Trabalho Prático - Fase 2

Diogo Pires, a93239 Gonçalo Soares, a93286
Marco Costa, a93283 Rita Teixeira, a89494

4 de agosto de 2023

Inteligência Artificial

Licenciatura em Engenharia Informática

Índice

Capítulo 1

Introdução

Na segunda fase deste trabalho procuramos explorar diferentes algoritmos de procura em grafos, bem como aprofundar outros conceitos da primeira fase. Para isso, criámos grafos e pontos de entrega nesses grafos, e usamos diferentes algoritmos para escolher caminhos entre dois pontos de entregas de encomendas. A partir desse caminho, escolhemos qual o veículo mais adequado para a entrega, e guardamos a informação relativa a essa entrega, bem como o percurso realizado.

Neste relatório começaremos por expor a formulação que fizemos do problema proposto pela equipa docente. Seguidamente, faremos referência às alterações que foram feitas ao trabalho que foi entregue na primeira fase do projeto. Posteriormente, iremos referir os algoritmos implementados no programa, assim como a heurística usada nos algoritmos de pesquisa informada. De seguida, será explicada a implementação das várias funcionalidades do programa. Imediatamente a seguir iremos mostrar alguns exemplos de execução do programa, bem como, alguns testes feitos pelo grupo. Por fim, faremos uma breve conclusão relativa ao trabalho feito.

Capítulo 2

Desenvolvimento

2.1 Formulação do problema

Nós formulamos este problema com um problema de agente único que pretende encontrar uma solução, de cada vez. Isto porque podemos ter vários estafetas a fazer entregas ao mesmo tempo, mas eles não se afetam uns aos outros. O tipo de informação é perfeita, porque o agente conhece o mapa desde o início do planeamento do circuito e é determinístico, porque não existe qualquer aleatoriedade no mapa. Também é estático, porque os locais de encomenda não se alteram, e discreto.

2.1.1 Estado inicial

O estado inicial é o estafeta no centro de distribuições da *Green Distribution*, com as encomendas por distribuir numa dada cidade.

2.1.2 Estado objetivo

O estado final ou objetivo é o estafeta no centro de distribuições da *Green Distribution*, com as encomendas atribuídas e possíveis de ser transportadas já entregues.

2.1.3 Operadores

Nome: Estafeta muda de local

- **Pré-condições:** Estafeta encontra-se num local válido do grafo.
- **Efeitos:** Estafeta muda o seu local atual, para outro que esteja conectado por um caminho.

Nome: Entrega uma encomenda.

- **Pré-condições:** O local onde o estafeta se encontra tem uma encomenda atribuída, no dia da entrega.
- **Efeitos:** A encomenda é considerada entregue, e o estafeta tem menos um sítio para visitar, e menos uma encomenda para entregar.

2.2 Alterações na base de conhecimento

Para podermos tratar dos caminhos, tivemos de acrescentar alguns campos às várias entidades que usamos durante o programa. Para além disso, agora criámos as entregas durante a execução do programa, tal como os circuitos.

2.3 Algoritmos de procura

Para o sistema encontrar um caminho possível para o estafeta, este dispõe de cinco algoritmos de procura diferentes, (dois de pesquisa informada e três de pesquisa não informada). O funcionamento dos algoritmos é inspirado nas aulas teóricas e práticas, e as complexidades temporais e espaciais estão descritas no fim do relatório.

2.3.1 Pesquisa não informada

Os algoritmos de pesquisa não informada, dividem-se em dois tipos: *depth-first* e *breadth-first*. Estes distinguem-se na forma como expandem os nodos, sendo que o *depth-first* se foca em expandir os nodos dos níveis inferiores primeiro, enquanto que o *breadth-first* dá prioridade aos do nível atual.

Uma caraterística dos algoritmos de pesquisa não informada é não terem informação sobre o grafo em geral, por isso não são tão eficientes quanto os algoritmos de pesquisa não informada. Os três primeiros algoritmos de pesquisa não informada não tem em conta as distâncias entre nodos, e são úteis porque descobrem se existe conexão entre dois locais. Sendo assim, as soluções que chegam para os caminhos não são as melhores em termos de custo, mas também são menos complexas.

Pesquisa em largura

Para a pesquisa em largura, procuramos visitar todos os nodos por níveis, acrescentando os nodos vizinhos a uma *queue* de nodos. Enquanto não encontramos o nodo de destino, continuamos a ir buscar nodos à *queue*.

Pesquisa em profundidade

Para a pesquisa em profundidade, utilizamos uma função recursiva, adaptando a função dada nas aulas. Começamos por procurar os nodos ligados à origem, e guardá-los numa *queue*. Depois, retiramos o primeiro nodo da *queue*, e procuramos outra vez, recursivamente, com a nova origem ser o nodo seguinte da *queue*. Nós guardamos a lista antes da chamada recursiva para o caso de a procura não chegar ao nodo pretendido, imitando o *back-trace* do prolog.

Busca Iterativa Limitada em Profundidade.

O funcionamento deste tipo de procura é muito parecido com o da pesquisa em profundidade, apenas alteramos uma componente da função para saber até que profundidade vai procurar. Assim, incrementamos sempre em um valor o nível de procura, até chegar à solução, caso exista.

Dijkstra

O algoritmo de *Dijkstra* é dos algoritmos mais famosos para encontrar o caminho mais curto. Baseia-se na análise do grafo para construir a árvore de caminhos mais curtos entre o nodo inicial e os outros nodos. Este algoritmo garante a descoberta do melhor caminho (se existir algum caminho).

2.3.2 Pesquisa informada

Os algoritmos de pesquisa informada diferem dos não informados pelo facto de que usam uma heurística que os ajuda a avaliar quais os melhores nodos a expandir. Por isto, estes algoritmos são do tipo *depth-first*, pois focam-se em expandir, apenas o nodo mais atrativo de cada nível. Isto ajuda na redução do tempo de execução bem como na memória consumida, pois não precisam de expandir todos os nodos, usando uma heurística como um critério de escolha. Em contrapartida, a desvantagem deste tipo de algoritmos é que não garante o melhor resultado. Mais ainda, a qualidade do resultado é dependente da qualidade da heurística, ou seja, uma heurística má gera resultados maus.

Heurística

Para a pesquisa informada foi necessário criar uma heurística que permita estimar o custo de seguir esse caminho. Para tal, foi decidido que a heurística que produz melhores resultados será a da distância em linha reta do nodo ao destino. Consideramos isto porque, a distância entre dois pontos é um cálculo que um computador consegue executar rapidamente enquanto faz a pesquisa. Isto evita que se crie uma tabela para cada nodo com as estimas de se chegar a todos os outros nodos, que teria de ser recalculado cada vez que se adiciona-se um nodo ao grafo.

Para além disso, a distância em linha reta, para o problema em questão, é uma heurística admissível, visto que a estima apenas pode ser mais baixa, (caso o nodo esteja próximo ao destino mas que ainda tenha de dar uma volta grande), ou igual ao custo real de chegar ao destino (caso haja uma ligação direta ao nodo destino).

Gulosa

O algoritmo guloso baseia-se no uso da estima do custo de um nodo para avaliar qual o melhor nodo a expandir. Este algoritmo tem como falha não considerar o caminho já percorrido, por isso não chega a resultados tão bons como os do A*. Para além disso, pode ser levado por maus caminhos caso a estima do nodo não seja representativa do custo até ao destino.

A*

O algoritmo *A** é das escolhas mais populares para a descoberta de caminhos. Este baseia-se na utilização de uma heurística do ponto atual para o nodo destino, que podemos designar como *h(nodo)*. Também é usado o custo do caminho desde do ponto inicial até o nodo atual, conhecido como *g(nodo)*. A forma como combinamos estes dois factores é através da função *f(nodo)* que é dada por *g(nodo) + h(nodo)*. Ao expandir o nodo atual este procura sempre o nodo com o menor valor de f e só termina quando tiver esgotado todos os nodos possíveis desde o ponto inicial ou se encontrar a solução. Devido ao facto

da heurística utilizada ser admissível, é garantido que este algoritmo encontra a melhor solução num período de tempo relativamente pequeno.

2.4 Implementação

2.4.1 Base de conhecimento

A nossa base de conhecimento é uma adaptação da base de conhecimento da primeira fase deste trabalho, porque tivemos de acrescentar algumas informações.

As alterações que fizemos são:

- Os locais passaram a ter coordenadas, úteis para a heurística dos algoritmos de pesquisa informada.
- Cada freguesia tem a localização do centro de entregas, a que chamamos origem. Cada entrega deve começar e acabar no centro de entregas.
- Os veículos tem um coeficiente de poluição, para que seja possível escolher entregas mais ecológicas.

2.4.2 Criação de circuitos

Para criarmos os circuitos, procuramos todos os estafetas com encomendas atribuídas, e verificamos que encomendas têm de entregar. Depois procuramos os locais de entrega dessas encomendas, e armazenamos essas informações juntas (local de entrega e encomenda). Depois, geramos possíveis percursos, que são conjuntos de locais de entrega que um estafeta tem de passar durante um dia. Também fazemos percursos que passam por mais que um local de entrega, isto é, um estafeta entrega mais que uma encomenda numa viagem. Por fim, trocamos a ordem dos locais de entrega em percursos com mais que uma paragem. A razão para trocarmos a ordem de entrega é descrita a seguir, a partir

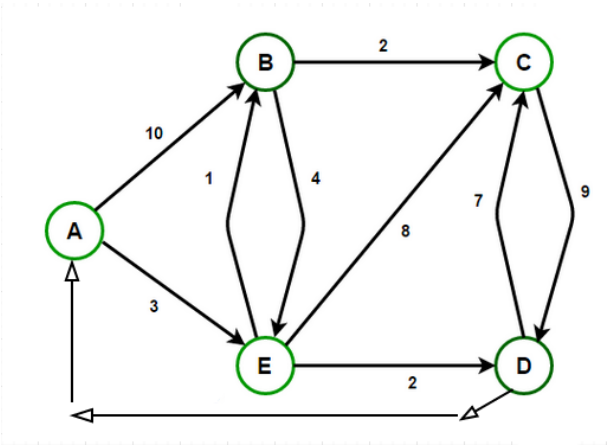

Figura 2.1: Exemplo nu grafo

da imagem anterior. Se quisermos entregar duas encomendas, uma no local E e outra no local D, a ordem de entrega importa. Se entregarmos primeiro D e depois E, o estafeta tem de realizar o percurso de A a D, voltar à base, e entregar E. Logo, é melhor entregar primeiro E, e depois entregar D, passando apenas uma vez pelo centro de distribuições.

No fim de calcular todos esses percursos, verificamos quais deles são possíveis. Um percurso só é possível se o peso total das encomendas dessa viagem for menor que o máximo que podemos transportar de cada vez. Esse máximo é definido pela capacidade do veículo que pode transportar mais carga. Depois, em função do critério de procura de

circuitos, velocidade ou ecologia, decidimos qual o melhor circuito, e veículo para realizar a entrega:

Velocidade

Se quisermos minimizar o tempo de entrega, procuramos o percurso com menor distância percorrida. Para explicar melhor o que é um percurso, utilizamos o grafo anterior. Suponhamos que temos de entregar no local E e C: um percurso possível é entregar cada encomenda individualmente; outro é entregar as duas na mesma viagem . Para calcularmos os caminhos entre os vários locais, usamos um algoritmo escolhido pelo utilizador. Depois de termos os melhores caminhos para entregar as encomendas, procuramos do veículo mais rápido até ao mais lento qual consegue entregar as encomendas de cada viagem.

Ecológica

Se quisermos ter entregas o mais ecológicas possíveis, temos de calcular quão poluente é cada percurso, e guardar o que consegue entregar todas as encomendas poluindo menos. Para calcularmos a poluição de cada percurso, começamos por ver quais os caminhos que tem de percorrer. Depois, vemos qual o veículo menos poluente que consegue fazer cada viagem, porque por exemplo, uma bicicleta não consegue entregar encomendas pesadas. Para comparar a poluição entre veículos, usamos o parâmetro ”coeficiente de poluição”. Por fim, para saber a poluição de cada viagem, e consequentemente do percurso, multiplicamos a distância pelo coeficiente de ”poluição do veículo”. Desta forma, privilegiamos entregas com meios de transporte mais ecológicos, mas valorizamos uma entrega mais poluente que consiga entregar muitas encomendas numa viagem curta. Por exemplo, é preferível ter um carro entregar dez encomendas numa distância curta que uma moto entregar cada encomenda individualmente, apesar de a mota ter um coeficiente de poluição menor.

Nas duas situações, caso não conclua os cálculos de melhor percurso em 5 segundos, o algoritmo para e fica-se com o melhor caminho encontrado até esse ponto. Para decidir a data de entrega da encomenda, somamos à data da encomenda um dia e o tempo de transporte. Decidimos também somar um dia para que as encomendas não sejam entregues no próprio dia, porque consideramos isso irrealista.

2.4.3 Geração de informação

Grafos

De forma a conseguir testar melhor o programa em diferentes cenários, foi decidido criar um algoritmo para gerar grafos aleatórios. Este cria uma cidade nova com o nome e número de nodos fornecidos. Para além disso, é necessário especificar a probabilidade do algoritmo criar uma conexão entre cada dois nodos.

Para garantir o bom funcionamento do programa, os grafos obedecem às seguintes regras:

- Cada nodo só se pode conectar aos 5 nodos mais próximos, ou à origem (caso não haja caminho possível de, ou para o nodo a partir da origem).
- A partir do Centro de Distribuição (origem) deve haver um caminho até todos os outros nodos;
- De todos os outros nodos deve haver um caminho de volta possível para o Centro de Distribuição;
- Todos os nodos devem ficar a uma distância mínima uns dos outros para permitir que estes sejam mais facilmente percetíveis pelo utilizador na imagem gerada.

Isto garante que é possível fazer entregas a todos os nodos do grafo e que o estafeta não fica preso no nodo de destino. Assim, após a geração aleatória dos nodos e das suas conexões, verificamos se um caminho de ida e volta é possível. Caso não o seja, é criado uma ligação direta no sentido adequado.

Para isto foi usado o algoritmo *depth-first*. Este mostra-se o melhor pelo facto de que, neste caso, não nos interessa que o caminho encontrado seja o melhor, apenas queremos verificar se um caminho existe.

No fim da geração do grafo, o algoritmo guarda uma imagem no formato PNG do grafo gerado, de forma a permitir que o utilizador tenha uma ideia mais clara deste. Esta imagem é criada na pasta em que o programa foi executado.

Encomendas

Para gerarmos encomendas, temos em conta as seguintes características: o peso máximo da encomenda deve estar compreendido entre 1 e metade do máximo que é possível transportar por um veículo, para haver sempre hipótese de levar duas encomendas; não devem haver IDs de encomendas repetidos, e o local da entrega tem de existir, associado a uma cidade.

A geração de encomendas é parametrizável, pois pode-se escolher uma cidade e o número de encomendas criadas. Por fim, guardamos as encomendas na base de conhecimento.

Atribuições e estafetas

Para criarmos atribuições, começamos por procurar as encomendas que ainda não foram atribuídas. Depois procuramos a cidade de destino dessa encomenda e vemos se já existe um estafeta encarregue das das entregas dessa cidade. Caso não exista um, criámos um estafeta com um id único, e associado a essa cidade. Fazemos este processo para cada encomenda não atribuída. Por fim, guardámos a atribuição na base de conhecimento, tal como os estafetas que tenham sido criados.

2.4.4 Circuitos mais usados

Para além das funcionalidades que já foram referidas, o programa permite obter informação relativa aos percursos mais usados, seja no volume/peso total acumulados de todas as entregas desse circuito, ou no número de vezes que este foi feito. Mais ainda, é possível obter a lista dos circuitos mais eficientes em termos de entregas por percurso do circuito.

Com esta informação o centro de distribuições poderá avaliar quais os circuitos que lhes geram maior lucro, bem como as localidades com maior número de encomendas. Isto ajuda a empresa a tomar decisões mais informadas.

2.5 Execução do programa

2.5.1 Dependências

Para executar corretamente o programa, é preciso instalar as seguintes dependências:

- Menu.
- Networkx.
- Psutil.

2.5.2 Informações de funcionamento do programa

Uma caraterística do nosso sistema é: uma encomenda não pode ser atribuída duas vezes; por isso, não é possível entregar a encomenda duas vezes. Sempre que se quiser gerar novas entregas, é preciso gerar novas encomendas.

2.5.3 Exemplos de execução do programa

Mostrar todas as encomendas

Como o nome desta opção do menu indica, esta lista as encomendas geradas pelo utilizador ou já definidas na base de conhecimento, assim como detalhes sobre as mesmas. Tal como será nas opções seguintes, para selecionar a funcionalidade que o utilizador deseja executar, é apenas necessário selecionar o número da opção.

```
****Green Distribution Management****

1. Mostrar todas as encomendas
2. Mostrar todas as entregas
3. Gerar encomendas
4. Entregar encomendas
5. Criar cidade de entrega
6. Mostrar circuitos mais produtivos
7. Mostrar circuito mais usado
8. Mostrar circuito com maior número de entregas por peso
9. Mostrar circuito com maior número de entregas por volume
10. Ativar/desativar modo de teste
11. Realizar testes para todos os algoritmos
12. Sair

>>> 1

Encomendas:
Id=0, clienteId=89, peso=10, volume=7, prazo=7, dataEncomenda=2022-01-05, idLocalEntrega=30

Id=1, clienteId=83, peso=40, volume=3, prazo=3, dataEncomenda=2022-01-06, idLocalEntrega=20
```

Figura 2.2: Opção 1 - Mostrar todas as encomendas

Mostrar todas as entregas

Quando selecionada a opção 2, esta irá apresentar uma lista de todas as encomendas que já foram entregues. Também são indicados os detalhes de cada entrega

```
****Green Distribution Management****

1. Mostrar todas as encomendas
2. Mostrar todas as entregas
3. Gerar encomendas
4. Entregar encomendas
5. Criar cidade de entrega
6. Mostrar circuitos mais produtivos
7. Mostrar circuito mais usado
8. Mostrar circuito com maior número de entregas por peso
9. Mostrar circuito com maior número de entregas por volume
10. Ativar/desativar modo de teste
11. Realizar testes para todos os algoritmos
12. Sair

>>> 2

Entregas:

Id da encomenda:  0
Estafeta: Marco
Data entrega: 2024-09-13
Transporte: barco
Peso: 10
Caminho: Local_18 Local_30 Local_36 Local_38 Local_26 Local_31 Local_13 Local_18
```

Figura 2.3: Opção 2 - Mostrar todas as entregas

Gerar encomendas

Através desta opção é possível adicionar encomendas a cidades (que são representadas na forma de grafos) já geradas

```
****Green Distribution Management****

1. Mostrar todas as encomendas
2. Mostrar todas as entregas
3. Gerar encomendas
4. Entregar encomendas
5. Criar cidade de entrega
6. Mostrar circuitos mais produtivos
7. Mostrar circuito mais usado
8. Mostrar circuito com maior número de entregas por peso
9. Mostrar circuito com maior número de entregas por volume
10. Ativar/desativar modo de teste
11. Realizar testes para todos os algoritmos
12. Sair

>>> 3

Lista dos grafos já gerados:
Vila_do_Conde
Porto
Insira o nome do grafo: Porto
Insira o número de encomendas: 10
Encomendas geradas com sucesso.
```

Figura 2.4: Opção 3 - Gerar encomendas

Entregar encomendas

Ao selecionar esta opção, o sistema irá entregar as encomendas previamente geradas através do método escolhido pelo utilizador. Todos estes métodos são descritos em detalhe nas secções anteriores do relatório. Um detalhe desta opção é o facto de o utilizador poder escolher que tipo de entrega deseja que seja realizada, rápida ou ecológica.

Para além de realizar a entrega das encomendas, este método também apresenta qual foi a memória necessária e o tempo para a execução do algoritmo selecionado.

```
****Green Distribution Management****

1. Mostrar todas as encomendas
2. Mostrar todas as entregas
3. Gerar encomendas
4. Entregar encomendas
5. Criar cidade de entrega
6. Mostrar circuitos mais produtivos
7. Mostrar circuito mais usado
8. Mostrar circuito com maior número de entregas por peso
9. Mostrar circuito com maior número de entregas por volume
10. Ativar/desativar modo de teste
11. Realizar testes para todos os algoritmos
12. Sair

>>> 4

Pretende usar o critério ecológico? (S)im/(N)ão s
Algoritmos implementados: dict_keys(['dfs', 'dfs_lim', 'bfs', 'gulosa', 'a_estrela', 'dijkstra'])
Nome do algoritmo a ser usado: bfs
Memoria de execução do algoritmo: 0.171875MB
Tempo de execução do algoritmo: 0.3358919620513916s
Circuito gerado com sucesso!
```

Figura 2.5: Opção 4 - Entregar encomendas

Criar cidade de entrega

A opção 5 permite a criação de uma nova cidade, e adição à base de conhecimento do sistema. Por conseguinte, o grafo associado à cidade é também criado.

De maneira a gerar uma nova cidade, o sistema irá pedir ao utilizador para inserir o nome da cidade, o número de nodos que terá e a probabilidade que terá de conexão entre cada nodo.

```
****Green Distribution Management****

1. Mostrar todas as encomendas
2. Mostrar todas as entregas
3. Gerar encomendas
4. Entregar encomendas
5. Criar cidade de entrega
6. Mostrar circuitos mais produtivos
7. Mostrar circuito mais usado
8. Mostrar circuito com maior número de entregas por peso
9. Mostrar circuito com maior número de entregas por volume
10. Ativar/desativar modo de teste
11. Realizar testes para todos os algoritmos
12. Sair

>>> 5

Insira o nome da cidade: Porto
Insira o número de nodos: 30
Insira a probabilidade de conexão: 50
Cidade criada com sucesso!
Cidade guardada com o nome "Porto.png"
```

Figura 2.6: Opção 5 - Criar cidade de entrega

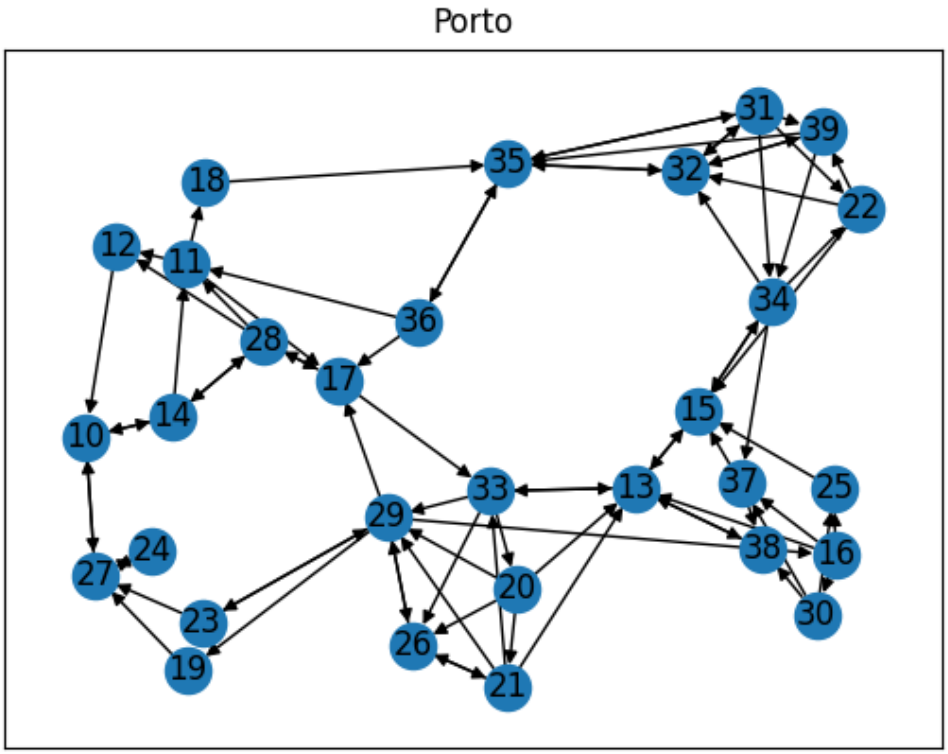

Figura 2.7: Grafo da cidade criada

2.5.4 Testes

De modo a efetuar testes no programa, começamos por gerar um grafo, ou usar o que já está definido na base de conhecimento. Os parâmetros de teste utilizados para gerar os grafos são: 10 nodos, e 10% de probabilidade de conexão.

Depois, temos de gerar as encomendas, sendo um número razoável 15 encomendas. Finalmente, é possível gerar os circuitos, e as entregas. Essa informação será disponibilizada num ficheiro de logs, ou após consultar os menus.

Modo teste

Com o objetivo de efetuar testes de uma forma mais eficaz foi criado o modo de teste, que permite que encomendas já entregues voltem a ser entregues. Assim, consegue-se testar os algoritmos de uma forma muito mais eficiente e não tão demorada. Para além disso, os testes tornam-se mais corretos, porque é possível testar no mesmo mapa, as mesmas encomendas, com diferentes algoritmos. De seguida foi adicionada a opção "realizar testes para todos os algoritmos"que automatiza os testes, de forma, a que todos os algoritmos sejam testados de forma igual e rápida.

Segue-se um exemplo da opção de "Realizar testes para todos os algoritmos"para o grafo representado na base de conhecimento.

```
De modo a correr todos os algoritmos nas mesmas condições o modo teste será ativado, caso não esteja!
Inicia-se a procura com o algoritmo dfs!
Memoria de execução do algoritmo: 0.00390625MB
Tempo de execução do algoritmo: 0.7817802429199219s
Circuito gerado com sucesso!
Inicia-se a procura com o algoritmo dfs_lim!
Memoria de execução do algoritmo: 0.0MB
Tempo de execução do algoritmo: 0.25147366523742676s
Circuito gerado com sucesso!
Inicia-se a procura com o algoritmo bfs!
Memoria de execução do algoritmo: 0.0MB
Tempo de execução do algoritmo: 0.022976398468017578s
Circuito gerado com sucesso!
Inicia-se a procura com o algoritmo gulosa!
Memoria de execução do algoritmo: 0.0625MB
Tempo de execução do algoritmo: 0.031050920486450195s
Circuito gerado com sucesso!
Inicia-se a procura com o algoritmo a_estrela!
Memoria de execução do algoritmo: 0.0MB
Tempo de execução do algoritmo: 0.07664871215820312s
Circuito gerado com sucesso!
Inicia-se a procura com o algoritmo dijkstra!
Memoria de execução do algoritmo: 0.0MB
Tempo de execução do algoritmo: 0.08531856536865234s
Circuito gerado com sucesso!
A repor o modo de teste...
```

Figura 2.8: Opção 11 - Realizar testes para todos os algoritmos

Resultados

Esta tabela de resultados trata da cidade de Vila do Conde, isto é, o grafo que está definido na base de conhecimento.

Estratégia	**Tempo (segundos)**	**Espaço (MB)**	**Indicador/ Custo**
DFS	0.7817	0.0039	91
DFS-LIM	0.2515	≈ 0	55
BFS	0.0230	≈ 0	73
Dijkstra	0.0853	≈ 0	55
Gulosa	0.0311	≈ 0	60
A-Estrela	0.0766	0.0625	55

As soluções dos algoritmos de procura que não tem em conta o custo dos caminhos não chegam necessariamente às soluções ótimas. Pode acontecer, mas é por acaso, como no caso do **DFS-LIM**. No entanto, os algoritmos que analisam o custo entre nodos dão os melhores caminhos entre dois locais. Em grafos relativamente pequenos e com poucas encomendas, o nosso programa encontra os melhores circuitos para entregar as encomendas, mas em situações com muitas encomendas e grafos extensos isso não é garantido. Isto acontece porque, ao aumentar o número de encomendas, as possibilidades de viagens aumentam bastante, aumentando os cálculos necessários para encontrar a melhor solução. Para além disso, como as hipóteses são muitas, não conseguimos analisar todas, mas chegamos a resultados satisfatórios porque ainda assim testamos bastantes casos dentro do tempo máximo definido.

Capítulo 3

Comentários finais e conclusão

No fim do trabalho, o grupo acredita que fez um trabalho satisfatório que, não só cumpre os requisitos pedidos no enunciado, como os excede, nomeadamente na geração dos grafos.

Apesar disto, o grupo acredita que poderia ter melhorado a forma como os percursos possíveis são gerados, eliminando alguns casos repetidos, mas isso teria de ser pensado desde o início. Teríamos de guardar todos os percursos calculados, e verificar se já foi calculado o custo, assim não seria necessário recalcular. Outra melhoria que gostaríamos de ter feito era *pruning* na análise de casos, para não avaliar percursos piores que o melhor descoberto até esse ponto. Mais ainda, a quantidade de encomendas podia aumentar caso o problema anterior fosse solucionado, possibilitando o cálculo de mais percursos.

Por fim, o grupo sai com um melhor conhecimento relativo aos algoritmos de procura sobre grafos e com mais uma ferramenta para o mercado de trabalho o que nos dá um maior destaque relativamente aos outros programadores.

Capítulo 4

Bibliografia

Slides apresentados nas aulas teóricas da unidade curricular de Inteligência Artificial

4.1 Packages utilizados

- Menu
- Networkx
- Psutil