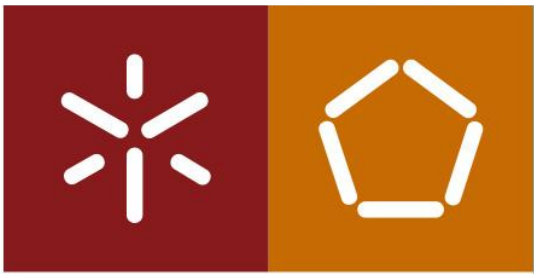

Universidade do Minho
Licenciatura em Engenharia Informática

Inteligência Artificial
Grupo 39

Green Distribution
2ª Fase

Daniela Cristina Miranda Carvalho - A93224
Eduardo Costa de Magalhães - A93301
Laura Nunes Rodrigues - A93169
Mariana Filipa da Silva Rodrigues - A93306




Índice

Introdução

Para a segunda parte do trabalho de grupo realizado no âmbito da unidade curricular Inteligência Artificial, foi nos pedido para implementar a funcionalidade que permite a sugestão de circuitos de entrega de encomendas para os estafetas definidos na primeira parte deste trabalho.

Para implementar esta funcionalidade, optámos por recorrer a grafos para representar uma região da cidade de Braga e à definição de algoritmos de procura em grafos para assim obter os percursos de entrega mais eficientes.

Por uma questão de tamanho da base de conhecimento escolhemos reduzir o número de freguesias a serem consideradas. No mapa abaixo podemos ver as freguesias ou agrupamentos escolhidos:

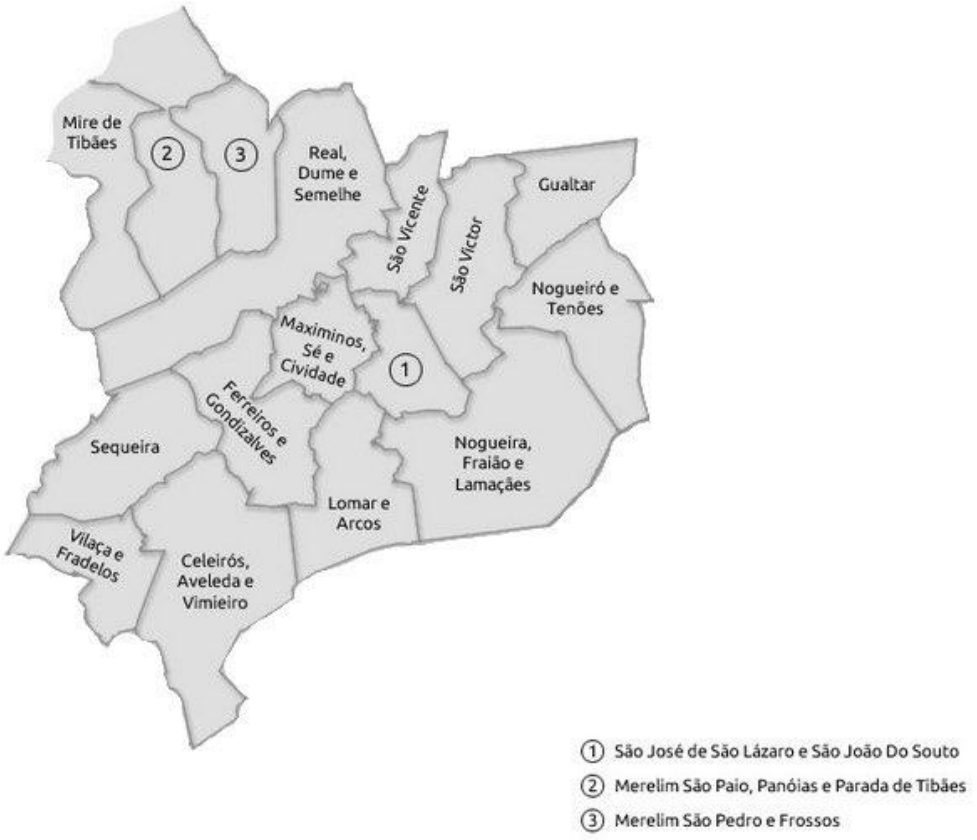

Fig. 1 - Alguns agrupamentos de freguesias da cidade de Braga

Freguesias:

1. São José de São Lázaro e São João do Souto
2. Merelim São Paio, Panóias e Parada de Tibães
3. Merelim São Pedro e Frossos
4. Nogueiró
5. Nogueira, Fraião e Lamaçães
6. Mire de Tibães
7. Real, Dume e Semelhe
8. São Vicente
9. São Vitor

10. Gualtar
11. Maximinos, Sé e Cividade
12. Sequeira
13. Ferreiros e Gondizalves
14. Lomar e Arcos
15. Celeirós, Aveleda e Vimieiro
16. Vilaça e Fradelos

Como pretendíamos definir apenas um exemplo, optámos por representar 16 das 37 freguesias de Braga.

Para obtermos os percursos, procurámos aplicar diferentes algoritmos de pesquisa sobre grafos. De forma complementar, avaliámos a performance e eficiência de cada um deles.

Aproveitámos para implementar funcionalidades extra que considerámos pertinentes e que adicionam aos objetivos definidos, tal como, começar a entrega de uma encomenda e concluir a entrega da mesma.

Representação da cidade de Braga

De forma a desenvolver um caso prático e pôr à prova os algoritmos que implementámos, definimos um grafo cujos nodos correspondem às freguesias mencionadas na secção anterior.

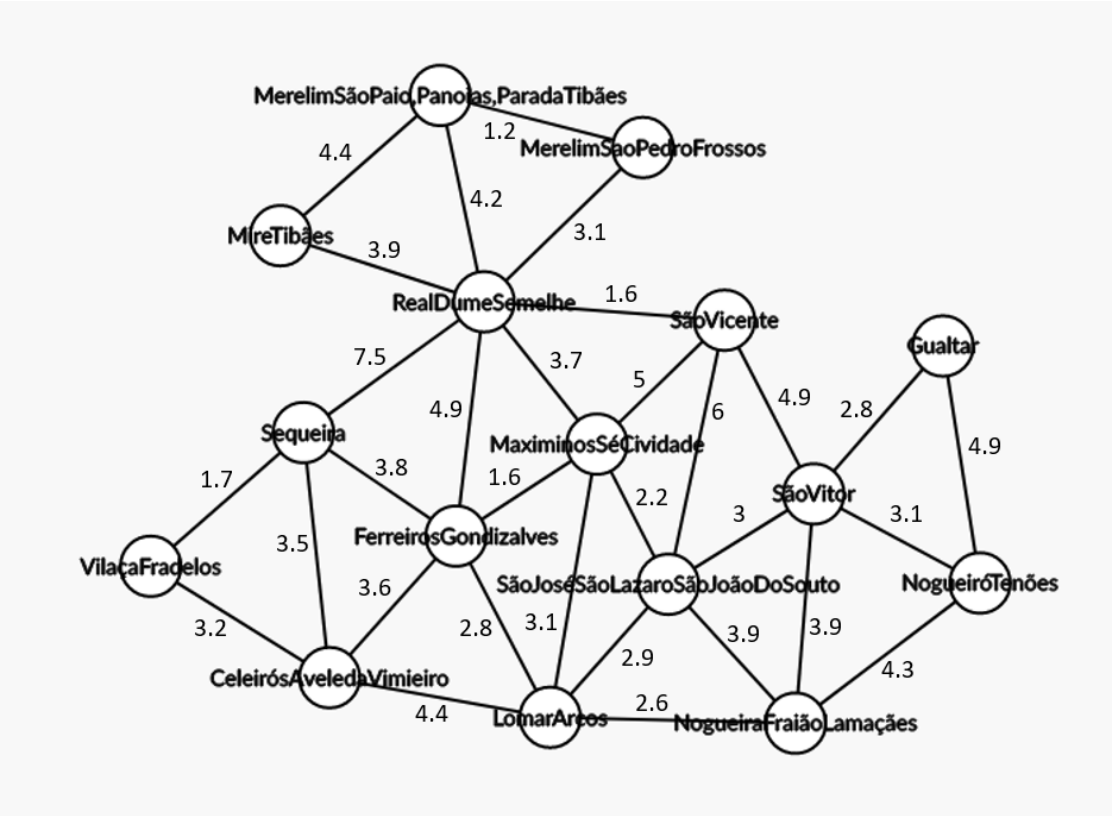

Fig. 2 - Grafo das freguesias escolhidas

Obtivemos um grafo não orientado pesado e completámos a base de conhecimento do nosso projeto com o grafo.

Tal como se verifica com a sede dos correios da cidade de Braga, a sede da Green Distribution situa-se em Maximinos, ou seja em “MaximinosSeECividade”.

Formulação do Problema

Como já foi mencionado, os algoritmos implementados tinham como objetivo obter circuitos que poderiam ser usados na entrega de encomendas, ou seja, obter um caminho entre dois nodos do grafo, um que seria a sede da Green Distribution e o destino da encomenda.

O circuito é complementado com um caminho de volta, isto é, com um caminho que parte do destino da encomenda e cujo objetivo é a sede.

A obtenção destes caminhos trata-se de um problema de pesquisa que pode ser formulado da seguinte maneira:

Tipo: Problema de estado único, uma vez que o ambiente é completamente observável e o agente saberá sempre em que estado estará

Estado: Qualquer freguesia representada no grafo ou a Sede

Estado inicial: Sede

Estado final: Qualquer uma das freguesias onde se pretenda entregar encomendas

Operadores: Deslocação entre cada freguesia, isto é, travessia entre as arestas do grafo

Custo: soma das distâncias entre as freguesias por onde passa para fazer as entregas

Solução: a solução ideal é o caminho (sequência de freguesias) que tem um custo menor, ou seja, menor soma de distâncias para fazer a entrega

Esta formulação aplica-se ao percurso feito pelo estafeta para fazer a entrega da encomenda, contudo, os algoritmos, como já foi mencionado, são aplicados no caminho de regresso do estafeta à sede. Nesse caso, o Estado Inicial será uma das freguesias do grafo e o estado objetivo a sede.

O circuito encontra-se completo quando se obtêm os dois caminhos.

Base de Conhecimento

Após analisarmos o problema apresentado, constatamos que, para representar o conhecimento necessário para o sistema e implementar as funcionalidades propostas nesta segunda parte do projeto, à nossa base de conhecimento seria necessário adicionar os seguintes predicados:

- viagem(Freguesia, Freguesia, Distância) -> {V,F}
  Representa as arestas do grafo das Freguesias apresentado acima. Uma viagem possui uma Freguesia inicial, uma Freguesia final e a Distância entre estas duas freguesias.
  Uma vez que a sede da Green Distribution se situa no agrupamento de freguesias de *MaximinosSeECividade*, a distância a este é igual a zero.

- estimativa(Freguesia, Distância) -> {V,F}
  Estimativas das distâncias da sede à respetiva Freguesia.

- percurso(Percurso,Volume,Peso) -> {V,F}
  Para cada Percurso existente associa-se a soma de todos os Volumes e Pesos das encomendas que seguirão o mesmo Percurso efetuado por um estafeta.

Observação: Os diversos valores das distâncias mencionadas acima foram obtidos com recurso ao Google Maps, sendo estes os dos percursos mais curtos possível.

Funcionalidades Implementadas

Para realizar esta fase do trabalho prático, implementámos os algoritmos de procura solicitados no enunciado (DFS, BFS, BILP, Gulosa e A*) que serão essenciais para obter um caminho entre um local e o destino e cuja formulação já foi mencionada.

Contudo, o pedido não era simplesmente obter um caminho entre uma freguesia e a sede, mas sim, obter um percurso completo para o estafeta, isto é, um ciclo cujo local de partida e de chegada é a sede e que passa no ponto de entrega. Definimos uma destas funções para cada tipo de algoritmo.

Sabendo que o custo de cada percurso obtido era apenas a distância total deste, definimos uma função que para a encomenda indicada, conhecendo a distância que essa encomenda terá de percorrer, determina que meio de transporte o estafeta deverá usar e quanto tempo irá demorar a percorrer esse percurso.

De forma a analisar o tempo de execução de cada algoritmo, implementámos predicados que obtêm o runtime de cada método de procura.

Como foi mencionado no enunciado, definimos ainda funções que permitem gerar um percurso com várias paragens, ou seja, dado uma lista de encomendas estes predicados irão gerar um percurso que poderá ser seguido pelo estafeta e que irá passar por todos os locais de entrega necessários. Definimos um predicado para cada algoritmo já mencionado.

De forma a determinar qual o percurso mais rápido e o percurso mais ecológico, definimos um predicado que para uma dada encomenda aplica todos os predicados definidos que permitem gerar um percurso completo e que compara os resultados obtidos, sendo que para obter o caminho mais

rápido compara o comprimento de cada percurso obtido e para obter o caminho mais ecológico compara o tempo que demora a percorrer o dado percurso.

Definimos, também, uma função que permite determinar qual o circuito com o maior número de entregas, tendo como fatores de avaliação o volume e o peso.

Como funcionalidades extras, definimos predicados que nos permitem adicionar encomendas e estafetas à base de conhecimento, começar a entrega de uma encomenda e ainda dar como concluída a entrega de uma encomenda.

Um dos novos predicados implementados foi *adicionaEstafeta: NumeroEstafeta -> {V,F}*. Este permite adicionar um estafeta com o número indicado, cuja Lista de Entregas por este feitas e a Lista de Entregas que este está a realizar se encontram vazias. Por defeito, a Prioridade deste estafeta inicialmente será 1, tendo, portanto, prioridade sobre os restantes estafetas.

O predicado *adicionaEncomenda: NumeroEncomenda, NumeroCliente, Peso, Volume, TempoMáximo, Freguesia, Data -> {V,F}* foi também criado. Este permite adicionar uma nova encomenda ao conhecimento, desde que ainda não exista nenhuma com o mesmo número na base de conhecimento, que a freguesia esteja definida no grafo já mencionado e que a data seja válida.

Para adicionar esta informação à base de conhecimento, recorremos a invariantes que tinham sido implementados na primeira fase deste trabalho prático e que permitem constatar que algumas condições se verificam quando se adicionam novos dados.

Optámos ainda por definir o predicado *comecaEntrega: Encomenda, Estafeta, Percurso, Distância, MeioTransporte, Tempo, Preco -> {V,F}* que, dado o número de uma encomenda, determina a informação necessária para dar início à entrega da mesma. Para tal, determina o percurso que deverá ser seguido para chegar ao local onde a encomenda se destina e para voltar à sede. Quando se obtém o percurso, obtêm-se também o meio de transporte que deverá ser usado e o tempo que irá demorar a percorrer o circuito. Calcula-se ainda o preço da encomenda, em função do meio de transporte, do peso e do tempo máximo estabelecido pelo cliente. Com estes dados, cria-se a *entregaIntermédia*. De seguida, determina-se o estafeta que fará esta entrega, sendo os fatores de decisão a disponibilidade do estafeta e a sua prioridade, e atualiza-se a lista de entregas intermédias do estafeta escolhido. Por fim, atualiza-se o conhecimento do predicado *percurso* com os novos dados obtidos.

A última funcionalidade extra que definimos foi a *concluiEntrega: Encomenda, TempoUsado, Classificacao, Data -> {V,F}*. O predicado mencionado permite fazer as alterações necessárias para que a encomenda seja dada como entregue. Estas alterações incluem adicionar à base de conhecimento a entrega associada a esta encomenda definida na primeira fase e atualizar a lista de entregas intermédias e a lista de entregas concluídas do estafeta responsável pela encomenda. Esta atualização consiste em retirar o número da encomenda da lista das entregas intermédias e adicioná-la à lista das encomendas entregues, atualizando também a prioridade do estafeta tendo em consideração o tempo que foi necessário para fazer a entrega.

Explicação dos algoritmos implementados

Procura não informada:

Estes algoritmos de procura encarregam-se de tentar encontrar o estado objetivo sem ter em consideração os custos das arestas quando escolhem o próximo nó a ser visitado.

- Profundidade (DFS - Depth-First Search)

Formalmente, um algoritmo de pesquisa em profundidade realiza uma procura não-informada que avança através da expansão do primeiro nodo do grafo, pesquisando o nodo objetivo num nível de profundidade cada vez maior, até que o alvo da pesquisa seja encontrado ou até encontrar um nodo que não possui mais nodos adjacentes. Posto isto, retrocede (backtrack) e começa no próximo nodo. São usadas estruturas simples para guardar o caminho visitado, sendo por isso, necessária pouca memória. No entanto, não garante que é retornada a melhor solução.

Definimos o predicado *resolve_DFS: Objetivo, Freguesia, Caminho, Custo -> {V,F}* que obtém o percurso desde a Freguesia em que se encontra (que pode ser a Sede) até a um dado Objetivo (que mais uma vez pode ser a Sede). O Caminho devolvido é o percurso que o estafeta deve fazer para ir de um ponto para outro, sendo o Custo a distância que irá percorrer.

O Predicado que permite gerar o circuito percorrido por um estafeta para entregar uma dada encomenda usando o algoritmo Depth-First Search é *gerarPercursoDFS: Encomenda, Percurso, Distância, Meio de Transporte, Tempo -> {V,F}*. Este predicado permite gerar um caminho de ida e de volta, assim como a distância que será percorrida pelo estafeta, o meio de transporte que este deverá utilizar para entregar a encomenda no periodo de tempo definido pelo cliente e quanto tempo demora a percorrer o percurso

- Largura (BFS - Breadth-First Search)

Efetuar uma pesquisa em largura primeiro (BFS), consiste em começar por visitar um nodo do grafo, visitar todos os seus adjacentes, avançar para o próximo nível de profundidade (para um nodo adjacente ao anterior) e efetuar o mesmo processo em cada nodo dessa mesma profundidade até encontrarmos uma solução (o nodo pretendido estar na lista de nodos visitados). Tornando-se assim um método de pesquisa muito sistemático. No entanto, trata-se de um algoritmo de pesquisa que, normalmente, demora muito tempo e que sobretudo ocupa muito espaço, uma vez que são necessárias estruturas pesadas para guardar todos os caminhos.

Para implementar este método de pesquisa definimos o predicado *resolve_BFS: Objetivo, Freguesia, Caminho, Custo -> {V,F}* que obtém o percurso desde a Freguesia em que se encontra (que pode ser a Sede) até a um dado Objetivo (que mais uma vez pode ser a Sede). O Caminho devolvido é o percurso que o estafeta deve fazer para ir de um ponto para outro, sendo o Custo a distância que irá percorrer.

O Predicado que permite gerar um percurso para uma encomenda usando o algoritmo Breath-First Search é *gerarPercursoBFS: Encomenda, Percurso, Distancia, Meio de Transporte, Tempo -> {V,F},* e que, simultaneamente, obtém o meio de transporte que deverá ser usado e o tempo que irá demorar a percorrer este percurso.

- Busca Iterativa Limitada em Profundidade

Um dos problemas da pesquisa Primeiro em Profundidade prende-se com a incapacidade desta lidar com caminhos muito longos. A Busca Iterativa com limite de profundidade procura evitar este problema fixando um limite máximo de profundidade (N) em que procura o nodo objetivo. Assim, este algoritmo consiste em aplicar o algoritmo de procura em profundidade **também** até um certo valor máximo de profundidade, parando quando encontra o nodo desejado ou quando passa da profundidade indicada. Esta profundidade é incrementada e o algoritmo repetido até que se encontre o nodo desejado ou até que a profundidade máxima seja a mesma que o limite definido.

Este algoritmo foi implementado através do predicado *resolve_BILP: Objetivo, Freguesia, Caminho, Custo, Limite -> {V,F}* que obtém o percurso desde a Freguesia em que se encontra (que pode ser a Sede) até a um dado Objetivo (que mais uma vez pode ser a Sede) dado um Limite do número de freguesias por onde pode passar. O Caminho devolvido é o percurso que o estafeta deve fazer para ir de um ponto para outro, sendo o Custo a distância que irá percorrer.

O Predicado que permite gerar um percurso para uma encomenda usando o algoritmo Busca Iterativa Limitada em Profundidade é *gerarPercursoBILP: Encomenda, Percurso, Distância, Meio de Transporte, Tempo, Limite -> {V,F}*.

Procura informada

A técnica de busca informada utiliza o conhecimento específico do problema para dar uma pista para a solução do problema. A pesquisa informada pode ser vantajosa em termos de custo, onde a otimização é alcançada com custos mais baixos de pesquisa.

- Gulosa (Greedy-Search)

Um algoritmo de pesquisa gulosa escolhe, em cada iteração, o nodo que parece estar a uma menor distância do objetivo final. O nodo escolhido passa a fazer parte da solução que o algoritmo constrói. No entanto, isto provoca uma maior suscetibilidade a falsos começos e portanto, não garante que encontra sempre a melhor solução. Todos os nós são mantidos em memória e é necessário detetar estados repetidos.

O algoritmo por trás desta procura é desenvolvido a partir do predicado *resolve_gulosa: Objetivo, Freguesia, Caminho/Custo -> {V,F}* que obtém o percurso desde a Freguesia em que se encontra (que pode ser a Sede) até a um dado Objetivo (que mais uma vez pode ser a Sede). O Caminho devolvido é o percurso que o estafeta deve fazer para ir de um ponto para outro, sendo o Custo a distância que irá percorrer.

O Predicado que permite gerar um percurso para uma encomenda usando a pesquisa Gulosa é *gerarPercursoGulosa: Encomenda, Percurso, Distância, Meio de Transporte, Tempo -> {V,F}*.

- A* (A estrela)

A pesquisa A* evita expandir caminhos que são caros. O algoritmo combina a pesquisa gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução. É, por isso, um algoritmo ótimo e completo, no entanto tem uma complexidade no tempo exponencial e mantém todos os nodos em memória.

De modo a implementar este algoritmo desenvolvemos o predicado *resolve_aestrela: Objetivo, Freguesia, Caminho/Custo -> {V,F}* que obtém o percurso desde a Freguesia em que se

encontra (que pode ser a Sede) até a um dado Objetivo (que mais uma vez pode ser a Sede). O Caminho devolvido é o percurso que o estafeta deve fazer para ir de um ponto para outro, sendo o Custo a distância que irá percorrer.

O Predicado que permite gerar um percurso para uma encomenda usando a pesquisa A* é *gerarPercursoAEstrela: Encomenda, Percurso, Distância, Meio de Transporte, Tempo -> {V,F}*.

Resultados

Concluída a implementação dos diferentes algoritmos pudemos comparar a execução destes, assim como os resultados dos mesmos. Fizemos uma análise a nível de tempo de execução, utilização de memória, custo (ou seja, distância percorrida para efetuar a entrega) e se conseguiu encontrar a melhor solução.

Os testes foram realizados para as freguesias de Gualtar e de Nogueiró e Tenões.

Estratégia	Tempo	Espaço	Custo	Encontrou a melhor solução?
DFS	O(b^m)	O(b*m)	G: 18,2 NT: 13,3	Não
BFS	O(b^d)	O(b^d)	G: 12,7 NT: 13	Não
BILP	O(b^l)	O(b*l)	G: 12,7 NT: 13	Não
Gulosa	Pior caso: O(b^m) Melhor caso: O(bd)	Pior caso: O(b^m) Melhor caso: O(bd)	G: 8 NT: 8,3	Nem sempre
A*	Exponencial	Nº de nodos	G: 8 NT: 8,3	Sim

Tabela 1 - Resultados de Tempo e Espaço esperados e Custos obtidos

Legenda:

- b: fator de ramificação
- d: profundidade da solução
- m: máxima profundidade do grafo
- l: a profundidade limite de pesquisa
- G: Gualtar
- NT: Nogueiró e Tenões

De seguida, nas imagens abaixo, encontram-se os testes realizados.

```
?- tempoDFS(RunTime,sede,nogueiroETenoes,P,D).
RunTime = 1,
P = [sede, maximinosSeECividade, realDumeESemelhe, saoVicente, saoVitor, nogueiroETenoes],
D = 13.3 .

?- tempoBFS(RunTime,sede,nogueiroETenoes,P,D).
RunTime = 5,
P = [sede, maximinosSeECividade, saoVicente, saoVitor, nogueiroETenoes],
D = 13.0 .

?- tempoBILP(RunTime,sede,nogueiroETenoes,P,D,10).
RunTime = 0,
P = [sede, maximinosSeECividade, saoVicente, saoVitor, nogueiroETenoes],
D = 13.0 .

?- tempoGulosa(RunTime,sede,nogueiroETenoes,P,D).
RunTime = 0,
P = [sede, maximinosSeECividade, saoJoseDeSaoLazaroESaoJoaoDoSouto, saoVitor, nogueiroETenoes],
D = 8.3 .

?- tempoAEstrela(RunTime,sede,nogueiroETenoes,P,D).
RunTime = 0,
P = [sede, maximinosSeECividade, saoJoseDeSaoLazaroESaoJoaoDoSouto, saoVitor, nogueiroETenoes],
D = 8.3 .
```

Fig. 3 - Tempos obtidos desde a Sede até Nogueiró e Tenões

```
?- tempoDFS(RunTime,sede,gualtar,P,D).
RunTime = 1,
P = [sede, maximinosSeECividade, realDumeESemelhe, saoVicente, saoVitor, nogueiroETenoes, gualtar]
D = 18.200000000000003 .

?- tempoBFS(RunTime,sede,gualtar,P,D).
RunTime = 5,
P = [sede, maximinosSeECividade, saoVicente, saoVitor, gualtar],
D = 12.7 .

?- tempoBILP(RunTime,sede,gualtar,P,D,10).
RunTime = 0,
P = [sede, maximinosSeECividade, saoVicente, saoVitor, gualtar],
D = 12.7 .

?- tempoGulosa(RunTime,sede,gualtar,P,D).
RunTime = 1,
P = [sede, maximinosSeECividade, saoJoseDeSaoLazaroESaoJoaoDoSouto, saoVitor, gualtar],
D = 8.0 .

?- tempoAEstrela(RunTime,sede,gualtar,P,D).
RunTime = 0,
P = [sede, maximinosSeECividade, saoJoseDeSaoLazaroESaoJoaoDoSouto, saoVitor, gualtar],
D = 8.0 .
```

Fig. 4 - Tempos obtidos desde a Sede até Gualtar

```
?- gerarPercursoDFS(e00001,Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, lomarEArcos, celeirosAveledaEVimieiro, sequeira, ferreirosEGondiz
alves, maximinosSeECividade, sede],
Dist = 16.4,
MT = bicicleta,
T = 2.043846153846154 .

?- gerarPercursoBFS(e00001,Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, lomarEArcos, celeirosAveledaEVimieiro, ferreirosEGondizalves, max
iminosSeECividade, sede],
Dist = 12.7,
MT = bicicleta,
T = 1.6738461538461538 .

?- gerarPercursoBILP(e00001,Percurso,Dist,MT,T,10).
Percurso = [sede, maximinosSeECividade, lomarEArcos, celeirosAveledaEVimieiro, ferreirosEGondizalves, max
iminosSeECividade, sede],
Dist = 12.7,
MT = bicicleta,
T = 1.6738461538461538 .

?- gerarPercursoGulosa(e00001,Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, ferreirosEGondizalves, celeirosAveledaEVimieiro, ferreirosEGondiz
alves, maximinosSeECividade, sede],
Dist = 10.4,
MT = bicicleta,
T = 1.32 .

?- gerarPercursoAEstrela(e00001,Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, ferreirosEGondizalves, celeirosAveledaEVimieiro, ferreirosEGondiz
alves, maximinosSeECividade, sede],
Dist = 10.4,
MT = bicicleta,
T = 1.32 .
```

Fig. 5 - Gerar percurso com os diferentes algoritmos de procura

```
?- gerarPercursoDFSVariasEncomendas([e00001,e00002],Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, lomarEArcos, celeirosAveledaEVimieiro, sequeira, ferreirosEGondiz
alves, maximinosSeECividade, realDumeESemelhe, saoVicente|...],
Dist = 73.3,
MT = mota,
T = 2.2106770648706138.

?-
|    gerarPercursoBFSVariasEncomendas([e00001,e00002],Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, lomarEArcos, celeirosAveledaEVimieiro, lomarEArcos, nogueiraFraia
oELamacaes, nogueiroETenoes, gualtar, saoVitor|...],
Dist = 31.7,
MT = mota,
T = 0.9819213045019497.

?- gerarPercursoBILPVariasEncomendas([e00001,e00002],Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, lomarEArcos, celeirosAveledaEVimieiro, lomarEArcos, nogueiraFraia
oELamacaes, nogueiroETenoes, gualtar, saoVitor|...],
Dist = 31.700000000000003,
MT = mota,
T = 0.9819213045019499.

?- gerarPercursoGulosaVariasEncomendas([e00001,e00002],Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, ferreirosEGondizalves, celeirosAveledaEVimieiro, ferreirosEGondiz
alves, maximinosSeECividade, saoJoseDeSaoLazaroESaoJoaoDoSouto, saoVitor, gualtar|...],
Dist = 26.4,
MT = mota,
T = 0.8143778801843319.

?- gerarPercursoAEstrelaVariasEncomendas([e00001,e00002],Percurso,Dist,MT,T).
Percurso = [sede, maximinosSeECividade, ferreirosEGondizalves, celeirosAveledaEVimieiro, lomarEArcos, sao
JoseDeSaoLazaroESaoJoaoDoSouto, saoVitor, gualtar, saoVitor|...],
Dist = 26.3,
MT = mota,
T = 0.811152073732719.
```

Fig. 6 - Gerar percurso para efetuar várias entregas com os diferentes algoritmos de procura

Através de uma análise dos tempos apresentados nas figuras 3 e 4, concluímos que os nossos algoritmos não estão de acordo com os valores esperados para o tempo, previa-se que os algoritmos estivessem ordenados decrescentemente relativamente ao tempo: DFS, BILP, BFS, Gulosa e A*. Por outras palavras, os testes que executamos não representam os valores "teóricos" de que estaríamos à espera, contudo, estes valores de tempo de execução podem ter sido influenciados pela definição do grafo, uma vez que o grau de ramificação pode ser influenciado pela mesma. Não obstante, podemos observar que, segundo a tabela apresentada anteriormente, seria de esperar que o tempo de execução do algoritmo profundidade fosse superior ao do algoritmo bilp dado que é expectável que m seja superior a l (legenda).

Dado que o bfs tem um elevado fator de ramificação, uma vez que este vai desenvolvendo os ramos "um a um" até que encontra uma solução, ou seja, a "árvore" que é gerada no bfs terá mais ramos e maiores que na dfs, em que a procura é feita sobretudo sobre um ramo, o valor de ramificação será menor apesar de a profundidade ser superior. Isto poderá ser a origem dos valores obtidos.

Em contrapartida, tal como previsto, os custos obtidos, ou seja, as distâncias percorridas entre a sede e as freguesias mencionadas, estão conforme os valores expectáveis, dado que os algoritmos informados obtêm as melhores soluções uma vez que têm em atenção os custos das arestas e tomam as suas decisões em função dos mesmos enquanto que as não informadas limitam-se a obter um caminho entre a origem e o destino.

Calculámos ainda o tempo que demoraria a percorrer os percursos, uma vez que este é um fator de produtividade e é pertinente para a decisão de um caminho. Contudo, estes valores de duração, normalmente, são proporcionais à distância do percurso, dado que são calculados em função da velocidade média de um veículo e da distância. Adicionalmente, como a encomenda para a qual se está a calcular o percurso é igual em todos os testes, a penalização na velocidade média do veículo provocada pelo peso desta não será relevante e não causará grande distinção nos tempos esperados.

Consideramos interessante que na maioria dos exemplos, a pesquisa Gulosa conseguiu obter soluções ótimas.

Terminamos a análise de resultados, com os percursos que seriam usados para fazer várias entregas de encomendas, ou seja, percursos que têm múltiplas paragens obrigatórias. Nestes, a diferença entre os valores obtidos pelos algoritmos informados e os algoritmos não informados torna-se ainda mais visível, sobretudo comparando os informados com a dfs.

Comentários Finais e Conclusão

Concluída a segunda parte do projeto de Inteligência Artificial pudemos observar que, tal como lecionado nas aulas, o algoritmo de pesquisa A* é o mais eficaz, não só por chegar sempre à solução ótima, mas também porque na maior parte das vezes é o método que tem menores complexidades tanto a nível temporal como espacial.

Por outro lado, como podemos observar na tabela apresentada no tópico anterior, por vezes conseguimos obter um percurso ótimo utilizando o algoritmo de pesquisa Gulosa.

Contudo, dos algoritmos de pesquisa não informados é improvável obter uma solução ótima, dado que, como já foi mencionado, estes pretendem apenas obter um caminho para o objetivo, não tendo em atenção o custo das arestas. São, por isso, mais pertinentes para a procura de caminhos em grafos não pesados. Dentro destes, podemos também constatar que os algoritmo de busca iterativa de profundidade limitada é dos algoritmos de procura não informada mais eficiente, dado que utiliza uma estrutura mais simples que o bfs, mas à semelhança desse escolhe o menor caminho entre a origem e o objetivo, ou seja, o caminho que passa por menos nodos.

Assim chegamos à conclusão que para uma pesquisa mais eficaz devemos usar o algoritmo A*, apesar de também ser bom o método da gulosa não é tão constante quanto à otimalidade da solução. Os restantes algoritmos podem obter boas soluções, mas é muito menos provável que tal aconteça.

Por último, a realização da segunda fase permitiu-nos aprofundar e pôr em prática as técnicas de formulação de problemas e a implementação de alguns algoritmos de procura lecionados nas aulas.

Pensamos que, de um modo geral, conseguimos atingir todos os objetivos propostos.