Universidade do Minho

Mestrado Integrado em Engenharia Informática
Licenciatura em Engenharia Informática

Inteligência Artificial

Trabalho Prático - Fase 2

Grupo 12

Ana Rebelo
(A90234)

Ana Murta
(A93284)

Ana Henriques
(A93268)

Joana Oliveira
(A87956)

5 de janeiro de 2022

Resumo

No presente relatório, serão apresentadas e descritas todas as decisões tomadas para solucionar a segunda fase da problemática proposta na Unidade Curricular de Inteligência Artificial.

A segunda fase deste trabalho prático centra-se na criação de um sistema de recomendação de circuitos de entrega de encomendas para o caso de estudo. Assim sendo, procedeu-se ao desenvolvimento das estratégias de procura propostas, através de algoritmos de pesquisa informada e de pesquisa não informada. Como tal, representou-se os diversos pontos de entrega (as freguesias anteriormente associadas, na primeira fase, à entrega de encomendas) em forma de grafo.

Para garantir o sucesso na resolução deste problema, a elaboração do trabalho prático deverá permitir, consoante a localização do ponto de partida (*Green Distribuition*), implementar um conjunto de funcionalidades que serão posteriormente enunciadas.

Conteúdo

Lista de Figuras

1 Introdução

A segunda fase do trabalho prático da disciplina de Inteligência Artificial foca-se, principalmente, no caso de estudo referente aos circuitos de entregas pelo centro de distribuição *Green Distribuition*, levando à atualização do sistema anteriormente desenvolvido.

Para o efeito, é concebido o acesso a uma base de conhecimento, que será atualizada com a representação dos diversos pontos de entrega em forma de grafo de modo a aplicar os diversos algoritmos de pesquisa informada e não informada, que possuem diferentes benefícios mediante os critérios especificados para o circuito a gerar.

Posto isto, o principal objetivo na resolução deste problema é conseguir elaborar um sistema de recomendação de circuitos de entregas para o caso de estudo.

2 Formulação do problema

O problema em causa é um problema de estado único, uma vez que o ambiente é determinístico, totalmente observável, e o agente — o estafeta — "sabe" exatamente o estado em que estará. Ou seja, o estafeta sabe precisamente o local da entrega da encomenda. Deste modo, a solução é reduzida à procura de um caminho do estado inicial ao estado objetivo.

- **Estado Inicial**: Green Distribuition
- **Estado Objetivo**: Green Distribuition
- **Estados**: Os vários pontos de entrega existentes
- **Operações**: Mover-se entre as várias freguesias e entregar as encomendas
- **Custo da solução**: É determinado pela distância entre as freguesias
- **Limitação**: Peso que condiciona o meio de transporte e, por sua vez, o tempo de entrega
- **Objetivos**: Encontrar o caminho mais rápido, utilizando as distâncias entre freguesias, **e** encontrar o circuito mais eficiente, tendo em conta a distância percorrida, o peso da encomenda e a velocidade média do meio de transporte utilizado. Deste modo, o circuito que permitir uma maior rapidez na entrega e uma menor distância percorrida será o mais eficiente.

3 Base de Conhecimento

A base de conhecimento tem como fontes de conhecimento as seguintes entidades:

- **encomenda**: IdEncomenda, IdEstafeta, Peso, Volume, PontoEntrega $\rightsquigarrow$ {$\mathbb{V}$,$\mathbb{F}$}
- **grafo**: grafo(ListaPontosEntrega, ListaArestas) $\rightsquigarrow$ {$\mathbb{V}$,$\mathbb{F}$}

3.1 Encomenda

Cada `encomenda` é caracterizada pelo seu identificador, pelo identificador do `estafeta` que a entregará, pelo peso, pelo volume e pelo ponto de entrega onde será entregue. Na primeira fase, estávamos perante um problema em que as encomendas já tinham sido entregues. Nesta segunda fase, considerou-se o problema das encomendas serem futuramente entregues. Como tal, a entidade `encomenda` passou a ser caracterizada por diferentes parâmetros.

Seguidamente, são apresentados alguns exemplos:

```
encomenda(100000000001,2,10,20,'São Victor').
encomenda(123456789000,3,16,15,'Gualtar').
encomenda(300145999366,2,5,30,'São Victor').
encomenda(411188745632,4,30,25,'Maximinos').
encomenda(512200534686,4,1,25,'Maximinos').
encomenda(611111154895,5,50,40,'Dume').
encomenda(792555468332,1,4,10,'Gualtar').
encomenda(800000000234,6,25,52,'Gualtar').
encomenda(910928382779,4,8,10,'Maximinos').
encomenda(100910101098,2,4,17,'Gualtar').
encomenda(113364968333,4,18,67,'São Victor').
encomenda(121203928222,7,10,28,'Maximinos').
encomenda(136666733413,4,20,41,'Tadim').
encomenda(142019203922,6,7,21,'Gualtar').
encomenda(150393815151,5,30,77,'São Vicente').
encomenda(169998372344,3,50,87,'Esporões').
encomenda(172123212430,6,9,18,'Maximinos').
encomenda(154268745963,8,33,12,'Lamas').
encomenda(774951266458,8,10,13,'Figueiredo').
encomenda(122457456652,9,42,40,'Figueiredo').
encomenda(901230054687,10,4,5,'Priscos').
encomenda(758846125940,10,6,21,'Tebosa').
encomenda(214568744021,10,3,20,'Lamas').
encomenda(115542001235,11,11,35,'Esporões').
```

3.2 Grafo

A base de conhecimento foi atualizada com a representação dos diversos pontos de entrega disponíveis no sitema, ou seja, as freguesias associadas à entrega de encomendas, em forma de grafo. Na figura 1, encontra-se ilustrado o grafo aplicado ao nosso problema:

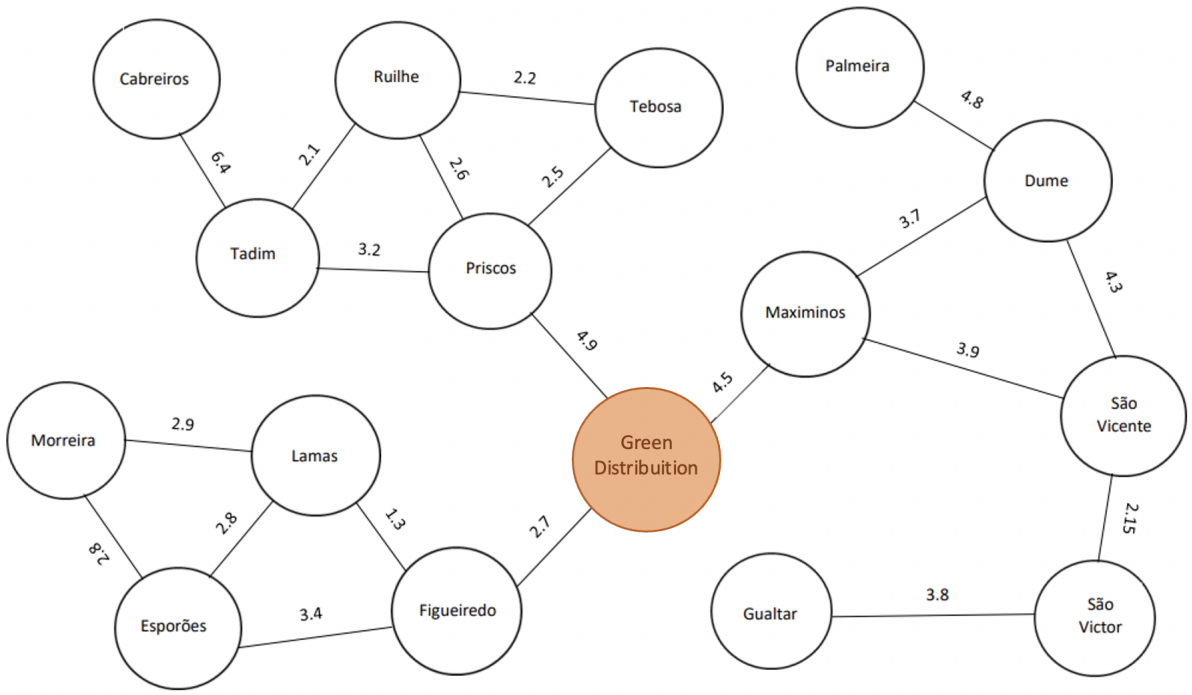

Figura 1: Ilustração do Grafo

A representação do grafo previamente ilustrado consistiu na implementação da entidade `grafo` para o caso em estudo, sendo esta apresentada a seguir:

```
g(grafo(['Green Distribuition','Tadim','Esporões','Maximinos','Dume',
         'São Vicente','São Victor','Gualtar','Figueiredo','Lamas',
         'Morreira','Palmeira','Tebosa','Priscos','Ruilhe','Cabreiros'],
        [aresta('Green Distribuition','Priscos',4.9),
         aresta('Green Distribuition','Maximinos',4.5),
         aresta('Green Distribuition','Figueiredo',2.7),
         aresta('Priscos','Tebosa',2.5),
         aresta('Priscos','Ruilhe',2.6),
         aresta('Priscos','Tadim',3.2),
         aresta('Tebosa','Ruilhe',2.2),
         aresta('Ruilhe','Tadim',2.1),
```

```
        aresta('Tadim','Cabreiros',6.4),
        aresta('Figueiredo','Lamas',1.3),
        aresta('Figueiredo','Esporões',3.4),
        aresta('Lamas','Esporões',2.8),
        aresta('Lamas','Morreira',2.9),
        aresta('Esporões','Morreira',2.8),
        aresta('Maximinos','Dume',3.7),
        aresta('Maximinos','São Vicente',3.9),
        aresta('Dume','São Vicente',4.3),
        aresta('Dume','Palmeira',4.8),
        aresta('São Vicente','São Victor',2.15),
        aresta('São Victor','Gualtar',3.8)])
).
```

Tal como é observável, cada aresta do grafo é caracterizada por um nodo inicial e por um nodo final, juntamente com a distância entre os dois nodos.

Cada circuito começa com a saída do estafeta, deslocando-se com um determinado meio de transporte, do centro de distribuição da *Green Distribuition* — `Estado Inicial`. É efetuada um paragem no ponto de entrega da encomenda e, posteriormente, o estafeta regressa ao centro de distribuição — `Estado Final`.

Por isto mesmo, foram implementados os seguintes predicados `goal/1` e `pontoEntrega/1`. Para além disto, foi construído o predicado `estima/2` para permitir a implementação dos algoritmos de pesquisa informada.

```
goal('Green Distribuition').

pontoEntrega('Tadim').
pontoEntrega('Esporões').
pontoEntrega('Maximinos').
pontoEntrega('Dume').
pontoEntrega('São Vicente').
pontoEntrega('São Victor').
pontoEntrega('Gualtar').
pontoEntrega('Priscos').
pontoEntrega('Cabreiros').
pontoEntrega('Figueiredo').
pontoEntrega('Lamas').
pontoEntrega('Morreira').
pontoEntrega('Palmeira').
pontoEntrega('Tebosa').
pontoEntrega('Ruilhe').
```

```
estima('Green Distribuition',0).
estima('Figueiredo',2.7).
estima('Priscos',4.5).
estima('Maximinos', 4.5).
estima('Dume',7.5).
estima('Palmeira',9).
estima('São Vicente',6.5).
estima('São Victor', 8).
estima('Gualtar',4).
estima('Tebosa',6.7).
estima('Ruilhe',7).
estima('Tadim',6.2).
estima('Cabreiros',11).
estima('Lamas',2).
estima('Morreira',5.7).
estima('Esporões',5).
```

4 Pesquisa Não Informada

4.1 Pesquisa em Profundidade (DFS - *Depth-First Search*)

A estratégia de procura em profundidade implementada devolve o circuito (desde a *Green Distribuition* ao ponto de entrega e, de regresso, ao local de partida) e a distância percorrida. Este circuito corresponde ao melhor caminho de acordo com as definições do algortimo em questão, que é conhecido por primeiro expandir sempre o nó mais profundo da árvore. A estratégia de procura em profundidade adotada para determinar o tempo de uma entrega segue um processo idêntico, a única diferença é que recebe o ID da encomenda e não a freguesia onde esta será entregue.

```
% # Caminho: Green Distribuition -> Ponto de Entrega -> Green Distribuition
% # Distância: Custo do Circuito Inteiro
resolveDFS(Nodo,Caminho,Distancia) :-
    profundidade(Nodo,[Nodo],CaminhoVolta,Dist),
    inverso(CaminhoVolta,CaminhoIda),
    append(CaminhoIda,[Nodo|CaminhoVolta],Caminho),
    Distancia is Dist*2.

% # IdEnc: ID da encomenda a entregar.
% # Tempo: Tempo do Circuito Inteiro.
resolveDFSTempo(IdEnc,Caminho,Tempo) :-
    encomenda(IdEnc,_,_,_,Nodo),
    profundidade(Nodo,[Nodo],CaminhoVolta,Dist),
    inverso(CaminhoVolta,CaminhoIda),
    append(CaminhoIda,[Nodo|CaminhoVolta],Caminho),
    tempoEntregaEncomenda(IdEnc,Dist,Tempo).

profundidade(Nodo,_,[],0) :- goal(Nodo).
profundidade(Nodo,Historico,[ProxNodo|Caminho],DistanciaT) :-
    g(G),adjacente(Nodo,ProxNodo,Distancia1,G),
    nao(membro(ProxNodo,Historico)),
    profundidade(ProxNodo,[ProxNodo|Historico],Caminho,Distancia2),
    DistanciaT is Distancia1 + Distancia2.
```

4.2 Pesquisa em Largura (BFS - *Breadth-First Search*)

O algoritmo de pesquisa em largura é definido por primeiro expandir o nó da raiz, depois os seus sucessores e assim sucessivamente. Deste modo, o algoritmo implementado devolve o circuito (desde a *Green Distribuition* ao ponto de entrega e, de regresso, ao local de partida) e a distância percorrida, respeitando a definição previamente mencionada. A estratégia de procura em largura adotada para calcular o tempo de uma entrega segue um processo idêntico, a única diferença é que recebe o ID da encomenda e não a freguesia onde esta será entregue.

```
% # Caminho: Green Distribuition -> Ponto de Entrega -> Green Distribuition
% # Distância: Custo do Circuito Inteiro.
resolveBFS(Nodo,Caminho,Distancia) :-
    goal(NodoFinal),
    largura(NodoFinal,[[Nodo]],CaminhoAux,Dist),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    inverso(CaminhoVolta,CaminhoIda),
    append(CaminhoIda,[Nodo|CaminhoVolta],Caminho),
    Distancia is Dist*2.

% # IdEnc: ID da encomenda a entregar.
% # Tempo: Tempo do Circuito Inteiro.
resolveBFSTempo(IdEnc,Caminho,Tempo) :-
    encomenda(IdEnc,_,_,_,Nodo),
    goal(NodoFinal),
    largura(NodoFinal,[[Nodo]],CaminhoAux,Dist),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    inverso(CaminhoVolta,CaminhoIda),
    append(CaminhoIda,[Nodo|CaminhoVolta],Caminho),
    tempoEntregaEncomenda(IdEnc,Dist,Tempo).

largura(NodoFinal,[[NodoFinal|T]|_],Caminho,0) :- inverso([NodoFinal|T],Caminho).
largura(NodoFinal,[Lista|Outros],Caminho,DistanciaT) :-
    g(G),Lista = [A|_],
    findall([X|Lista],(NodoFinal \== A, adjacente(A,X,_,G),nao(membro(X,Lista))),Novos),
    adjacente(A,X,Distancia1,G),
    concatena(Outros,Novos,Todos),
    largura(NodoFinal,Todos,Caminho,Distancia2),
    DistanciaT is Distancia1 + Distancia2.
```

4.3 Busca Iterativa Limitada em Profundidade

O algoritmo que executa a busca iterativa limitada em profundidade é bastante similar ao algoritmo DFS, diferindo, contudo, num limite. Isto é, esta estratégia de procura é caracterizada por estabelecer um limite (em profundidade) no número de nós a procurar, sendo a melhor (entre as outras estratégias de pesquisa não informada) para problemas com um grande espaço de pesquisa e em que a profundidade da solução não é conhecida. Tal como os outros algoritmos DFS e BFS, este também devolve o circuito (desde a *Green Distribuition* ao ponto de entrega e de regresso ao local de partida) e a distância percorrida. A estratégia de procura limitada em profundidade aplicada para calcular o tempo de uma entrega segue um processo idêntico, a única diferença é que recebe o ID da encomenda e não a freguesia onde esta será entregue.

```
% # Caminho: Green Distribuition -> Ponto de Entrega -> Green Distribuition
% # Distância: Custo do Circuito Inteiro.
% # Limite - Número limite de nós a procurar.
resolveLimitada(Nodo,Caminho,Distancia,Limite) :-
    profundidadeLimitada(Nodo,[Nodo],CaminhoAux,Dist,Limite),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    inverso(CaminhoVolta,CaminhoIda),
    append(CaminhoIda,[Nodo|CaminhoVolta],Caminho),
    Distancia is Dist*2.

% # IdEnc: ID da encomenda a entregar.
% # Tempo: Tempo do Circuito Inteiro.
resolveLimitadaTempo(IdEnc,Caminho,Tempo,Limite) :-
    encomenda(IdEnc,_,_,_,Nodo),
    profundidadeLimitada(Nodo,[Nodo],CaminhoAux,Dist,Limite),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    inverso(CaminhoVolta,CaminhoIda),
    append(CaminhoIda,[Nodo|CaminhoVolta],Caminho),
    tempoEntregaEncomenda(IdEnc,Dist,Tempo).

profundidadeLimitada(Nodo,_,[],0,_) :- goal(Nodo).
profundidadeLimitada(Nodo,Historico,[ProxNodo|Caminho],DistanciaT,Limite) :-
    Limite > 0,g(G),
    adjacente(Nodo,ProxNodo,Distancia1,G),
    nao(membro(ProxNodo,Historico)),
    Limite1 is Limite-1,
    profundidadeLimitada(ProxNodo,[ProxNodo|Historico],Caminho,Distancia2,Limite1),
    DistanciaT is Distancia1 + Distancia2.
```

5 Pesquisa Informada

5.1 Pesquisa Gulosa (*Greedy Search*)

A pesquisa gulosa assume, como estratégia, a expansão do nó de menor custo, escolhendo aquele que lhe parece melhor no momento, considerando a estimativa determinada para cada nó do grafo (ou seja, cada ponto de entrega e o centro de distribuição). Como não é feita uma acumulação dos custos de cada nó do circuito, a escolha do nó de menor custo no momento nem sempre garante que se encontre a solução ótima e, por isso, não pondera as consequências de cada decisão. A heurística adotada como auxílio a esta pesquisa foi a distância em linha reta desde o nodo atual até ao ponto de entrega. A metodologia de procura gulosa aplicada para determinar o tempo de uma entrega segue um processo idêntico, a única diferença é que recebe o ID da encomenda e não a freguesia onde esta será entregue.

```
% # Caminho: Green Distribuition -> Ponto de Entrega -> Green Distribuition
% # Custo: Custo do Circuito Inteiro.
resolveGulosa(Nodo,Caminho/Custo) :-
    estima(Nodo,Estima),
    agulosa([[Nodo]/0/Estima],CaminhoIda/CustoIda/_),
    inverso(CaminhoIda,CaminhoAux),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    append(CaminhoIda,CaminhoVolta,Caminho),
    Custo is CustoIda*2.

agulosa(Caminhos,Caminho) :-
    obter_melhor(Caminhos,Caminho),
    Caminho = [Nodo|_]/_/_,
    goal(Nodo).
agulosa(Caminhos,Solucao) :-
    obter_melhor(Caminhos,MelhorCaminho),
    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
    expande_gulosa(MelhorCaminho,Expandidos),
    append(OutrosCaminhos,Expandidos,NovosCaminhos),
    agulosa(NovosCaminhos,Solucao).

expande_gulosa(Caminho,Expandidos) :-
    findall(NovoCaminho,adjacenteV2(Caminho,NovoCaminho), Expandidos).

% # IdEnc: ID da encomenda a entregar.
% # Custo: Tempo do Circuito Inteiro.
resolveGulosaTempo(IdEnc,Caminho/Custo) :-
    encomenda(IdEnc,_,_,_,Nodo),
    estimaTempoEnc(IdEnc,Nodo,Estima),
```

```
    velocidadeEncomenda(IdEnc,V),
    agulosa_tempo(V,[[Nodo]/0/Estima],CaminhoIda/CustoIda/_),
    inverso(CaminhoIda,CaminhoAux),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    append(CaminhoIda,CaminhoVolta,Caminho),
    Custo is CustoIda*2.

agulosa_tempo(_,Caminhos,Caminho) :-
    obter_melhor(Caminhos,Caminho),
    Caminho = [Nodo|_]/_/_,
    goal(Nodo).
agulosa_tempo(V,Caminhos,Solucao) :-
    obter_melhor(Caminhos,MelhorCaminho),
    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
    expande_gulosa_tempo(V,MelhorCaminho,Expandidos),
    append(OutrosCaminhos,Expandidos,NovosCaminhos),
    agulosa_tempo(V,NovosCaminhos,Solucao).

expande_gulosa_tempo(V,Caminho,Expandidos) :-
    findall(NovoCaminho,adjacenteV3(V,Caminho,NovoCaminho),Expandidos).
```

5.2 Pesquisa A Estrela (*A* Search*)

Por último, o algoritmo A* evita expandir caminhos mais caros e, como tal, combina a formalidade do algoritmo de Dijkstra com a pesquisa em largura (BFS), minimizando a soma do caminho já efetuado com o mínimo previsto que falta até à solução. Consequentemente, tem um comportamento ótimo e completo já que devolve sempre a solução de menor custo. À semelhança da gulosa, o A* também devolve o circuito efetuado (desde a *Green Distribuition* ao ponto de entrega e de regresso ao local de partida) e a distância percorrida. A metodologia de procura A* aplicada para determinar o tempo de uma entrega segue um processo idêntico, a única diferença é que recebe o ID da encomenda e não a freguesia onde esta será entregue.

```
% # Caminho: Green Distribuition -> Ponto de Entrega -> Green Distribuition
% # Custo: Custo do Circuito Inteiro.
resolveAEstrela(Nodo,Caminho/Custo) :-
    estima(Nodo,Estima),
    aestrela([[Nodo]/0/Estima],CaminhoIda/CustoIda/_),
    inverso(CaminhoIda,CaminhoAux),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    append(CaminhoIda,CaminhoVolta,Caminho),
    Custo is CustoIda*2.

aestrela(Caminhos,Caminho) :-
```

```
    obter_melhor(Caminhos,Caminho),
    Caminho = [Nodo|_]/_/_,
    goal(Nodo).

aestrela(Caminhos,SolucaoCaminho) :-
    obter_melhor(Caminhos,MelhorCaminho),
    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
    expande_aestrela(MelhorCaminho,ExpCaminhos),
    append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
    aestrela(NovoCaminhos,SolucaoCaminho).

expande_aestrela(Caminho,ExpCaminhos) :-
    findall(NovoCaminho,adjacenteV2(Caminho,NovoCaminho),ExpCaminhos).

% # IdEnc: ID da encomenda a entregar.
% # Custo: Tempo do Circuito Inteiro.
resolveAEstrelaTempo(IdEnc,Caminho/Custo) :-
    encomenda(IdEnc,_,_,_,Nodo),
    estimaTempoEnc(IdEnc,Nodo,Estima),
    velocidadeEncomenda(IdEnc,V),
    aestrela_tempo(V,[[Nodo]/0/Estima],CaminhoIda/CustoIda/_),
    inverso(CaminhoIda,CaminhoAux),
    apagaCabeca(CaminhoAux,CaminhoVolta),
    append(CaminhoIda,CaminhoVolta,Caminho),
    Custo is CustoIda*2.

aestrela_tempo(_,Caminhos,Caminho) :-
    obter_melhor(Caminhos,Caminho),
    Caminho = [Nodo|_]/_/_,
    goal(Nodo).
aestrela_tempo(V,Caminhos,SolucaoCaminho) :-
    obter_melhor(Caminhos,MelhorCaminho),
    seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
    expande_aestrela_tempo(V,MelhorCaminho,ExpCaminhos),
    append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
    aestrela_tempo(V,NovoCaminhos,SolucaoCaminho).

expande_aestrela_tempo(V,Caminho,ExpCaminhos) :-
    findall(NovoCaminho,adjacenteV3(V,Caminho,NovoCaminho),ExpCaminhos).
```

6 Funcionalidades

6.1 Funcionalidade 1

"Gerar os circuitos de entrega, caso existam, que cubram um determinado território."

O predicado ***todosOsCaminhosTerritorio*** permite, dado um território, calcular todos os circuitos que por lá passam.

Primeiramente, são obtidos todos os pontos de entrega possíveis contidos no nosso grafo. De seguida, através do predicado ***todosOsCaminhosAux***, para cada ponto de entrega, são calculados todos os circuitos entre a *Green Distribuition* e o ponto que passem no território em questão. No final, são, então, concatenados todos estes caminhos e obtida a solução final.

```
todosOsCaminhosTerritorio(Territorio,L) :-
    solucoes(Ponto,pontoEntrega(Ponto),Pts),
    todosOsCaminhosAux(Territorio,Pts,L).
```

6.2 Funcionalidade 2

"Identificar quais os circuitos com maior número de entregas (por volume e peso)."

O predicado ***circuitoMaiorNumEntregasPorPeso*** apresenta os circuitos com mais encomendas entregues com um determinado peso e o predicado ***circuitoMaiorNumEntregasPorVolume*** com um determinado volume.

Através do predicado ***allCaminhos*** são obtidos todos os caminhos existentes no nosso grafo, através do algoritmo DFS, com início na *Green Distribuition*. Posto isto, são então invocados, nos respetivos predicados, o ***maiorNumEntregasPorPeso*** e o ***maiorNumEntregasPorVolume***, que permitem obter o valor máximo de entregas feitas num circuito conforme o parâmetro. Assim sendo, são chamados, respetivamente, os predicados ***circuitoMaiorNumEntregasPorPesoAux*** e ***circuitoMaiorNumEntregasPorVolumeAux***, com os quais é possível descobrir o(s) caminho(s), cujo número de entregas é igual ao calculado anteriormente. Por fim, através do predicado ***geraCircuitos***, os caminhos obtidos são transformados em circuitos.

```
circuitoMaiorNumEntregasPorPeso(Peso,C,T) :-
    allCaminhos(R),
    maiorNumEntregasPorPeso(Peso,R,T),
    circuitoMaiorNumEntregasPorPesoAux(Peso,T,R,C1),
    geraCircuitos(C1,C).
```

```
circuitoMaiorNumEntregasPorVolume(Vol,C,T) :-
    allCaminhos(R),
    maiorNumEntregasPorVolume(Vol,R,T),
    circuitoMaiorNumEntregasPorVolumeAux(Vol,T,R,C1),
    geraCircuitos(C1,C).
```

6.3 Funcionalidade 3

"Comparar circuitos de entrega tendo em conta os indicadores de produtividade (distância e tempo)."

O predicado ***produtividade*** devolve a distância percorrida e o tempo despendido para entregar uma encomenda com um determinado algoritmo de pesquisa.

Para o efeito, recorre ao predicado ***estrategiaProcura***, para calcular a distância percorrida, determinando primeiro a freguesia onde a encomenda será entregue, e ao predicado ***estrategiaProcuraTempo***, para calcular o tempo despendido.

```
% # 1 - DFS
% # 2 - BFS
% # 3 - Limitada em Profundidade
% # 4 - Gulosa
% # 5 - A*

produtividade(IdEnc,Distancia,Tempo,1) :-
    encomenda(IdEnc,_,_,_,Freg),
    estrategiaProcura(Nodo,Caminho,Distancia,1),
    estrategiaProcuraTempo(IdEnc,Caminho,Tempo,1).

produtividade(IdEnc,Distancia,Tempo,2) :-
    encomenda(IdEnc,_,_,_,Freg),
    estrategiaProcura(Freg,Caminho,Distancia,2),
    estrategiaProcuraTempo(IdEnc,Caminho,Tempo,2).

produtividade(IdEnc,Distancia,Tempo,3) :-
    encomenda(IdEnc,_,_,_,Freg),
    estrategiaProcura(Freg,Caminho,Distancia,3),
    estrategiaProcuraTempo(IdEnc,Caminho,Tempo,3).

produtividade(IdEnc,Distancia,Tempo,4) :-
    encomenda(IdEnc,_,_,_,Freg),
    estrategiaProcura(Freg,Caminho,Distancia,4),
    estrategiaProcuraTempo(IdEnc,Caminho,Tempo,4).

produtividade(IdEnc,Distancia,Tempo,5) :-
```

```
    encomenda(IdEnc,_,_,_,Freg),
    estrategiaProcura(Freg,Caminho,Distancia,5),
    estrategiaProcuraTempo(IdEnc,Caminho,Tempo,5).
```

6.4 Funcionalidade 4

"Escolher o circuito mais rápido (usando o critério da distância)."

O ***circuitoMaisRapido*** permite, segundo o algoritmo, escolher o circuito mais curto.

Este invoca o ***circuitoMaisRapidoAux*** que, utilizando como argumentos a freguesia da encomenda e um algoritmo, calcula todos os caminhos e retorna apenas o mais rápido.

```
circuitoMaisRapido(IdEnc,Alg,C,D) :-
    encomenda(IdEnc,_,_,_,Freg),
    circuitoMaisRapidoAux(Freg,C,D,Alg).
```

6.5 Funcionalidade 5

"Escolher o circuito mais ecológico (usando critério de tempo)."

O predicado ***circuitoMaisEficiente*** permite, consoante o algoritmo, escolher o circuito que leva menos tempo.

É apenas invocado o predicado ***circuitoMaisEficienteAux***, que, tal como na funcionalidade anterior, calcula todos os caminhos utilizando o algoritmo escolhido e, retorna o mais ecológico.

```
circuitoMaisEficiente(IdEnc,Alg,C,T) :-
    circuitoMaisEficienteAux(IdEnc,C,T,Alg).
```

6.6 Funcionalidade Extra

"Identificar quais os circuitos com maior número de entregas."

O predicado ***circuitosMaiorNumEntregas*** apresenta os circuitos com mais encomendas entregues.

Através do predicado ***allCaminhos*** são obtidos todos os caminhos existentes no nosso grafo, através do algoritmo DFS, com início na Green Distribuition. Posto isto, é, então, invocado o

maiorNumEntregasCircuito, que permite obter o valor máximo de entregas feitas num circuito. Assim sendo, é chamado o ***circuitosMaiorNumEntregasAux***, com o qual é possível descobrir o(s) caminho(s), cujo número de entregas é igual ao calculado anteriormente. Por fim, através do predicado ***geraCircuitos***, os caminhos obtidos são transformados em circuitos.

```
circuitosMaiorNumEntregas(MaxE,L) :-
    allCaminhos(R),
    maiorNumEntregasCircuito(R,MaxE),
    circuitosMaiorNumEntregasAux(R,MaxE,L1),
    geraCircuitos(L1,L).
```

7 Predicados Auxiliares

O ficheiro *auxiliares2.pl* contém os predicados necessários ao desenvolvimento das funcionalidades principais desta parte do projeto. Estes são, então, apresentados de seguida.

7.1 Escolha do Transporte

O predicado ***meioDeTransporteUsado*** permite, com base no peso total a ser transportado numa viagem, indicar qual o meio de transporte mais adequado.

```
meioDeTransporteUsado(PesoTotal,Transporte) :-
    ((PesoTotal =< 5 -> Transporte = 'Bicicleta');
     (PesoTotal > 5, PesoTotal =< 20 -> Transporte = 'Mota');
     (PesoTotal > 20, PesoTotal =< 100 -> Transporte = 'Carro')).
```

7.2 Velocidade do Transporte

O predicado ***velocidadeEntrega*** permite calcular, com base no transporte utilizado e no peso total a ser transportado, qual a velocidade a que uma entrega deverá ser feita.

```
velocidadeEntrega(Transporte,Peso,Velocidade) :-
    ((Transporte == 'Bicicleta' -> Velocidade is 10 - Peso * 0.7);
     (Transporte == 'Mota' -> Velocidade is 35 - Peso * 0.5);
     (Transporte == 'Carro' -> Velocidade is 25 - Peso * 0.1)).
```

7.3 Tempo de Entrega de uma Encomenda

O predicado ***tempoEntregaEncomenda*** permite, através do predicado ***velocidadeEncomenda***, decidir o transporte a ser utilizado para entregar uma encomenda, bem como a velocidade a que deve ser feito e com isto, calcular quanto tempo irá demorar.

```
tempoEntregaEncomenda(IdEnc,D,T) :-
    velocidadeEncomenda(IdEnc,Velocidade),
    T is D/Velocidade.

velocidadeEncomenda(IdEnc,Velocidade) :-
    encomenda(IdEnc,_,Peso,_,_),
    meioDeTransporteUsado(Peso,Transporte),
    velocidadeEntrega(Transporte,Peso,Velocidade).
```

7.4 Número de Entregas num Circuito

O predicado ***numEntregasCircuito*** permite saber o número total de entregas feitas num dado circuito, somando o número de encomendas entregues em cada freguesia do mesmo.

```
numEntregasCircuito([],0,0).
numEntregasCircuito([Freg],PesoTotal,N) :-
    findall((IdEnc,Peso),encomenda(IdEnc,_,Peso,_,Freg),L),
    somaPesos(L,PesoTotal),
    comprimento(L,N).
numEntregasCircuito([Freg|T],PesoTotal,N) :-
    findall((IdEnc,Peso),encomenda(IdEnc,_,Peso,_,Freg),L),
    somaPesos(L,PesoTotal1),
    comprimento(L,N1),
    numEntregasCircuito(T,PesoTotal2,N2),
    PesoTotal is PesoTotal1 + PesoTotal2,
    N is N1 + N2.
```

8 Main

O predicado ***main*** é responsável por interagir com o utilizador, criando um *loop*, em que, primeiramente, são dadas três opções ao usuário: uma para aceder às funcionalidades da 1ª fase, outra para aceder às da 2ª fase e, ainda, a opção para sair do *loop*. Entretanto, dependendo da escolha do utilizador, ser-lhe-á apresentado o menu específico para cada uma das fases.

```
-------------MENU-------------
1. Funcionalidades - Fase 1
2. Funcionalidades - Fase 2
0. Sair
------------------------------

Escolha um:
|: 2.


------------------------------------------------------------FASE2-------------------------------------------------------------
1. Gerar os circuitos de entrega, caso existam, que cubram um determinado território.
2. Representar os diversos pontos de entrega (freguesias) disponíveis em forma de grafo.
3. Identificar quais os circuitos com maior número de entregas (por volume e peso).
4. Comparar circuitos de entrega tendo em conta os indicadores de produtividade.
5. Escolher o circuito mais rápido para entregar uma dada encomenda, utilizado um algoritmo de pesquisa.
6. Escolher o circuito mais ecológico para entregar uma dada encomenda, utilizado um algoritmo de pesquisa.
7. Identificar quais os circuitos com maior número de entregas.
0. Sair
-------------------------------------------------------------------------------------------------------------------------------

Escolha um:
|:
```

Figura 2: Menu Principal e SubMenu da Segunda Fase

Para cada uma das funcionalidades, os dados a fornecer, para que estas possam serem executadas, são diferentes, bem como o formato em que os *outputs* são imprimidos. Abaixo, encontra-se ilustrada a apresentação do resultado de executar cada uma das funcionalidades:

A execução da **funcionalidade 1** exige a introdução de um território, introdução essa feita do seguinte modo: 'Lamas', por exemplo. Posteriormente, serão apresentados todos os circuitos de entrega que cubrem esse território.

```
Escolha um:
|: 1.

Indique o território a consultar (entre aspas):
|: 'Gualtar'.

[[Green Distribuition,Maximinos,Dume,São Vicente,São Victor,Gualtar,São Victor,São Vicente,Dume,Maximinos,Green Distribuition],
[Green Distribuition,Maximinos,São Vicente,São Victor,Gualtar,São Victor,São Vicente,Maximinos,Green Distribuition]]
```

Figura 3: Output da Funcionalidade 1

Após ser executada a **funcionalidade 2**, o utilizador poderá visualizar a representação das diversas freguesias em forma de grafo.

```
Escolha um:
|: 2.

Representação dos pontos de entrega em forma de grafo:

grafo([Green Distribuition,Tadim,Esporões,Maximinos,Dume,São Vicente,São Victor,Gualtar,Figueiredo,Lamas,Morreira,Palmeira,Tebosa
,Priscos,Ruilhe,Cabreiros],[aresta(Green Distribuition,Priscos,4.9),aresta(Green Distribuition,Maximinos,4.5),aresta(Green Distri
buition,Figueiredo,2.7),aresta(Priscos,Tebosa,2.5),aresta(Priscos,Ruilhe,2.6),aresta(Priscos,Tadim,3.2),aresta(Tebosa,Ruilhe,2.2)
,aresta(Ruilhe,Tadim,2.1),aresta(Tadim,Cabreiros,6.4),aresta(Figueiredo,Lamas,1.3),aresta(Figueiredo,Esporões,3.4),aresta(Lamas,E
sporões,2.8),aresta(Lamas,Morreira,2.9),aresta(Esporões,Morreira,2.8),aresta(Maximinos,Dume,3.7),aresta(Maximinos,São Vicente,3.9
),aresta(Dume,São Vicente,4.3),aresta(Dume,Palmeira,4.8),aresta(São Vicente,São Victor,2.15),aresta(São Victor,Gualtar,3.8)])
```

Figura 4: Output da Funcionalidade 2

Para executar a **funcionalidade 3**, o utilizador tem de optar entre um dos critérios: peso ou volume. Assim que efetuar esta escolha, ser-lhe-á pedido a inserção de um valor. Mediante este valor, será mostrado os circuitos com maior número de entregas tendo em conta o critério selecionado.

```
Escolha um:
|: 3.

Peso(0) ou Volume(1)?:
|: 0.

Insira o peso/volume:
|: 50.

Número de entregas: 2
Circuito(s) com maior número de entregas:

[[Green Distribuition,Maximinos,Dume,Palmeira,Dume,Maximinos,Green Distribuition],[Green Distribuition,Maximinos,São Vicente,Dume
,Palmeira,Dume,São Vicente,Maximinos,Green Distribuition]]
```

Figura 5: Output da Funcionalidade 3

Tal como explicitado anteriormente, a execução da **funcionalidade 4** exige a introdução do ID da encomenda. Posteriormente, serão apresentados os resultados referentes à distância percorrida e ao tempo despendido, por cada algoritmo de pesquisa, para entregar a dita encomenda.

```
Escolha um:
|: 4.

Indique o ID da encomenda a entregar:
|: 300145999366.

Profundidade (DFS):
*** Distância: 21.1 km
*** Tempo: 1.623076923076923 horas

Largura (BFS):
*** Distância: 35.5 km
*** Tempo: 2.730769230769231 horas

Limitada em Profundidade:
*** Distância: 21.1 km
*** Tempo: 1.623076923076923 horas

Gulosa (Greedy):
*** Distância: 21.1 km
*** Tempo: 3.246153846153846 horas

A Estrela (A*):
*** Distância: 21.1 km
*** Tempo: 3.246153846153846 horas
```

Figura 6: Output da Funcionalidade 4

Para executar a **funcionalidade 5**, é necessário introduzir o ID da encomenda. A seguir, o usuário deverá optar pelo algoritmo de pesquisa com o qual deseja obter o caminho mais rápido para entregar aquela encomenda. A execução da **funcionalidade 6** tem um apresentação idêntica à da funcionalidade 5, com a única diferença que será determinado o circuito mais eficiente.

```
Escolha um:
|: 5.

Indique o ID da encomenda:
|: 300145999366.

---------Algoritmos de Pesquisa---------
1. Profundidade (DFS)
2. Largura (BFS)
3. Limitada em Profundidade
4. Gulosa (Greedy)
5. A Estrela (A*)
----------------------------------------

Escolha um:
|: 4.

Caminho mais rápido: [Green Distribuition,Maximinos,São Vicente,São Victor,São Vicente,Maximinos,Green Distribuition]
*** Distância: 21.1 km
```

Figura 7: Output da Funcionalidade 5

```
Escolha um:
|: 6.

Indique o ID da encomenda:
|: 300145999366.

---------Algoritmos de Pesquisa---------
1. Profundidade (DFS)
2. Largura (BFS)
3. Limitada em Profundidade
4. Gulosa (Greedy)
5. A Estrela (A*)
----------------------------------------

Escolha um:
|: 5.

Caminho mais eficiente: [Green Distribuition,Maximinos,São Vicente,São Victor,São Vicente,Maximinos,Green Distribuition]
*** Tempo: 3.246153846153846 horas
```

Figura 8: Output da Funcionalidade 6

9 Análise dos Resultados

Tal como é observável na Figura 9, para o ponto de entregas `Lamas`, o algoritmo `A*` é aquele que, juntamente com o algoritmo `gulosa`, apresenta melhores resultados em todos os parâmetros em estudo: tempo de execução, espaço, indicador/custo e se encontrou a melhor solução.

Estratégia	Tempo de Execução (segundos)	Espaço	Indicador/Custo	Encontrou a melhor solução?
DFS	0.000	15936	17.799999999999997	Não
BFS	0.009	17568	22.0	Não
Limitada em Profundidade	0.000	9440	17.799999999999997	Não
Gulosa	0.000	4560	8.0	Sim
A*	0.000	4560	8.0	Sim

Figura 9: Tabela dos resultados para o ponto de entrega *Lamas*

No entanto, o algoritmo `gulosa` nem sempre devolve a solução de menor custo e, como tal, pode não ter um comportamento ótimo, ao contrário do `A*`. Já o algoritmo de pesquisa em largura, `BFS`, é o que mostra os piores resultados em todos os critérios anteriormente enunciados.

Esta análise comparativa entre as diferentes estratégias de procura só foi possível através do predicado `obterEstatisticas/1`, que recebe o tipo de algoritmo a avaliar.

```
?- obterEstatisticas(1).
% 276 inferences, 0.000 CPU in 0.000 seconds (95% CPU, 2816327 Lips)
Memória usada: 15936
Custo: 17.799999999999997
true .

?- obterEstatisticas(2).
% 518 inferences, 0.000 CPU in 0.000 seconds (92% CPU, 1925651 Lips)
Memória usada: 17568
Custo: 22.0
true .

?- obterEstatisticas(3).
% 174 inferences, 0.000 CPU in 0.000 seconds (79% CPU, 2718750 Lips)
Memória usada: 9440
Custo: 17.799999999999997
true .

?- obterEstatisticas(4).
% 186 inferences, 0.000 CPU in 0.000 seconds (93% CPU, 1309859 Lips)
Memória usada: 4560
Custo: 8.0
true .

?- obterEstatisticas(5).
% 186 inferences, 0.000 CPU in 0.000 seconds (92% CPU, 1500000 Lips)
Memória usada: 4560
Custo: 8.0
true .
```

Figura 10: Estatísticas dos algoritmos de pesquisa para o ponto de entrega *Lamas*

10 Conclusão

A realização desta segunda fase do trabalho prático permitiu consilidar e pôr em prática os conceitos abordados nas aulas, relativamente ao desenvolvimento de estratégias de procura informada e não informada. Isto permitiu analisar comparativamente os algoritmos de pesquisa implementados em vários parâmetros (nomeadamente: tempo de execução, utilização de memória e se encontra a melhor solução), chegando à conclusão que estes têm diferentes benefícios consoante os critérios especificados para o circuito a gerar.

As maiores dificuldades do grupo centraram-se no cálculo do tempo de cada entrega (na identificação do caminho mais ecológico), uma vez que este implica ter em atenção vários fatores, como a velocidade e o caminho retornado pelo algoritmo de pesquisa adotado. Todavia, o grupo considera tê-las ultrapassado com sucesso, apresentando uma boa resolução para o problema. O único requesito que não foi possível cumprir foi a questão de um estafeta poder efetuar mais do que uma entrega (levar mais de um encomenda) em cada viagem (circuito).

Posto isto, sentimos que a realização deste trabalho prático foi concluída com sucesso já que foram implementadas todas as funcionalidades pedidas e, ainda, uma adicional. Como trabalho futuro, gostaríamos de implementar mais algumas funcionalidades extra e, talvez, explorar outros tipos de algoritmos.

11 Referências Bibliográficas

- Stuart Russell e Peter Norvig
  Artificial Intelligence - A Modern Approach, 4ª edição.

- Cesar Analide, Paulo Novais e José Neves
  Sugestões para a Elaboração de Relatórios, novembro de 2001.