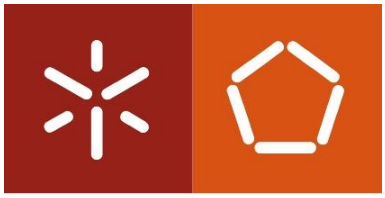


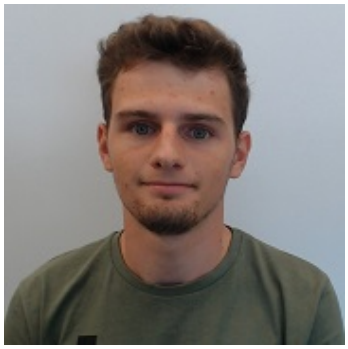
Universidade do Minho
Licenciatura em Engenharia Informática

Inteligência Artificial

Serviço de Gestão de Centro de Distribuição

Programação em lógica

Grupo 50



Bohdan Malanka
a93300

Diogo Rebelo
a93278

Henrique Alvelos
a93316

Teresa Fortes
a93250

20 de julho de 2022

Conteúdo

1 Introdução

No âmbito da Unidade Curricular de Inteligência Artificial, realizamos este trabalho prático de modo a aprofundar os conhecimentos relativamente a um sistema de representação de conhecimento e raciocínio, com capacidade para caracterizar um universo de discurso na área da logística de distribuição de encomendas.

Naturalmente, um dos objetivos é a utilização mais aprofundada da linguagem de programação em lógica, *Prolog*, também lecionada durante as aulas práticas. Neste contexto, perante o enunciado colocado, pretende-se inevitável e transversalmente promover a resolução de problemas, através de mecanismos de raciocínio relevantes.

2 Resumo

A Ecologia é um tema relevante e que, cada vez mais, marca o seu lugar nos dias de hoje. Com efeito, o trabalho em questão prende-se com o desenvolvimento de uma Base de Conhecimento alusiva a um sistema de gestão de distribuição de encomendas em contexto sustentável. Assim sendo, uma associação, *Green Distribution*, visa transportar um conjunto de encomendas efetuadas por um ou mais clientes, numa certa região incentivando à utilização, preferencialmente, de meios de transporte mais sustentáveis.
Então, determinados estafetas, responsáveis pela distribuição da encomenda, são recompensados proporcionalmente à sua taxa ecológica, ou valor ecológico global. Então, é de extrema relevância a implementação das várias funcionalidades solicitadas no respetivo enunciado, de modo a ver concretizado com sucesso a boa gestão da empresa.

Igualmente importante nesta secção, é referir que, ao longo do desenvolvimento de todo o trabalho (mais propriamente, a sua vertente mais prática: o código em si), foram surgindo algumas dúvidas em relação à melhor forma de representação da cada entidade e o modo de obtenção de cada resultado das *queries*.

Contudo, tiramos partido de várias funções já definidas pela própria linguagem para uma resolução bem conseguida e que nos pareceu mais eficaz.

Naturalmente, surge detalhada a explicação de cada modo de resolução das *queries* e a forma como se chegou ao resultado apresentado.

3 Implementação - Código

3.1 Arquitetura da aplicação

A solução construída está estruturada de acordo com a seguinte figura.

A seguir à mesma, encontra-se a informação em relação a cada ficheiro existente e a sua representação no contexto do trabalho.

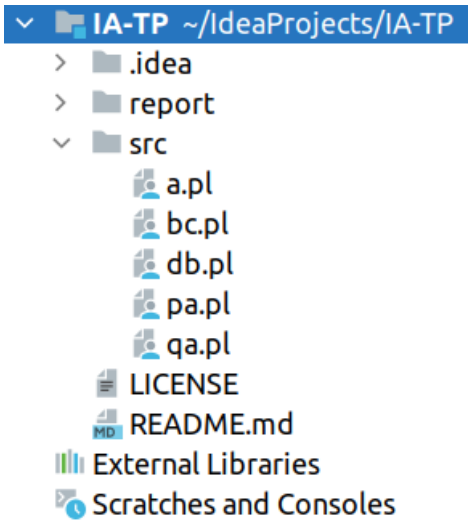


Figura 1: Arquitetura da aplicação desenvolvida.

Descrição de cada ficheiro

- **`a.pl`:** "a" de "algoritmos", é o ficheiro que contém todo o código desenvolvido sobre os algoritmos de pesquisa/procura utilizados.
- **`bc.pl`:** "bc" de "base de conhecimento", é o ficheiro que contém todo o código relativo ao conjunto de factos criados.
- **`db.pl`:** "bd" de "database", é o ficheiro que contém o código que define o conjunto de queries solicitadas e mais algumas criadas pelo grupo.
- **`pa.pl`:** "pa" de "predicados auxiliares", é o ficheiro que contém todo o código relativo ao conjunto de predicados auxiliares, isto é, a extensão de predicados que são utilizados como auxílio nas queries.
- **`qa.pl`:** "qa" de "queries de atualização", é o ficheiro que contém todo código alusivo às queries de atualização, isto é, que permitem adicionar, remover ou atualizar informação na aplicação.

3.2 Base de Conhecimento

Antes de começar a realizar as funcionalidades mínimas, tentamos perceber a forma como declarar conhecimento.

Como referido anteriormente, a modo como a Base de Conhecimento está organizada é importante não só em termos de compreensão do respetivo código, como também influencia a dificuldade de implementação de cada funcionalidade.

Assim sendo, selecionou-se a forma de organização da informação que facilitaria a posterior implementação. Seguem-se, então, as diferentes estruturas:

3.2.1 Transporte: `transporte(tipo, valorEco)`

Transporte é o meio usado pelo estafeta para se deslocar até ao destino da encomenda.

- Só tem 3 opções: bicicleta, moto e carro;
- Para diferenciar o valor económico, adicionou-se outro elemento. A bicicleta, moto e carro têm valor 3,2,1, respetivamente.

3.2.2 Estafeta: `estafeta(idEstafeta, sumEcologico, sumAvaliacao,nEntregas)`

O Estafeta é o funcionário da empresa que entrega encomendas.

- Para podermos diferenciar de estafeta para estafeta, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*;
- Para responder à *query* 1, foi necessário usar um parâmetro que funciona como acumulador, *sumEcologico*. Este valor acumula por cada vez que o estafeta usar algum tipo de transporte;
- De modo a responder à *query* 6 foi necessário usar outro parâmetro que também funciona como acumulador, e este corresponde à soma de todas as avaliações dos clientes aos quais entregou encomendas;
- Obviamente que, para fazer a média de valores, é necessário outro valor que corresponde ao número de entregas que o estafeta fez.

3.2.3 Cliente: `cliente(idCliente, freguesia,rua, nPedidosEntrega)`

O Cliente é uma pessoa que aderiu aos serviços da empresa.

- Para podermos diferenciar de cliente para cliente, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*;
- Para responder à *query* 5, foi necessário usar dois parâmetros cujos nomes são muito explicativos: freguesia e rua.

  Além disso, foi necessário outro constituinte de modo a verificar quantas entregas aquele cliente fez.

3.2.4 Encomenda: `encomenda(id, peso, volume, precobase)`

A encomenda é um pacote que vai ser entregue ao cliente pelo estafeta.

- Para podermos diferenciar de encomenda para encomenda, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*.
- De modo a resolver a *query* 10, o peso é necessário.

3.2.5 Cliente: `cliente(idCliente, freguesia,rua, nPedidosEntrega)`

O Cliente é uma pessoa que aderiu aos serviços da empresa.

- Para podermos diferenciar de cliente para cliente, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*;
- Para responder à *query* 5, foi necessário usar dois parâmetros cujos nomes são muito explicativos: freguesia e rua.

  Além disso, foi necessário outro constituinte de modo a verificar quantas entregas aquele cliente fez.

3.2.6 Entrega: `entrega(idEstafeta, idCliente, idEncomenda, dataE(A,M,D), dataR(A,M,D), transporte, preco, avaliacao)`

A entrega é uma espécie de relatório sobre os detalhes de um envio de uma encomenda.

- Para saber quem entregou a encomenda, é necessário saber o *ID* do estafeta;
- O envio teve que ser para alguém, daí ter o *ID* do cliente;
- Para verificar as encomendas não entregues, é necessário um parâmetro para a data de Envio;
- De modo a saber que a encomenda foi entregue, há um parâmetro para a data de receção. Caso não tenha sido entregue, o valor é dataR(0,0,0);
- O estafeta teve de usar um meio de transporte para se deslocar, logo também existe um argumento para o transporte;
- Para saber o lucro (*query* 4) da empresa num determinado dia, adicionamos um parâmetro para o preço;
- O cliente precisa de avaliar o trabalho do estafeta, daí ter um espaço para tal.

3.3 Queries Fase 1

Nesta parte, segue-se o conjunto da informação relacionada com o raciocínio por trás de cada *query* desenvolvida para a Fase 1 do trabalho, assim como detalhes e/ou aspetos que o grupo considera relevantes para a sua compreensão.

3.3.1 Query 1: Identificar o estafeta que utilizou mais vezes um meio de transporte mais ecológico

Assumimos que o estafeta mais ecológico seria aquele cujo valor da divisão entre o somatório do valor seu ecológico com o seu total de entregas fosse o maior.

Apesar de o objetivo ser a obtenção do estafeta com o maior valor ecológico, perante a existência de estafetas com igual valor, poderíamos tomar duas decisões: manter esta espécie de "duplicados" ou não.

Primeiramente, desenvolvemos o primeiro predicado, mas apercebemo-nos que o segundo seria mais correto.

Desta forma, construíram-se dois predicados diferentes:

obter_estafeta_mais_eco_sem_dups(Final)
Perante dois estafetas com o mesmo valor ecológico (e o maior da lista), na ordenação decrescente, há remoção destes duplicados. Neste caso, obtém-se como estafeta mais ecológico a primeira ocorrência na lista (o que surge à cabeça).
obter_estafeta_mais_eco_com_dups(Final)
Perante dois estafetas com o mesmo valor ecológico (e o maior da lista), na ordenção decrescente, não há remoção destes duplicados e a este valor ecológico máximo surge associado o nome de todos os estafetas com aquele respetivo valor.

3.3.2 *Query* 2: Identificar que estafetas entregaram determinada(s)

quemEntregou([IdEncomenda|T],IdCliente,Estafetas)
O nosso pensamento foi pesquisar quem entregou determinada encomenda ao cliente com o nome especificado e adicionar umpar (ID estafeta, ID encomenda) à medida que a pesquisa na base de conhecimento acontecia. No caso de a encomenda não estar associada ao cliente, a função simplesmente ignora.

3.3.3 *Query* 3: Identificar os clientes servidos por um determinado estafeta

quemServidos(IdEstafeta,Clientes)
O raciocínio adotado foi selecionar todas as entregas que o estafeta recebido como parâmetro fez, devolvendo assim uma lista de IDs de clientes sem duplicados.

3.3.4 *Query* 4: Calcular o valor faturado pela Green Distribution num determinado dia

quantoLucrou(Ano,Mes,Dia,Sum)
Visto que a entrega tem um elemento relacionado com o preço, o algoritmo foi pegar nesses elementos cujas entregas foram no dia selecionado e somar tudo.

3.3.5 *Query* 5: Identificar quais as zonas (e.g., rua ou freguesia) com maior volume de entregas por parte da Green Distribution

Neste caso, o objetivo é para cada entrega, formar uma lista de pares (Freguesia-Número de Encomendas), ordenando-a de acordo com o segundo elemento do par. No nosso caso, a freguesia é a Key e o outro valor é o Value.

Então, agrupa-se os pares pela respetiva chave (e.g. `[(gualtar-2,gualtar-3,tenoes-4,...]`) e, depois, somam-se valores de encomendas da mesma freguesia (e.g. `[(gualtar-5,tenoes-4,...]`). Tendo-se esta lista, trocam-se em todos os pares a Key com o Value, ordenando-os por default utilizando o keysort/2 (que não remove duplicados!). O algoritmo é semelhante para as ruas.

obter_zonas_mais_povoadas(HotSortedFinal, 1)
Obtém as ruas com mais entregas registadas. O primeiro argumento é a respetiva lista, o segundo serve apenas para indicar que a ordenação se dá por rua.
obter_zonas_mais_povoadas(HotSortedFinal, 2)
Obtém as freguesias com mais entregas registadas. O primeiro argumento é a respetiva lista, o segundo serve apenas para indicar que a ordenação se dá por freguesia.

3.3.6 *Query* 6: Calcular a classificação média de satisfação de cliente para um determinado estafeta

avaliacaoMedia(IdEstafeta, Media)
Dado que o estafeta tem dois parâmetros relacionados com a média de satisfação (`sumAvaliacao` e `nEntregas`) , basta dividir o primeiro valor pelo segundo.

3.3.7 *Query* 7: Identificar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo

nEntregadasTransporte(AnoI,MesI,DiaI,AnoF,MesF,DiaF,Lista)
O raciocínio foi o seguinte: começamos por filtrar as entregas que se encontram no intervalo de tempo dado e retiramos a lista de transportes. Com a ajuda da função auxiliar `ocorrências2Tuplas` transformamos essa lista obtida por uma com tuplos, onde se encontra cada transporte e as respetivas suas ocorrências.

3.3.8 *Query* 8: Identificar o número total de entregas pelos estafetas, num determinado intervalo de tempo

nEntregadas(AnoI,MesI,DiaI,AnoF,MesF,DiaF,Len)
Presumimos que o número total de entregas realizadas pelos estafetas eram as efetuadas com sucesso, então selecionamos, para uma lista, aquelas em que a data de receção estivesse no intervalo de tempo. Por fim, calculámos o tamanho da lista.

3.3.9 *Query* 9: Calcular o número de encomendas entregues e não entregues pela *Green Distribution*, num determinado período de tempo

A respetiva *query* obtém o conjunto de encomendas entregues (E) e não entregues (NE), num determinado período de tempo introduzido pelo utilizador.

Estas informações numéricas são mostradas na forma de um par (E,NE), visível no último parâmetro da query.

obter_entregas_no_periodo(AnoI,MesI,DiaI,AnoF,MesF,DiaF,(E,NE))
Para obter cada informação numérica usam-se dois predicados auxiliares que têm estrutura semelhante. O que auxilia no cálculo das encomendas entregues só verifica se para cada encomenda o envio se encontra no período recebido, agrupando as encomendas numa lista. Depois é obtido o tamanho da lista, tendo-se o número total. O predicado que auxilia no cálculo das encomendas não entregues só tem de verificar se a data de envia da encomenda é posterior à do utilizador e se a data de receção pelo cliente tem o formato (0,0,0).

3.3.10 *Query* 10: Calcular o peso total transportado por estafeta num determinado dia

pesoTotal(IdEstafeta, dataR(A,M,D), Total)
Para obter o peso total transportado por um estafeta criamos um predicado que recebe como parâmetros o ID desse estafeta, a data do dia e a resposta na variável Total. Assim, fazemos um *findall* das entregas desse dia recolhendo o ID das encomendas numa lista. Depois, com a ajuda da função auxiliar `pesoTotalAux`, transformamos essa lista por peso das respetivas encomendas e no final somamos os pesos todos.

3.4 Queries e Fundamentos - Fase 2

Nesta parte, segue-se o conjunto da informação relacionada com o raciocínio por trás de cada query desenvolvida para a Fase 2 do trabalho, assim como detalhes e/ou aspetos que o grupo considera relevantes para a sua compreensão.

3.4.1 Funcionalidade 1: Gerar os circuitos de entrega, caso existam, que cubram um determinado território (e.g. rua ou freguesia)

A funcionalidade em questão gera já o circuito tendo em conta os indicadores de produtividade mencionados na query 4. Assim o circuito está pronto a usar.

```
293 forneceCircuito(LRuas,LIDenc,Circuito) :-
294   getPesoTotal(LIDenc,Peso),
295   getVolumeTotal(LIDenc,Volume),
296   getDistanciaTotal(LRuas,DistTotal),
297   getMenorTempo(LIDenc,Prazo),
298   tempoEntregaMedio(Peso,bicicleta,DistTotal,TempoEntrega),
299   ((Peso =< 5, verificaDentroPrazo(Prazo,TempoEntrega)) ->
300   Circuito = circuito(LRuas,LIDenc,bicicleta,Peso,Volume,DistTotal,TempoEntrega),!;
301   tempoEntregaMedio(Peso,moto,DistTotal,TempoEntrega2),
302   (Peso =< 20, verificaDentroPrazo(Prazo,TempoEntrega2)) ->
303   Circuito = circuito(LRuas,LIDenc,moto,Peso,Volume,DistTotal,TempoEntrega2),!;
304   tempoEntregaMedio(Peso,carro,DistTotal,TempoEntrega3),
305   Circuito = circuito(LRuas,LIDenc,carro,Peso,Volume,DistTotal,TempoEntrega3),!).
```

Figura 2: Código que fornece um circuito.

Note-se que esta query apenas é responsável por fornecer um circuito para a(s) encomenda(s).

Sendo assim, começa-se por calcular o peso total da lista de encomendas especificadas, passando-se à obtenção do volume total das mesmas e da distância total a percorrer, consoante a lista de ruas recebidas, e chega-se ao calculo do prazo mínimo e cumprir.

De seguida, começa por calcular o tempo para o caminho recebido, começando no meio de transporte mais ecológico (a bicicleta).

Se o peso total da(s) encomenda(s) for superior ao que a bicicleta suporta ou se o prazo com a bicicleta não for cumprido, passa-se ao calculo do tempo para entregar a encomenda com o próximo transporte mais ecológico (a moto).

Verifica-se o prazo novamente. Se alguma das condições falhar, segue-se para o cálculo do tempo para o carro, o transporte menos ecológico.

Por fim, se uma das 2 primeiras condições se verificar, então, forma-se de imediato o circuito com os dados calculados anteriormente e com o transporte mais ecológico selecionado e respetivo tempo de entrega.

3.4.2 Requisito 2: Representação dos diversos pontos de entrega em forma de grafo, tendo em conta que apenas se devem ter localizações (rua e/ou freguesia) disponíveis

O predicado indica uma aresta existente entre as ruas que recebe como parâmetro.
O código que permite tal representação é:

```
135 %Extensão do predicado move: MoradaO,          MoradaD,                    CustoDistancia,  CustoTempo -> {V,F}
136 move(rua(rua_green_distribution,centro),       rua(rua_D_Sebastiao_onde_estas,nogueiro),   6,  10).
137 move(rua(rua_green_distribution,centro),       rua(rua_dos_alivios,nogueiro),              5,  5).
138 move(rua(rua_D_Sebastiao_onde_estas,nogueiro), rua(rua_dos_alivios,nogueiro),              2,  3).
139 move(rua(rua_D_Sebastiao_onde_estas,nogueiro), rua(rua_dos_aflitos,nogueiro),              3,  4).
140 move(rua(rua_dos_alivios,nogueiro),            rua(rua_dos_aflitos,nogueiro),              3,  5).
141 move(rua(rua_dos_aflitos,nogueiro),            rua(avn_da_liberdade,gualtar),              6,  10).
142 move(rua(rua_D_Sebastiao_onde_estas,nogueiro), rua(rua_da_ceramica,gualtar),               8,  10).
143 move(rua(rua_da_ceramica,gualtar),             rua(avn_da_liberdade,gualtar),              2,  5).
```

Figura 3: Código que representa o grafo.

3.4.3 Query 3: Identificar quais os circuitos com maior número de entregas (por volume e peso)

Query 3 (versão 1 e versão 2)
Para a respetiva query, foram realizados dois algoritmos diferentes no que diz respeito aos dados recebidos. A primeira versão tem em conta os predicados existentes na base de conhecimento, já a segunda versão recebe uma lista qualquer de circuitos.
maisEntregasPeso(Circuito) & maisEntregasVolume(Circuito)
Começa por formar uma lista com pares (Peso/Volume Total - Circuito), ordenando a respetiva lista por ordem descrescente de peso/volume. Agrupa circuitos com o mesmo peso/volume. Remove todas as keys da lista, deixando só os circuitos. Obtém a cabeça da lista que é o circuito com maior peso/volume total.
maisEntregasPeso_ListReceived(LCircuitos,Circuito) & maisEntregasVolume_ListReceived(LCircuitos,Circuito)
O algoritmo é extremamente parecido com o utilizado anteriormente, a única diferença persiste no que se faz inicialmente: em vez de se basear na base de conhecimento, percorre cada elemento da lista recebida.

3.4.4 *Queries* 4, 5 e 6

Uma vez que a query 4 utiliza funções das queries 5 e 6, consideramos relevante apresentar o raciocínio conjunto.

- 4: Comparar circuitos de entrega tendo em conta os indicadores de produtividade;
- 5: Escolher o circuito mais rápido (usando o critério da distância);
- 6: Escolher o circuito mais ecológico (usando um critério de tempo);

Query 4
comparaCircuitos(L)
Imprime a informação. Apresentando o circuito devolvido pela função seguinte. Apresenta o circuito mais ecológico e mais rápido.
comparaCircuitosAux([circuito(A,B,C,D,E,DistTotal,F)|L])
Recebendo uma lista com vários circuitos, a função serve apenas para imprimir no ecrã os conjunto dos argumentos de cada predicado circuito, ou seja, de cada elemento da lista recebida.
Queries 5 e 6
escolheCircuito([(circuito(A,B,C,D,F,DistTotal,E))|T],Metodo,Circuito)
Faz a seleção final do circuito, chamando a função em questão, consoante o estafeta pretenda o mais rápido ou ecológico.
escolheCircuitoAux([circuito(A,B,C,D,F,DistTotal,E)|T],DT,rapido,Circuito)
Seleciona o melhor circuito em termos de distância, dos circuitos recebidos. Para isso, vai comparando a distância total de cada circuito, verificando a menor até ao momento de comparação.
escolheCircuitoAux([circuito(A,B,C,D,F,DistTotal,E)|T],DT,ecologico,Circuito)
Seleciona o melhor circuito em termos de tempo de entrega dos circuitos recebidos. Para isso, vai comparando o tempo de entrega de cada circuito, verificando o menor até ao momento de comparação.

4 Problema de Procura

4.1 Formulação

O modo como o grupo formulou o problema surge bem exemplificado na imagem que surge a seguir.

Resumidamente, cada nodo do grafo representa uma rua existente, à exceção do nodo inicial/-raíz que representa a localização do centro de distribuição.

Cada aresta tem como informação representada a respetiva distância e tempo.

A informação, em código, é:

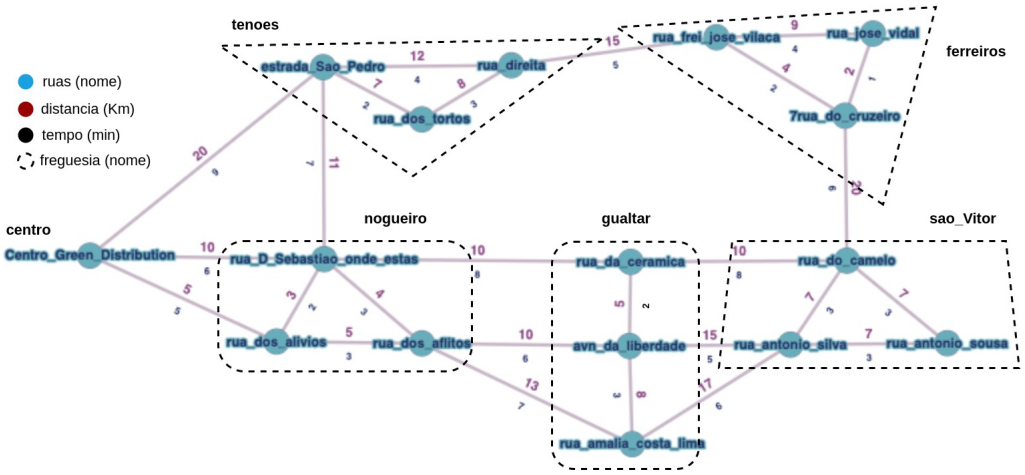


Figura 4: Representação das respetivas ruas e freguesias em grafo.

- **`rua(nome da rua, nome da freguesia):`** informação que consta em cada nodo.
- **`move(rua(...), rua(...), custo distância, custo tempo):`** informação que consta em cada aresta, ou seja, que estabelece a ligação entre os nodos.
- Cada caminho surge representado como uma lista de ruas. O mesmo poderia ser representado como uma lista de move's, todavia, na maioria dos casos, não se usa essa forma de representação.

4.2 Estratégias de Procura

Seguem-se aspetos relevantes alusivos ao método adotado para cada procura. Incluiu-se alguma informação sobre cada algoritmo de procura (informada ou não informada) utilizado.

Pesquisa
Não Informada
DFS (Depth First Search)
Ideia base: primeiro em profundidade. Avançar no grafo/árvore (em profundidade) enquanto for possível; Senão, recuar (backtrack) e tentar alternativamente outros caminhos (em profundidade também). Estrutura de dados requerida muito simples e leve (lista com nodos visitados ou caminho atual). Não garante o caminho mais curto ou com menos passos em primeiro lugar.
BFS (Breath First Search)
Ideia base: primeiro em largura. A partir de um nodo são explorados todos os nós adjacentes. Posteriormente, são explorados os nodos acessíveis através dos adjacentes (nível seguinte) e assim sucessivamente. Estrutura de dados requerida pesada: requerido armazenar todos os caminhos que ainda podem ser expandidos (fila de caminhos). Garante o caminho com menos passos encontrado em primeiro lugar.
IDDS (Iterative Deepening Depth-first Search)
Ideia base: forma de pesquisa em profundidade com progressão do limite utilizado. Não é necessário preocupar-se com como definir o limite de profundidade. Ótimo em relação ao facto de encontrar a solução mais curta primeiro. Pode não terminar.
Informada
A* (A star)
Ideia Base: possibilidade de utilizar uma função estimativa com a contabilização de custos acumulados conhecidos. Primeiro caminho produzido é obrigatoriamente o melhor. Sensível à distância do nodo atual ao nodo destino.
Greedy
Expande o nodo que parece ser o mais próximo do objetivo. Com uma heurística decente dirige-se praticamente diretamente ao objetivo. Melhor caso com tempo e espaço lineares. Não garante solução ótima.

Esta tabela reúne aspetos gerais bem sabidos sobre os algoritmos de pesquisa, a adaptação e explicação adaptada ao nosso código apresenta-se a seguir.

DFS

Funcionamento: não há muito a acrescentar em relação ao que surge anteriormente. A pesquisa ocorre em profundidade, enquanto isso for possível. Fornecendo-se a Origem (Orig) e o Destino (Dest), o algoritmo vai visitando os nodos e guardando-os numa lista (LA). É testada a existência de ciclos, para não existir circularidade e o caminho final é invertido. O caminho que ele constrói é apenas o primeiro que encontra, o que significa que não tem em conta distância ou tamanho do caminho.
Aplicação: Quando o estafeta pretende entregar certa(s) encomenda(s), precisa de saber o caminho a seguir, segundo um algoritmo por si selecionado. Sendo assim, indicando a lista de encomendas e o algoritmo a usar, é-lhe apresentado o circuito que deve seguir, assim como o meio de transporte a usar. É neste contexto que é utilizado.
Complexidade: Tempo: $O(V + E)$ Espaço: $O(V)$

BFS

Funcionamento: A pesquisa ocorre em largura, em que é explorado um nó e todos os seus adjacentes, de forma sucessiva. Há uma regra, a `getMove(Nodo1,Nodo2)`, que verifica a existência de nodos adjacentes e foi a única adaptação a fazer ao algoritmo.
Aplicação: Do mesmo modo que o algoritmo anterior, este surge no contexto de seleção de um caminho.
Complexidade: Tempo: $O(V + E)$ & Espaço: $O(V)$, onde V é o número de vértices e E é o número de arestas.

IDDS

Funcionamento: fornecendo a rua de origem, de destino e o limite utilizado, o algortimo inicia a sua procura decrementando o limite até que o objetivo seja cumprido.
Aplicação: Do mesmo modo que o algoritmo anterior, este surge no contexto de seleção de um caminho.
Complexidade: Tempo: $O(b^d)$ & Espaço: $O(d)$, onde b é o fator de ramificação e d é a profundidade do objetivo (destino).

A*

Funcionamento: demonstra ser o melhor dos três. Não só apresenta os melhores tempos, em geral, como também está parametrizado para apenas demonstrar o melhor caminho em termos de distância. No entanto, apesar destas vantagens, a heurística que implementamos para a estimativa utilizada em cada iteração não é necessariamente ideal, pois foi introduzida manualmente sem cálculos.
Aplicação: Do mesmo modo que o algoritmo anterior, este surge no contexto de seleção de um caminho.
Complexidade: Tempo: $O(b^d)$ & Espaço: $O(b^d)$, onde b é o fator de ramificação e d é a profundidade do objetivo (destino).

Gulosa

Funcionamento: é em tudo igual ao A*, exceto na escolha do próximo nodo. Aqui, apenas teremos em consideração o valo heurístico de cada nodo, e não o custo da aresta ou o custo guardado até agora. Assim, as soluções não são ótimas, mas sim, como é óbvio, heurísticas. Portanto, em termos de performance de tempo e de espaço é muito semelhante ao A*, mas a solução é normalmente menos próxima da ideal.
Aplicação: Do mesmo modo que o algoritmo anterior, este surge no contexto de seleção de um caminho.
Complexidade: Tempo: $O(b^d)$ & Espaço: $O(b^d)$, onde b é o fator de ramificação e d é a profundidade do objetivo (destino).

4.3 Análise de Resultados

Para a realização deste aspeto criaram-se as respetivas regras que medem a *performance* de cada algoritmo de pesquisa e correram-se todos com o centro de distribuições de ponto inicial e o nodo mais distante do grafo(Rua António Sousa, São Victor) anotando os resultados na tabela.

A pesquisa A* não encontrou a melhor solução porque achamos que é por causa das estimas que introduzimos manualmente e podem estar mal.

Uma solução seria, o predicado rua() ter coordenadas como parâmento e assim o cálculo da linha reta seria mais fácil.

Resultado	Tempo (ms)	Espaço(bytes)	Indicador /Custo	Encontrou a melhor solução? (Sim/Não)
DFS	Instantâneo	3744	Não	Não
IDDS	Instantâneo	3752 (Lim=10)	Não	Não
BFS	9	80176	Não	Não
Greedy	1	6448	Sim	Não
A*	1	8544	Sim	Não

```
?- statisticsBF(rua(rua_green_distribution,centro),rua(rua_antonio_sousa,sao_Victor),Caminho,Mem).
% 27,142 inferences, 0.009 CPU in 0.009 seconds (100% CPU, 2904311 Lips)
Caminho = [rua(rua_green_distribution, centro), rua(rua_D_Sebastiao_onde_estas, nogueiro), rua(rua_da_ceramica, gualtar),
 rua(rua_do_camelo, sao_Victor), rua(rua_antonio_sousa, sao_Victor)],
Mem = 80176 .

?- statisticsDF(rua(rua_green_distribution,centro),rua(rua_antonio_sousa,sao_Victor),Caminho,Mem).
% 88 inferences, 0.000 CPU in 0.000 seconds (93% CPU, 813452 Lips)
Caminho = [rua(rua_green_distribution, centro), rua(rua_D_Sebastiao_onde_estas, nogueiro), rua(rua_dos_alivios, nogueiro)
, rua(rua_dos_aflitos, nogueiro), rua(avn_da_liberdade, gualtar), rua(rua_antonio_silva, sao_Victor), rua(rua_antonio_sou
sa, sao_Victor)],
Mem = 3744 .

?- statisticsDFLimitado(rua(rua_green_distribution,centro),rua(rua_antonio_sousa,sao_Victor),Caminho,10,Mem).
% 94 inferences, 0.000 CPU in 0.000 seconds (93% CPU, 807928 Lips)
Caminho = [rua(rua_green_distribution, centro), rua(rua_D_Sebastiao_onde_estas, nogueiro), rua(rua_dos_alivios, nogueiro)
, rua(rua_dos_aflitos, nogueiro), rua(avn_da_liberdade, gualtar), rua(rua_antonio_silva, sao_Victor), rua(rua_antonio_sou
sa, sao_Victor)],
Mem = 3752 .

?- statisticsGulosa(rua(rua_green_distribution,centro),rua(rua_antonio_sousa,sao_Victor),(Caminho,Custo),Mem).
% 209 inferences, 0.001 CPU in 0.001 seconds (99% CPU, 184586 Lips)
Caminho = [rua(rua_green_distribution, centro), rua(rua_D_Sebastiao_onde_estas, nogueiro), rua(rua_da_ceramica, gualtar),
 rua(rua_do_camelo, sao_Victor), rua(rua_antonio_sousa, sao_Victor)],
Custo = 25,
Mem = 6448 .

?- statisticsAEstrela(rua(rua_green_distribution,centro),rua(rua_antonio_sousa,sao_Victor),(Caminho,Custo),Mem).
% 284 inferences, 0.001 CPU in 0.001 seconds (99% CPU, 478865 Lips)
Caminho = [rua(rua_green_distribution, centro), rua(rua_D_Sebastiao_onde_estas, nogueiro), rua(rua_da_ceramica, gualtar),
 rua(avn_da_liberdade, gualtar), rua(rua_antonio_silva, sao_Victor), rua(rua_antonio_sousa, sao_Victor)],
Custo = 24,
Mem = 8544 .

?-
```

Figura 5: Resultados dos respetivos algoritmos

5 Comentários Finais, Conclusão e Trabalho Futuro

Em relação à concretização dos objetivos propostos, o grupo confirma que estes se encontram bem estabelecidos e concretizados. O grupo considera que o modo como o enunciado estava feito proporcionava abordagens diferentes de desenvolvimento, o que levou a várias reflexões e toma de decisões sobre qual(ais) a(s) melhor(es) forma(s) de implementar certa funcionalidade.

Na construção e desenvolvimento do trabalho houve solidez e organização, permitindo posterior concretização.

Contudo, sabemos que, no conhecimento positivo, quando fazemos expansão, ou seja, inserimos novos predicados, não actualizamos o resto dos predicados, porque eles estão relacionados entre si. Em suma, consideramos que o trabalho foi bem conseguido, não só porque realizamos todas as *queries*, requisitos e funcionalidades solicitados, como também adicionamos mais algumas funcionalidades, por exemplo, a atualizar parâmetros aos estafetas, clientes e transporte, o fornecimento do circuito com mais encomendas entregues por peso e volume (adaptando-se ao circuito recebido e não apenas aos existentes na base de conhecimento).

6 Referências Bibliográficas

1. Russell and Norvig (2009). Artificial Intelligence - A Modern Approach, 3rd edition, ISBN-13: 9780136042594;

2. Costa E., Simões A., (2008), Inteligência Artificial-Fundamentos e Aplicações, FCA, ISBN: 978-972-722-34