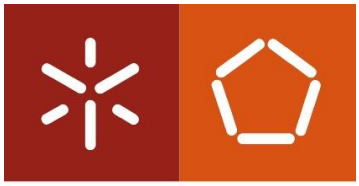

# Universidade do Minho

Mestrado Integrado de Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio
(2ºSemestre - 2019/20)

# Relatório SRCR - Trabalho Individual

A84414 João Pedro Gonçalves Correia

Braga

1 de Junho de 2020

# Resumo

Este relatório tem como objetivo descrever a solução criada para a problemática proposta pelos docentes da cadeira de Sistemas de Representação de Conhecimento e Raciocínio.

A meta do trabalho consistiu na representação e modelação do sistema de transportes do concelho de Oeiras e na aplicação de algoritmos de pesquisa sobre a rede criada.

Foram utilizados algoritmos de pesquisa não informada e pesquisa informada, utilizando como heurísticas as informações geográficas existentes nos datasets fornecidos.

Para tal foi necessário processar os datasets fornecidos de forma a representar sob a forma de um grafo a rede de transportes e, posteriormente, implementar um conjunto de algoritmos que atuem sobre esse grafo.

Este processo culminará num sistema de construção de percursos que respeitará as limitações indicadas pelo utilizador.

# Conteúdo

# Lista de Figuras

# Capítulo 1

# Introdução

Este trabalho teve como objetivo a representação do sistema de transportes de Oeiras e a aplicação de algoritmos de pesquisa sobre este, recorrendo ao sistema de inferência do prolog como base da construção destes caminhos.

O sistema de transportes consistirá, assim, na existência de paragens e carreiras que as ligam entre si. Estas estão representadas em vários datasets que serão processados e de onde será extraída a informação relevante para a construção da representação do sistema de transportes.

Para a elaboração dos percursos foram adaptados vários algoritmos, tanto de pesquisa não informada como informada, que terão diferentes vantagens consoante as especificações do percurso a gerar.

# Capítulo 2

# Descrição do Trabalho e Análise de Resultados

O trabalho está divido em cinco diferentes componenentes que representarão diferentes processos de manipulação do grafo dos transportes.

O primeiro componente será o processo de **pré-processamento**, onde os ficheiros .xlxs fornecidos serão processados e transformados em ficheiros .csv que o prolog poderá carregar e interpretar.

O componente do **carregamento dos dados** estará encarregue de carregar a informação existente nos csv's processados para a base de informação do prolog, criando as ligações necessárias para a elaboração do grafo.

O componente da **pesquisa não informada** implementará algoritmos de pesquisa não informada que criarão caminhos consoante as restrições indicadas pelo utilizador.

O componente da **pesquisa informada** implementará algoritmos que utilizem heurísticas para obter os melhores caminhos consoante a heurística utilizada.

O componente do **cálculo do tempo de viagem** utilizará métricas de tráfego para ao cálculo aproximado do tempo de viagem.

## 2.1 Pré-processamento

Foram fornecidos dois ficheiros .xlsx diferentes como dataset base para o projeto. Estes foram:

- Um ficheiro referente a **todas as paragens**. Este contém, numa só sheet, todas as paragens, assim como toda a informação relativa a cada uma;
- Um ficheiro referente a **cada carreira**. Este contém várias sheets, cada uma correspondente a uma carreira, em que as paragens estão organizadas de forma sequencial de forma a representar a sua organização dentro da carreira. Também este ficheiro contém toda a informação relativa a cada paragem.

Cada paragem contém, associada a si, um conjunto de informações:

1. GID(id geográfico);
2. Latitude;
3. Longitude;
4. Estado de conservação;
5. Tipo de abrigo;
6. Se a paragem tem publicidade;
7. Operadora;
8. Número da carreira;
9. Código de rua;
10. Nome da rua;
11. Freguesia.

Nestes dois ficheiros foram detetadas um conjunto de anomalias que deviam ser resolvidas, nomeadamente:

- Encoding errado dos caracteres com acento.
- Informação inexistente nos campos de algumas paragens.
- Carreiras identificadas no ficheiro das paragens que não existem no ficheiro das carreiras.

Para resolver estes problemas foi utilizada a biblioteca de manipulação de datasets *Pandas* da linguagem Python. Esta possui uma API de tratamento de ficheiros sobre a forma de dataframes que permite abstrair o formato dos ficheiros que lhe são fornecidos.

Para tratar dos encodings usou-se a função **df.replace** para substituir cada corrupção pelo verdadeiro caracter.

Figura 2.1: Tratamento de caracteres corrompidos

```
def muda_encoding_paragens(df):
    lista_colunas = ['Nome da Rua','Freguesia'] #colunas em que existe corrupção de caracteres

    for coluna in lista_colunas:
        df[coluna] = df[coluna].str.replace('Ã£','ã')
        df[coluna] = df[coluna].str.replace('Ã³','ó')
        df[coluna] = df[coluna].str.replace('Ã©','é')
        df[coluna] = df[coluna].str.replace('Ã¢','â')
        df[coluna] = df[coluna].str.replace('Ã§','ç')
        df[coluna] = df[coluna].str.replace('Ãª','ê')
        df[coluna] = df[coluna].str.replace('Ãº','ú')
        df[coluna] = df[coluna].str.replace('Ã¡','á')
        df[coluna] = df[coluna].str.replace('Ã','Á')
        df[coluna] = df[coluna].str.replace('Ã','í')

    return df
```

Quanto ao problema da falta de informação em algumas entradas, o pandas representa estes campos como NaN e o seu tratamento irá depender de qual campo seja. Se o NaN se encontrar nos campos GID, latitude, longitude ou carreira, estas entradas são eliminadas. Caso contrário, o campo é marcado como desconhecido e o seu tratamento dependerá do predicado no prolog que utilize esse campo.

Figura 2.2: Tratamento de campos desconhecidos

```
# tratará dos valores desconhecidos nas colunas nos exceis das adjacencias da forma que se julge relevante
def processa_nans_adjacencias(df):
    colunas_onde_se_elimina = ["gid","latitude","longitude","Carreira"]

    df = df.dropna(subset = colunas_onde_se_elimina)
    # nas restantes colunas substitui por Desconhecido
    df = df.replace(np.nan,'Desconhecido', regex=True)

    return df


def processa_nans_paragens(df):
    colunas_onde_se_elimina = ["gid","latitude","longitude"]

    df = df.dropna(subset = colunas_onde_se_elimina)
    # nas restantes colunas substitui por Desconhecido
    df = df.replace(np.nan,'Desconhecido', regex=True)

    return df
```

Por fim, em relação ao problema das carreiras inexistentes no ficheiro de todas as paragens, a estratégia consistiu em fazer com que cada um dos ficheiros .xlsx contenha apenas os campos estritamente necessários. Assim, removeu-se o campo das carreiras do ficheiro de todas as paragens e removeram-se todos os campos de informação suplementar no ficheiro das carreiras, passando a ser este a ditar a existência de carreiras.

Figura 2.3: Tratamento das carreiras inexistentes

```
#devolve um dataframe apenas com as colunas desejadas, no dataframe das adjacências
def seleciona_colunas_adjacencia(df):
    colunas_a_manter = ["gid","latitude","longitude","Carreira"]
    df = df[colunas_a_manter]
    return df


def seleciona_colunas_paragens(df):
    colunas_a_manter = ["gid","latitude","longitude","Estado de Conservacao","Tipo de Abrigo",
    "Abrigo com Publicidade?","Operadora", "Codigo de Rua",
    "Nome da Rua","Freguesia"]
    df = df[colunas_a_manter]
    return df
```

Os dataframes processados foram depois guardados sob a forma de ficheiros .csv. O ficheiro com todas as paragens ficou guardado com o nome *paragens.csv* enquanto que o dataframe das carreiras foi separado nas várias sheets que o compunham e cada uma foi guardada com o nome da carreira.

## 2.2 Carregamento dos dados

O carregamento dos dados para o prolog foi feito através do predicado read_csv_file de uma biblioteca do swi-prolog. No carregamento do csv das paragens, cada linha é inserida na base de dados sob a forma do predicado paragem/10. Este conterá os seguintes termos:

1. GID(id geográfico);
2. Latitude;
3. Longitude;
4. Estado de conservação;
5. Tipo de abrigo;
6. Se a paragem tem publicidade;
7. Código de rua;
8. Nome da rua;
9. Freguesia.

No carregamento dos vários csv's relativos às carreiras, as ligações entre duas paragens ficam representadas através do predicado adjacencia/4. Os campos deste predicado serão:

1. GID da paragem atual;
2. GID da paragem à qual a paragem atual está ligada;
3. Distância entre as duas paragens;
4. Número da carreira.

A distância entre as paragens é calculada usando o algoritmo da distância euclidiana entre as suas latitudes e longitudes.

## 2.3 Pesquisa não-informada

Os algoritmos de pesquisa não informada foram aplicados aos seguintes requerimentos do trabalho:

- Calcular um trajeto entre dois pontos;
- Selecionar apenas algumas das operadoras de transporte para um determinado percurso;
- Excluir um ou mais operadores de transporte para o percurso;
- Identificar quais as paragens com o maior número de carreiras num determinado percurso;
- Escolher o percurso com o menor número de paragens
- Escolher o percurso que passe apenas por abrigos com publicidade;
- Escolher o percurso que passe apenas por paragens abrigadas;
- Escolher um ou mais pontos intermédios por onde o percurso deverá passar;
- Listar as ruas e freguesias por onde um caminho passa (*extra*).

### 2.3.1 Calcular um trajeto entre dois pontos

Foram desenhados dois algoritmos diferentes para a pesquisa não-informada: um algoritmo de pesquisa *Depth first* e um algoritmo que constroi o caminho a partir do destino até à partida.

Cada um destes algoritmos apresenta diferentes vantagens e desvantagens: o algoritmo depth first demora mais a encontrar uma solução quando o caminho é bastante complexo, porém, quando encontra, encontra também rapidamente várias alternativas. O algoritmo que constrói o caminho a partir do destino encontra o primeiro caminho mais rapidamente, porém quando se tentam encontrar alternativas, demora mais tempo e muitas delas são bastantes semelhantes.

Foi ainda implementado um algoritmo breadth first, porém, para evitar gastos excessivos de memória, teve-se de discartar a opção de conter informação sobre as carreiras, pois isso aumentaria exponencialmente o tempo de pesquisa. Visto que esta procura apresentou um tempo de pesquisa ordens de magnitude superior às outras duas procuras implementadas, preferiu-se desenvolver as duas primeiras soluções para problemas mais complexos. Além do mais, esta implementação do algoritmo não permite obter diferentes alternativas.

| | Depth first | Contrução inversa | Breadth first |
|---|---|---|---|
| Começo | Partida | Destino | Partida |
| Complexidade | $O(V+E)$ | $O(V+E)$ | $O(V+E)$ |
| Optima? | Não | Não | Não |
| Completa? | Sim | Sim | Sim |
| Rapidez a encontrar a primeira solução | Moderada | Elevada | Reduzida |
| Diversidade de soluções? | Elevada | Reduzida | Não-existente |
| Viabilidade da primeira solução? | Moderada | Moderada | Elevada |

Tabela 2.1: Tabela comparativa das várias resoluções propostas (Pesquisa não-informada)

### 2.3.2 Seleção de operadoras

Para a adição desta restrição, foi necessário alterar os algoritmos de forma a que apenas seguissem por paragens que fossem operados por certas operadoras. Como a informação das adjacências e das paragens está separada em dois predicados diferentes, é primeiro necessário verificar os gid's das paragens adjacentes, usando o predicado adjacencia/4, depois, para esses gid's, através do predicado paragem/10, ver qual a operadora que opera nessa paragem. Por fim verifica-se se essa operadora pertence à lista fornecida pelo utilizador.

Figura 2.4: Seleção de um caminho que apenas seja operado pelas operadoras indicadas

```
adjacencia( Paragem, ProxParagem,DistanciaParagem,Carreira),
paragem(Paragem,_,_,_,_,_,Operador,_,_,_),
member(Operador,ListaOperadores),
```

### 2.3.3 Exclusão de operadoras

Para a exclusão de certas operadoras, o processo é muito semelhante ao anterior, porém agora passa-se a verificar que a operadora escolhida não pertence ao conjunto fornecido.

Figura 2.5: Seleção de um caminho que não seja operado pelas operadoras indicadas

```
adjacencia( Paragem, ProxParagem,DistanciaParagem,Carreira),
paragem(Paragem,_,_,_,_,_,Operador,_,_,_),
\+ member(Operador,ListaOperadores),
```

### 2.3.4 Seleção das paragens com mais carreiras

Este algoritmo deverá devolver, para além do trajeto, a lista das paragens ordenadas pelo número de carreiras que aí passam. O processo consistirá em converter cada paragem num tuplo (GID, número de carreiras que passam nesse GID), usando o seguinte algoritmo:

Figura 2.6: Cálculo de quantas carreiras passam por uma paragem

```
quantas_carreiras(GID,NumeroCarreiras):- findall((GID),(adjacencia(GID,_,_,_)),L),
                                         length(L,NumeroCarreiras).


converte_tuplo_para_n_carreiras((GID,_),(GID,NCarreiras)):-
    quantas_carreiras(GID,NCarreiras).
```

É depois utilizado o algoritmo de sort do swi prolog, comparando o segundo termo dos pares em ordem descendente.

Figura 2.7: Ordenação pelo número de carreiras

```
maplist(converte_tuplo_para_n_carreiras, Solucao,ListaASerOrdenada),
sort(2,  @>=, ListaASerOrdenada,  ListaOrdenadaPorNumeroDeCarreiras),
distancia_para_tempo(Distancia,Tempo).
```

### 2.3.5 Seleção do caminho com menor número de paragens

Este problema é substancialmente mais complicado que os restantes apresentados, visto que a primeira solução encontrada pelo algoritmo de pesquisa depth first poderá não satisfazer o pedido.

Para este propósito, podem ser tomadas duas diferentes opções: ou um algoritmo de pesquisa não informada que terá de encontrar todos os caminhos e depois verificar qual é o que satisfaz a a condição imposta, ou um algoritmo de pesquisa informada. Visto que a heurística que se teria de utilizar teria de informar a quantas paragens se está do destino, isso derrotaria o propósito do algoritmo, pois todos os caminhos teriam de ser calculados previamente.

Como tal, foi utilizado o algoritmo de depthfirst. A partir deste geram-se todas as soluções e usa-se o algoritmo de sort nativo do prolog para ordenar as soluções pelo seu número de paragens. Como forma de melhorar a performance, este agoritmo não vai ter em conta as diferentes carreiras que

passam por uma paragem, pois isso iria aumentar exponencialmente o número de soluções. Também foi removida a possibilidade de as carreiras seguirem em ambos os sentidos, pela mesma exata razão.

Figura 2.8: Ordenação pelo número de paragens

```
menor_caminho_paragens(Comeco, Destino,MenorCaminho,DistanciaCaminho,Tempo,NumeroParagens):-
    findall((Solucao,Distancia),resolve_df2(Comeco,Destino,Solucao,Distancia),L),
    maplist(adiciona_numero_paragens,L,ListaComNumeroParagens),
    sort(2,  @<, ListaComNumeroParagens,  ListaOrdenadaPorDistancia),
    nth0(0,ListaOrdenadaPorDistancia,(MenorCaminho,DistanciaCaminho,NumeroParagens)),
    distancia_para_tempo(DistanciaCaminho,Tempo).

adiciona_numero_paragens((Caminho,Distancia),(Caminho,Distancia,NParagens)):-
    length(Caminho,NParagens).
```

### 2.3.6 Seleção de caminhos apenas com publicidade

Para satisfazer esta condição, adiciona-se uma condição extra na seleção da paragem seguinte, como já foi feito em cima. Quando se encontram paragens com valores desconhecidos no campo da publicidade, é assumido que estas não contém publicidade.

Figura 2.9: Seleção de paragens com publicidade

```
adjacencia( Paragem, ProxParagem,DistanciaParagem,Carreira),
paragem(ProxParagem,_,_,_,_,'Yes',_,_,_,_),
```

### 2.3.7 Seleção de caminhos abrigados

Visto que o campo relativo ao abrigo não é binário, como era o caso na publicidade, tem de se verificar se o campo relativo ao abrigo tem o valor 'Aberto dos lados' ou 'Fechado dos Lados', vistos erem estes os que se classificam como abrigados.

Figura 2.10: Seleção de paragens abrigadas

```
adjacencia( Paragem, ProxParagem,DistanciaParagem,Carreira),
paragem(ProxParagem,_,_,_,TipoAbrigo,_,_,_,_,_),
member(TipoAbrigo,['Aberto dos Lados','Fechado dos Lados']),
```

### 2.3.8 Escolha de pontos intermédios

Esta opção adiciona a obrigação de um caminho conter um dado conjunto de paragens no percurso. Para obter este fim, é utilizado o predicado contem_paragens, que verifica que cada paragen está contida no percurso, independentemenata da carreira a que pertence o caminho.

Figura 2.11: Seleção de paragens intermédias

```
contem_paragens([Paragem|T],Caminho):
    member((Paragem,_),Caminho),
    contem_paragens(T,Caminho).

contem_paragens([],_).
```

### 2.3.9 Listar as ruas e freguesias por onde um caminho passa

De forma a aproveitar toda a informação fornecida no dataset, esta opção permite também saber as ruas e freguesias por onde um caminho passa. Isso é feito reccorrendo ao penúltimo e ao último campo do predico paragem/10.

Figura 2.12: Listagem de ruas e freguesias

```
resolve_df_lista_ruas_e_freguesias(Comeco,Destino,Solucao,Distancia,Tempo,ListaRuas,ListaFreguesias):-
    depthfirst( [], Comeco, Destino,SolucaoInvertida,Distancia),
    reverse(SolucaoInvertida,Solucao),
    distancia_para_tempo(Distancia,Tempo),
    cria_lista_ruas_e__Freguesias(Solucao,ListaRuasComDuplicados,ListaFreguesiasComDuplicados),
    sort(ListaFreguesiasComDuplicados,ListaFreguesias),
    sort(ListaRuasComDuplicados,ListaRuas).



cria_lista_ruas_e__Freguesias([(Paragem,_)|T],[Rua|T2],[Freguesia|T3]):-
    paragem(Paragem,_,_,_,_,_,_,_,Rua,Freguesia),
    cria_lista_ruas_e__Freguesias(T,T2,T3).

cria_lista_ruas_e__Freguesias([],[],[]).
```

## 2.4 Pesquisa informada

Os algoritmos de pesquisa informada terão de utilizar heurísticas para a auxiliar a pesquisa e decisão de que paragens escolher a seguir. Neste trabalho, a heurística que foi utilizada foi a distância em linha reta desde a paragem atual até à paragem destino. Esta será obtida calculando a distância euclidiana entre as latitudes e longitudes dos pontos.

### 2.4.1 Cálculo do percurso mais rápido

Para este problema foram abordadas três diferentes alternativas:

- Findall de todas as soluções de um algoritmo de pesquisa não informada;
- Pesquisa Gulosa;
- Algorimo A*.

A opção de usar o findall utilizará o algoritmo de pesquisa depth first. Esta deverá encontrar todos os caminhos e depois ordenará a lista de possíveis percursos tendo em conta a distância.

O algoritmo de pesquisa gulosa utilizará a heurística fornecida e, a cada iteração, escolherá o ponto cuja heurística seja menor.

Por último, o algoritmo A* terá em conta a distância real percorrida, mas também utilizará heurísticas para determinar que caminho expandir.

O algoritmo de pesquisa gulosa foi alterado de forma a evitar a formação de ciclos.

O seguinte quadro apresenta as diferenlas entre os algoritmos:

| | Findall DF | Gulosa | A* |
|---|---|---|---|
| Complexidade | $O(N^V)$ | $O(exp)$ | $O(exp)$ |
| Informada? | Não | Sim | Sim |
| Optima? | Sim | Não | Sim |
| Completa? | Sim | Sim | Sim |
| 183 - 182 | 11 | <1 | 31 |
| 183 - 609 | 10 | <1 | 41 |

Tabela 2.2: Tabela comparativa das várias resoluções propostas (Procura do menor caminho)

## 2.5 Cálculo da distância e tempo de viagem

Para obter a distância total da viagem vão-se somando as distâncias existentes no último termo do predicado adjacencia/4.

Este valor é depois usado para obter o tempo de viagem num algoritmo que tem em conta mais três fatores:

- Quantidade de tráfego;
- Dia da semana;
- Hora do dia.

Para o cálculo da variação de tráfego durante o dia, foram utilizados os valores médio da velocidade dos transportes públicos da cidade de Boston. Esta cidade foi escolhida por ter um número de população semelhante ao concelho de lisboa e Oeiras somados, dois concelhos vizinhos que, principalmente na hora de ponta terão muito tráfego partilhado.

Quanto ao dia da semana, foi feita uma distinção entre os dias úteis da semana e os dias do fim de semana, visto que estes últimos apresentam tráfego em menor quantidade quando comparado com o dos dias úteis.

Por fim, em relação à hora do dia é feita uma penalização severa na velocidade média nas horas de ponta e uma penalização ligeira noutras horas de alto tráfego. Como tal, as velocidades médias nas várias partes da semana serão:

- 30km/h às 8,9,17 e 18h nos dias úteis;
- 35km/h às 10,13,14,16e 19h nos dias úteis;
- 40km/h às 8,9,16,17e 18h ao fim de semana;
- 45km/h nas alturas restantes.

Este algoritmo está presente no algoritmo *predicados_auxiliates.pl*. O tempo de viagem é devolvido sobre a forma de um tuplo (nº de horas, nº de minutos).

São utilizados os predicados get_time, stamp_time e date_time_value do swi-prolog para a obtenção da hora do dia e do dia da semana.

O algoritmo de cálculo do tempo de viagem está presente no anexo A.1

# Capítulo 3

# Conclusão

Em conclusão, todas as funcionalidades foram implementadas com sucesso, tendo sindo exploradas diferentes alternativas à resolução dos problemas, tendo em conta diversos problemas, como tempo de execução e gestão de memória.

Durante a execução deste projeto, a maior dificuldade consistiu na adaptação dos algoritmos, em relação à sua utilização de memória, dada a escala do grafo gerado.

Em futuras iterações do trabalho, gostaria de implementar mais algoritmos de pesquisa, tanto informada como não-informada, assim como uma funcionalidade que permitissse visualizar graficamente o caminho escolhido.

# Capítulo 4

# Referências

## 4.1 Referências Bibliográficas

[Novais, 2020] Novais, Paulo,
"Métodos de resolução de problemas e de procura"
Universidade do Minho, 2020

[RUSSEL,2010], RUSSEL S. J., Norvig, P., & Davis, E.
Artificial intelligence: a modern approach. 3rd ed.
Upper Saddle River, NJ, 2010

## 4.2 Referências Eletrónicas

Acesso a ficheiros CSV utilizando o ficheiro prolog. Disponível em:
https://www.swi-prolog.org/pldoc/man?section=csv. Acesso em: 15 mai. 2020.

Dealing with time and date. Disponível em:
https://www.swi-prolog.org/pldoc/man?section=timedate. Acesso em: 15 mai. 2020.

A* search algorithm. Disponível em:
https://en.wikipedia.org/wiki/A*_search_algorithm. Acesso em 18 mai. 2020.

# Apêndice A

# Mecânica de cálculo do tempo de viagem

```
%calcula quanto tempo demora a percorrer a dada distância
distancia_para_tempo(Distancia,(Horas,Minutos)):-
    hora_do_dia(Hora,DiaDaSemana),
    velocidade_media(Hora,DiaDaSemana,VelocidadeMedia),
    DistanciaArredondada is round(Distancia),
    Horas is DistanciaArredondada//VelocidadeMedia,
    DistanciaRestante is DistanciaArredondada - Horas*VelocidadeMedia,
    Minutos is round((60*DistanciaRestante)/VelocidadeMedia).
```

Figura A.1: Mecânica de cálculo do tempo de viagem