Inteligência Artificial – Trabalho Prático

Distribuição de alimentos durante uma catástrofe natural

Ana Carolina Penha Cerqueira	A104188
Humberto Gil Azevedo Sampaio Gomes	A104348
João Pedro de Vasconcelos Torres	A95748
José António Fernandes Alves Lopes	A104541
José Rodrigo Ferreira Matos	A100612

3 de janeiro de 2025

Avaliação por pares

Ana Carolina Penha Cerqueira	A104188	DELTA = 0
Humberto Gil Azevedo Sampaio Gomes	A104348	DELTA = 0
João Pedro de Vasconcelos Torres	A95748	DELTA = 0
José António Fernandes Alves Lopes	A104541	DELTA = 0
José Rodrigo Ferreira Matos	A100612	DELTA = 0

1 Descrição do problema

Durante um furação na cidade de Braga, foram vários os edifícios que ficaram sem água e sem energia elétrica. Por este motivo, poucos comerciantes puderam abrir portas, e foi necessário distribuir alimentos pela cidade. Na freguesia de São Vítor, foram estabelecidos vários pontos de recolha: Tribunal, Escola Secundária Alberto Sampaio, Instituto de Nanotecnologia, Cemitério de Monte d'Arcos, Universidade de Minho, Escola de Medicina da Universidade do Minho (Olimpo) e a loja Happy China. Estes locais devem ser supridos por um centro de distribuição, que se localiza no Bar Académico.

Cada ponto de recolha tem a sua urgência, ou seja, a partir de um determinado instante de tempo, pode já ser demasiado tarde para entregar os bens necessários. Surge um problema de otimização, onde se pretende distribuir bens ao máximo número de pontos de recolha, antes de cada ponto de recolha fechar.

Cada ponto de recolha precisa de um número exato de cada produto, e o centro de distribuição faz as várias entregas utilizando um número limitado de veículos. Cada produto tem o seu peso, e cada veículo tem um limite de peso que pode transportar, sendo necessário distribuir-se os bens pelos veículos de um modo correto. Ademais, os veículos viajam a diferentes velocidades, têm diferentes limites de distância, e nem todos os veículos são capazes de viajar em todas as condições meteorológicas. Deste modo, é necessário escolher-se os veículos a utilizar para fazer cada entrega de um modo eficiente.

Figura 1: Grafo utilizado, com o centro de distribuição assinalado a azul e os pontos de recolha a vermelho.

2 Formulação do problema

2.1 Problema de distribuição

Considere-se a topologia apresentada acima, e os produtos, veículos, e pontos de recolha com as características enunciadas no anexo 10.1. Os mesmos dados podem ser encontrados em formato JSON no anexo 10.2, tal como são importados pela aplicação.

Um estado deste problema pode ser definido como $E = (t, V_c, V_a, P, (R_1, R_2, \ldots, R_n))$, onde:

- t é o instante de tempo atual;
- V_c é o multiconjunto de veículos no centro de distribuição;
- V_a é o multiconjunto de veículos em andamento;
- P é o multiconjunto de produtos no centro de distribuição;
- $R_1, R_2, \ldots, R_n$ são os pontos de recolha;

Cada veículo em andamento pode decomposto em $\beta = (t_c, v)$, onde:

- t_c é o instante de tempo em que o veículo regressa ao centro de distribuição;
- v é o veículo em viagem.

Ademais, cada ponto de recolha pode ser decomposto em $R_k = (t_k, P_k)$, onde:

- t_k é o instante de tempo máximo onde o ponto de recolha pode receber bens.
- P_k é o multiconjunto de produtos em falta no ponto de recolha.

O problema de distribuição pode ser formulado do seguinte modo:

Tipo de problema: Como as condições meteorológicas variam durante a simulação, o ambiente é não determinístico. Como o espaço de estados é conhecido, o problema é de contingência.

Estado inicial: $E = (0, V_c, \varnothing, P, (R_1, R_2, \ldots, R_7))$, onde V_c, P, e todo o R_k são definidos de acordo com os conteúdos do anexo 10.1. Exemplificando um elemento de cada tipo, tem-se:

- $V_c = \{\text{Pessoa} : 5, \text{Mota} : 2, \text{Carro} : 2\}$;
- $P = \{\text{Garrafa de água} : 200, \ldots, \text{Botija de butano} : 10\}$;
- $R_1 = (300, \{\text{Garrafa de água} : 1\})$ (tribunal);
- ...

Operadores de mudança de estado:

Uma possível mudança de estado resulta do envio de um veículo para um ponto de recolha. Considere-se um ponto de recolha $R_k = (t_k, P_k)$, um conjunto de produtos P_s tal que $P_s \subset P$ e $P_s \subset P_k$, e um veículo v cujo limite de peso não seja inferior à soma dos pesos dos produtos em P_s.

Caso exista um caminho de custo t_v entre o centro de distribuição e o ponto de recolha (ver subproblema 2.2), o veículo chegará ao ponto de recolha em t_v, e retornará ao centro de distribuição em $2t_v$, visto que o grafo do problema é orientado. Caso $t + t_v \leq t_k$ (o veículo chega ao ponto de recolha antes deste fechar), é possível a mudança para o seguinte estado:

$$E = (t, V_c - \{v\}, V_a \cup \{(t + 2t_v, v)\}, P - P_s, (\ldots, (t_k, P_k - P_s), \ldots)),$$

onde o símbolo de subtração denota a diferença entre multiconjuntos. Considerou-se, que os produtos enviados, apesar de ainda não terem chegado ao ponto de recolha, já não são precisos pelo mesmo. Tal pode ser feito porque se tem a certeza que o veículo enviado chegará a tempo ao ponto de recolha.

Outra possível mudança de estado é a espera por um veículo. Seja $\beta = (t_c, v) \in V_a$ o veículo com o menor valor de tempo de chegada (t_c) em V_a. É, então, possível transitar para o seguinte estado:

$$E = (t_c, V_c \cup \{v\}, V_a - \{\beta\}, P, (R_1, R_2, \ldots, R_n))$$

Estado objetivo: Tratando-se de um problema de otimização, o(s) estado(s) objetivo, desconhecido(s), é/são aquele(s) que apresenta(m) o menor número de pontos de recolha com pelo menos um produto em necessidade.

Custo da solução: Como o problema é de otimização, o que importa é o estado da solução, e não o custo do caminho para alcançá-lo.

2.2 Subproblema de procura de caminho

Considere-se o grafo apresentado, o nó correspondente ao centro de distribuição, o, e o nó correspondente a um ponto de recolha, d. Considere-se também um veículo com um limite de combustível $c_{\max}$, um limite de condições meteorológicas $m_{\max}$, e uma velocidade v. Um estado deste problema pode ser dado como $E = (c, n)$, um nível de combustível junto à posição atual do veículo. O problema de procura de caminho pode ser formulado do seguinte modo:

Tipo de problema: Como as condições meteorológicas variam durante a simulação, o ambiente é não determinístico. Como o espaço de estados é conhecido, o problema é de contingência.

Estado inicial: $E = (c_{\max}, o)$.

Operador de mudança de estado: Seja n' um nó adjacente a n. O combustível necessário para viajar entre n e n' é dado por $c_n = d(n, n') \times (1 + m(n, n'))$, onde d calcula a distância entre dois nós e m as condições meteorológicas numa aresta entre dois nós, um valor entre 0,0 e 1,0. Caso $c_n \leq c$, e $m(n, n') \leq m_{\max}$, é possível transitar para o estado $E = (c - c_n, n')$.

Estado final: $E = (c, d)$, onde c, o combustível restante, é um qualquer valor não-negativo.

Custo da solução: O custo da solução é o tempo de viagem total, ou seja, a soma dos tempos de viagem em cada aresta percorrida. O tempo de viagem numa aresta é dado por:

$$t_{n \to n'} = \frac{d(n, n') \times (1 + m(n, n'))}{v}$$

3 Obtenção e representação do mapa

O mapa da freguesia de São Vítor é obtido com uma *query* à *Overpass API*, do projeto *OpenStreetMap*, feita através da biblioteca Python `requests`. Foi desenvolvido um sistema de *caching* para não ser necessária a transferência do mapa no início todas as execuções da aplicação desenvolvida. Do mapa, apenas são extraídas estradas (`ways` com a *tag* `highway`), e são ignorados todos os outros tipos de via.

O *parsing* do XML obtido da *query* à API é feito com as bibliotecas `BeautifulSoup` e `lxml`. Nesta etapa, o mapa pode ser configurado como sendo um grafo orientado (os sentidos das estradas devem ser respeitados) ou não (é possível dirigir em contramão). No mapa da topologia apresentado, optou-se por esta segunda opção, uma vez que se considerou que o código de

estrada pode ser ignorado durante uma catástrofe natural onde são várias as estradas cortadas.

Um grafo é armazenado como uma matriz de adjacência esparsa. Um dicionário associa nós (origens de arestas) a outro dicionário, que associa nós (destinos de arestas) a custos. Por exemplo, a expressão `edges[o][d] = c` quer dizer que existe uma aresta entre `o` e `d` de custo `c`. Num mapa, `o` e `d` são inteiros, identificadores de nós na base de dados do *OpenStreetMap*, e `c` é um valor de vírgula flutuante. Um mapa armazena também outro dicionário, que associa identificadores de nós às suas coordenadas. Estas, durante a importação do mapa, são calculadas, de acordo com a projeção de Mercator, com base na latitude e longitude de cada nó.

Por último, as condições meteorológicas num mapa são representadas por outro dicionário, que associa arestas (pares origem-destino) a valores de vírgula flutuante entre 0.0 e 1.0, os extremos do bom e do mau tempo, respetivamente. Para uma geração de condições meteorológicas em gradiente (sem grandes picos ou vales nos valores), ruído Perlin (através da biblioteca Python `perlin-noise`) foi utilizado.

4 Algoritmos de procura

Foram implementados diversos algoritmos de procura, para poderem ser comparados em aspetos de qualidade da solução gerada e de desempenho computacional. Nesta secção, procura-se enumerar os algoritmos implementados e descrever algumas das otimizações desenvolvidas.

4.1 Procura em profundidade

Este algoritmo foi implementado com algumas diferenças em relação à sua implementação *standard*. Em primeiro lugar, é armazenado um conjunto de nós visitados, que é utilizado para garantir que o algoritmo termina, mesmo em grafos com ciclos. De seguida, o algoritmo também foi implementado iterativamente, e não recursivamente. Devido ao grande tamanho de uma *frame* da *stack* em Python, o limite de chamadas recursivas é bastante baixo, pelo que a implementação iterativa se provou necessária para caminhos longos. Por último, o algoritmo não expande nós que ultrapassem o limite de combustível do veículo a realizar a procura, diminuindo o número de nós visitados quando não existe um caminho que o veículo seja capaz de realizar.

4.2 Procura iterativa

Este algoritmo, através de chamadas sucessivas ao algoritmo de procura em profundidade (com uma profundidade máxima definida), encontra o caminho entre dois nós com o menor número de saltos. No entanto, o algoritmo de procura em profundidade descrito anteriormente não pode ser utilizado para uma procura iterativa. O uso de um conjunto para registar os nós visitados e evitar ciclos faz com que, após *back tracking*, alguns caminhos possíveis não sejam considerados, uma vez que envolvem nós já visitados. Logo, a verificação de ciclos na procura em profundidade limitada é feita através da consulta do caminho até ao nó atual (e se a sua expansão não causará um ciclo) [1]. Esta verificação mais lenta e o maior número de nós visitados (é possível que o mesmo nó seja visitado várias vezes) fazem com que este algoritmo exija um tempo de processamento muito superior aos restantes, pelo que não será considerado para os testes realizados.

4.3 Procura em largura

Este algoritmo encontra o caminho entre dois nós com o menor número de saltos. Para um melhor desempenho, foi usada uma lista duplamente ligada para a implementação da fila de espera deste algoritmo. Deste modo, tanto as operações de *push* como *pop* podem ser executadas em tempo constante. Tal como na procura em profundidade, o algoritmo não expande nós que ultrapassem o limite de combustível do veículo a realizar a procura.

4.4 Procura de custo uniforme / Algoritmo de Dijkstra

Este algoritmo encontra o caminho com menor custo entre dois nós. Para a implementação da fila de prioridades necessária para este algoritmo, foi utilizado um *heap* binário, que apresenta melhor desempenho do que listas. Tal como os outros algoritmos, não são expandidos nós fora do alcance do combustível do veículo a realizar a procura, diminuindo o número de nós visitados quando não existe um caminho que o veículo seja capaz de realizar.

4.5 Procura gulosa

Este algoritmo é muito semelhante à procura em profundidade, com a diferença de que a ordem de expansão dos vizinhos de um nó é determinada por uma heurística. Tal como na implementação da procura em profundidade, o algoritmo foi implementado iterativamente,

e também não expande nós fora do alcance do combustível do veículo a realizar a procura. Uma outra otimização, exclusiva à procura gulosa, relaciona-se com a heurística de distância cartesiana. Como os valores da heurística apenas são utilizados para comparação entre si, visto que a raiz quadrada é uma função monótona crescente, o seu cálculo não é necessário, melhorando o desempenho do algoritmo:

$$dx_1^2 + dy_1^2 < dx_2^2 + dy_2^2 \Rightarrow \sqrt{dx_1^2 + dy_1^2} < \sqrt{dx_2^2 + dy_2^2}$$

4.6 A*

A única diferença entre este algoritmo e o algoritmo de Dijkstra é a ordem dos nós na fila de espera, que passam a estar ordenados pela soma da sua distância à origem da procura com a sua heurística. Quando a heurística é cartesiana, já não é possível evitar o cálculo das raízes quadradas como na procura gulosa, visto que é necessário que a heurística não sobreestime o custo entre dois nós caso se deseje que o algoritmo devolva a solução ótima de um problema de procura [1].

4.7 Comparação dos algoritmos de procura

Para comparar o desempenho e a qualidade das soluções dos vários algoritmos, estes foram executado, no grafo apresentado, entre o centro de distribuição e os vários pontos de recolha de bens. Os testes foram realizados com as condições meteorológicas desativadas, considerando um veículo de alcance ilimitado. Ademais, foram testados todos os algoritmos com exceção da procura iterativa (devido ao seu longo tempo de execução), e os algoritmos de procura informada foram testados com duas heurísticas, uma baseada na distância cartesiana e a outra na distância de Manhattan. Foram várias as métricas recolhidas, e os dados completos podem ser observados no anexo 10.3. Aqui, apresentam-se as somas das métricas para todos os destinos, procurando-se apresentar uma visão mais geral dos resultados obtidos:

	Σ_{Tempo} (ms)	Σ_{Custo}	$\Sigma_{N\text{nós}}$	$\Sigma_{N\text{visitas}}$
DFS	72,78	196564,98	8408	31799
BFS	37,01	13118,15	356	29391
Dijkstra	97,58	12411,22	405	36109
Greedy (Cart)	73,47	40882,16	1805	12357
Greedy (Man)	52,14	20596,48	732	9468
A* (Cart)	25,58	12411,22	405	7570
A* (Man)	7,67	12596,46	428	2830

Tabela 1: Somas das diversas métricas recolhidas nos testes dos vários algoritmos de procura.

Como esperado, tanto o algoritmo de Dijkstra como o A* (com heurística cartesiana) são capazes de encontrar as soluções ótimas dos problemas de procura. No entanto, o A* não exige nem tanto tempo de execução nem tanta memória, uma vez que o uso da heurística regula a ordem de visita dos nós e permite que um menor número destes tenha de ser visitado.

Quando o A* é utilizado com a heurística baseada na distância de Manhattan, nem sempre resulta na solução ótima. Isso é esperado, visto que apenas é garantido que o algoritmo A* seja ótimo quando a heurística não sobreestima o custo entre nós [1], o que não é verdade com a distância de Manhattan. No entanto, esta heurística é de cálculo mais rápido e, no grafo utilizado, reduz o número de nós visitados significativamente, diminuindo consideravelmente o tempo de execução e o uso de memória do A* (quando comparado com a heurística cartesiana). Considerando isto e o facto de que as soluções obtidas apresentam um custo muito próximo do ótimo, este algoritmo é o que melhor equilibra entre tempo de execução e qualidade da solução.

Nos problemas de procura apresentados, caso não fosse possível implementar algoritmos de procura informada, a procura em largura não seria uma má escolha de algoritmo. Ao contrário do algoritmo de Dijkstra, a procura em largura não é ótima, visto que o grafo é pesado. No entanto, as soluções encontradas são próximas da ótima, e tanto o tempo de processamento necessário como os requisitos de memória (número de visitas) são menores do que os do algoritmo de Dijkstra, tornando este algoritmo viável caso a solução ótima não seja necessária. Uma forma de obter a solução ótima com a procura em largura seria uma pré-discretização do grafo, ou seja, inserir nós em arcos longos para garantir que todos os arcos têm o mesmo custo / comprimento.

Por último, os algoritmos de procura em profundidade e de procura gulosa foram os que apresentaram as piores soluções, visto que não têm em consideração o custo das soluções encontradas, e apenas se "preocupam" com arranjar uma solução satisfatória. A procura gulosa, devido ao uso de uma heurística, gera, a partir da origem, caminhos na direção do destino, diminuindo o seu

comprimento (e consequentemente o custo da solução) e o número de nós visitados (requirindo menos memória). Por outro lado, a procura em profundidade, como não utiliza uma heurística, várias vezes gera uma solução que consiste em dar uma "volta" ao mapa antes de chegar ao destino.

5 Algoritmo para empacotamento

Para o envio de recursos para um local, é necessário distribuí-los por veículos, Surge um problema de empacotamento, onde se procura distribuir vários produtos, cada um com um peso, por vários veículos, cada um com um limite de peso e um custo de envio, procurando maximizar-se o peso enviado, enquanto se minimiza o custo de envio dos veículos.

Na aplicação desenvolvida, este problema é resolvido com um algoritmo genético. Um indivíduo, que correspondente a uma distribuição dos produtos pelos veículos, é constituído por tantos genes quantos produtos. Cada gene é um número inteiro, e corresponde ao veículo em que o produto deve ser colocado. Segue-se, abaixo, o exemplo de um indivíduo, onde todos os produtos são colocados no veículo 0, com exceção do 3º produto, que é colocado no veículo 1.

0	0	1	0	0

Tabela 2: Exemplo um indivíduo no algoritmo genético utilizado para empacotamento.

A função de *fitness* utilizada devolve 0 caso pelo menos um dos veículos ultrapasse o seu limite de preço. Quando esta condição não é violada, procura-se, por esta ordem, maximizar o peso enviado, minimizar o custo dos veículos enviados, e minimizar o número de veículos enviados. Para ordenar estes valores, garantiu-se, através de multiplicações, que estes estavam em diferentes ordens de grandeza:

$$\mathcal{F}(X) = \begin{cases} 0, & \text{se há veículos sobrecarregados} \\ \text{Peso}(X) \times 10^6 - \text{Custo}(X) - \text{N}_{\text{veículos}}(X), & \text{caso contrário} \end{cases}$$

A operação de cruzamento é simples: é escolhido um índice onde os cromossomas dos pais são divididos, e geram-se dois filhos, cada um com uma primeira parte do cromossoma de um dos pais e com uma segunda parte do outro. A função de mutação, no entanto, já é mais complexa. Uma possível mutação é a alteração do veículo no qual um produto é colocado. No entanto,

para evitar que o algoritmo convirja para colocar todos os produtos num veículo de maior custo quando um de menor custo existe, uma possível mutação é a alteração de veículo de todos os produtos no mesmo veículo.

6 Resolução do problema de distribuição

O problema de distribuição é um problema de otimização computacionalmente complexo de resolver. Só a avaliação de uma solução exige a resolução de vários subproblemas de procura e de empacotamento. Logo, o algoritmo desenvolvido para o resolver é guloso, e apenas calcula e avalia uma única solução. No entanto, o algoritmo "pondera" bastante sobre cada passo da solução, procurando diminuir ineficiências em cada um. O algoritmo pode ser descrito pelos seguintes passos:

1. Caso não haja veículos no centro de distribuição, o algoritmo espera pelo primeiro veículo a chegar. O algoritmo suporta (e tira partido) de vários veículos que chegam em simultâneo.
2. O algoritmo calcula caminhos para o ponto de recolha de maior urgência, testando todos os tipos de veículo disponíveis.
3. Caso nenhum veículo seja capaz de chegar ao destino a tempo, o algoritmo decide entre desistir do centro de distribuição em teste e esperar por mais veículos. Irá escolher a segunda opção se e só se houver veículos à espera diferentes dos presentes no centro de distribuição.
4. Caso haja veículos capazes de chegar a tempo ao ponto de recolha, o algoritmo de empacotamento é executado, e procura-se enviar o peso máximo possível nos veículos disponíveis.
5. Caso não haja mais produtos a entregar a um centro de distribuição, ele é descartado, e o algoritmo procurará suprir o de maior prioridade seguinte.

Este algoritmo é de execução rápida, e encontra, na maior parte das vezes, a solução ótima do problema dado. As raras vezes restantes podem ser atribuídas a "azar" na geração de números aleatórios para o algoritmo genético de empacotamento ou para as alterações meteorológicas dinâmicas.

Apesar deste algoritmo simples encontrar boas soluções para o problema dado, o mesmo poderia não ser verdade para outros problemas, com diferentes veículos e combinações de produtos. Este algoritmo assume que o algoritmo de empacotamento deixará pouco espaço vazio nos veículos. Assim, serão raras situações onde possa compensar enviar um veículo que passará por dois

pontos de recolha, mas que demorará mais tempo a regressar ao centro de distribuição. Caso os veículos tivessem capacidades muito superiores, ou os pontos de recolha tivessem necessidade de um menor número de produtos, a possibilidade de enviar um veículo a mais do que um ponto de recolha já deveria ser considerada. Para avaliar se seria mais compensador enviar um veículo a passar por mais do que um ponto ou esperar pelo seu regresso antes de o reenviar, o algoritmo para resolução do problema de distribuição teria de avaliar vários caminhos na árvore de estados, um processo mais complexo e mais computacionalmente intensivo. Alternativamente, poderia ser possível o desenvolvimento de um heurística para determinar que nó deve ser visitado de seguida, mas a conceção de uma boa heurística o mais universal possível é bastante complicada.

7 Apresentação da solução do problema

Quando a aplicação desenvolvida inicia, conforme os argumentos providenciados, pode ser gerada uma solução de um problema de distribuição, ou pode ser carregada uma solução de um ficheiro. Em ambos os casos, a solução será apresentada ao utilizador na interface que se procura descrever nesta secção. Esta interface foi desenvolvida com recurso à biblioteca `pygame`.

Inicialmente, é apresentado o mapa com o centro de distribuição e os pontos de recolha. A tecla `W` pode ser utilizada para alternar entre mostrar as condições meteorológicas ou não. As setas do teclado podem utilizadas para mover o mapa na janela, e tanto o *scroll* do rato como as teclas `+` e `-` podem ser utilizadas para controlar o fator de ampliação. Clicar num nó resulta na impressão de uma hiperligação para a saída textual do programa. Esta hiperligação redireciona o utilizador para o *Open Street Maps*, permitindo que este veja em maior detalhe o mapa associado ao grafo apresentado.

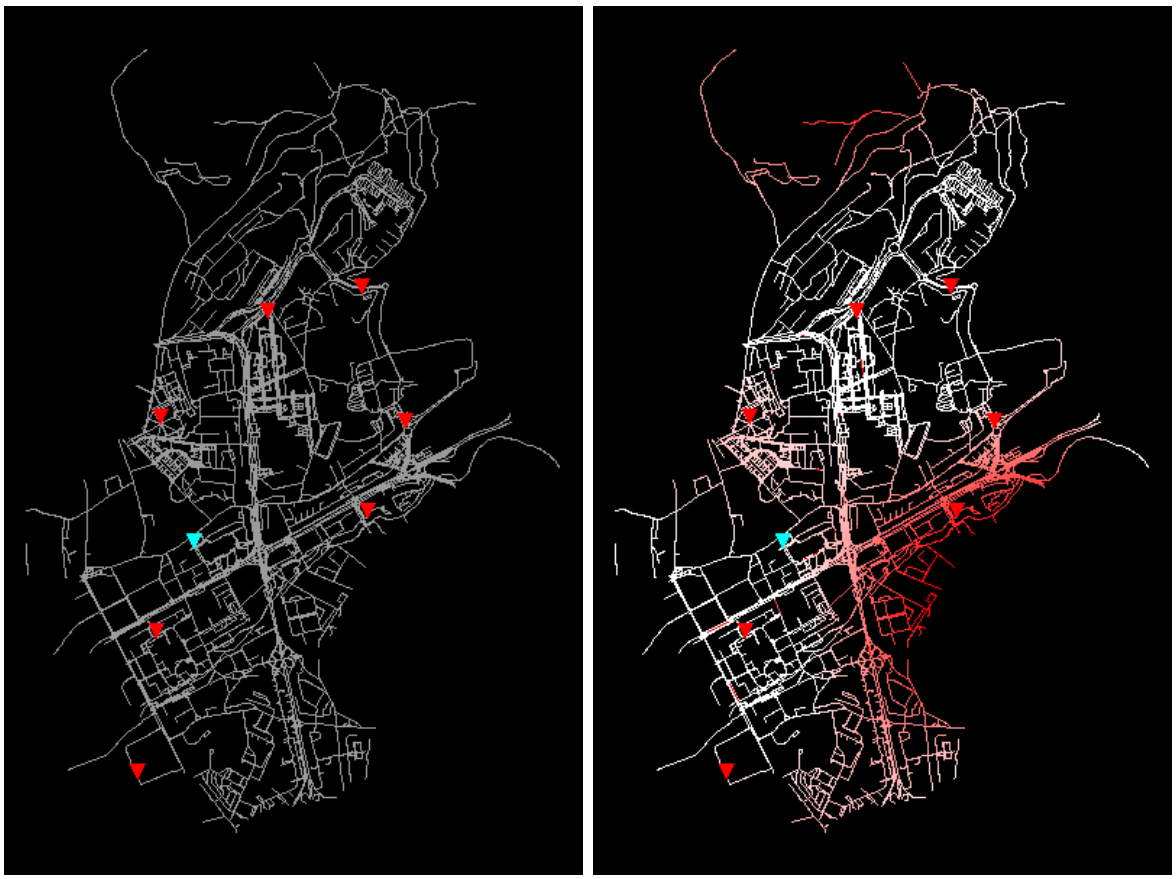

Figura 2: Apresentação do centro de distribuição e dos pontos de recolha na UI, sem e com condições meteorológicas visíveis.

As teclas `Enter` e `Backspace` são utilizadas para navegar entre os diapositivos da apresentação da solução. Um diapositivo possível é a apresentação dos resultados de um algoritmo de procura. O caminho, quando existente, é apresentado a azul, e os nós visitados são apresentados a vermelho. Quando premida a tecla `A`, a interface anima o funcionamento do algoritmo, mostrando os nós pela ordem que foram visitados.

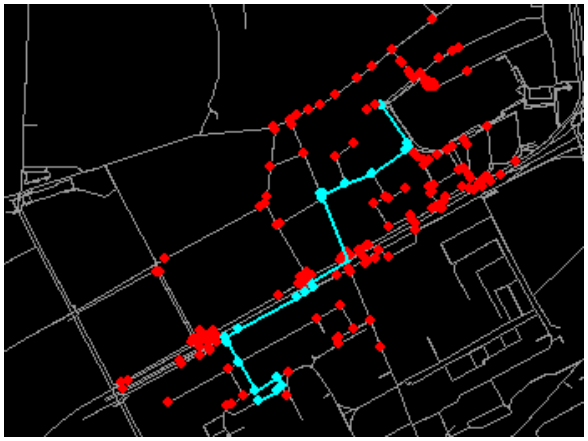

Figura 3: Apresentação dos resultados de um algoritmo de procura na UI.

A interface também é capaz de apresentar o resultado do algoritmo de empacotamento, a satisfação de entregas aos pontos de recolha, e a incapacidade de satisfazer um ponto de recolha a tempo.

Figura 4: Apresentação dos resultados do algoritmo de empacotamento, da satisfação de uma entrega a um ponto de recolha, e da incapacidade de satisfazer um ponto de recolha a tempo.

8 Conclusão

Em suma, neste projeto, fomos capazes de abstrair um problema real, formalizando-o como um problemas de grafos. Implementámos diversos algoritmos de procura e adaptá-mo-los ao contexto em que foram utilizados, o que permitiu que os mesmos fossem melhor otimizados do que suas versões genéricas. Além disso, realizámos testes empíricos a comparar estes algoritmos, e chegámos a conclusões que corroboram os conhecimentos teóricos na área: entre os algoritmos testados, o A* foi consistentemente o mais veloz, visto que a adição de uma heurística ao algoritmo de Dijkstra permite diminuir significativamente o número de nós visitados. Ademais, também utilizámos este trabalho prático como oportunidade para implementar algoritmos lecionados na UC de Inteligência Artificial, mas nunca implementados nas aulas práticas, como o algoritmo genético que desenvolvemos para empacotamento. Por último, desenvolvemos uma UI que mostra o resultado do problema resolvido, e serve também como ferramenta pedagógica

no ensino dos vários algoritmos de procura. Inclusive, ajudou-nos a depurar erros nas nossas implementações dos algoritmos de Dijkstra e A*.

Apesar de serem várias as funcionalidades implementadas, existem possíveis pontos de melhoria no nosso trabalho. Em primeiro lugar, o algoritmo genético para empacotamento beneficiaria de alguma afinação, visto que há certos cenários de teste onde tem muita dificuldade a chegar à solução ótima por muito que se aumente o número de gerações, sinalizando que mudanças à função de *fitness* e às operações de seleção, cruzamento e mutação possam ser bem-vindas. Ademais, seria interessante comparar a qualidade das soluções deste algoritmo com as soluções resultantes do uso estratégias gulosas bem conhecidas. Por último, também é possível melhorar o algoritmo que decide por que ordem os produtos são distribuídos pelos pontos de recolha, com a adição da possibilidade do mesmo veículo visitar mais do que um ponto de recolha, e a implementação de uma heurística / exploração de vários ramos para chegar a uma melhor solução do problema.

9 Bibliografia

[1] S. Russel and P. Norvig, *Artificial Intelligence – A Modern Approach*, 4th ed. USA: Pearson, 2021.

10 Anexos

10.1 Dados para a resolução do problema

	Combustível (m)	Peso máximo (kg)	Pior tempo	Velocidade (m/s)	Quantidade disponível
Pessoa	2000,0	10,0	1,0	2,0	5
Moto	6000,0	20,0	0,9	1,5	2
Carro	10000,0	30.0	0,7	1,0	2

Tabela 3: Características dos veículos.

Considerou-se que, durante uma catástrofe naturais, quanto maior o veículo, maior será a dificuldade de o manobrar em mau tempo, pelo que carros e motocicletas são mais lentas do que pessoas a pé.

	Peso (kg)	Quantidade disponível
Garrafa de água	2,0	200
Saco de farinha	3,0	100
Carne congelada	4,0	40
Vegetais congelados	5,0	40
Botija de butano	9,0	10

Tabela 4: Características dos produtos.

	Tempo limite (s)
Tribunal	300,0
ESAS	2000,0
INL	7000,0
Cemitério	7500,0
UMinho	8000,0
Olimpo	12000,0
Happy China	15000,0

Tabela 5: Instantes de tempo limite onde os pontos de recolha podem receber bens.

	Garrafa de água	Saco de farinha	Carne congelada	Vegetais congelados	Botija de butano
Tribunal	1	0	0	0	0
ESAS	5	2	1	0	1
INL	0	0	1	1	1
Cemitério	0	0	0	0	3
UMinho	6	3	2	2	0
Olimpo	0	30	0	0	0
Happy China	1	0	0	0	0

Tabela 6: Necessidades de cada ponto de recolha

10.2 Dados para a resolução do problema em format JSON


```
{
    "map": {
        "location": "Braga (São Vítor)",
```

```
    "directed": false,
    "enable-weather": true
},

"distribution-center": {
    "name": "Bar Académico",
    "node": 439359565,
    "vehicles": {
        "person": 5,
        "motorcycle": 2,
        "car": 2
    },
    "products": {
        "Water bottle": 200,
        "Bag of flour": 100,
        "Frozen meat": 40,
        "Frozen vegetables": 40,
        "Butane bottle": 10
    }
},

"delivery-targets": {
    "Tribunal": {
        "name": "Tribunal",
        "node": 1605296772,
        "time-limit": 300,
        "products": {
            "Water bottle": 1
        }
    },
    "Escola Secundária Alberto Sampaio": {
        "name": "Escola Secundária Alberto Sampaio",
        "node": 4090416170,
        "time-limit": 2000,
        "products": {
            "Water bottle": 5,
            "Bag of flour": 2,
            "Frozen meat": 1,
            "Butane bottle": 1
        }
    },
    "Instituto de Nanotecnologia": {
        "name": "Instituto de Nanotecnologia",
```

```
        "node": 1607596811,
        "time-limit": 7000,
        "products": {
            "Frozen meat": 1,
            "Frozen vegetables": 1,
            "Butane bottle": 1
        }
    },
    "Cemitério": {
        "name": "Cemitério",
        "node": 2361505854,
        "time-limit": 7500,
        "products": {
            "Butane bottle": 3
        }
    },
    "Universidade do Minho": {
        "name": "Universidade do Minho",
        "node": 227110575,
        "time-limit": 8000,
        "products": {
            "Water bottle": 6,
            "Bag of flour": 3,
            "Frozen meat": 2,
            "Frozen vegetables": 2
        }
    },
    "Olimpo": {
        "name": "Olimpo",
        "node": 1193997031,
        "time-limit": 12000,
        "products": {
            "Bag of flour": 30
        }
    },
    "Happy China": {
        "name": "Happy China",
        "node": 4674683655,
        "time-limit": 15000,
        "products": {
            "Water bottle": 1
        }
    }
```

```
},

"vehicles": {
    "person": {
        "name": "person",
        "max_fuel": 2000.0,
        "max_weight": 10.0,
        "worst_weather": 1.0,
        "speed": 2.0,
        "image": "res/person.bmp"
    },
    "motorcycle": {
        "name": "motorcycle",
        "max_fuel": 6000.0,
        "max_weight": 20.0,
        "worst_weather": 0.9,
        "speed": 1.5,
        "image": "res/motorcycle.bmp"
    },
    "car": {
        "name": "car",
        "max_fuel": 10000.0,
        "max_weight": 30.0,
        "worst_weather": 0.7,
        "speed": 1.0,
        "image": "res/car.bmp"
    }
},

"products": {
    "Water bottle": {
        "name": "Water bottle",
        "weight": 2.0,
        "color": [0, 255, 255]
    },
    "Bag of flour": {
        "name": "Bag of flour",
        "weight": 3.0,
        "color": [200, 200, 200]
    },
    "Frozen meat": {
        "name": "Frozen meat",
        "weight": 4.0,
```

```
            "color": [150, 0, 0]
        },
        "Frozen vegetables": {
            "name": "Frozen vegetables",
            "weight": 5.0,
            "color": [0, 0, 100]
        },
        "Butane bottle": {
            "name": "Butane bottle",
            "weight": 9.0,
            "color": [200, 200, 0]
        }
    }
}
```

10.3 Resultados da comparação dos algoritmos de procura

Tempo (ms)	Tribunal	ESAS	INL	Cemitério	UMinho	Olimpo	Happy China	Σ
DFS	0,54	21,73	6,32	15,55	6,99	10,06	11,59	72,78
BFS	1,09	4,85	2,35	3,11	7,24	9,87	8,5	37,01
Dijkstra	5,87	18,69	10,04	8,24	16,59	20,87	17,28	97,58
Greedy (Cart)	0,2	55,76	0,37	15,53	0,84	0,45	0,32	73,47
Greedy (Man)	0,14	0,52	0,33	49,75	0,78	0,37	0,25	52,14
A* (Cart)	0,65	2,55	1,61	3,25	1,62	12,85	3,05	25,58
A* (Man)	0,36	1,35	1,17	2,55	0,54	1,04	0,66	7,67

Tabela 7: Tempo de CPU para a execução de cada algoritmo de procura, entre o Bar Académico e os destinos apresentados.

Custo	Tribunal	ESAS	INL	Cemitério	UMinho	Olimpo	Happy China	Σ
DFS	5315,16	4970,64	32428,01	40269,65	28219,21	42881,52	42480,79	196564,98
BFS	1010,32	2200,89	1683,98	1250,39	2242,19	2741,26	1989,12	13118,15
Dijkstra	1010,32	2158,87	1432,5	1250,39	1884,8	2717,2	1957,14	12411,22
Greedy (Cart)	1277,28	2305,31	2250,09	23491,25	4802,74	3797,38	2958,11	40882,16
Greedy (Man)	1291,62	2712,37	2671,13	2231,37	4934,5	3797,38	2958,11	20596,48
A* (Cart)	1010,32	2158,87	1432,5	1250,39	1884,8	2717,2	1957,14	12411,22
A* (Man)	1028,1	2179,62	1566,73	1250,39	1884,8	2717,2	1969,62	12596,46

Tabela 8: Custo das soluções dos vários algoritmos de procura, entre o Bar Académico e os destinos apresentados.

Nós na solução	Tribunal	ESAS	INL	Cemitério	UMinho	Olimpo	Happy China	Σ
DFS	226	199	1384	1697	1157	1884	1861	8408
BFS	25	44	33	37	56	95	66	356
Dijkstra	25	68	35	37	74	96	70	405
Greedy (Cart)	45	49	91	1228	176	128	88	1805
Greedy (Man)	41	63	101	121	190	128	88	732
A* (Cart)	25	68	35	37	74	96	70	405
A* (Man)	26	58	68	37	74	96	69	428

Tabela 9: Número de nós nas soluções dos vários algoritmos de procura, entre o Bar Académico e os destinos apresentados.

Nós visitados	Tribunal	ESAS	INL	Cemitério	UMinho	Olimpo	Happy China	Σ
DFS	255	8888	2881	6839	3136	4547	5253	31799
BFS	914	3857	2048	2593	5646	7738	6595	29391
Dijkstra	2379	6779	3909	3210	5929	7734	6169	36109
Greedy (Cart)	46	8885	99	2894	216	129	88	12357
Greedy (Man)	42	120	101	8766	221	130	88	9468
A* (Cart)	216	786	537	1061	519	3501	950	7570
A* (Man)	133	453	457	947	220	362	258	2830

Tabela 10: Número de nós visitados na execução dos vários algoritmos de procura, entre o Bar Académico e os destinos apresentados.