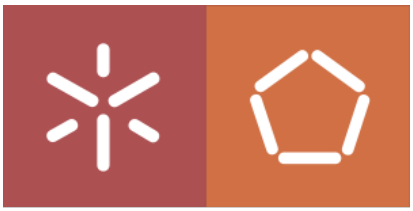

Universidade do Minho
Escola de Engenharia

Universidade do Minho

Licenciatura em Engenharia Informática

Inteligência Artificial - Relatório Fase 2

Grupo 30

Pedro Aquino Martins de Araújo (A90614)
Ricardo Miguel Santos Gomes (A93785)
Rui Pedro Gomes Coelho (A58898)
Vasco Baptista Moreno (A94194)

Janeiro, 2022

Capítulo 1

Resumo

O presente relatório pretende apoiar e descrever o trabalho desenvolvido na Fase 2 do trabalho proposto na Unidade Curricular de Inteligência Artificial. Seguindo o trabalho previamente desenvolvido no decurso da Fase 1, o presente trabalho tem como objetivo atualizar o sistema desenvolvido, integrando funcionalidades que permitam recomendar circuitos de entrega para as encomendas.

Assim como na fase anterior, a solução desenvolvida foi criada com recurso à programação em lógica PROLOG, integrando uma base de conhecimento e um conjunto de predicados para modelar e atualizar o sistema em conformidade com os requisitos propostos.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 2

Introdução

O trabalho aqui apresentado procura dar continuidade ao exercício resolvido na primeira fase. Na Fase 1 foi pedida a modelação de um sistema de representação de conhecimento e raciocínio capaz de retratar o universo de discurso na área da logística de distribuição de encomendas. Agora, partindo da modelação desse universo, procura estender-se a solução fornecida, integrando novas funcionalidades que permitam determinar quais os circuitos de entregas mais adequados.

Assim sendo, o presente relatório procura dar suporte à solução desenhada para a determinação dos circuitos a ser utilizados pela *Green Distribution*. Tal como na fase anterior, a programação em lógica PROLOG foi a ferramenta selecionada para o efeito, uma vez que permitiu, por um lado, retirar partido dos mecanismos de inferência fornecidos pelo PROLOG e, por outro lado, sedimentar e aprofundar os conhecimentos previamente adquiridos.

Capítulo 3

Preliminares

O desenvolvimento da solução proposta, tal como foi previamente referido, recorre à programação em lógica PROLOG. Como tal, a representação do conhecimento foi efetuada com recurso a Lógica e a sua manipulação é elaborada através de mecanismos de Inferência. Como tal, a compreensão destas duas componentes é essencial para o acompanhamento do trabalho desenvolvido. Adicionalmente, a compreensão do mecanismo de inferência em PROLOG é imprescindível, pelo que a leitura de documentação é recomendada.

Capítulo 4

Descrição do Trabalho e Análise de Resultados

A presente seção do relatório pretender dar a conhecer a estratégia adotada para desenvolver o sistema de recomendação de circuitos para o sistema desenvolvido na primeira fase de entrega. Assim sendo, e considerando o sistema já desenvolvido, foram efetuadas algumas alterações na base de conhecimento, de modo a acomodar as funcionalidades a implementar.

Tal como na fase anterior, a primeira parte consiste no desenvolvimento da Base de Conhecimento, onde são dados a conhecer os predicados que integram definem o conhecimento preexistente acerca do universo em estudo – cf. Anexo A.1 para consultar o predicado *entrega* que define as encomendas presentes na base de conhecimento. Seguidamente, foi desenvolvida uma componente responsável pelo sistema de inferência, que implementa as funcionalidades propostas. Por fim, os predicados auxiliares foram inseridos numa componente isolada. Esta segregação permite, em primeira parte, uma leitura e compreensão mais acessível das funcionalidades disponibilizadas. Adicionalmente, pretende que permita algum tipo de reutilização dos predicados, tentando que estes sejam o a mais genéricos possível.

Posto isto, de modo a concretizar o sistema em estudo, foi simulada uma rede de entregas para a *Green Distribution*. Para tal, foi construído um grafo, Figura 4.1, onde os diversos nodos representam as freguesias nas quais a empresa de distribuição efetua entrega de encomendas. Entre estas freguesias encontram-se estabelecidas ligações, que correspondem às arestas do grafo, representativas dos trajetos possíveis a serem tomados para percorrer o grafo. A cada uma dessas arestas está associado um valor, que representa a distância percorrida por um estafeta ao optar por essa ligação entre dois nós – distância concetualizada em quilómetros. Estas ligações são bidirecionais, significando que uma aresta definida entre dois nodos não impõe o deslocamento numa direção específica no grafo. Adicionalmente, no grafo encontra-se ainda presente um nó correspondente à localização da central de distribuição da *Green Distribution*.

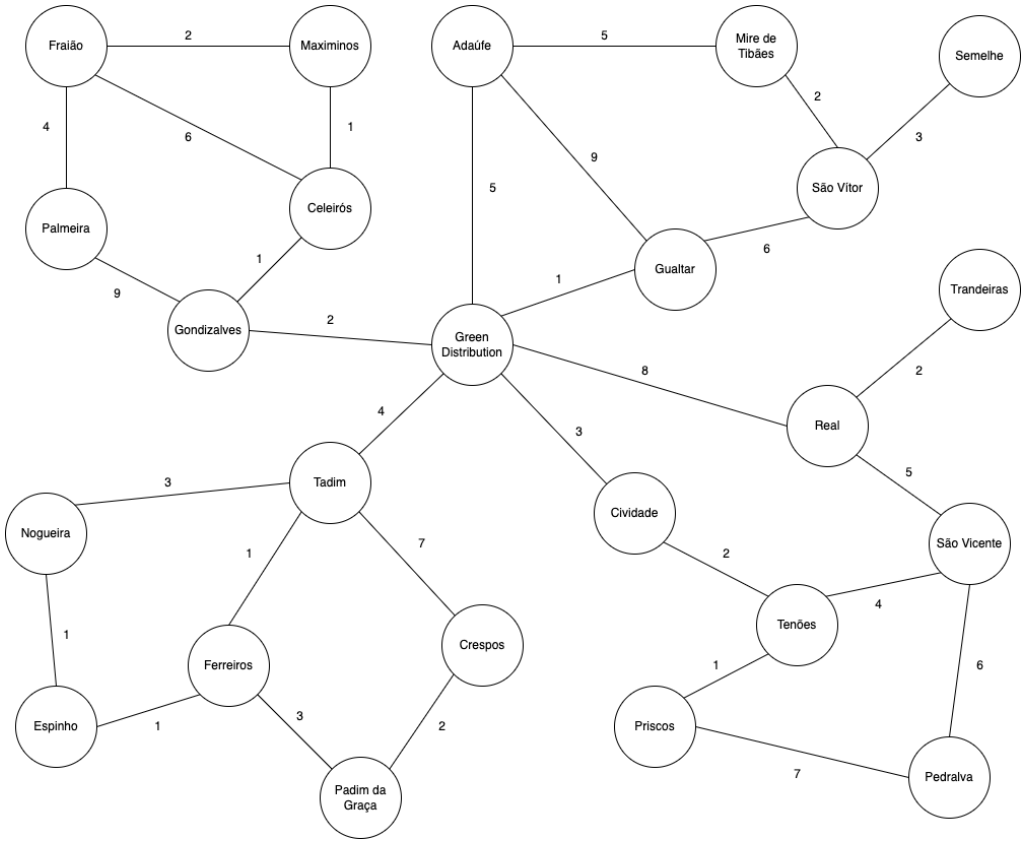

Figura 4.1: Grafo da rede de distribuição da *Green Distribution*.

O nodo da *Green Distribution* corresponde ao ponto de partida e de chegada de todos os circuitos admissíveis para os estafetas, i.e., qualquer trajeto definido para a entrega de uma encomenda inicia-se pela saída do centro de distribuições até à morada de destino, terminando, apenas, aquando o regresso do estafeta à central da *Green Distribution*.

Tal como foi concetualizado na fase anterior, as entregas podem ser efetuadas com recurso aos seguintes meios de transporte:

- Bicicleta: sendo considerado o meio de transporte mais ecológico, que viaja a uma velocidade média de 10 km/h, podendo transportar no máximo encomendas até 5 Kg;
- Mota: viajam a uma velocidade média de 35 km/h, transportando encomendas com um limite máximo de 20 Kg;
- Carro: sendo tido como o meio de transporte menos ecológico, viaja a uma velocidade média de 25 km/h, transportando um peso máximo de 100 Kg.

Visando determinar a comparação e avaliação dos diversos circuitos possíveis foram selecionadas duas métricas: tempo de entrega e distância percorrida. Com estas medidas é possível comparar os diversos circuitos possíveis, facilitando a escolha dos que melhor se adequam para efetuar a entrega atempada das encomendas, sempre de um modo eficiente. Assim sendo, as métricas foram definidas do seguinte modo:

- Tempo de entrega: corresponde ao tempo total despendido em viajam desde que o estafeta sai da central de distribuição, até voltar ao centro da *Green Distribution* após ter entregue a encomenda;
- Distância percorrida: corresponde à distância total percorrida desde que o estafeta sai do centro, até regressar, após a entrega da encomenda.

No que diz respeito ao tempo de entrega foram estipuladas um conjunto de restrições a considerar no decurso da entrega. Assim sendo, cada veículo possui uma penalização na sua velocidade média, associada ao peso da encomenda que se encontra em distribuição:

- A *bicicleta* sofre um decréscimo de 0.7 km/h por cada Kg que transporta;
- A *mota* sofre um decréscimo de 0.5 km/h por cada Kg que transporta;
- O *carro* sofre um decréscimo de 0.1 km/h por cada Kg que transporta;

O grafo apresentado na Figura 4.1 permite efetuar diversas pesquisas, não informadas, relativamente aos circuitos de entrega. Contudo, foram, posteriormente, consideradas heurísticas para a determinação destes circuitos. Particularmente, a pesquisa informada toma como heurística as estimativas presentes nos triângulos verdes da Figura 4.2, que correspondem à distância em linha reta, descrita em quilómetros, entre um nodo e a central de distribuição, i.e., *Green Distribution*.

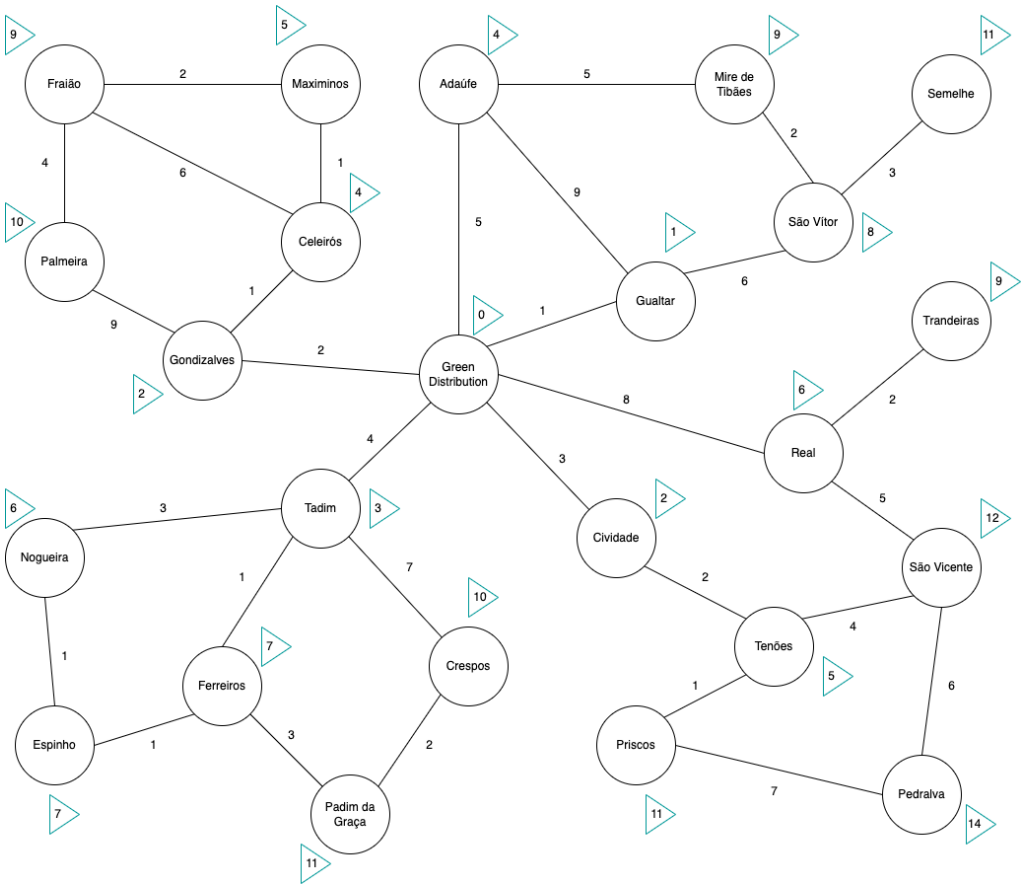

Figura 4.2: Grafo da rede de distribuição da *Green Distribution*.

4.1 Base de Conhecimento

Tomando como ponto de partido a base de conhecimento proposta para a primeira fase de entrega, foi gerada uma nova base de conhecimento. A primeira alteração na Base de Conhecimento passou por incluir a representação do grafo da rede de distribuição presente na Figura 4.1, com recurso ao predicado *move* – cf. Anexo A.2. Adicionamente, foi inserido o predicado *adjacente* para que, de um modo simples, fosse possível verificar se dois nós da rede se encontram ligados – notemos que este predicado permite estabelecer a bidirecionalidade das arestas, sem haver necessidade de a descrever exaustivamente na base de conhecimento.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% adjacente: Nodo, ProximoNodo, Custo -> {V,F}
adjacente(Nodo, ProxNodo, C) :- move(Nodo, ProxNodo, C).
adjacente(Nodo, ProxNodo, C) :- move(ProxNodo, Nodo, C).
```

Adicionalmente, para efetuar pesquisas no grafo de acordo com as estimativas realizadas, foi incorporado o predicado *estima*, que define a distância entre um dado nó até ao nó que corresponde à *Green Distribution* – cf. Anexo A.3.

4.2 Funcionalidades do sistema

De acordo com os enunciados apresentados, a modelação do sistema de distribuição da *Green Distribution* deve suportar um conjunto de funcionalidades que permitam operar sobre o mesmo. Numa primeira fase, o sistema desenvolvido permite obter um vasto conjunto de informações acerca da distribuição de encomendas, como, por exemplo, a determinação da zona de entrega com maior volume de encomendas. Nesta fase, o objetivo centra-se em determinar circuitos de entrega, respeitando um conjunto de regras. Assim sendo, foram desenvolvidos vários predicados para o efeito, recorrendo a estratégias de pesquisa informada e não informada para o efeito[1].

O primeiro predicado desenvolvido permite determinar os circuitos de entrega, caso existam, que cubram um determinado território, i.e., enunciar qual o trajeto realizado por um estafeta, iniciando-o na *Green Distribution*, até à freguesia onde é efetuada a entrega, voltando no fim à central de distribuição *Green Distribution*. Para tal, foi criado o predicado *geraCircuitosObjetivo*, que determina todos os circuitos disponíveis no grafo.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado geraCircuitosObjetivo: Nodo, Caminhos -> {V,F}
geraCircuitosObjetivo(NodoObjetivo, Caminhos) :-
        findall(Caminho,geraCircuito(NodoObjetivo, Caminho, C), Caminhos).
```

O predicado desenvolvido recorre ao predicado auxiliar *geraCircuito*, que permite determinar um circuito para o nó objetivo, através da pesquisa em profundidade – cf. Anexo A.4.

Seguidamente, foram incorporadas as funcionalidades que permitem identificar o top de N circuitos que entregaram um maior peso total de encomendas e o top N circuitos que entregaram o maior número de encomendas. Para tal foram criados os predicados *identificarCircuitosPeso*

[1]Para consultar os algoritmos de pesquisa informada e não informada usados nos predicados descritos ao longo do relatório consultar Anexo A.5

e *identificarCircuitosNEntregas*. Ambos os predicados seguem a mesma lógica de construção: inicialmente são calculados todos os circuitos da rede, uma vez colecionado, também, a variável de interesse, i.e., peso ou volume de encomendas respetivamente, são ordenados os diversos circuitos, por ordem decrescente, sendo filtrados posteriormente os N primeiros circuitos.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado identificarCircuitosPeso: TopN, Circuitos -> {V,F}
identificarCircuitosPeso(N, Circuitos) :-
        geraCircuitosObjetivo(NodoObjetivo, Caminhos),
        calcularPesosCircuitosTodos(Caminhos, [H|T]),
        ordenarCaminhos(T, [H], CircuitosOrd),
        escolheN(CircuitosOrd, N, [], Circuitos).

% Extensão do predicado identificarCircuitosNEntregas: TopN, Circuitos -> {V,F}
identificarCircuitosNEntregas(N, Circuitos) :-
        geraCircuitosObjetivo(NodoObjetivo, Caminhos),
        calcularNEntregasCircuitosTodos(Caminhos, [H|T]),
        ordenarCaminhos(T, [H], CircuitosOrd),
        escolheN(CircuitosOrd, N, [], Circuitos).
```

Os predicados *compara_circuitos* e *compara_circuitos_estafeta* permitem, através dos indicadores de produtividade, comparar os circuitos de entrega. Assim sendo, são analisados : os tempos de entrega verificando se há veículos que cumprem com o prazo, e a distância percorrida seguindo as regras previamente mencionadas.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado compara_circuitos: Nodo, Caminhos -> {V,F}
compara_circuitos(NodoObjetivo,Circuitos) :-
        geraCircuitosComCustos(NodoObjetivo,Circuitos1),
        compara_circuitos_aux2(NodoObjetivo,Circuitos1,[],Circuitos).

% Extensão do predicado compara_circuitos_estafeta: Nodo, Estafeta, Caminhos -> {V,F}
compara_circuitos_estafeta(NodoObjetivo,Estafeta,Circuitos) :-
        geraCircuitosComCustos(NodoObjetivo,Circuitos1),
        compara_circuitos_aux1(NodoObjetivo,Estafeta,Circuitos1,[],Circuitos).
```

Para a construção da funcionalidade, foram desenvolvidos predicados auxiliares que permitem, findo o cálculo, obter uma lista com as informações relevantes dos circuitos gerados para um dado nó – cf. Anexo A.4. Dentro dos diversos predicados criados, podem ser salientados os predicados *calcula_tempo* e *desconto_velocidade*, que sãos usado para atualizar a velocidade dos veículos aquando das entregas, em função do peso das encomendas.

```
%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% calcula_tempo: Veiculo, Distancia, Peso, Tempo -> {V,F}
calcula_tempo(Veiculo,Distancia,PesoEnc, Tempo) :-
    velMed(Veiculo,Vel),
    desconto_velocidade(Veiculo,Vel,PesoEnc, VelDesconto),
    Tempo is (Distancia / VelDesconto) + (Distancia / Vel).
```

```
%-------------------------------- - - - - - - - - - - - - - - - - - -
% desconto_velocidade: Veiculo, Velocidade, Peso, NovaVelocidade -> {V,F}
desconto_velocidade(bicicleta, Vel, Peso, NewVel) :-
    NewVel is Vel - (0.7*Peso).
desconto_velocidade(mota, Vel, Peso, NewVel) :-
    NewVel is Vel - (0.5*Peso).
desconto_velocidade(carro, Vel, Peso, NewVel) :-
    NewVel is Vel - (0.1*Peso).
```

Adicionalmente, o sistema é ainda capaz de determinar os melhores circuitos, em função de critérios diferentes: determina o circuito mais rápido e determina o circuito mais ecológico. Para tal foram desenvolvidos um conjunto de predicados que, através de pesquisa informada, determinam os melhores circuitos.

Assim sendo, o predicado *top_faster_circuits* recorre ao critério da distância percorrida para encontrar o caminho mais rápido a ser percorrido, tendo como base as encomendas presentes na base de conhecimento. Para tal, o predicado começa por determinar os circuitos possíveis para as entregas definidas. Após a remoção de eventuais repetições, os circuitos são ordenados em função da sua distância.

```
%-------------------------------- - - - - - - - - - - - - - - - - - -
% Extensão do predicado top_faster_circuits: Caminhos -> {V,F}
top_faster_circuits(Circuits) :-
        faster_circuits(Cs),
        retiraDestinosRepetidos(Cs,[],NewCs),
        sortDistances(NewCs,Circuits).

%-------------------------------- - - - - - - - - - - - - - - - - - -
% faster_circuits: Circuitos -> {V,F}
faster_circuits(Circuitos) :-
    findall(
        (Estafeta,NodoObjetivo,Distancia,Caminho),
        (entrega(Estafeta, _ ,_, _, _ , NodoObjetivo, _ , _ , _ ),
        resolve_aestrela(NodoObjetivo,Caminho/Distancia)),Circuitos).
```

No que diz respeito ao circuito mais ecológico de cada entrega, o predicado *most_ecologic_circuit* recorre ao critério de tempo e à ordem de veículos considerados mais ecológicos para encontrar a solução. Para obter os N circuitos mais ecológicos, recorre ao predicado *topN_most_eco*.

```
%-------------------------------- - - - - - - - - - - - - - - - - - -
% Extensão do predicado topN_most_eco: Caminhos, TopN -> {V,F}
topN_most_eco(Circuits, N) :-
        most_ecologic_circuit(Cs),
        get5eco(Cs,[],Circuits,N,1).
```

4.3 Teste das funcionalidades

A Figura 4.3 demonstra um exemplo de testes efetuados para averiguar o correto funcionamento das diversas funcionalidades providenciadas pelo sistema.

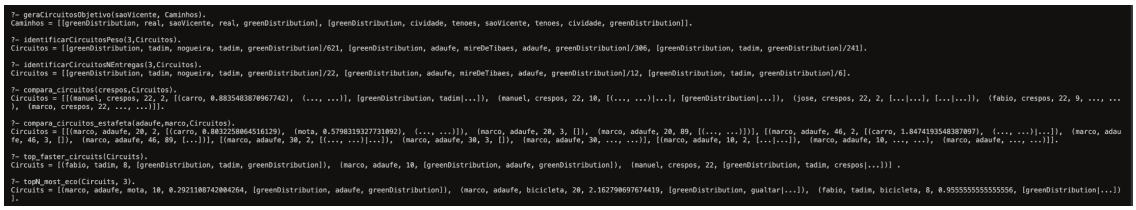

Figura 4.3: Teste das funcionalidades com recurso ao SWI-Prolog

4.4 Medidas de performance

Para medir o tempo de execução e a memória usada pelos predicados, foi desenvolvido o predicado *measure_performance* que permite medir e apresentar o desempenho de um dado predicado. O exemplo de código concreto apresentado permite obter as métricas para a estratégia de procura em profundidade, cujo nó objetivo corresponde à freguesia de Padim da Graça[2].

```
%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% Cálculo de tempo de execução e memória utilizada por um predicado
measure_performance() :-
        statistics(runtime, [TimeSinceStart | [TimeSinceLastCall]]),
        statistics(global_stack,[M1,L1]),
        profundidade(padimDaGraca,Caminho,Custo), % predicado a ser testado
        statistics(runtime, [NewTimeSinceStart | [ExecutionTime]]),
        statistics(global_stack,[M2,L2]),
        write('Execution took '), write(ExecutionTime), write(' ms.'), nl,
        write('Used memory '), write(L2), write(' Kb.'), nl.
```

A Tabela 4.1 demonstra as medidas registadas para avaliar a performance das diversas estratégias de procura utilizadas. O valor de custo apresentado corresponde à distância total percorrida para a solução encontrada com o algoritmo de pesquisa.

Tabela 4.1: Medidas de performance para as diversas estratégias de procura usadas.

Estratégia	Tempo (ms)	Espaço (Kb)	Custo	Solução Ótima
Profundidade	0	13568	18	Não
Profundidade com limite	0	28904	18	Não
Largura	1	56288	18	Não
Gulosa	0	11992	18	Não
A*	1	9904	12	Sim

Como seria expectável, a pesquisa em profundidade, com ou sem limite, necessita de menor quantidade de memória comparativamente à pesquisa em largura. Tal deve-se pelo modo como os nós da árvore de procura são expandidos e pelo facto de haver poda da árvore aquando a pesquisa em profundidade. De igual modo, a pesquisa informada com o algoritmo A* encontra a solução ótima, o que, para o teste efetuado, corresponde a um circuito com 12 unidades de distância.

[2]Para a realização de todos os testes elaborados para a medição de tempo e memória foi considerado como nodo objetivo "padimDaGraca". Para a procura em profundidade com limite, tomou-se o limite máximo como sendo de 3 níveis.

No que diz respeito ao tempo medido, são observadas poucas diferenças. Os dados registados são reflexo da base de conhecimento utilizada, pelo que o aumento de factos na base, pode levar a diferenças mais significativas quer a nível da memória, quer ao nível do tempo registados.

Capítulo 5

Conclusões e Sugestões

Ao longo do presente relatório foram descritos os diversos passos adotados pelo grupo para desenvolver um sistema de representação de conhecimento e raciocínio, capaz de retratar o universo de discurso na área da logística de distribuição de encomendas, com particular foco na determinação de circuitos de distribuição das mesmas. Para tal, a Base de Conhecimento preparada na primeira fase de entrega foi adaptada, incorporando, agora, um grafo representativo das zonas de entregas da *Green Distribution*.

Adicionalmente, foram introduzidas estimativas para explorar diversas técnicas de pesquisa, informada e não informada, no grafo da rede de distribuição. Posto isto, foram desenvolvidas um conjunto de funcionalidades e predicados, descritos ao longo do relatório, que permitem operar o grafo, determinando circuitos possíveis de entrega.

O trabalho aqui apresentado corresponde aos requisitos pedidos, disponibilizando uma série de funcionalidades que trabalha sobre a Base de Conhecimento criada. O presente sistema encontra-se apto a ser estendido a casos mais complexos, através da inclusão de predicados adicionais, com outras técnicas de pesquisa.

Capítulo 6

Bibliografia

[1] ANALIDE, Cesar, NOVAIS, Paulo
"Sugestões para a Redacção de Relatórios Técnicos"
Relatório Técnico, Departamento de Informática, Universidade do Minho, Portugal, 2011

[2] BRATKO, Ivan
"PROLOG: Programming for Artificial Intelligence"
United States, Pearson Education (US), 2000

[3] COSTA, Ernesto, SIMOES, Anabela
"Inteligência Artificial-Fundamentos e Aplicações"
FCA, ISBN: 978-972-722-34 2008

[4] RUSSELL, Stuart, NORVIG, Peter
"Artificial Intelligence - A Modern Approach, 3rd edition"
ISBN-13: 9780136042594, 2009

Apêndice A

Grafo

A.1 Encomendas

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado entrega: estafeta, tipoEncomenda, pesoEnc, prazo,
%       cliente, freguesia, classificacao, dataEncomenda, dataEntrega -> {V, F}
entrega(manuel, comida, 2, 1, maria, 'crespos', 4, 11/10/2021/10, 11/10/2021/11).
entrega(jose, comida, 4, 1, ana, 'nogueira', 5, 11/10/2021/12, 11/10/2021/13).
entrega(fabio, roupa, 12, 24, filipa, 'tadim', 5, 12/10/2021/13, 12/10/2021/19).
entrega(marco, movel, 2, 48, cristina, 'adaufe', 3, 11/10/2021/17, 14/10/2021/10).
entrega(jose, comida, 4, 1, ana, 'nogueira', 5, 11/10/2021/20, 11/10/2021/21).
entrega(manuel, comida, 1, 1, ana, 'nogueira', 5, 11/10/2021/19, 11/10/2021/20).
entrega(fabio, comida, 2, 1, ana, 'nogueira', 3, 12/10/2021/21, 12/10/2021/22).
entrega(manuel, comida, 1, 1, ana, 'nogueira', 5, 11/10/2021/13, 11/10/2021/15).
entrega(manuel, roupa, 10, 18, maria, 'crespos', 3, 12/10/2021/11, 13/10/2021/10).
entrega(manuel, comida, 3, 0.5, cristina, 'adaufe', 4, 12/10/2021/14, 12/10/2021/15).
entrega(manuel, movel, 74, 30, filipa, 'tadim', 5, 13/10/2021/14, 16/10/2021/10).
entrega(manuel, comida, 3, 0.5, cristina, 'adaufe', 4, 15/10/2021/18, 15/10/2021/19).
entrega(jose, roupa, 17, 4, ana, 'nogueira', 2, 15/10/2021/18, 17/10/2021/10).
entrega(jose, comida, 2, 1, maria, 'crespos', 3, 14/10/2021/11, 14/10/2021/12).
entrega(jose, comida, 3, 0.5, filipa, 'tadim', 1, 12/10/2021/10, 12/10/2021/12).
entrega(jose, movel, 74, 30, filipa, 'tadim', 5, 13/10/2021/9, 16/10/2021/10).
entrega(jose, movel, 60, 5, ana, 'nogueira', 3, 14/10/2021/15, 15/10/2021/10).
entrega(fabio, movel, 45, 17, ana, 'nogueira', 2, 15/10/2021/15, 18/10/2021/10).
entrega(fabio, roupa, 9, 5, maria, 'crespos', 3, 12/10/2021/16, 14/10/2021/10).
entrega(fabio, comida, 4, 2, filipa, 'tadim', 1, 12/10/2021/21, 12/10/2021/22).
entrega(fabio, movel, 53, 25, cristina, 'adaufe', 2, 13/10/2021/4, 15/10/2021/10).
entrega(fabio, comida, 2, 1, ana, 'nogueira', 3, 12/10/2021/13, 12/10/2021/14).
entrega(marco, roupa, 3, 1, ana, 'nogueira', 2, 14/10/2021/9, 14/10/2021/18).
entrega(marco, comida, 4, 0.5, maria, 'crespos', 4, 12/10/2021/10, 12/10/2021/12).
entrega(marco, comida, 3, 0.5, cristina, 'adaufe', 5, 12/10/2021/11, 12/10/2021/12).
entrega(marco, movel, 74, 30, filipa, 'tadim', 5, 13/10/2021/11, 16/10/2021/10).
entrega(marco, movel, 89, 23, cristina, 'adaufe', 5, 15/10/2021/14, 19/10/2021/10).
```

A.2 Definição do grafo

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensão do predicado move: LocalidadeOigem, LocalidadeDestino,
%       CustoDistancia, CustoTempo -> {V,F}
move(fraiao,maximinos,2).
move(fraiao,celeiros,6).
move(fraiao,palmeira,4).
move(maximinos,celeiros,1).
move(celeiros,gondizalves,1).
move(palmeira,gondizalves,9).
move(gondizalves,greenDistribution,2).
move(adaufe,mireDeTibaes,5).
move(adaufe,gualtar,9).
move(adaufe,greenDistribution,5).
move(mireDeTibaes,saoVitor,2).
move(gualtar,saoVitor,6).
move(gualtar,greenDistribution,1).
move(saoVitor,semelhe,3).
move(trandeiras,real,2).
move(real,saoVicente,5).
move(real,greenDistribution,8).
move(saoVicente,pedralva,6).
move(saoVicente,tenoes,4).
move(pedralva,priscos,7).
move(tenoes,cividade,2).
move(tenoes,priscos,1).
move(cividade,greenDistribution,3).
move(padimDaGraca,crespos,2).
move(padimDaGraca,ferreiros,3).
move(crespos,tadim,7).
move(ferreiros,tadim,5).
move(ferreiros,espinho,1).
move(tadim,nogueira,3).
move(tadim,greenDistribution,4).
move(espinho,nogueira,1).
```

A.3 Definição das estimativas

```
%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensão do predicado estima: Localidade, EstimaDistancia -> {V,F}
estima(fraiao, 9).
estima(maximinos, 5).
estima(palmeira, 10).
estima(celeiros, 4).
estima(gondizalves, 2).
estima(adaufe, 4).
estima(mireDeTibaes, 9).
estima(saoVitor, 8).
estima(gualtar, 1).
estima(semelhe, 11).
```

```
estima(trandeiras, 9).
estima(real, 6).
estima(saoVicente, 12).
estima(pedralva, 14).
estima(priscos, 11).
estima(tenoes, 5).
estima(cividade, 2).
estima(crespos, 10).
estima(padimDaGraca, 11).
estima(ferreiros, 7).
estima(espinho, 7).
estima(nogueira, 6).
estima(tadim, 3).
estima(greenDistribution, 0).
```

A.4 Predicados auxiliares

```
%-------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado geraCircuito: Nodo, Caminho, Custo -> {V,F}
% Gera um circuito para um determinado local
geraCircuito(NodoObjetivo, Caminho, C) :-
    inicial(Inicio),
    profundidadeprimeiroInicial(NodoObjetivo, Inicio, [Inicio], [H|T], C2),
    inverso([H|T], [NodoRetirar|Resto]),
    append([H|T], Resto, Caminho),
    C is C2 * 2.

%-------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado geraCircuitosComCustos: Nodo, Circuito -> {V,F}
% Gera todos os circuito para um determinado local, com o respetivo custo
geraCircuitosComCustos(NodoObjetivo, Caminhos) :-
    findall((Caminho,C),geraCircuito(NodoObjetivo, Caminho, C), Caminhos).

%-------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado calcularPesosCircuitosTodos: Lista1, Lista2 -> {V,F}
calcularPesosCircuitosTodos([], []).
calcularPesosCircuitosTodos([Circuito1|T], [CircuitoPeso|OutrosCircuitosPeso]) :-
    calculaPesoCircuito(Circuito1, CircuitoPeso),
    calcularPesosCircuitosTodos(T, OutrosCircuitosPeso).

%-------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado calculaPesoCircuito: Lista1, Lista2 -> {V,F}
% Retira a greenDistribution das contas
calculaPesoCircuito([H|T], [H|T]/PesoTotal) :-
    calculaPesoCircuitoAuxiliar(T, PesoTotal).

%-------------------------------- - - - - - - - - - - - - - - - - -
% Extensão do predicado geraCircuitosComCustos: Caminho, Peso -> {V,F}
% Calcula os pesos associados às freguesias do circuito
calculaPesoCircuitoAuxiliar([], 0).
calculaPesoCircuitoAuxiliar([Freguesia|T], PesoTotal) :-
```

```
    findall(Peso, entrega(_, _, Peso, _, _, Freguesia, _, _, _), ListaPesos),
    somatorio(ListaPesos, Peso1),
    calculaPesoCircuitoAuxiliar(T, Peso2),
    PesoTotal is Peso1 + Peso2.

%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado faster_circuits: Circuitos -> {V,F}
% Calcula o circuito mais rápido de cada entrega
faster_circuits(Circuitos) :-
    findall(
        (Estafeta,NodoObjetivo,Distancia,Caminho),
        (entrega(Estafeta, _ ,_, _, _ , NodoObjetivo, _ , _ , _ ),
        resolve_aestrela(NodoObjetivo,Caminho/Distancia)),Circuitos).

%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado most_ecologic_circuit: Circuitos -> {V,F}
% Determina o circuito mais ecológico de cada entrega
most_ecologic_circuit(Circuitos) :-
    findall(
        (Estafeta,NodoObjetivo,Veiculo,Distancia,Tempo,Caminho),
        (entrega(Estafeta,_, Peso, Prazo, _ , NodoObjetivo, _ , _ , _ ),
        geraCircuitosComCustos(NodoObjetivo,Lista),
        geraVeiculos(Lista,Peso,Prazo,[],Veiculos),
        getMostEco(Veiculos,(Tempo,Distancia,Caminho,Veiculo))),
        Circuitos).

%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado get5eco: Lista1, Custo1, Lista2, Numero, Index -> {V,F}
get5eco(_,C,C,N,_) :-
    N =< 0,!.
get5eco(Lista,CAux,Circuits,Nmax,I) :-
    Nmax > 0,
    veiculoIndice(V,I),
    member((_,_,V,_,_,_),Lista),
    getListaVeiculo(V,Lista,[],ListaV),
    expandList(CAux,Nmax,ListaV, NewL,NewN),
    NewI is I + 1,
    get5eco(Lista,NewL,Circuits,NewN,NewI),
    !.

%-------------------------------- - - - - - - - - - -  -  -  -  -  -
% Extensão do predicado getListaVeiculo: Veículo, Lista1, Lista2, Lista3 -> {V,F}
getListaVeiculo(_,[],L,L) :- !.
getListaVeiculo(V,[(Estafeta,NodoObjetivo,Veiculo,Distancia,Tempo,Caminho)|R],L,L1) :-
    V == Veiculo,
    getListaVeiculo(V,R,[(Estafeta,NodoObjetivo,Veiculo,Distancia,Tempo,Caminho)|L],L1),
    !.
getListaVeiculo(V,[(Estafeta,NodoObjetivo,Veiculo,Distancia,Tempo,Caminho)|R],L,L1) :-
    getListaVeiculo(V,R,L,L1).
```

A.5 Algoritmos de procura

```
%------------------------------------------------------------------------
%---------------                    Profundidade
%------------------------------------------------------------------------

%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado profundidade: Nodo, Caminho, Custo -> {V,F}
% Determina os caminhos disponíveis com os respetivos custos através
% de pesquisa em profundidade
profundidade(NodoObjetivo, CaminhoTodo, C) :-
    inicial(Inicio),
    profundidadeprimeiroInicial(NodoObjetivo, Inicio, [Inicio], Caminho, C2), !,
    duplicaCaminho(Caminho, CaminhoTodo),
        C is C2 * 2.

profundidadeprimeiroInicial(NodoObjetivo, Inicio,_, [Inicio, NodoObjetivo], C) :-
    adjacente(Inicio, NodoObjetivo, C).
profundidadeprimeiroInicial(NodoObjetivo, Inicio, Historico, [Inicio, ProxNodo|Caminho], C) :-
    adjacente(Inicio, ProxNodo, C1),
    nao(membro(ProxNodo, Historico)),
    profundidadeprimeiro(NodoObjetivo, ProxNodo, [ProxNodo|Historico], Caminho, C2),
        C is C1 + C2.

profundidadeprimeiro(NodoObjetivo, NodoAtual, _ , [NodoObjetivo], C1) :-
    adjacente(NodoAtual, NodoObjetivo, C1), !.
profundidadeprimeiro(NodoObjetivo, NodoAtual, Historico, [ProxNodo|Caminho], C) :-
    adjacente(NodoAtual, ProxNodo, C1),
    nao(membro(ProxNodo, Historico)),
    profundidadeprimeiro(NodoObjetivo, ProxNodo, [ProxNodo|Historico], Caminho, C2),
    C is C1 + C2.

% Extensão do predicado melhorProfundidade: Nodo, Caminho, Custo -> {V,F}
% Determina o melhor caminho através de pesquisa em profundidade
melhorProfundidade(NodoObjetivo, Caminho, Custo) :-
    findall(
        (Caminhos, Custos),
        profundidade(NodoObjetivo, Caminhos, Custos), L),
    minimo(L, [],(Caminho, Custo)), !.

% Extensão do predicado duplicaCaminho: Caminho1, Caminho2 -> {V,F}
% Duplica os nodos do caminho
duplicaCaminho(Caminho, CaminhoTodo) :-
    duplicaCaminhoAuxiliar(Caminho, [], CaminhoInverso),
    append(Caminho, CaminhoInverso, CaminhoTodo).

% Extensão do predicado duplicaCaminhoAuxiliar: Lista1, Lista2, Caminho -> {V,F}
% Calcula o caminho inverso, sem o nodo objetivo
duplicaCaminhoAuxiliar([], [_|T], T).
duplicaCaminhoAuxiliar([H|T], Aux, CaminhoInverso) :-
    duplicaCaminhoAuxiliar(T, [H|Aux], CaminhoInverso).
```

```
%-------------------------------------------------------------------------
%---------------          Profundidade com limite
%-------------------------------------------------------------------------

%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensão do predicado profundidadeLimite: Nodo, Limite, Caminho, Custo -> {V,F}
% Determina os caminhos disponíveis com os respetivos custos através
% de pesquisa em profundidade com limite
profundidadeLimite(NodoObjetivo, Limite, CaminhoTodo, C) :-
    inicial(Inicio),
    profundidadeprimeiroLimiteInicial(NodoObjetivo, Limite, Inicio, [Inicio], Caminho, C2), !,
    duplicaCaminho(Caminho, CaminhoTodo),
    C is C2 * 2.

profundidadeprimeiroLimiteInicial(NodoObjetivo,_, Inicio,_, [Inicio, NodoObjetivo], C) :-
    adjacente(Inicio, NodoObjetivo, C).
profundidadeprimeiroLimiteInicial(NodoObjetivo, Limite, Inicio,
    Historico, [Inicio, ProxNodo|Caminho], C) :-
        adjacente(Inicio, ProxNodo, C1),
        nao(membro(ProxNodo, Historico)),
        length([ProxNodo|Historico], Tam),
        Tam - 1 < Limite,
        profundidadeprimeiroLimite(NodoObjetivo, Limite, ProxNodo, [ProxNodo|Historico],
            Caminho, C2),
        C is C1 + C2.

profundidadeprimeiroLimite(NodoObjetivo, _, NodoAtual, _, [NodoObjetivo], C1) :-
    adjacente(NodoAtual, NodoObjetivo, C1), !.
profundidadeprimeiroLimite(NodoObjetivo, Limite, NodoAtual, Historico,
    [ProxNodo|Caminho], C) :-
        adjacente(NodoAtual, ProxNodo, C1),
        nao(membro(ProxNodo, Historico)),
        length([ProxNodo|Historico], Tam),
        Tam - 1 < Limite,
        profundidadeprimeiroLimite(NodoObjetivo, Limite, ProxNodo, [ProxNodo|Historico],
            Caminho, C2),
        C is C1 + C2.

% Extensão do predicado melhorProfundidadeLimite: Nodo, Limite, Caminho, Custo -> {V,F}
% Determina o melhor caminho através de pesquisa em profundidade com limite
melhorProfundidadeLimite(NodoObjetivo, Limite, Caminho, Custo) :-
    findall((SS, CC), profundidadeLimite(NodoObjetivo, Limite, SS, CC), L),
    minimo(L, [], (Caminho, Custo)), !.

%-------------------------------------------------------------------------
%---------------                  Largura
%-------------------------------------------------------------------------

%------------------------------ - - - - - - - - - - - - - - - - - -
% Extensão do predicado largura: Nodo1, Nodo2, Custo -> {V,F}
% Determina os caminhos disponíveis com os respetivos custos através de pesquisa em largura
largura(Dest, Caminho, Custos) :-
    inicial(Orig),
```

```
    largura2(Dest,[[Orig/0]],Cam),
    !,
    somaCustos(Cam, Custos2),
    retiraCustos(Cam, [NodoRetirar|Resto]),
    inverso([NodoRetirar|Resto], CaminhoIncompleto),
    append(CaminhoIncompleto, Resto, Caminho),
    Custos is Custos2 * 2.

% Extensão do predicado largura2: Nodo, Lista1, Lista2 -> {V,F}
largura2(Dest,[[Dest/Cus|T]|_],[Dest/Cus|T]).
largura2(Dest,[LA|Outros],Cam) :-
    LA=[Act|_],
    Act = Atual/Cus,
    findall(
        [X/C|LA],
        (Dest\==Atual,adjacente(Atual,X, C),
        \+membroCustos(X,LA)),Novos),
    append(Outros,Novos,Todos),
    largura2(Dest,Todos,Cam).

% Extensão do predicado somaCustos: Lista, Custo -> {V,F}
somaCustos([], 0).
somaCustos([_/Custo|Resto], Custos) :-
    somaCustos(Resto, Custos2),
    Custos is Custo + Custos2.

% Extensão do predicado somaCustos: Lista1, Lista2 -> {V,F}
retiraCustos([], []).
retiraCustos([Localidade/_|Resto], [Localidade|Outras]) :- retiraCustos(Resto, Outras).

% Extensão do predicado somaCustos: Lista1, Lista2 -> {V,F}
membroCustos(Local, [Local/Cus|Resto]).
membroCustos(Local, [Nodo/Cus|Resto]) :-
    membroCustos(Local, Resto).

%-------------------------------------------------------------------------
%---------------                  Gulosa
%-------------------------------------------------------------------------

%------------------------------ - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado resolve_gulosa: Nodo, Lista -> {V,F}
% Determina os caminhos disponíveis com os respetivos custos através de pesquisa gulosa
resolve_gulosa(Nodo, Caminho/Custo) :-
        estima(Nodo, Estima),
        agulosa([[Nodo]/0/Estima], [GD|T]/Custo2/_), !,
        inverso([GD|T], [NodoRetirar|CaminhoInverso]),
        append([GD|T], CaminhoInverso, Caminho), %junta sem o NodoObjetivo duplicado
        Custo is Custo2 * 2.

% Extensão do predicado agulosa: Lista1, Lista2 -> {V,F}
agulosa(Caminhos, Caminho) :-
        obtem_melhor_g(Caminhos, Caminho),
        Caminho = [Nodo|_]/_/_,
        objetivo(Nodo).
```

```
agulosa(Caminhos, SolucaoCaminho) :-
        obtem_melhor_g(Caminhos, MelhorCaminho),
        seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
        expande_gulosa(MelhorCaminho, ExpCaminhos),
        append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    agulosa(NovoCaminhos, SolucaoCaminho).

% Extensão do predicado obtem_melhor_g: Lista1, Lista2 -> {V,F}
obtem_melhor_g([Caminho], Caminho) :- !.
obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho) :-
        Est1 =< Est2, !,
        obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
obtem_melhor_g([_|Caminhos], MelhorCaminho) :-
        obtem_melhor_g(Caminhos, MelhorCaminho).

% Extensão do predicado expande_gulosa: Lista1, Lista2 -> {V,F}
expande_gulosa(Caminho, ExpCaminhos) :-
        findall(NovoCaminho, adjacente2(Caminho,NovoCaminho), ExpCaminhos).

%-------------------------------------------------------------------------
%---------------                    A*
%-------------------------------------------------------------------------

%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% Extensão do predicado resolve_aestrela: Nodo, Lista -> {V,F}
% Determina os caminhos disponíveis com os respetivos custos através de pesquisa A*
resolve_aestrela(Nodo, Caminho/Custo) :-
        estima(Nodo, Estima),
        aestrela([[Nodo]/0/Estima], [GD|T]/Custo2/_), !,
        inverso([GD|T], [NodoRetirar|CaminhoInverso]),
        append([GD|T], CaminhoInverso, Caminho),
        Custo is Custo2 * 2.

% Extensão do predicado aestrela: Lista1, Lista2 -> {V,F}
aestrela(Caminhos, Caminho) :-
        obtem_melhor(Caminhos, Caminho),
        Caminho = [Nodo|_]/_/_,
        objetivo(Nodo).
aestrela(Caminhos, SolucaoCaminho) :-
        obtem_melhor(Caminhos, MelhorCaminho),
        seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
        expande_aestrela(MelhorCaminho, ExpCaminhos),
        append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela(NovoCaminhos, SolucaoCaminho).

% Extensão do predicado obtem_melhor: Lista1, Lista2 -> {V,F}
obtem_melhor([Caminho], Caminho) :- !.
obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho) :-
        Custo1 + Est1 =< Custo2 + Est2, !,
        obtem_melhor([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
obtem_melhor([_|Caminhos], MelhorCaminho) :-
        obtem_melhor(Caminhos, MelhorCaminho).
```

```
% Extensão do predicado expande_aestrela: Lista1, Lista2 -> {V,F}
expande_aestrela(Caminho, ExpCaminhos) :-
        findall(NovoCaminho, adjacente2(Caminho,NovoCaminho), ExpCaminhos).
```