Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Informática

Inteligência Artificial

Green Distribution
Logística de Distribuição de Encomendas

Grupo 25:
João Machado, A89510
Carolina Vila Chã, A89495
David Duarte, A93253
Daniel Faria, A81997

Novembro 2021

Resumo

Neste relatório encontra-se detalhada a resolução do sistema desenvolvido para a primeira fase do projeto.

O objetivo principal do projeto *Green Distribution* é desenvolver um sistema de representação de conhecimento e raciocínio com capacidade para caraterizar um universo de discurso na área da logística de distribuição de encomendas.

Conteúdo

1 Introdução

Vivemos em tempos onde a ecologia é um tópico cada vez mais relevante no nosso dia-a-dia. A empresa *Green Distribution* é um serviço de entrega de encomendas e tem como objetivo principal privilegiar sempre o meio de entrega mais ecológico. Assim, foi desenvolvido um sistema de representação de conhecimento e raciocínio que demonstra as funcionalidades subjacentes à utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

Neste sistema, existem várias entidades: veículo, estafeta, encomenda, freguesia, rua, cliente e entrega. A principal diferença entre encomenda e entrega, é que a encomenda é o pedido do cliente, enquanto que o estafeta faz a entrega desse pedido, ou seja, faz e entrega da encomenda.

O sistema deverá ter implementadas as seguintes funcionalidades básicas:

- Identificar o estafeta que utilizou mais vezes um meio de transporte mais ecológico
- Identificar que estafetas entregaram determinada(s) encomenda(s) a um determinado cliente
- Identificar os clientes servidos por um determinado estafeta
- Calcular o valor faturado pela *Green Distribution* num determinado dia
- Identificar quais as zonas (rua ou freguesia) com maior volume de entregas por parte da *Green Distribution*
- Calcular a classificação média de satisfação de cliente para um determinado estafeta
- Identificar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo
- Identificar o número total de entregas pelos estafetas, num determinado intervalo de tempo
- Calcular o número de encomendas entregues e não entregues pela *Green Distribution*, num determinado intervalo de tempo
- Calcular o peso total transportado por um estafeta num determinado dia

No entanto, o conhecimento a tratar pode ser estendido, pelo que se implementaram as seguintes funcionalidades extra:

-

2 Preliminares

A linguagem utilizada neste projeto é prolog. é uma linguagem de tipo declarativo, onde a lógica do programa é expressa em termos de relações, e onde o cálculo é feito à base da consulta dessas relações. É uma linguagem de uso geral que é especialmente associada com a inteligência artificial e linguística computacional.

Em prolog existe apenas um tipo de dados, o termo, cuja natureza depende de como foi declarado, podendo ser um de vários objetos:

- Átomos - Um átomo é uma sequência de letras, números e/ou *underscore*, iniciando com uma letra minúscula. Pode também conter espaços e/ou ser começado por uma letra maiúscula, mas para tal deverá estar entre aspas. Por exemplo, `"Green Distribution", cli1`.
- Números - Um número é uma sequência de dígitos, sendo também permitidos os sinais de `.` para números reais, `-` para números negativos e **e** para números em notação científica. Por exemplo, `123 , 3.14 , 5e10`.
- Variáveis - Essencialmente o mesmo que átomos, no entanto iniciadas por letra maiúscula (sem a inclusão de aspas como seria nos átomos). Por exemplo, `Estafeta , P`.

2.1 Termos Compostos

Um termo composto consiste em uma cabeça - que é necessariamente um átomo - e parâmetros listados entre parênteses e separados por vírgulas.

Existem dois tipos principais de termos compostos utilizados neste projeto:

- Listas - são definidas por construção recursiva, por exemplo, `[1,2,3,4]`.
- Strings - são sequências de caracteres entre aspas, por exemplo, `"String"`.

2.2 Factos

Os factos são sempre verdadeiros, e são compostos por uma cabeça e por argumentos. Por exemplo, `rua(rua1, freguesia1)`. Através deste facto sabemos que rua1 e freguesia1 estão relacionados pelo predicado rua. A interpretação, neste caso, é que rua1 é uma rua da freguesia1.

2.3 Regras

Regras podem também ser chamadas de cláusulas, e expressam relações entre factos. Por exemplo, `mae(A, B) :- filha(B, A)`. Esta regra expressa que o predicado mae(A,B) é apenas verdadeiro se filha(B,A) for verdadeiro.

2.4 Questões

As questões são expressões colocadas ao interpretador, que vai tentar encontrar algum predicado ou facto que satisfaça a questão.

Por exemplo, sabendo que `estafeta(est1)` é o único estafeta da base de conhecimento:

```
% Questão: Encontra X tal que X é estafeta
? - estafeta(X).
X = est1 .

% Questão: Existe est3 na lista de estafetas?
? - estafeta(est3).
false.
```

3 Descrição do Trabalho e Análise de Resultados

3.1 Construção da Base de Conhecimento

Para a base de conhecimento foram criadas várias entidades, por exemplo:

```
% veiculo(Veiculo, Carga, Velocidade, Preço)
veiculo(bicicleta, 5, 10, 5).

% estafeta(IdEstafeta, Nome)
estafeta(est1, 'Lomberto Felgado').

% encomenda(IdEncomenda, DataHora, Tempo de entrega, Peso, Volume,
↪  Preço, Rua, IdCliente)
encomenda(enc1, 1/1/2021/18/30, 2, 10, 5, 56, rua1, cli1).

% freguesia(IdFreguesia)
freguesia(f1).

% rua(IdRua, IdFreguesia)
rua(rua1, f1).

% cliente(IdCLiente, Nome, IdRua).
cliente(cli1, 'Filmina Ribano', rua1).

% entrega(IdEntrega, IdEncomenda, IdEstafeta, Classificação(0-5),
↪  Veiculo)
entrega(ent1, enc1, est1, 5, bicicleta).
```

3.2 Funcionalidades Necessárias

Esta secção dedica-se a explicar os predicados que permitem criar novas entidades (e.g. estafeta, cliente, encomenda, etc.), entre outros, para que sejam utilizados durante a execução do programa.

Uma vez que todos os predicados `criar_(entidade)` seguem estruturas similares, segue apenas um exemplo:

```
criar_entrega(EntId, EncId, EstId, Class, Veiculo) :-
    (entrega(EntId,_,_,_,_) -> write("Id da entrega já existe.");
        encomenda(EncId,_,_,_,_,_,_),
        estafeta(EstId,_),
        veiculo(Veiculo,_,_,_),
        Class =< 5, Class >= 0,
        assert(entrega(EncId, EncId, EstId, Class, Veiculo))).
```

Embora o predicado verifique se já existe um Id igual no sistema, esta verificação de unicidade de valores será feita mais eficazmente com a introdução de Invariantes na próxima fase.

3.3 Funcionalidades Base

3.3.1 Estafeta que utilizou mais vezes o meio de transporte mais ecológico

Para a concretização deste predicado foram utilizados três predicados auxiliares. São eles o **elemento_mais_frequente**, **elemento_mais_frequente_aux** e o **frequencia**.

A ideia geral da implementação deste predicado passa por criar uma lista de estafetas, em que cada uma dessas entradas representa uma entrega, pelo que é uma lista onde ocorrem valores repetidos de estafeta.

Primeiro, e com o objetivo de selecionar o veículo mais ecológico que possa carregar o peso da encomenda, o sistema tenta criar essa lista apenas para as entregas que foram feitas com o veículo mais ecológico. Caso não hajam entregas feitas com esse veículo, o algoritmo tenta o mesmo mas agora para o segundo veículo mais ecológico, e assim sucessivamente.

Uma vez criada essa lista resta agora devolver o elemento da lista que ocorre mais vezes nela devolvendo assim o estafeta que mais vezes utilizou um meio de transporte mais ecológico.

```
f1_aux(Elem,usainBolt) :-
    findall(Estafeta,
        entrega(_,_,Estafeta,_,usainBolt),Lista),
        elemento_mais_frequente(Lista,Elem).
f1_aux(Elem,bicicleta) :-
```

```
    findall(Estafeta,
    entrega(_,_,Estafeta,_,bicicleta),Lista),
    elemento_mais_frequente(Lista,Elem).
(...)
f1_aux(Elem,fogetao) :-
    findall(Estafeta,
    entrega(_,_,Estafeta,_,fogetao),Lista),
    elemento_mais_frequente(Lista,Elem).
```

3.3.2 Identificar que estafetas entregaram determinadas encomendas a um determinado cliente

Dado um cliente, este predicado devolve os pares **(Estafeta, Encomenda)** que correspondem às encomendas desse cliente.

```
f2_estafetasCliente(Cliente,R):-
    findall((Encomenda, Estafeta),
        (entrega(_, Encomenda, Estafeta,_,_),
        encomenda(Encomenda, _,_,_,_,_,_,_,Cliente)),
        R).
```

3.3.3 Identificar os clientes servidos por um determinado estafeta

Trata-se de um predicado simples em que dado um estafeta é calculada a lista de todos os clientes que serviu.

```
f3_clientesEstafeta(E,R):-
    findall(Cliente,
        (entrega(_, Encomenda, E,_,_),
        encomenda(Encomenda,_,_,_,_,_,_,_,Cliente)),
        L),
    sort(L,R).
```

3.3.4 Valor faturado pela *Green Distribution* num determinado dia

Este predicado recebe como argumento um dia, e com a ajuda da função **findall** calcula uma lista dos preços de todas as encomendas que já foram entregues num determinado dia.

```
f4_faturacaoDia(D/M/A,R):-
    findall(Preco,
        (entrega(_, Encomenda, Estafeta,_,_),
        encomenda(Encomenda, D/M/A/_/_,_,_,Preco,_,_,_)), Precos),
    sumlist(Precos,R).
```

3.3.5 Identificar as zonas com maior volume de entregas por parte da *Green Distribution*

Este predicado consiste em, dado um tipo de zona (rua ou freguesia), devolver uma lista ordenada das ruas ou freguesias da que tem mais volume de entregas para a que tem menos volume de entregas, fazendo uso do predicado **f5_aux** para o efeito.

```
f5_zonasMaiorVolume(rua,RuasN):-
        findall(Rua,
            (entrega(_,Encomenda,_,_,_),
            encomenda(Encomenda,_,_,_,_,_,_,Rua,_)),
            Ruas),
        f5_aux(Ruas,RuasN).


f5_zonasMaiorVolume(freguesia,FreguesiasN):-
        findall(Freguesia,
            (entrega(_,Encomenda,_,_,_),
            encomenda(Encomenda,_,_,_,_,_,_,Rua,_),
            rua(Rua,Freguesia)),
            Freguesias),
        f5_aux(Freguesias,FreguesiasN).
```

3.3.6 Cálculo da classificação média de satisfação de cliente para um dado estafeta

Utilizando a função **findall**, e fixando o estafeta passado como argumento, é criada a lista *Classes* que terá todas as avaliações do estafeta mencionado. De seguida basta efetuar o somatório desses valores e dividir pelo tamanho da lista obtendo assim o valor médio.

```
f6_classificacaoMedia(E,R) :-
    findall(Class, entrega(_,_,E,Class,_), Classes),
    length(Classes, Length),
    Length =\= 0,
    sumlist(Classes, Sum),
    R is div(Sum, Length).
```

3.3.7 Número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo

Para um dado veículo são selecionadas todas as entregas feitas com este. Essas entregas são filtradas por data com a ajuda do predicado auxiliar **datahora_intervalo**. Uma vez obtida a lista de todas as entregas realizadas com um veículo especificado é calculado o tamanho desta utilizando a função **length**.

```
f7_entregasVeiculoIntervalo(V,Ii,If,R) :-
    findall(Entrega,
    (entrega(Entrega, Encomenda,_,_,V),
        encomenda(Encomenda, I,_,_,_,_,_,_,_),
        datahora_intervalo(I, Ii, If)),
    Lista),
    length(Lista,R).
```

3.3.8 Número total de entregas pelos estafetas num determinado intervalo de tempo

Dadas duas datas, um início e um fim, é calculado o número total de entregas nesse intervalo de tempo. Para isso é calculada uma lista de todas as entregas filtradas por data, com a ajuda do predicado auxiliar **datahora_intervalo**, que dada uma data verifica se esta se encontra entre duas outras.

Uma vez criada a lista resta apenas calcular o seu tamanho e tem-se assim o número total de entregas pelos estafetas num determinado intervalo de tempo.

```
f8_entregasEstafetaIntervalo(Ii, If, R) :-
    findall(Entrega,
    (entrega(Entrega, Encomenda,_,_,_),
        encomenda(Encomenda, I,_,_,_,_,_,_,_),
```

```
5           datahora_intervalo(I, Ii, If)),
6       Lista),
7       length(Lista,R).
```

3.3.9 Número de encomendas entregues e não entregues pela Green Distribution, num determinado período de tempo

Para determinar o valor pedido o grupo implementou um predicado que calcula duas listas. Uma que terá as encomendas entregues num determinado intervalo e outra que será o total de encomendas entregues e não entregues. Desta forma basta apenas subtrair à **length** do total a **length** da lista das entregues e obtém-se assim as não entregues.

```
1  f9_encomendasEntreguesIntervalo(Ii, If) :-
2      findall(Entrega, (entrega(Entrega, Encomenda,_,_,_) ,
       ↪  encomenda(Encomenda, I,_,_,_,_,_,_,_), datahora_intervalo(I, Ii,
       ↪  If)), Entregas),
3      % Sabendo quantas foram entregues, então o resto não foi entregue,
       ↪  logo Total - Entregue = NEntregue
4      findall(EncID, (encomenda(EncID, I,_,_,_,_,_,_,_),
       ↪  datahora_intervalo(I, Ii, If)), EncomendasTotal),
5      length(Entregas, E),
6      length(EncomendasTotal, ET),
7      NE is ET - E,
8      write("Encomendas entregues: "), write(E), write("\nEncomendas não
       ↪  entregues: "), write(NE).
```

3.3.10 Peso total transportado por um estafeta num determinado dia

Este predicado consiste em criar uma lista dos pesos de todas as entregas de um estafeta, utilizando para isso a função **findall** juntamente com os predicados **entrega** e **encomenda** para obter as entregas e filtrar por dia.

Criada essa lista, resta agora efetuar o somatório de todos os seus valores obtendo assim o peso total transportado por um estafeta num determinado dia.

```
1  f10_pesoEstafetaDia(Estafeta,D/M/A,R) :-
2      findall(Peso,
3          (entrega(_, Encomenda, Estafeta,_,_),
```

```
4           encomenda(Encomenda, D/M/A/_/_,_,Peso,_,_,_)), Pesos),
5       sumlist(Pesos,R).
```

3.4 Funcionalidades Extra

3.4.1 Menu

Foi implementado um menu que permite ao utilizador ler sobre as funcionalidades do problema e inserir dados de modo mais fácil de utilizar.

```
?- menu.
1 - Estafeta que utilizou um meio de transporte mais ecológico mais
↪  vezes
2 - Estafetas que entregaram determinada(s) encomenda(s) a um
↪  determinado cliente
3 - Clientes servidos por determinado cliente
4 - Valor faturado pela Green Distribution num determinado dia
5 - Zonas com maior volume de entregas por parte da Green Distribution
6 - Classificação média de satisfação de cliente para um determinado
↪  estafeta
7 - Número total de entregas pelos diferentes meios de transporte num
↪  determinado intervalo de tempo
8 - Número total de entregas pelos estafetas num determinado intervalo
↪  de tempo
9 - Número de encomendas entregues e não entregues pela Green
↪  Distribution num determinado período de tempo
10 - Peso total transportado por estafeta num determinado dia
0 - Sair
```

Cada número executa a função respetiva. Por exemplo:

```
call_f10:-
    write('Estafeta: '),nl,
    read(Estafeta),
    write('Data no formato: D/M/A (separado por barra): '),nl,
    read(Data),
    f10_pesoEstafetaDia(Estafeta,Data,R),
    write('O peso total transportado por '), write(Estafeta),
```

```
    write(' é: '), write(R),nl,nl.
```

3.4.2 Mais veículos

Foram adicionados mais veículos, nomeadamente:

- Usain Bolt - O mais amigo do ambiente, apenas pode carregar 1kg, com uma velocidade média de 45km/h, com preço de 20 euros.
- Rolls Royce - Menos ecológico que o carro, pode carregar 150kg, com uma velocidade média de 120km/h, com preço de 80 euros.
- Jato - Pouco ecológico, carrega até 200kg, com uma velocidade média de 800km/h, e com um preço de 200 euros.
- Fogetão - Modelado a partir do foguetão real Saturn V, o fogetão é o veículo menos ecológico, no entanto é capaz de carregar 48600kg e atinge uma velocidade média de 24944km/h. A sua utilização custa 185 milhões de euros.

3.5 Predicados Auxiliares

3.5.1 datahora

Este predicado devolve a data atual no formato Dia/Mês/Ano/Hora/Minuto.

Este formato foi escolhido ao invés do tipo **date** nativo ao prolog, pois foi considerado ser mais simples de utilizar para este projeto, dado que **date** contém uma série de argumentos que seriam desnecessários para grande parte do trabalho.

```
1 datahora(D/M/A/H/Min) :-
2     get_time(Epoch),
3     stamp_date_time(Epoch, DateTime, local),
4     date_time_value(year, DateTime, A),
5     date_time_value(month, DateTime, M),
6     date_time_value(day, DateTime, D),
7     date_time_value(hour, DateTime, H),
8     date_time_value(minute, DateTime, Min).
```

3.5.2 datahoramenor

Este predicado determina se a primeira **datahora** é menor que a segunda. Para tal, calcula o número de segundos que ocorreram desde o *Epoch* (1 de janeiro de 1970) e compara esses valores.

```
datahoramenor(D1/M1/A1/H1/Mi1, D2/M2/A2/H2/Mi2) :-
    date_time_stamp(date(A1,M1,D1,H1,Mi1,0,0,-,-), Stamp1),
    date_time_stamp(date(A2,M2,D2,H2,Mi2,0,0,-,-), Stamp2),
    R1 is integer(Stamp1),
    R2 is integer(Stamp2),
    R1 =< R2.
```

3.5.3 datahora_intervalo

Verifica se uma **datahora** se encontra num dado intervalo de tempo.

```
datahora_intervalo(I, Ii, If) :-
    datahoramenor(Ii, I),
    datahoramenor(I, If).
```

3.5.4 preco

Determina o preço de uma encomenda.

```
preco(TLimite, Veiculo, P) :-
    veiculo(Veiculo,_,_,PrecoVeiculo),
    P is 5 + 48 - TLimite + PrecoVeiculo.
```

3.5.5 veiculo_encomenda

Este predicado calcula o veículo mais apropriado para a entrega, de acordo com o peso.

```
veiculo_encomenda(Peso, bicicleta) :- Peso =< 5.
veiculo_encomenda(Peso, mota) :- Peso =< 20.
veiculo_encomenda(Peso, carro) :- Peso =< 100.
veiculo_encomenda(_, _) :- write("Demasiado peso.").
```

3.5.6 determinarVeiculo

Determina veículo a usar para uma entrega baseando-se no peso da encomenda, que segue a seguinte fórmula: `5 + (48 - Tempo_de_Entrega) + Preço_do_Veículo`, onde 5 é o preço base de utilização do serviço.

```
preco(TLimite, Peso, P) :-
    veiculo_encomenda(Peso, Veiculo),
    veiculo(Veiculo,_,_,PrecoVeiculo),
    P is 5 + 48 - TLimite + PrecoVeiculo.
```

3.5.7 elemento_mais_frequente

Calcula o elemento mais frequente numa lista.

```
elemento_mais_frequente([],0).
elemento_mais_frequente(Lista, E) :-
    sort(Lista, [H|T]),
    elemento_mais_frequente_aux(Lista, [H|T], HN, Freq),
    E = HN.
```

3.5.8 elemento_mais_frequente_aux

Complementa o predicado **elemento_mais_frequente** verificando que o elemento devolvido é de facto o mais frequente em comparação com outros da lista.

```
elemento_mais_frequente_aux(Lista, [H], H, Frequencia) :-
    frequencia(H, Lista, Frequencia).
elemento_mais_frequente_aux(Lista, [H|T], H, Fx) :-
    frequencia(H, Lista, Fx),
    elemento_mais_frequente_aux(Lista, T, HCauda, Frequencia),
    Frequencia =< Fx.
elemento_mais_frequente_aux(Lista, [H|T], Elemento, Frequencia) :-
    frequencia(H, Lista, Fx),
    elemento_mais_frequente_aux(Lista, T, Elemento, Frequencia),
    Frequencia > Fx.
```

3.5.9 frequencia

Calcula a frequência de um elemento numa lista, complementando assim o predicado **elemento_mais_frequente_aux**.

```
frequencia(E, [], 0).
frequencia(E, [E|T], F) :- frequencia(E, T, F1), F is F1 + 1.
frequencia(E, [H|T], F) :- E \= H,
    frequencia(E, T, F).
```

3.5.10 f5_aux

Predicado auxiliar de **f5_zonasMaiorVolume**, que a partir de uma lista de ruas/freguesias vai criar uma nova lista, sem repetições, por ordem de frequência de elementos, do maior para o menor.

```
f5_aux([],[]).
f5_aux(ListaAll,[Elem|ListaFinal]):-
        elemento_mais_frequente(ListaAll,Elem),
        delete(ListaAll,Elem,ListaIterada),
        f5_aux(ListaIterada,ListaFinal).
```

4 Conclusões e Sugestões

Através da realização deste projeto, acreditamos ter criado um sistema de representação de conhecimento e raciocínio capaz de caraterizar um universo de discurso na área da logística de distribuição de encomendas.

O obstáculo encontrado mais significativo foi a utilização da linguagem prolog, uma vez que é uma linguagem de programação cuja estrutura e modo de execução é bastante diferente das outras linguagens até agora aprendidas. No entanto, este projeto permitiu-nos consolidar o que foi aprendido durante aulas de IA.

De futuro, esperamos tornar o sistema mais representativo do modo de funcionamento da Green Distribution, e esperamos também conseguir que seja mais realista e se aproxime mais de como este tipo de empresas de entregas funcionam no mundo real.