# Relatório do Exercício 2

Rodolfo Silva (a81716)

Braga, Junho de 2020

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre/2019-2020)

## Resumo

Este projeto resume-se na criação de um sistema de recomendação de transportes públicos, nomeadamente de autocarros, onde se aplica o uso de diversos algoritmos de pesquisa em grafos, e visa a solidificar o conhecimento relativo ao paradigma da programação em lógica e as suas aplicações no mundo.

# Conteúdo

# Lista de Figuras

# 1 Introdução

O principal objetivo deste projeto consiste na solidificação da utilização da linguagem de programação em lógica *Prolog*, no âmbito da pesquisa em grafos usando algoritmos de procura.

O tema em estudo é a área dos serviços públicos para a sugestão de sequência de paragens a percorrer para completar um trajeto, com o objectivo de aprofundar o conhecimento relativo à linguagem *Prolog* e as suas diversas aplicações no mundo exterior.

## 2 Preliminares

Antes de dar início ao desenvolvimento de um sistema de recomendação de transporte público, nomeadamente de autocarros, foi necessário realizar uma breve análise aos ficheiros *.csv* que foram fornecidos, e aos requisitos impostos no enunciado.

Esta análise visa estudar a melhor forma de armazenar a informação das Paragens/Adjacências necessária à realização dos requisitos, bem como a identificação de uma linguagem de programação adequada para o fazer.

Por fim, decidiu recorrer-se à linguagem *Python* para realizar a extração dos dados presentes nos ficheiros *.csv*, que foram posteriormente organizados em três diferentes conjuntos de frases, que serão brevemente explicitadas.

# 3 Breve Descrição do Projeto Realizado

De seguida, encontram-se os três tipos de frases utilizados para representar o conhecimento das paragens, tal como um pequeno exemplo:

- **nextTo**: $Paragem1, Paragem2 \rightarrow \{V, F\}$
- **adjacente**: $Paragem1, Paragem2, Carreira \rightarrow \{V, F\}$
- **paragem**: $Gid, Latitude, Longitude, Estado\ de\ Conservao,$
  $Tipo\ de\ Abrigo, Existncia\ de\ publicidade, Operadora, Carreiras,$
  $Codigo\ de\ Rua, Nome\ da\ Rua, Freguesia \rightarrow \{V, F\}$

```
paragem(750,-103210.92,-101837,bom,fechado_dos_lados,no,lt,[119,125],1634,'estrada de talaíde','porto salvo').
paragem(752,-103260.70410270982,-101287.68122082386,bom,void,no,lt,[106,119],262,'estrada de leceia','porto salvo').
adjacente(750,752,119).
nextTo(750,752).
```

Pequena amostragem de como o conhecimento foi guardado.

Algumas liberdades foram tomadas na realização do *parsing* dos ficheiros, de forma a facilitar o uso da base de conhecimento, nomeadamente:

- O encoding dos ficheiros *.csv* foi alterado.
- Campos vazios encontrados no decorrer do *parsing* passaram a tomar o valor '*void*'.
- O campo das carreiras na frase paragem foi colocado dentro de uma lista.
- Os últimos dois campos da frase paragem foram encapsulados dentro de apóstrofes, permitindo o seu uso sem possíveis desformatações.

## 3.1 Algoritmos

De modo a conseguir obter resultados conclusivos e concretos a partir da realização deste projeto, foram realizados três algoritmos diferentes, *Breadth first search*, Procura em A* e por fim *Greedy Search*.

Tambem foi desenvolvido um algoritmo de procura em profundidade, mas devido ao seu mau desempenho foi removido.

### 3.1.1 Breadth first search

Este algoritmo representa o algoritmo mais simples que foi implementado. Consta de um algoritmo não informado que vai verificar todas as paragens adjacentes a uma outra paragem, de modo a encontrar a paragem destino.

Todos as paragens pela qual o algoritmo passa vão ser adicionadas a uma lista de paragens visitadas, que é posteriormente utilizada como o caminho final. Este caminho irá, mais adiante, criar problemas, dado existirem múltiplas paragens dentro do caminho que não lhe pertencem. Desta forma, foi implementado um predicado que visa remover as paragens que não pertencem ao caminho final.

```
bfs([Node|_],Node,History,Caminho) :- remDupes(Node,History,[],Caminho).
bfs([Node|RestQ],Goal,History,Caminho) :- findall(NextNode , ( near(Node,NextNode),\+member(NextNode,History),\+member(NextNode,RestQ) ), Sucessores),
                                          sort(Sucessores,Succ),
                                          append(RestQ,Succ,Queue),
                                          bfs(Queue,Goal,[Node|History],Caminho).


remDupes(Node,[],NewV,[Node|NewV]).
remDupes(Node,[Head|Tail],NewV,Caminho) :- (near(Node,Head) -> remDupes(Head,Tail,[Node|NewV],Caminho) ; remDupes(Node,Tail,NewV,Caminho) ).

near(Nodo1,Nodo2) :- nextTo(Nodo1,Nodo2) ; nextTo(Nodo2,Nodo1).
```

Algoritmo Breadth First.

### 3.1.2 Procura em A*

O algoritmo A* Informado foi implementado neste projeto usando como Estima o cálculo da distância Euclidiana entre a longitude e latitude da paragem origem e destino.

Este algoritmo consiste na utilização da Estima calculada para conseguir ordenar a lista de espera, de modo a que o caminho mais próximo do destino seja o caminho que se vai desenvolver. Isto permite, frequentemente, encontrar o caminho mais curto entre uma Origem e um Destino.

```
aestrela(Caminhos, Goal ,Caminho) :- obtem_melhor(Caminhos, Caminho, Goal),
                                     Caminho = [Goal|_]/_/_.
aestrela(Caminhos, Goal ,SolucaoCaminho) :- obtem_melhor(Caminhos, MelhorCaminho, Goal),
                                            seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
                                            expande_aestrela(MelhorCaminho, Goal,ExpCaminhos),
                                            append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
                                            aestrela(NovoCaminhos, Goal ,SolucaoCaminho).


expande_aestrela(Caminho, Goal ,ExpCaminhos) :- findall(NovoCaminho, adj(Caminho,NovoCaminho, Goal), ExpCaminhos).


adj( [Nodo|Caminho]/Custo/_ , [ProxNodo,Nodo|Caminho]/NovoCusto/Est , Goal) :- near(Nodo, ProxNodo),\+ member(ProxNodo, Caminho),
                                                                               euclidean(Nodo,ProxNodo,PassoCusto),
                                                                               NovoCusto is Custo + PassoCusto,
                                                                               euclidean(ProxNodo, Goal, Est).
```

Algoritmo A*.

### 3.1.3 Greedy Search

O algoritmo Greedy Informado vai procurar todos os adjacentes possíveis de uma certa paragem e, de seguida, irá ordenar a lista de espera de modo a que o caminho com o menor número de paragens esteja sempre na cabeça da lista.

Isto permite calcular o caminho que usa menos paragens entre dois pontos com eficiência medíocre.

```
greedy([[Goal|Resto]/N|Soln],Goal,Visitados,Path/Paragens) :- inverso([Goal|Resto],Path), Paragens is N.
greedy([[Node|Resto]/N|Soln],Goal,Visitados,CP) :- findall(ProxNodo,(near(Node,ProxNodo),\+member(ProxNodo,Visitados)),Nodos),
                                                   buildLists([Node|Resto]/N,Nodos,[],New),
                                                   append(Soln,New,MiddleMan),
                                                   mysort(MiddleMan,[],FinalList),
                                                   greedy(FinalList,Goal,[Node|Visitados],CP).

mysort([],Temp,Final) :- inverso(Temp,Final).
mysort([MM/N|Resto],Temp,FList) :- getLowest(MM,N,Resto,Lowest) , delete_each(Lowest,[MM/N|Resto],MiddleMan) , mysort(MiddleMan,[Lowest|Temp],FList).

getLowest(LowestSoFar,LowestValue,[],LowestSoFar/LowestValue).
getLowest(LowestSoFar,LowestValue,[Caminho/Valor|Cauda],Lowest) :- ( Valor < LowestValue -> getLowest(Caminho,Valor,Cauda,Lowest) ;
                                                                                            getLowest(LowestSoFar,LowestValue,Cauda,Lowest)).

buildLists(_,[],List,List).
buildLists(Lista/N,[Nodo|Resto],NewList,Resultado) :- NN is N+1 , buildLists(Lista/N,Resto,[[Nodo|Lista]/NN|NewList],Resultado).
```

Algoritmo Greedy.

## 3.2 Resposta a Questões Colocadas

Nesta secção, vai ser efetuada uma pequena descrição das alterações que foram realizadas aos algoritmos de forma a conseguir a realização das *queries* propostas.

### 3.2.1 Calcular um trajeto entre dois pontos

Para a primeira *query* a ser realizada, calcular o trajeto entre dois pontos sem restrições, foi usado o Algoritmo *BreadthFirst* sem alterações para a realização desta *query*.

### 3.2.2 Selecionar uma operadora de transporte para um determinado percurso

Na segunda *query* a ser realizada, foi utilizado tanto o Algoritmo *BreadthFirst* como o Algoritmo A*, unica mudança feita aos algoritmos foi a inclusão de um predicado que verifica se uma paragem pertence a operadora dada como argumento.

Predicado que verifica se Nodo pertence a operadora desejada.

```
operadoraValida(Nodo,Operadora) :-
                    paragem(Nodo,_,_,_,_,_,Operadora,_,_,_,_).
```

### 3.2.3 Escolher o percurso que passe apenas por paragens abrigadas

No que diz respeito à terceira *query* a ser realizada, recorreu-se ao Algoritmo *BreadthFirst* e ao Algoritmo A*, sendo que a alteração realizada aos algoritmos foi a inclusão de um predicado que verifica se uma paragem era abrigada.

Considerou-se como abrigadas as paragens que no campo de "Tipo de Abrigo" continham 'fechado_dos_lados' ou 'aberto_dos_lados'.

Predicado que verifica se a paragem possui abrigo

```
abrigoValido(Nodo) :-
                paragem(Nodo,_,_,_,fechado_dos_lados,_,_,_,_,_,_);
                paragem(Nodo,_,_,_,aberto_dos_lados,_,_,_,_,_,_).
```

### 3.2.4 Escolher o percurso que passe apenas por abrigos com publicidade

A quarta *query* a ser realizada foi desenvolvida usando tanto o Algoritmo *BreadthFirst* como o Algoritmo A*. A única alteração realizada está na inclusão de um predicado que verifica se uma paragem tem publicidade.

Predicado que verifica se a paragem tem publicidade.

```
publicidadeValida(Node) :- paragem(Node,_,_,_,_,yes,_,_,_,_,_).
```

### 3.2.5 Identificar quais as paragens com o maior número de carreiras num determinado percurso

A quinta *query* a ser realizada foi implementada apenas com o Algoritmo $BreadthFirst$, sendo que, não houve nenhuma mudança ao algoritmo em si, mas depois da sua conclusão, foi realizado um *sort* a fim de ordenar as paragens do caminho por ordem decrescente de carreiras.

Predicado usado para ordenar o resultado.

```
ordenaC([],MiddleMan,Resultado) :- inverso(MiddleMan,Resultado).
ordenaC(Caminho,MiddleMan,Resultado) :-
        getGreater(Caminho,-1,-1,Greatest),
        delete_each(Greatest,Caminho,NewCaminho),
        ordenaC(NewCaminho,[Greatest|MiddleMan],Resultado).
```

### 3.2.6 Excluir um ou mais operadores de transporte para o percurso

A sexta *query* a ser realizada foi desenvolvida recorrendo aos algoritmos $BreadthFirst$ e A*. A única alteração efetuada aos algoritmos está na inclusão de um predicado que verifica se uma paragem não pertence a nenhuma das operadoras fornecidas como argumento.

Predicado que verifica se não pertence as operadoras fornecidas.

```
operadoraNope(Node,[]).
operadoraNope(Node,[Op|Cauda]) :-
  (paragem(Node,_,_,_,_,_,Op,_,_,_,_) -> !,fail ;
  operadoraNope(Node,Cauda)).
```

### 3.2.7 Escolher o percurso mais rápido (usando critério da distância)

Na sétima *query*, recorreu-se ao Algoritmo A*, sem nenhuma alteração, visto que o algoritmo já procura por ele próprio o caminho com a menor distância, distância essa calculada pela soma das distancias Euclidianas de cada par de paragens pertencentes ao caminho resultante.

#### 3.2.8 Escolher o menor percurso (usando critério menor número de paragens)

Para a oitava *query*, foi utilizado o Algoritmo *Greedy*, sem nenhuma alteração realizada, uma vez que o algoritmo em si já procura o caminho com menos paragens possíveis entre dois pontos.

#### 3.2.9 Escolher um ou mais pontos intermédios por onde o percurso deverá passar

A nona *query* a ser realizada foi desenvolvida com o uso do Algoritmo *Greedy*, sem alterações significativas ao algoritmo.

O motivo para o uso deste algoritmo em particular, está no facto de que este permite encontrar o caminho com menos paragens entre cada 2 pontos, o que facilita a geração de um caminho interligado.

Tal interligação foi atingida através do cálculo do caminho entre a Origem e a cabeça da lista de paragens obrigatórias; o consequente cálculo entre todos os caminhos da lista e, por fim, o cálculo do caminho entre o último elemento da lista com o destino.

# 4 Análise de Resultados

Após vários testes realizados aos diferentes algoritmos, chegou-se à conclusão demonstrada na tabela que se segue.

O Algoritmo *DepthFirst* muito raramente produzia resultados devido à profundidade do grafo.

O Algoritmo *BreadthFirst* produz resultados praticamente para quase todos as combinações de pares Origem/Destino que se testou, algumas vezes o resultado era ótimo pelo simples facto de haver uma solução no início do grafo. Quando isso não se verificava, o resultado raramente era ótimo.

O Algoritmo A*, apesar de ter sempre resultados ótimos, sofria também alguns dos problemas do *DepthFirst* em que, devido ao tamanho do grafo, demora algum tempo a concluir.

O Algoritmo *Greedy* consegue ter resultados ótimos e, para uma árvore de conhecimento do tamanho utilizado, conseguia ter uma eficiência entre tempo e resultados melhor que o A*. Uma vez que este algoritmo testa todos os caminhos possíveis, é provável que, com o crescimento do grafo, essa eficiência não se mantenha.

| **Algoritmo** | **Completo** | **Ótimo** | **Tempo** | **Ciclos Infinitos** |
|---|---|---|---|---|
| DFS | Não | Raramente | Mau | Não |
| BFS | Sim | Ocasionalmente | Bom | Não |
| A* | Sim | Sim | Intermédio | Não |
| Greedy | Não | Sim | Intermédio | Não |

# 5 Conclusões e Sugestões

O desenvolvimento deste projeto permitiu o aprimoramento do conhecimento relativamente ao paradigma da programação em lógica, fortalecendo a noção da sua utilidade na resolução de diversos problemas.

Tendo em conta que todas as funcionalidades propostas foram implementadas com sucesso, considera-se este projeto um sucesso.

Em suma, foi possível solidificar os conhecimentos na linguagem *Prolog* e também compreender a utilidade deste paradigma na resolução de problemas que necessitem de uma abordagem imparcial.

# 6 Referências


- Leonardo Bottaci. Introductory notes on Prolog and AI Search.

# 7 Anexos

```
| ?- help.
All Functions
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminho2Pontos(Start,Goal,Caminho) - Calcula um trajeto entre dois pontos
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoComOperadora(Start,Goal,Operadora,Caminho) - Calcula um trajeto entre dois pontos usando apenas uma Operadora
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoComOperadoraInf(Start,Goal,Operadora,Caminho) - Calcula um trajeto entre dois pontos usando apenas uma Operadora, pesquisa informada
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoAbrigo(Start,Goal,Caminho) - Calcula um trajeto entre dois pontos usando apenas paragens abrigadas
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoAbrigoInf(Start,Goal,Caminho) - Calcula um trajeto entre dois pontos usando apenas paragens abrigadas, pesquisa informada
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoPublicidade(Start,Goal,Caminho) - Calcula um trajeto entre dois pontos usando apenas paragens com publicidade
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoPublicidadeInf(Start,Goal,Caminho) - Calcula um trajeto entre dois pontos usando apenas paragens com publicidade, pesquisa informada
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoSemOperadora(Start,Goal,[],Caminho) - Calcula um trajeto entre dois pontos excluindo o uso de X operadoras
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoSemOperadoraInf(Start,Goal,[],Caminho) - Calcula um trajeto entre dois pontos usando apenas paragens com publicidade, pesquisa informada
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoCarreira(Start,Goal,Caminho,Ordem) - Calcula um trajeto entre dois pontos e depois ordena as paragens por numero total de carreiras
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoMaisCurto(Origem, Goal, Caminho/Custo) - Calcula o trajeto mais curto entre dois pontos (em termos de distÃ¢ncia), pesquisa informada
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoMenosParagens(Origem, Goal, Caminho/Paragens) - Calcula o trajeto com menos paragens entre dois pontos, pesquisa informada
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
caminhoComParagens(Start,Goal,Paragens,Caminho) - Calcula um trajeto que passa por todas as paragens dentro de "Paragens"
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
yes
```

Predicados para operar perante o conhecimento.

```
| ?- caminho2Pontos(722,410,C).
C = [722,723,751,762,756,208,797,191,795,796,828,287,1004,290,279,289,288,388,387,386,385,389,441,402,410] ?
yes
```

Caminho entre 2 paragens com BFS.

```
| ?- caminhoMaisCurto(722,410,C).
C = [722,723,751,762,756,208,797,191,795,796,828,287,1004,290,289,288,388,387,386,385,389,440,275,410]/6224.043333934534 ?
```

Caminho entre 2 paragens com A*.

```
| ?- caminhoMenosParagens(722,410,C).
C = [722,723,751,762,756,208,797,191,795,796,828,287,1004,290,289,288,388,387,386,385,389,441,402,410]/24 ?
```

Caminho entre 2 paragens com Greedy.

```
| ?- caminhoComOperadoraInf(131,467,vimeca,Caminho).
Caminho = [131,129,763,754,183,171,799,599,860,861,858,859,610,336,357,334,339,347,86,85,341,342,345,363,335,457,458,490,491,56,655,654,78,467] ?
yes
```

Caminho com operadora com algoritmo informado.

```
| ?- caminhoSemOperadoraInf(467,858,[scotturb],Caminho).
Caminho = [467,78,654,655,56,491,490,458,457,335,363,345,342,341,85,86,347,339,334,357,336,610,859,858] ?
yes
```

Caminho sem operadora com algoritmo informado.

```
| ?- caminhoAbrigo(131,245,Caminho).
Caminho = [131,129,763,754,183,791,595,594,185,89,107,237,250,261,597,953,248,244,245] ?
yes
| ?- caminhoAbrigoInf(131,245,Caminho).
Caminho = [131,129,763,754,183,791,595,594,185,107,250,597,953,248,243,245] ?
yes
```

Comparação entre caminho com abrigo, informado e não informado.

```
| ?- caminhoCarreira(131,128,Caminho,Ordem).
Caminho = [131,129,763,754,183,171,172,162,161,156,147,736,745,128],
Ordem = [183,147,736,745,128,171,172,162,161,156,129,131,763,754] ?
yes
```

Caminho entre 2 pontos e paragens ordenadas por numero de carreiras.