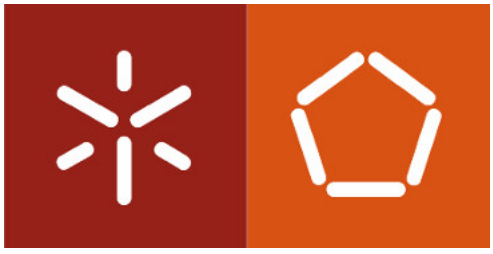

Universidade do Minho
Licenciatura em Engenharia Informática

Inteligência Artificial

Fase 2 - Grupo 38

André Silva - A87958 Armando Silva - A87949
João Nunes - A87972

Janeiro 2022



Conteúdo

Lista de Figuras

1 Introdução

No âmbito da Unidade Curricular de Inteligência Artificial, foi desenvolvido um sistema de representação de conhecimento e raciocínio capaz de completar o programa construído na fase 1 sobre a área de logística de distribuição de encomendas.

Nesta 2ª fase foi-nos proposto a atualização do sistema anteriormente desenvolvido com a implementação de um sistema de recomendação de circuitos de entrega de encomendas. Isto é, a implementação de algoritmos de procura com a intenção de encontrar os melhores caminhos de um ponto de origem a um ponto de destino, em grafos.

Ao longo deste relatório, será explicado o problema e todos os algoritmos de procura utilizados para a obtenção de diferentes soluções, estas que serão posteriormente comparadas.

2 Problema

2.1 Formulação

A figura seguinte representa um grafo de localidades, no distrito de Braga, contendo as distâncias entre as localidades e a estimativa de distância (quadrado verde) de cada localidade até à localidade final, Amares (em verde, localidade onde está o centro de distribuições *Green Distribution*). De salientar, que também está implementado um custo e uma estimativa relacionada com o tempo que demora até chegar à localidade, estes são calculadas no sistema tendo em conta o peso da encomenda e o veículo utilizado.

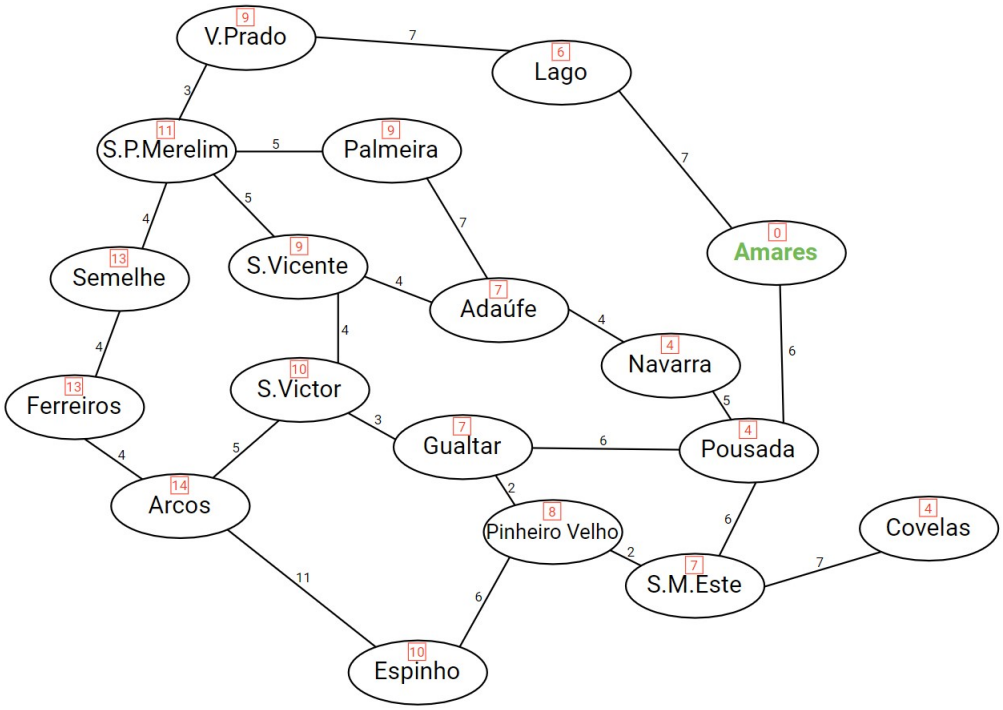

Figura 1. Grafo do cenário construído

Estado inicial: Qualquer localidade presente no grafo (depende do circuito)

Estado objetivo: Amares

Operadores:

Nome	Pré-condição	Efeito
Conduzir entre localidades	Destino escolhido está ligado à origem	Move para a localidade de destino

Custo: Cada ação tem o custo representado por distância (apresentado no grafo) ou tempo (calculado no sistema) em cada aresta.

2.2 Estratégias

Para a resolução do problema utilizamos diferentes algoritmos de procura que estão divididos em dois tipos.

2.2.1 Procura não informada

Algoritmos que apenas usam as informações disponíveis na definição inicial do problema. Explicamos a seguir três algoritmos que pertencem a este tipo.

Largura (BFS)

Ao contrario da procura em profundidade, a procura em largura começa por percorrer todos os nós adjacentes e só depois percorre os nos que são acessíveis a partir desses adjacentes. Deste modo é garantido que encontramos o caminho com menos passos para um dado nodo. Só deve ser utilizado em pequenos problemas pois requer muita memoria para guardar os caminhos que ainda faltam expandir.

Profundidade (DFS)

Na procura em profundidade tentamos sempre expandir para um nó mais profundo da árvore, e caso isso não seja possível recuamos e tentamos outros caminhos. Este algoritmo permite obter todos os caminhos possíveis desde um vértice inicial. No entanto, não pode ser usado para árvores com profundidade infinita e não garante que o caminho encontrado seja o mais curto. É um bom algoritmo para problemas com muitas soluções pois requer pouca memoria.

Busca iterativa limitada em Profundidade (Iterative-DS)

Este tipo de procura é semelhante á procura em profundidade. No entanto, visa resolver o problema de não poder ser usado em árvores com profundidade infinita ao definir um nível máximo de profundidade de procura.

2.2.2 Procura informada

Algoritmos que recebem informação adicional sobre o estado do objetivo final. Esta informação é obtida através de de funções que estimam o quão perto o estado atual está do objetivo final.

Gulosa

A estratégia de procura gulosa usa uma estimativa do caminho mais curto para o estado objetivo para decidir qual o nodo a expandir. Pode retornar uma solução ou falhar.

Estrela (A*)

Tal como a procura gulosa esta usa uma estimativa do caminho mais curto ate ao estado objetivo, mas a este soma o custo do caminho desde o nó inicial até ao nodo em que estamos, usando a seguinte função:

$$f(n) = g(n) + h(n)$$

g(n) = custo do caminho desde o nodo inicial até n.
h(n) = estimativa de custo desde o nodo n até ao nodo objetivo.
f(n) = custo estimado de uma solução que passa por n.

2.2.3 Complexidade dos algoritmos

Para analisarmos a complexidade dos algoritmos apresentados acima, temos de ter em conta que esta depende sempre de alguns fatores. Esses fatores são os seguintes:

- b - máximo fator de ramificação (número máximo de sucessores de um nó)
- d - profundidade da melhor solução
- m - máxima profundidade do espaço de estados

A seguir apresentamos duas tabelas com os vários pontos de análise da complexidade das estratégias.

Estratégia	Completo	Tempo	Espaço	Melhor solução
BFS	Sim, se *b* finito	O(*b*^*d*+1)	O(*b*^*d*+1)	Sim, se custos forem iguais
DFS	Não	O(*b*^*m*)	O(*bm*)	Não, devolve a primeira solução
Iterative-DS	Sim	O(*b*^*d*)	O(*b*^*d*)	Sim, se custos forem iguais
Gulosa	Não, pode entrar em ciclos	Pior caso: O(*b*^*m*) Melhor caso: O(*bd*)	Pior caso: O(*b*^*m*) Melhor caso: O(*bd*)	Nem sempre encontra
A*	Sim	Número de nodos com g(*n*)+h(*n*)<=C*	Número de nodos com g(*n*)+h(*n*)<=C*	Sim

3 Implementação

3.1 Base de conhecimento

Para representarmos todo o problema no nosso sistema, decidimos implementar o seguinte o conhecimento:

- **aresta:** LocalidadeOrigem, LocalidadeDestino, Distância.
- **estimaD:** Localidade, EstimaDistância.
- **objetivo:** Localidade.
- **transporte:** Veículo, Peso, VelocidadeMédia.

De salientar, que aproveitamos o conhecimento das encomendas já construído na fase 1 deste projeto. Este que foi aumentado para abrangermos mais localidades, o que permite realizarmos mais testes no final da implementação.

3.1.1 Aresta

As arestas são caracterizadas pela localidade origem, a localidade destino e a distância entre essas mesmas localidades.

```
aresta(amares, pousada, 6).
aresta(amares, lago, 7).
aresta(pousada, saomamedeeste, 6).
aresta(pousada, navarra, 5).
aresta(pousada, gualtar, 6).
aresta(covelas, saomamedeeste, 7).
aresta(saomamedeeste, pinheirovelho, 2).
aresta(navarra, adaufe, 4).
aresta(gualtar, pinheirovelho, 2).
aresta(gualtar, saovictor, 3).
aresta(pinheirovelho, espinho, 6).
aresta(adaufe, palmeira, 7).
aresta(adaufe, saovicente, 4).
aresta(saovictor, saovicente, 4).
aresta(saovictor, arcos, 5).
aresta(espinho, arcos, 11).
aresta(palmeira, saopedromerelim, 5).
aresta(saovicente, saopedromerelim, 5).
aresta(arcos, ferreiros, 4).
aresta(lago, vilaprado, 7).
aresta(vilaprado, saopedromerelim, 3).
aresta(saopedromerelim, semelhe, 4).
aresta(ferreiros, semelhe, 4).
```

Figura 2. Arestas

3.1.2 EstimaD

A estimaD caracteriza a estimativa de distância do local dado até ao local do centro de distribuições, que neste caso é em Amares. É caracterizada pela localidade e pela estima.

```
estimaD(amares, 0).
estimaD(pousada, 4).
estimaD(lago, 6).
estimaD(covelas, 4).
estimaD(saomamedeeste, 7).
estimaD(navarra, 4).
estimaD(gualtar, 7).
estimaD(pinheirovelho, 8).
estimaD(adaufe, 7).
estimaD(saovictor, 10).
estimaD(espinho, 10).
estimaD(palmeira, 9).
estimaD(saovicente, 9).
estimaD(arcos, 14).
estimaD(semelhe, 13).
estimaD(vilaprado, 9).
estimaD(saopedromerelim, 11).
estimaD(ferreiros, 13).
```

Figura 3. Estimativas da distância

3.1.3 Objetivo

O objetivo representa a localidade do centro de distribuições *Green Distribution*.

```
objetivo(amares).
```

Figura 4. Objetivo

3.1.4 Transporte

Representamos aqui os três diferentes veículos que podem ser utilizados para o transporte das encomendas. É caracterizado pelo tipo de veículo, o peso máximo que consegue transporte e a sua velocidade média.

```
transporte(bicicleta, 5, 10).
transporte(moto, 20, 35).
transporte(carro, 100, 25).
```

Figura 5. Transportes

3.2 Indicadores de produtividade

Foi-nos proposta a utilização de dois indicadores de produtividade no nosso sistema, sendo eles a distância percorrida e o tempo demorado para a entrega de uma encomenda.

Distância percorrida

Distância total percorrida para entregar a encomenda. Esta informação está representada diretamente no grafo entre cada aresta. Está também representada uma estimativa de distância desde qualquer localidade até à localidade do centro de distribuições da *Green Distribution*, em Amares.

Tempo de entrega

Tempo total demorado para entregar uma encomenda. Esta informação não está representada no grafo, sendo necessário efetuar o respetivo cálculo.

Começamos por escolher o veículo que vai transportar a encomenda através do predicado *escolheVeiculo*. Este vai primeiramente calcular a velocidade média, predicado *velocidadeMedia*, dos três diferentes veículos tendo em conta o peso da entrega. Depois vai calcular com a velocidade e a estimativa de distância total, até ao local da entrega, o tempo máximo que demorará a chegar ao destino. Caso o tempo esteja dentro do prazo estipulado na entrega, então esse vai ser o veículo escolhido para a entrega. De salientar, que temos sempre preferência pelo transporte por bicicleta, depois por moto e por último recurso a utilização do carro.

```
% ----------------------------------------
% Escolha do veículo a usar tendo em conta o peso da entrega
% escolheVeiculo: Distância, TempoMáximo, Peso, VelocidadeMédia, Veículo
escolheVeiculo(Distancia, TempoMaximo, Peso, VelocidadeMedia, Veiculo) :-
    velocidadeMedia(Peso, bicicleta, VB),
    velocidadeMedia(Peso, moto, VM),
    velocidadeMedia(Peso, carro, VC),
    (TempoMaximo > Distancia/VB, Peso =< 5 -> Veiculo = bicicleta, VelocidadeMedia is VB, !;
    TempoMaximo > Distancia/VM, Peso =< 20 -> Veiculo = moto, VelocidadeMedia is VM, !;
    TempoMaximo > Distancia/VC, Peso =< 100 -> Veiculo = carro, VelocidadeMedia is VC, !).

% ----------------------------------------
% Cálculo da velocidade média do veículo tendo em conta o peso
% velocidadeMedia: Peso, Veículo, VelocidadeMédia
velocidadeMedia(Peso, Veiculo, V) :-
    Veiculo == bicicleta -> V is (10 - Peso*0.7);
    Veiculo == moto -> V is (35 - Peso*0.5);
    Veiculo == carro -> V is (25 - Peso*0.1).
```

Figura 6. Predicados *escolheVeiculo* e *velocidadeMedia*

Por fim é realizado o cálculo do tempo que demorará a realizar o caminho de uma localidade para outra, pelo predicado *custoT*. Também é feito o cálculo da estimativa de tempo que demorará a chegar à localidade do centro de distribuições da *Green Distribution*.

```
% ----------------------------------------
% Cálculo do custo do tempo com base na próxima localidade e veículo
% custoT: Peso, Localidade, PróximaLocalidade, Veículo, CustoTempo
custoT(Peso, Localidade, ProxLocalidade, Veiculo, CustoT) :-
    ligacaoC(Localidade, ProxLocalidade, Distancia),
    velocidadeMedia(Peso, Veiculo, VM),
    CustoT is Distancia/VM.

% ----------------------------------------
% Cálculo da estima do tempo com base na localidade e veículo
% estimaT: Peso, Localidade, Veículo, EstimaTempo
estimaT(Peso, Localidade, Veiculo, EstimaT) :-
    estimaD(Localidade, Distancia),
    velocidadeMedia(Peso, Veiculo, VM),
    EstimaT is Distancia/VM.
```

Figura 7. Predicados *custoT* e *estimaT*

3.3 Funcionalidades

As funcionalidades pedidas no enunciado foram todas desenvolvidas no ficheiro 'funcionalidades.pl'. Estas necessitam de outras regras que estão desenvolvidas em 'auxiliares.pl', como por exemplo os algoritmos.

Foram desenvolvidas as seguintes funcionalidades, tendo em conta a localização do ponto de partida (centro de distribuições):

1. Gerar os **circuitos de entrega**, caso existam, que cubram um determinado território;
2. Identificar quais os circuitos com **maior número de entregas** (por volume e peso);
3. Comparar circuitos de entrega tendo em conta os **indicadores de produtividade**;
4. Escolher o circuito **mais rápido** (usando o critério da distância);
5. Escolher o circuito **mais ecológico** (usando um critério de tempo).

Gerar circuitos de entrega que cubram um determinado território

Com a intenção de obtermos todos os circuitos possíveis que abrangem uma dada localidade presente no grafo, desenvolvemos o predicado *geraCircuitos*. Este obtém utilizando o algoritmo BFS todos os caminhos possíveis que passa pela localidade dada. Devolve o número total de circuitos e uma lista com todos esses circuitos encontrados.

```
% ------------------------------------------------------------------------------------------
% FUNCIONALIDADE 1: Gerar os circuitos de entrega que cubram um determinado território
% Teste: geraCircuitos(saovictor, Lista, Tamanho).
% ------------------------------------------------------------------------------------------
% Gera todos os trajetos possíveis que passam por uma dada freguesia
% geraCircuitos: Freguesia, Lista, TamanhoLista
geraCircuitos(Freguesia, L, Length) :-
    todosCaminhos(L1),
    todosCaminhosTerritorio(Freguesia, L1, L),
    length(L, Length).
```

Figura 8. Predicado *geraCircuitos*

```
% ----------------------------------------
% Obtém todos os caminhos possíveis para todas as freguesias de todas as encomendas
% até chegar a Amares (freguesia do centro de distribuições)
% todosCaminhos: Solução
todosCaminhos(L) :-
    findall(Freguesia, encomenda(_, _, _, _, _, Freguesia, _, _, _, _), L1),
    getAllCaminhos(L1, L).

getAllCaminhos([], []).
getAllCaminhos([C|T], L) :-
    findall(Lista, resolve_lp(0, C, amares, _, _, Lista/_), L1),
    getAllCaminhos(T, L2),
    append(L1, L2, L).
```

Figura 9. Predicado *todosCaminhos* e *getAllCaminhos*

```
% ----------------------------------------
% Obtém todos os caminhos que passam por uma certa freguesia
% todosCaminhosTerritorio: Freguesia, CaminhosPossíveis, Solução
todosCaminhosTerritorio(_, [], []).
todosCaminhosTerritorio(Freguesia, [Caminho|T], L) :-
    (member(Freguesia, Caminho) ->
        todosCaminhosTerritorio(Freguesia, T, L1),
        append([Caminho], L1, L);
        todosCaminhosTerritorio(Freguesia, T, L)).
```

Figura 10. Predicado *todosCaminhosTerritorio*

Identificar os circuitos com maior número de entregas, por volume e peso

De forma a obtermos os circuitos com o maior número de entregas, por volume e peso, que passam por uma certa localidade, implementamos os predicados *circuitoComMaisEntregaPeso* e *circuitoComMaisEntregaVolume*. Este vai utilizar o algoritmo BFS para encontrar todos os caminhos que contenham a localidade dada construindo uma lista de pares com o respetivo volume ou peso e com o respetivo caminho. É depois realizada a ordenação dessa lista e devolvidos os três caminhos com maior número de volume ou peso possível.

```
% Identificar quais os circuitos com maior número de entregas por peso
% circuitosComMaisEntregaPeso: Freguesia, Caminho1, Caminho2, Caminho3
circuitosComMaisEntregaPeso(Freguesia, C1, C2, C3) :-
    geraCircuitosSemVolta(Freguesia, L1),
    circuitosComMaisEntregaAUX(0, L1, C),
    sort(0, @>, C, [C1, C2, C3|_]).      ArmandoBSilva99, 12 minutes ago

% Identificar quais os circuitos com maior número de entregas por volume
% circuitosComMaisEntregaVolume: Freguesia, Caminho1, Caminho2, Caminho3
circuitosComMaisEntregaVolume(Freguesia, C1, C2, C3) :-
    geraCircuitosSemVolta(Freguesia, L1),
    circuitosComMaisEntregaAUX(1, L1, C),
    sort(0, @>, C, [C1, C2, C3|_]).

% Variável 'Opcao': 0 -> Peso ; 1 -> Volume
circuitosComMaisEntregaAUX(_, [], []).
circuitosComMaisEntregaAUX(Opcao, [Caminho], [(Max,Caminho3)]) :-
    contaPesoVolumeCaminho(Opcao, Caminho, Max),
    reverse(Caminho, Caminho1),
    apagaPrimeiro(Caminho, Caminho2),
    append(Caminho1, Caminho2, Caminho3).
circuitosComMaisEntregaAUX(Opcao, [Caminho|T], L) :-
        contaPesoVolumeCaminho(Opcao, Caminho, Count),
        circuitosComMaisEntregaAUX(Opcao, T, Ls),
        reverse(Caminho, Caminho1),
        apagaPrimeiro(Caminho, Caminho2),
        append(Caminho1, Caminho2, Caminho3),
        append([(Count, Caminho3)], Ls, L).
```

Figura 11. Predicados *circuitosComMaisEntregaPeso*, *circuitosComMaisEntregaVolume* e auxiliar

```
% ----------------------------------------
% Calcula o Volume ou Peso de um caminho
% Variável 'Opcao': 0 -> Peso ; 1 -> Volume
% contaPesoVolumeCaminho: Caminho, CálculoFinal
contaPesoVolumeCaminho(_, [], 0).
contaPesoVolumeCaminho(Opcao, [Freguesia], Max) :- contaPesoVolume(Opcao, Freguesia, Max).
contaPesoVolumeCaminho(Opcao, [Freguesia|T], Max) :-
    contaPesoVolume(Opcao, Freguesia, Max1),
    contaPesoVolumeCaminho(Opcao, T, Max2),
    Max is Max1 + Max2.

% ----------------------------------------
% Soma o peso ou volume de todas as encomendas de uma Freguesia
% Variável 'Opcao': 0 -> Peso ; 1 -> Volume
% contaPesoVolume: Freguesia, Soma
contaPesoVolume(Opcao, Freguesia, Sum) :-
    (Opcao == 0 ->
        findall(Peso, (encomenda(_,_,_, Peso, _, Freguesia, _,_,_,_)), L),
        sum_list(L, Sum);
    Opcao == 1 ->
        findall(Volume, (encomenda(_,_,_, _, Volume, Freguesia, _,_,_,_)), L),
        sum_list(L, Sum)).
```

Figura 12. Predicados *contaPesoVolumeCaminho* e *contaPesoVolume*

Comparar circuitos de entrega tendo em conta os indicadores de produtividade

Desenvolvemos o predicado *comparaCircuitos* que nos permite obter uma boa comparação entre as diferentes estratégias de procura implementadas, tendo em conta os indicadores de produtividade. Este predicado recebe qualquer encomenda e devolve a localidade, o primeiro caminho que os vários algoritmos obtiveram e o respetivo indicador de produtividade.

```
% Comparar circuitos de entrega obtidos através de cada algoritmo, tendo em conta os indicadores de produtividade
% Variável 'Custo': 0 -> Distância ; 1 -> Tempo
% comparaCircuitos: IdentificadorCusto, IDEncomenda, Freguesia, TravessiaBFS, TravessiaDFS, TravessiaIDS, TravessiaG, TravessiaAE
comparaCircuitos(Custo, IDEncomenda, Freguesia, TravessiaBFS, TravessiaDFS, TravessiaIDS, TravessiaG, TravessiaAE) :-
    encomenda(IDEncomenda,_,_,_,_,Freguesia,_,_,_,_),
    (Custo == 0 ->
        geraCircuitosAlgoritmos(bfs, 0, IDEncomenda, TravessiaBFS),
        geraCircuitosAlgoritmos(dfs, 0, IDEncomenda, TravessiaDFS),
        geraCircuitosAlgoritmos(ids, 0, IDEncomenda, TravessiaIDS),
        geraCircuitosAlgoritmos(gulosa, 0, IDEncomenda, TravessiaG),
        geraCircuitosAlgoritmos(aestrela, 0, IDEncomenda, TravessiaAE);
    Custo == 1 ->
        geraCircuitosAlgoritmos(bfs, 1, IDEncomenda, TravessiaBFS),
        geraCircuitosAlgoritmos(dfs, 1, IDEncomenda, TravessiaDFS),
        geraCircuitosAlgoritmos(ids, 1, IDEncomenda, TravessiaIDS),
        geraCircuitosAlgoritmos(gulosa, 1, IDEncomenda, TravessiaG),
        geraCircuitosAlgoritmos(aestrela, 1, IDEncomenda, TravessiaAE)
    ).
```

Figura 13. Predicado *comparaCircuitos*

```
% ----------------------------------------
% Gera as soluções com o uso de algoritmos informados e não informados
% Variável 'Algoritmo', identifica o algoritmo a utilizar
% Variável 'Custo': 0 -> Distância ; 1 -> Tempo
% geraCircuitosAlgoritmos: Algoritmo, IdentificarCusto, IDEncomenda, Travessia
geraCircuitosAlgoritmos(Algoritmo, Custo, IDEncomenda, Travessia) :-
    Custo >= 0, Custo =< 1,
    (Algoritmo \== 'gulosa', Algoritmo \== 'aestrela' ->
        geraCircuitosNI(Algoritmo, Custo, IDEncomenda, Travessia);
        geraCircuitosI(Algoritmo, Custo, IDEncomenda, Travessia)).
```

Figura 14. Predicado *geraCircuitosAlgoritmos*

```
% ----------------------------------------
% Gera circuitos recebendo a encomenda, utiliza algoritmos não informados (BFS, DFS, IDS)
% Variável 'Algoritmo', identifica o algoritmo a utilizar
% Variável 'Custo': 0 -> Distância ; 1 -> Custo
% geraCircuitosNI: Algoritmo, IdentificadorCusto, IDEncomenda, Travessia/CustoF
geraCircuitosNI(Algoritmo, Custo, IDEncomenda, Travessia/CustoF) :-
    encomenda(IDEncomenda, _, _, Peso, _, Freguesia, _, _, Prazo, _),
    estimaD(Freguesia, Distancia),
    escolheVeiculo(Distancia, Prazo, Peso, _, Veiculo),
    (Algoritmo == 'bfs' ->
        resolve_lp(Custo, Freguesia, amares, Peso, Veiculo, Travessia/CustoF);
    Algoritmo == 'dfs' ->
        resolve_pp(Custo, Freguesia, Peso, Veiculo, Travessia/CustoF);
    Algoritmo == 'ids' ->
        resolve_pil(Custo, Freguesia, Peso, Veiculo, 10, Travessia/CustoF)).
```

Figura 15. Predicado *geraCircuitosNI*

```
% ----------------------------------------
% Gera circuitos recebendo a encomenda, utiliza algoritmos informados (Gulosa, Estrela)
% Variável 'Algoritmo', identifica o algoritmo a utilizar
% Variável 'Custo': 0 -> Distância ; 1 -> Custo
% geraCircuitosI: Algoritmo, IdentificadorCusto, IDEncomenda, Travessia/CustoF
geraCircuitosI(Algoritmo, Custo, IDEncomenda, Travessia/CustoF) :-
    encomenda(IDEncomenda, _, _, Peso, _, Freguesia, _, _, Prazo, _),
    estimaD(Freguesia, Distancia),
    escolheVeiculo(Distancia, Prazo, Peso, _, Veiculo),
    (Algoritmo == 'gulosa' ->
        resolve_gulosa(Custo, Freguesia, Veiculo, Peso, Travessia/CustoF);
    Algoritmo == 'aestrela' ->
        resolve_aestrela(Custo, Freguesia, Veiculo, Peso, Travessia/CustoF)).
```

Figura 16. Predicado *geraCircuitosI*

Circuito mais rápido

Para o cálculo do circuito mais rápido foi considerado o circuito com a menor distância percorrida. Para tal, construímos o predicado *circuitoMaisRapido* que obtem todos os circuitos possíveis através da estratégia DFS e devolve o circuito com a menor distância calculada.

```
% Obter o circuito mais rápido usando o critério da distância
% circuitoMaisRapido: IDEncomenda, Freguesia, Travessia
circuitoMaisRapido(IDEncomenda, Freguesia, Travessia) :-
    encomenda(IDEncomenda, _, _, _, _, Freguesia, _, _, _, _),
    obtemMelhor(0, Freguesia, _, _, Travessia).
```

Figura 17. Predicado *circuitoMaisRapido*

Circuito mais ecológico

Para o cálculo do circuito mais ecológico foi considerado o circuito com o menor tempo obtido depois da entrega. O tempo entre cada localidade é calculado através do predicado *custoT* explicado anteriormente, relembrando que utiliza sempre o veículo mais ecológico possível. Para obtermos o circuito pretendido, primeiro obtemos todos os circuitos possíveis através da estratégia DFS e devolvemos o circuito com o menor tempo calculado. Para tal, foi desenvolvido o predicado *circuitoMaisEscologico*.

```
% Obter o circuito mais ecológico usando o critério de tempo
% circuitoMaisEcologico: IDEncomenda, Freguesia, Peso, Veiculo, Travessia
circuitoMaisEcologico(IDEncomenda, Freguesia, Peso, Veiculo, Travessia) :-
    encomenda(IDEncomenda, _, _, Peso, _, Freguesia, _, _, Prazo, _),
    estimaD(Freguesia, Distancia),
    escolheVeiculo(Distancia, Prazo, Peso, _, Veiculo),
    obtemMelhor(1, Freguesia, Peso , Veiculo, Travessia).
```

Figura 18. Predicado *circuitoMaisEcologico*

4 Resultados

4.1 Funcionalidades

Exibimos os resultados obtidos após a realização das várias funcionalidades desenvolvidas no sistema, de modo a comprovarmos o funcionamento das mesmas.

Gerar circuitos de entrega que cubram um determinado território

Utilizamos o seguinte comando de teste *geraCircuitos(saovictor, Lista, Tamanho).*.

```
?- geraCircuitos(saovictor, Lista, Tamanho).
Lista = [[amares, pousada, gualtar, saovictor, saovicente, saopedromerelim, vilaprado, lago|...], [amares, pousada, saomamedeeste, pinheirovelho, gualtar, saovictor, saovicente|...], [amares, pousada, gualtar, saovictor, saovicente, adaufe|...], [amares, pousada, gualtar, saovictor, arcos|...], [amares, pousada, saomamedeeste, pinheirovelho|...], [amares, pousada, gualtar|...], [amares, pousada|...], [amares|...], [...|...]|...],
Tamanho = 349 .
```

Figura 19. Teste realizado para a funcionalidade 1

Identificar os circuitos com maior número de entregas, por volume e peso

Utilizamos os seguintes comandos de teste *circuitosComMaisEntregaPeso(vilaprado, P1, P2, P3).* e *circuitosComMaisEntregaVolume(palmeira, P1, P2, P3).*, respetivamente para o peso e o volume.

```
?- circuitosComMaisEntregaPeso(vilaprado, P1, P2, P3).
P1 =  (155, [amares, lago, vilaprado, saopedromerelim, semelhe, ferreiros, arcos, espinho|...]),
P2 =  (153, [amares, lago, vilaprado, saopedromerelim, semelhe, ferreiros, arcos, saovictor|...]),
P3 =  (143, [amares, lago, vilaprado, saopedromerelim, palmeira, adaufe, navarra, pousada|...]) .

?- circuitosComMaisEntregaVolume(palmeira, P1, P2, P3).
P1 =  (394, [amares, lago, vilaprado, saopedromerelim, semelhe, ferreiros, arcos, espinho|...]),
P2 =  (394, [amares, lago, vilaprado, saopedromerelim, palmeira, adaufe, saovicente, saovictor|...]),
P3 =  (384, [amares, lago, vilaprado, saopedromerelim, semelhe, ferreiros, arcos, espinho|...]) .
```

Figura 20. Teste realizado para a funcionalidade 2

Comparar circuitos de entrega tendo em conta os indicadores de produtividade

Utilizamos os seguintes comandos de teste *comparaCircuitos(0, 3, Freguesia, BFS, DFS, IDS, G, AE).* e *comparaCircuitos(1, 16, Freguesia, BFS, DFS, IDS, G, AE).*, respetivamente para comparar circuitos tendo em conta a distância e o tempo.

```
?- comparaCircuitos(0, 3, Freguesia, BFS, DFS, IDS, G, AE).
Freguesia = ferreiros,
BFS = IDS, IDS = AE, AE = [amares, pousada, gualtar, saovictor, arcos, ferreiros, arcos, saovictor|...]/48,
DFS = [amares, pousada, saomamedeeste, pinheirovelho, espinho, arcos, saovictor, saovicente|...]/128,
G = [amares, pousada, navarra, adaufe, palmeira, saopedromerelim, semelhe, ferreiros|...]/70 .

?- comparaCircuitos(1, 16, Freguesia, BFS, DFS, IDS, G, AE).
Freguesia = semelhe,
BFS = IDS, IDS = [amares, lago, vilaprado, saopedromerelim, semelhe, saopedromerelim, vilaprado, lago|...]/1.5272727272727273,
DFS = [amares, pousada, saomamedeeste, pinheirovelho, espinho, arcos, saovictor, saovicente|...]/4.363636363636363,
G = [amares, pousada, navarra, adaufe, palmeira, saopedromerelim, semelhe, saopedromerelim|...]/2.254545454545455,
AE = [amares, lago, vilaprado, saopedromerelim, semelhe, saopedromerelim, vilaprado, lago|...]/1.5272727272727271 .
```

Figura 21. Teste realizado para a funcionalidade 3

Circuito mais rápido

Utilizamos o seguinte comando de teste *circuitoMaisRapido(14, Freguesia, Caminho).*.

```
?- circuitoMaisRapido(14, Freguesia, Caminho).
Freguesia = covelas,
Caminho = [amares, pousada, saomamedeeste, covelas, saomamedeeste, pousada, amares]/38.
```

Figura 22. Teste realizado para a funcionalidade 4

Circuito mais ecológico

Utilizamos os seguintes comandos de teste *circuitoMaisEcologico(5, Freguesia, Peso, Veiculo, Caminho)*, *circuitoMaisEcologico(6, Freguesia, Peso, Veiculo, Caminho).* e *circuitoMaisEcologico(13, Freguesia, Peso, Veiculo, Caminho).*, para obtermos caminhos em que são escolhidos diferentes transportes.

```
?- circuitoMaisEcologico(5, Freguesia, Peso, Veiculo, Caminho).
Freguesia = espinho,
Peso = 2,
Veiculo = bicicleta,
Caminho = [amares, pousada, gualtar, pinheirovelho, espinho, pinheirovelho, gualtar, pousada|...]/4.651162790697675.

?- circuitoMaisEcologico(6, Freguesia, Peso, Veiculo, Caminho).
Freguesia = adaufe,
Peso = 8,
Veiculo = moto,
Caminho = [amares, pousada, navarra, adaufe, navarra, pousada, amares]/0.967741935483871.

?- circuitoMaisEcologico(13, Freguesia, Peso, Veiculo, Caminho).
Freguesia = pousada,
Peso = 25,
Veiculo = carro,
Caminho = [amares, pousada, amares]/0.5333333333333333.
```

Figura 23. Teste realizado para a funcionalidade 5

4.2 Estatísticas

Apresentamos os resultados obtidos após a execução da mesma funcionalidade para todos os algoritmos, de forma a que seja possível realizar uma análise comparativa homogénea entre todas as estratégias. Para tal, cada algoritmo vai obter o primeiro caminho do ID de encomenda 3.

Completamos a análise, com o recurso às regras predefinidas pelo PROLOG *time* e *statistics* de forma a obtermos a memória e o tempo passado sobre a execução de cada algoritmo.

```
?- statisticsBFS(1, 3, Caminho, Memoria).
% 1,865 inferences, 0.001 CPU in 0.001 seconds (100% CPU, 3663788 Lips)
Caminho = [amares, pousada, gualtar, saovictor, arcos, ferreiros, arcos, saovictor|...]/5.161290322580645,
Memoria = 36176 .

?- statisticsDFS(1, 3, Caminho, Memoria).
% 433 inferences, 0.000 CPU in 0.000 seconds (99% CPU, 3665981 Lips)
Caminho = [amares, pousada, saomamedeeste, pinheirovelho, espinho, arcos, saovictor, saovicente|...]/13.763440860215052,
Memoria = 6432 .

?- statisticsIDS(1, 3, Caminho, Memoria).
% 621 inferences, 0.000 CPU in 0.000 seconds (98% CPU, 4473225 Lips)
Caminho = [amares, pousada, gualtar, saovictor, arcos, ferreiros, arcos, saovictor|...]/5.161290322580645,
Memoria = 3416 .

?- statisticsGulosa(1, 3, Caminho, Memoria).
% 522 inferences, 0.000 CPU in 0.000 seconds (98% CPU, 3920626 Lips)
Caminho = [amares, pousada, navarra, adaufe, palmeira, saopedromerelim, semelhe, ferreiros|...]/7.526881720430107,
Memoria = 8288 .

?- statisticsAEstrela(1, 3, Caminho, Memoria).
% 1,498 inferences, 0.000 CPU in 0.000 seconds (100% CPU, 3246162 Lips)
Caminho = [amares, pousada, gualtar, saovictor, arcos, ferreiros, arcos, saovictor|...]/5.161290322580645,
Memoria = 27488 .
```

Figura 24. Estatísticas obtidas em cada algoritmo

Depois de demonstrada a execução dos testes, e obtidas todas as estatísticas pretendidas, construímos a seguinte tabela.

Estratégia	Tempo (segundos)	Espaço	Profundidade /Custo	Melhor solução
BFS	0.001	36176	5.1613	Sim
DFS	0.000	6432	13.7634	Não
Iterative-DS	0.000	3416	5.1613	Sim
Gulosa	0.000	8288	7.5269	Não
A*	0.000	27488	5.1613	Sim

Figura 25. Tabela comparativa dos algoritmos

Como é possível observar, tanto o algoritmo BFS, Iterative-DS e A* encontraram a melhor solução, mas em termos de espaço utilizado são bastantes diferentes. Neste caso, temos a Iterative-DS a obter o menor espaço utilizado, a seguir temos A* e por fim a BFS. Isto pode ter acontecido devido à não complexidade do grafo construído no inicio do problema. De salientar, que a BFS é a que demora mais tempo e ocupa mais espaço a obter a solução.

5 Conclusão

Dado por concluída a 2ª e última fase deste projeto, consideramos importante realizar uma análise crítica, contendo os pontos positivos e negativos, considerar possibilidades para trabalho futuro, e ainda, realizar uma avaliação final do trabalho realizado.

O cenário construído pelo grupo não está muito completo, como se pode ver nos resultados das estatísticas, mas possui uma enorme proximidade à realidade, abrindo hipóteses para os algoritmos de pesquisa encontrarem diversas soluções com diferentes custos. Todas as funcionalidades pedidas estão operacionais e a organização e documentação escrita em todos os ficheiros também é um ponto positivo no nosso trabalho.

O grupo sentiu as maiores dificuldades na implementação e adaptação dos diversos algoritmos de pesquisa ao cenário construído, pois envolveu a adição de diferentes fatores a todos os algoritmos.

Para trabalho futuro, o grupo considera que uma expansão em termos de número de localidades utilizadas no cenário poderia ser realizada na tentativa de obter uma melhor distinção das estratégias. A adição de mais funcionalidades também seria uma boa maneira de completar ainda mais o projeto.

Por fim, consideramos que o trabalho final foi satisfatório dado que contém todas as funcionalidades propostas sendo capaz de obter diferentes soluções para diferentes estratégias.