base no menor valor de h(n), ou seja, a localidade com maior urgência será priorizada, sem considerar o custo já percorrido.

A heurística tem as seguintes propriedades:

- Admissibilidade: Uma heurística é admissível se nunca superestimar o custo real de alcançar o objetivo. No nosso caso, a heurística é admissível porque a urgência nunca pode ser negativa, e a fórmula garante que a urgência maior resulte em uma heurística menor.
- Consistência: A heurística é consistente (ou monótona) se, para qualquer nó 'n' e o seu sucessor 'n''', o valor da heurística satisfizer a condição:

$$h(n) \leq c(n, n') + h(n')$$

Onde *c(n, n')* é o custo da transição de *n* para *n'*. A nossa heurística mantém esta propriedade, pois baseia-se sempre na urgência das localidades, o que não depende de transições de custo diretas, mas sim de um fator fixo.

6. Conclusões

O desenvolvimento deste trabalho prático evidenciou o potencial dos algoritmos de procura na resolução de problemas complexos e críticos, como a distribuição de mantimentos em cenários de emergência humanitária. A implementação e análise de estratégias de procura, tanto informadas quanto não informadas, permitiram não apenas compreender as diferenças de desempenho entre os métodos, mas também identificar as vantagens específicas de cada abordagem em cenários distintos.

A modelação do problema como um grafo, considerando elementos como distâncias, acessibilidade e restrições de veículos, demonstrou-se eficaz para integrar os desafios logísticos e dinâmicos do contexto proposto. A integração de heurísticas, especialmente no algoritmo A*, destacou-se pela eficiência na priorização de áreas críticas, garantindo que as soluções encontradas respeitassem as limitações operacionais e otimizassem os recursos disponíveis.

Além disso, a inclusão de eventos dinâmicos, como bloqueios de rotas e alterações meteorológicas, reforçou a capacidade adaptativa do sistema, aproximando-o das condições reais enfrentadas em situações de catástrofe. Este componente dinâmico não apenas aumentou a complexidade do problema, mas também validou a robustez das soluções encontradas.

No entanto, devido a uma má alocação do tempo durante o desenvolvimento, não conseguimos inserir a consideração do pavimento nas estradas, o que teria sido importante para uma análise mais completa das rotas. A inclusão dessa variável teria possibilitado uma avaliação mais precisa das rotas acessíveis para diferentes tipos de transporte, como camiões, drones e helicópteros.

Em suma, este trabalho demonstrou que a aplicação de técnicas de inteligência artificial, aliadas a uma modelação rigorosa e a simulações realistas, podem oferecer soluções práticas e eficientes para problemas de grande impacto social. A experiência adquirida reforça a importância de continuar a explorar e desenvolver sistemas inteligentes para apoiar a tomada de decisão em cenários críticos, promovendo uma gestão mais eficaz de recursos e um impacto positivo nas comunidades afetadas.