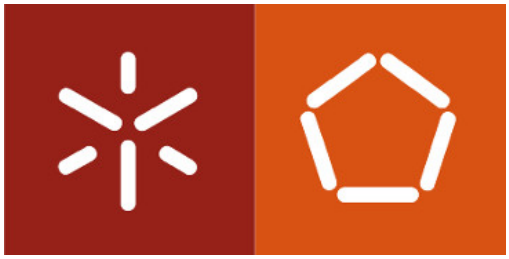

Universidade do Minho

Mestrado integrado em Engenharia Informática

COMPONENTE INDIVIDUAL

Diogo Pinto Ribeiro

A84442

Sistemas de Representação de Conhecimento e Raciocínio

3º ano, 2º semestre

Departmento de Informática

5 de Junho de 2020

Resumo

O presente trabalho foca-se na aplicação de métodos de pesquisa em problemas reais, sendo que o caso de estudo consiste na rede de transportes públicos do concelho de Oeiras. Foi elaborado um parser para interpretar os dados e gerar o respetivo código Prolog, sendo que também foram desevolvidos métodos de pesquisa informada e não-informada para a pesquisa de caminhos entre diversas paragens.

Índice

1 Introdução

Neste trabalho foi-nos proposto o problema da rede de transportes do concelho de Oeiras. Perante este problema, o nosso objetivo é transformar esta rede num grafo, de modo a conseguir efetuar pesquisas nele. Neste trabalho vamo-nos focar em responder a certas questões recorrendo a dois métodos de pesquisa: a pesquisa não-informada e a pesquisa informada. Para a implementação destas pesquisas e estruturas foi usado Prolog, sendo que esse é um dos objetivos do trabalho. No entanto, os dados fornecidos não se encontram já prontos a importar para o sistema, sendo que é necessário desenvolver um Parser que possa gerar estruturas já processadas. Após a implementação de ambos os métodos e depois de responder às questões propostas, procedeu-se também a uma comparação entre os dois métodos de pesquisa de modo a compreender quais as diferenças entre cada um relativamente ao problema em questão, além das diferenças estruturais. Dada a liberdade que nos foi dada em certos aspetos de implementação, surgiu logo inicialmente uma questão: dar preferência a uma distância menor entre o destino e origem, ou, dar preferência a considerar o melhor caminho aquele que passa por menos paragens. Devido ao facto de o autocarro, num cenário realista, perder bastante tempo em paragens, e, ao facto de apenas calcularmos a distância Euclidiana, optou-se por considerar que o caminho ideal é aquele que passa por menos paragens e não aquele que percorrer uma menor distância.

2 Dados do Problema

Para melhor entender o problema em questão, é necessário analisar com detalhe os dados fornecidos. Foram-nos fornecidos dois datasets no formato xlsx, sendo que cada um contém a mesma informação representada de maneiras distintas: o ficheiro paragens.xlsx contém uma lista de todas as paragens a considerar, bem como a informação respetiva a cada uma; o ficheiro adjacencias.xlsx contém a informação relativa a carreiras, isto é, informa-nos quais as paragens que cada carreira percorre. Seguem-se dois excertos destes datasets para melhor se compreender o conjunto de dados:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	gid	latitude	longitude	Estado de (	Tipo de Abrigo	Abrigo com Publicidad	Operadora	Carreira	Codigo de Rua	Nome da Rua	Freguesia		
2	79	-107011.5	-95214.57	Bom	Fechado dos Lados	Yes	Vimeca	01	103	Rua Damião de Góis	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
3	593	-103777.0	-94637.67	Bom	Sem Abrigo	No	Vimeca	01	300	Avenida dos Cavaleiros	Carnaxide e Queijas		
4	499	-103758.4	-94393.36	Bom	Fechado dos Lados	Yes	Vimeca	01	300	Avenida dos Cavaleiros	Carnaxide e Queijas		
5	494	-106803.2	-96265.84	Bom	Sem Abrigo	No	Vimeca	01	389	Rua São João de Deus	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
6	480	-106757.3	-96240.22	Bom	Sem Abrigo	No	Vimeca	01	389	Rua São João de Deus	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
7	957	-106911.1	-96261.15	Bom	Sem Abrigo	No	Vimeca	01	399	Escadinhas da Fonte da Maruja	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
8	366	-106021.3	-96684.5	Bom	Fechado dos Lados	Yes	Vimeca	01	411	Avenida Dom Pedro V	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
9	365	-106016.1	-96673.87	Bom	Fechado dos Lados	Yes	Vimeca	01	411	Avenida Dom Pedro V	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
10	357	-105236.9	-96664.4	Bom	Fechado dos Lados	Yes	Vimeca	01	1279	Avenida Tomás Ribeiro	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
11	336	-105143.5	-96690.32	Bom	Fechado dos Lados	Yes	Vimeca	01	1279	Avenida Tomás Ribeiro	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
12	334	-105336.0	-96668.68	Bom	Fechado dos Lados	Yes	Vimeca	01	1279	Avenida Tomás Ribeiro	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
13	251	-104487.6	-96548.01	Bom	Fechado dos Lados	Yes	Vimeca	01	1279	Avenida Tomás Ribeiro	Carnaxide e Queijas		
14	469	-106613.4	-96288	Bom	Fechado dos Lados	Yes	Vimeca	01	1288	Rua Rodrigo Albuquerque e Melo	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
15	462	-106636.2	-96302.04	Bom	Sem Abrigo	No	Vimeca	01	1288	Rua Rodrigo Albuquerque e Melo	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
16	44	-104458.5	-94926.22	Bom	Fechado dos Lados	Yes	Vimeca	01,13,15	1134	Largo Sete de Junho de 1759	Carnaxide e Queijas		
17	78	-107008.5	-95490.23	Bom	Fechado dos Lados	Yes	Vimeca	01,02,06,14	118	Alameda Hermano Patrone	Algés, Linda-a-Velha e Cruz Quebrada-Dafundo		
18	609	-104226.4	-95797.22	Bom	Fechado dos Lados	Yes	Vimeca	01,02,07,10,12,13,15	327	Avenida do Forte	Carnaxide e Queijas		
19	599	-104296.7	-95828.26	Bom	Fechado dos Lados	Yes	Vimeca	01,02,07,10,12,13,15	327	Avenida do Forte	Carnaxide e Queijas		
20	595	-103725.6	-95975.2	Bom	Fechado dos Lados	Yes	Vimeca	01,02,07,10,12,13,15	354	Rua Manuel Teixeira Gomes	Carnaxide e Queijas		
21	185	-103922.8	-96235.62	Bom	Fechado dos Lados	Yes	SCoTTURB	01,02,07,10,12,13,15	354	Rua Manuel Teixeira Gomes	Carnaxide e Queijas		
22	250	-104031.0	-96173.83	Bom	Fechado dos Lados	Yes	Vimeca	01,02,07,10,12,13,15	1113	Avenida de Portugal	Carnaxide e Queijas		
23	107	-103972.3	-95981.88	Bom	Fechado dos Lados	Yes	Vimeca	01,02,07,10,12,13,15	1113	Avenida de Portugal	Carnaxide e Queijas		
24	953	-104075.8	-95771.82	Bom	Fechado dos Lados	Yes	Vimeca	01,02,07,10,12,13,15	1116	Avenida Professor Dr. Reinaldo dos San	Carnaxide e Queijas		
25	594	-103879.9	-95751.23	Bom	Fechado dos Lados	No	Vimeca	01,02,07,10,12,13,15	1116	Avenida Professor Dr. Reinaldo dos San	Carnaxide e Queijas		
26	597	-104058.9	-95839.14	Bom	Fechado dos Lados	Yes	Vimeca	01,02,07,10,12,13,15	1137	Rua Tenente-General Zeferino Sequeira	Carnaxide e Queijas		

Figura 1: Screenshot paragens.xlsx

	A	B	C	D	E	F	G	H	I	J	K
1	gid	latitude	longitude	Estado de Conservacao	Tipo de Abrigo	Abrigo com	Operadora	Carreira	Codigo de Rua	Nome da Rua	Freguesia
2	116	-101520.29	-100001.26	Bom	Fechado dos Lados	Yes	Carris	171	252	Avenida Infante Dom Henrique	Barcarena
3	119	-101709.63	-100014.88	Bom	Fechado dos Lados	Yes	LT	171	978	Avenida de Santo AntÃ³nio de Tercena	Barcarena
4	118	-101728.86	-100021.08	Bom	Fechado dos Lados	No	LT	171	978	Avenida de Santo AntÃ³nio de Tercena	Barcarena
5	104	-101753.46	-99755.19	Bom	Fechado dos Lados	Yes	LT	171	261	Rua da Juventude	Barcarena
6	711	-101764.30649856283	-98424.15159847475	Bom	Sem Abrigo	No	LT	171	216	Estrada Consiglieri Pedroso	Barcarena
7	105	-101764.82	-99761.18	Bom	Fechado dos Lados	Yes	LT	171	261	Rua da Juventude	Barcarena
8	125	-101787.42	-98423.54	Bom	Fechado dos Lados	Yes	LT	171	216	Estrada Consiglieri Pedroso	Barcarena
9	115	-101877.84	-99707.56	Bom	Sem Abrigo	No	LT	171	1006	Rua AntÃ³nio Quadros	Barcarena
10	120	-101884.83	-100069.82	Bom	Fechado dos Lados	Yes	LT	171	978	Avenida de Santo AntÃ³nio de Tercena	Barcarena
11	121	-101894.85	-100053.16	Bom	Sem Abrigo	No	LT	171	978	Avenida de Santo AntÃ³nio de Tercena	Barcarena
12	1012	-101927.83891266519	-99709.84354381096	Bom	Sem Abrigo	No	LT	171	1006	Rua AntÃ³nio Quadros	Barcarena
13	127	-101949.9	-98542.91	Bom	Fechado dos Lados	Yes	LT	171	216	Estrada Consiglieri Pedroso	Barcarena
14	715	-101966.52	-98573.78	Bom	Fechado dos Lados	Yes	LT	171	216	Estrada Consiglieri Pedroso	Barcarena
15	134	-102017.79	-99652.24	Bom	Fechado dos Lados	Yes	LT	171	219	Estrada da Cruz dos Cavalinhos	Barcarena
16	122	-102021.07	-99964.5	Bom	Sem Abrigo	No	LT	171	978	Avenida de Santo AntÃ³nio de Tercena	Barcarena
17	141	-102028.47	-99961.71	Bom	Sem Abrigo	No	LT	171	978	Avenida de Santo AntÃ³nio de Tercena	Barcarena
18	143	-102122.63	-99975.95	Bom	Sem Abrigo	No	LT	171	241	Estrada das Fontainhas	Barcarena
19	744	-102136.13485160771	-98663.30880207638	Bom	Fechado dos Lados	No	LT	171	216	Estrada Consiglieri Pedroso	Barcarena
20	142	-102137.2	-99979.69	Bom	Sem Abrigo	No	LT	171	241	Estrada das Fontainhas	Barcarena
21	135	-102185.42	-99474.62	Bom	Sem Abrigo	No	LT	171	219	Estrada da Cruz dos Cavalinhos	Barcarena
22	136	-102207.02	-99467.54	Bom	Sem Abrigo	No	LT	171	219	Estrada da Cruz dos Cavalinhos	Barcarena
23	152	-102231.41	-98789.31	Bom	Fechado dos Lados	Yes	LT	171	216	Estrada Consiglieri Pedroso	Barcarena
24	139	-102277.41	-100088.41	Bom	Sem Abrigo	No	LT	171	241	Estrada das Fontainhas	Barcarena

Figura 2: Screenshot adjacencias.xlsx

Agora, que já sabemos melhor quais os dados com que iremos lidar, podemos começar a efetuar certas decisões. Para este problema, irá ser considerada uma Carreira como um circuito efetuado por um determinado autocarro, num sentido apenas, e que chegando à ultima paragem, regressa à primeira paragem, reiniciando o percurso.

Nesta fase do trabalho ficámos a perceber a importância da qualidade dos dados que iremos importar para o sistema. Qualquer discrepância nos valores afeta o resultado final, e todo o sistema sofre se os dados tiverem pouca qualidade.

3 Parser

Para processar os dados que vimos na secção anterior, foi necessário construir um pequeno Parser que permitisse ao mesmo tempo transformar os dados contidos no ficheiro xlsx em código Prolog e ao mesmo tempo modelar alguns aspetos. Para isso optou-se por construir o Parser em **Python**, utilizando uma biblioteca adicional para ler e tratar os ficheiros xlsx, **pandas**.

3.1 Estruturas

A partir da análise dos dados surgiram as seguintes estruturas necessárias ao funcionamento do sistema:

3.1.1 Paragens

A primeira estrutura representa uma paragem dentro do sistema. Para representar esta entidade dentro do sistema necessitamos dos seguintes campos: **gid**, **Latitude**, **Longitude**, **Estado de Conservação**, **Tipo de Abrigo**, **Abrigo com Publicidade**, **Operadora** e o **Código de Rua**. As paragens são geradas a partir do ficheiro paragens.xlsx, e, cada linha de código em Prolog toma o seguinte aspeto:

```
?- insereParagem(gid,lat,lon,estado,tipo,pub,operadora,codrua).
```

3.1.2 Ruas

De modo a evitar repetir ruas, achou-se uma boa prática criar uma estrutura só para as ruas, sendo que cada rua consiste nos seguintes campos: **Código de Rua**, **Nome da Rua** e **Freguesia**. As ruas são geradas a partir do ficheiro paragens.xlsx, e, cada linha de código em Prolog toma o seguinte aspeto:

```
?- insereRua(codrua,nome,freguesia).
```

Através do campo Código de Rua é possível efetuar a ligação entre uma Paragem e a Rua em que esta se situa.

3.2 Adjacências

Para ser possível representar um grafo, é necessário que existam arestas. No nosso caso, uma aresta é representada pelo predicado percurso, sendo que é composto por 4 parâmetros: **paragem**, **paragem seguinte**, **carreira**, e **distancia**. Esta aresta representa o percurso que um autocarro de uma determinada carreira faz entre 2 paragens consecutivas. O código gerado para cada percurso tem a seguinte forma, sendo que inclui o predicado evolução:

```
?- evolucao(percurso(paragem1,paragem2,carreira,distancia)).
```

Estas arestas são provenientes do ficheiros adjacencias.xlsx. As distâncias são distâncias euclidianas tal como sugeridas no enunciado.

3.3 Estimativas

Para a pesquisa informada são necessárias estimativas como vamos ver mais à frente, sendo que cada estimativa contém o identificador de um paragem, bem como a distancia dessa mesma paragem à paragem de destino. De momento sempre que quisermos utilizar um destino diferente na pesquisa informada, temos que gerar as estimativas de novo. O código gerado para cada estimativa tem o seguinte aspeto, sendo que inclui o predicado evolução:

```
?- evolucao(estimativa(gid,distancia)).
```

Todo código gerado por este parser é colocado num ficheiro dados.pl que irá ser importado dentro do sistema, permitindo por sua vez utilizar os dados fornecidos.

4 Definição do Problema

Dadas as paragens e as suas respetivas adjacências, é possível construir uma estrutura representativa do problema, na forma de um grafo. Os nós do grafo são constituídos pelo **gid** da paragem, permitindo assim aceder à informação de cada paragem mantendo o grafo simples. Os arcos do grafo são representados pelo predicado **percurso**, que inclui a paragem de onde parte, a paragem seguinte, a carreira que realiza a viagem, e a distância entre as duas paragens. Assim sendo, o grafo produzido com este formato tem o seguinte aspeto:

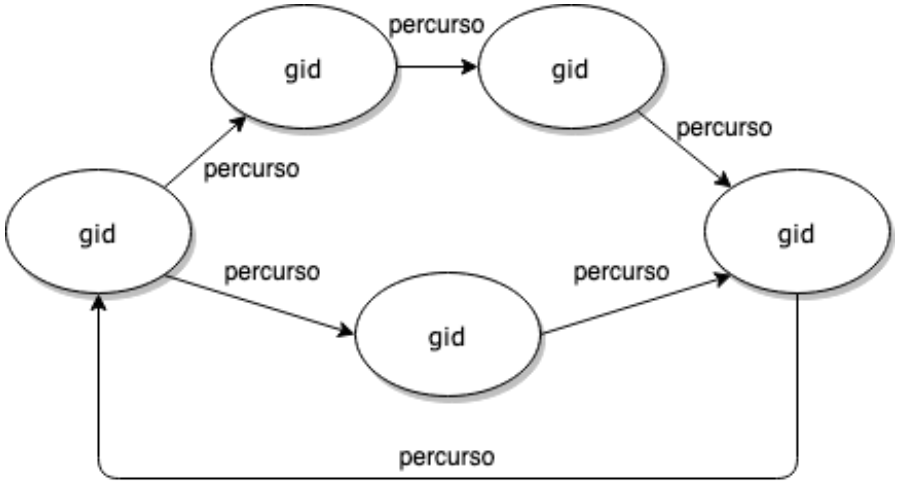

Figura 3: Exemplo de um grafo

Nos ficheiros não se encontra nenhuma informação sobre um ligação entre 2 paragens ser bidirecional ou não, nem sobre o que acontece depois de chegar à ultima paragem. De modo a simular melhor aquilo que é a realidade, considera-se que um percurso efetuado por uma carreira apenas circula em uma direção. Poderá existir um percurso em sentido oposto, mas teria de ser efetuado por uma carreira diferente. Com o mesmo fim, decidiu-se que quando um autocarro chega à ultima paragem, regressa à primeira paragem da sua carreira, efetuando assim um circuito. Apesar de serem apenas pormenores, estas características têm um forte impacto no sistema, mas tornam-no mais semelhante à realidade.

Dada a estrutura que irá ser utilizada para representar os dados, é possível avançar para a implementação de pesquisas sobre o grafo como iremos ver de seguida.

5 Pesquisa Não-Informada

Os algoritmos de pesquisa não-informada são aqueles ao qual não é dada qualquer informação para além da sua definição. No nosso caso, para este tipo de pesquisa, são fornecidas as seguintes informações: **paragem inicial**, **paragem final** e o conjunto de todos os **percursos** entre paragens.

É agora necessário analisar as várias estratégias de procura de modo a saber quais aquelas que mais nos interessam para o problema em questão.

5.1 Pesquisa Depth-First

A pesquisa Depth-First como o nome indica, expande-se sempre para o nó profundo na fronteira atual. A imagem a seguir ilustra o seu funcionamento:

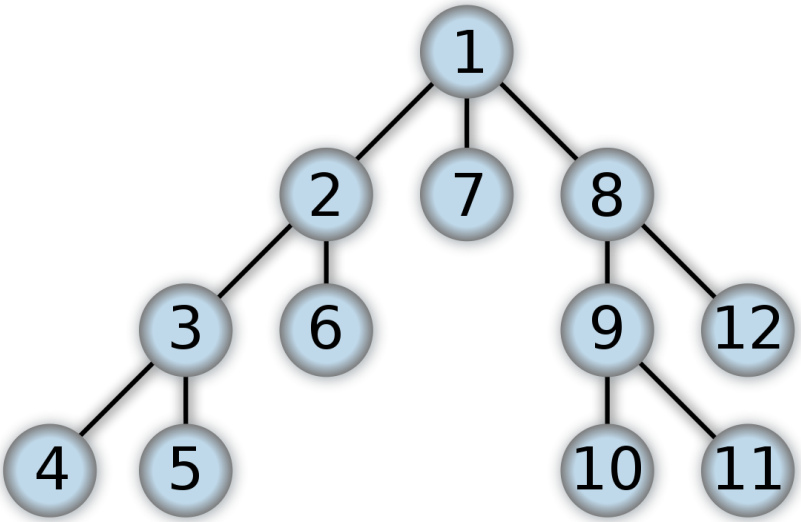

Figura 4: Pesquisa Depth-First *Depth-first search* 2020

Inicialmente, esta procura poderia parecer uma boa opção, no entanto rapidamente percebeu-se que não o era. A pesquisa Depth-First **não é completa nem ótima**, mas tem uma **complexidade espacial linear**. Embora possa ser uma boa opção em termo de memória ocupada, este método falha com o nosso caso. Ao chegar à ultima estação de uma carreira, existe um percurso que liga o fim da carreira ao

início, o que constitui um *loop*. Assim sendo, em diversos casos podemos nunca encontrar uma solução, logo este método não é viável.

5.2 Pesquisa Breadth-First

A pesquisa Breadth-First é uma pesquisa relativamente simples em que o nó raiz é expandido, sendo que de seguida os seus sucessores são expandidos, seguindo-se sempre essa lógica. A imagem a seguir ilustra o seu funcionamento:

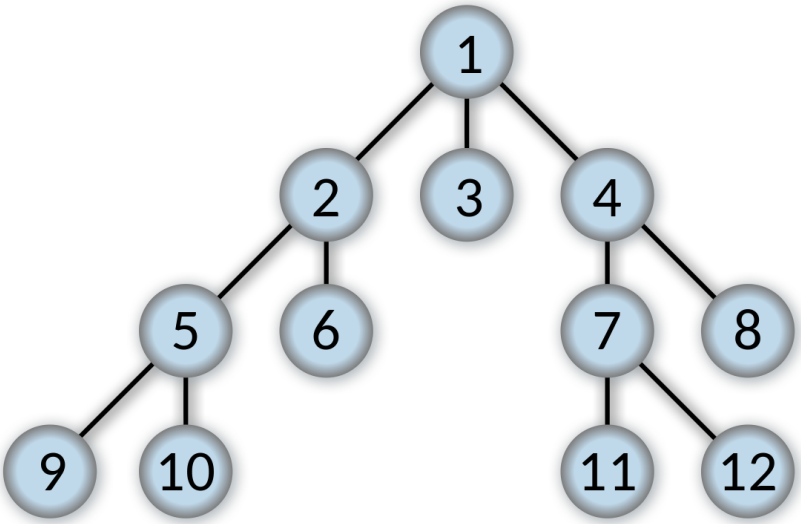

Figura 5: Pesquisa Breadth-First *Breadth-first search* 2020

Este algoritmo é bastante interessante para o nosso caso. Em primeiro lugar garante-nos uma pesquisa bastante sistemática. Em contrapartida, é um algoritmo algo lento e pesado, sendo esse o seu principal problema. O algoritmo Breadth-First é **completo**, tem um custo unitário para cada passo **ótimo**, mas tem uma **complexidade espacial exponencial**. Sendo que esta solução nos permite encontrar as soluções pretendidas, iremos então optar por seguir com esta abordagem sempre que recorrermos a pesquisas não-informadas.

Sabendo agora qual o algoritmo que pretendemos implementar em Prolog, optou-se por adicionar um predicado com a função de "juntar"tanto a pesquisa, como a impressão do resultado, sendo que em casos específicos podem existir alterações

nos parâmetros de modo a ampliar a pesquisa. De seguida segue-se o código base para uma pesquisa breadth-first relativa ao nosso grafo:

5.2.1 Implementação

```
% ---------- Pesquisa Breadth-First ----------
% Efetua a pesquisa breadth-first sem qualquer tipo de restricoes
% Parametros: paragem inicial e paragem final
viagembf(ParagemI,ParagemF) :-
        pesquisabf(ParagemI,ParagemF).

pesquisabf(NodoInicial,NodoFinal) :-
        bfs(NodoInicial,NodoFinal,S),
        duracao(S,D),
        escreve_resultado(S,D).

bfs(X,Y,P) :- bfsr(Y,[n(X,[])],[n(X,[])],R),
        reverse(R,P).

bfsr(Y,[n(Y,P)|_],_,P).

bfsr(Y,[n(S,P1)|Ns],C,P) :- findall(n(S1,[[S,S1,A]|P1]),
        (percurso(S,S1,A,T), \+ member(n(S1,_),C)), Es),
        append(Ns,Es,O),
        append(C,Es,C1),
        bfsr(Y,O,C1,P).
```

6 Pesquisa Informada

Os algoritmos de pesquisa informada são aqueles que podem ter acesso a uma função heurística que estima o custo de uma solução a partir de um determinado nodo. No nosso caso, para este tipo de pesquisa, a função heurística que contém a estimativa do custo de uma solução a partir de um determinado nodo é calculada no **parser**, sendo que sempre que pretendermos alterar o destino será necessário recalcular as estimativas e voltar a importar os dados. Uma das possíveis melhorias futuras passaria por calcular as estimativas dentro do sistema. Essa funcionalidade apenas não foi implementado devido à maior importância da implementação do algoritmo de pesquisa.

6.1 Pesquisa A*

A pesquisa A* (A-Estrela) é o algoritmo de procura **best-first** mais conhecido. Este algoritmo avalia os nós ao combinar g(n), o custo para alcançar o nodo, e h(n), a estimativa do custo de chegar do nodo à solução. Dado isto, obtemo f(n), que é o custo da solução mais barata que passa por n:

$$f(n) = g(n) + h(n)$$

O algoritmo A* é tanto **completo** como **ótimo**, desde que a função eurística h(n) seja consistente. Isto evidencia a importância do papel que os dados têm no nosso sistema. No entanto, o principal problema desta pesquisa encontra-se na **complexidade que o algoritmo apresenta relativamente ao espaço**, que o torna algo proibitivo. Devido a esta complexida espacial, foi necessário procurar um algoritmo que nos permita contornar o problema da complexidade espacial. No entanto implementou-se ainda assim a pesquisa a-estrela devido ao facto de ser uma pesquisa muito boa para o problema em questão:

6.1.1 Implementação

```
% ---------- Pesquisa A-Estrela ----------
pesquisaae(NodoInicial,NodoFinal) :-
    evolucao(destino(NodoFinal)),
    resolve_aestrela(NodoInicial,S,C),
    involucao(destino(NodoFinal)),
    write(S), nl, write(C).

resolve_aestrela(Nodo,Caminho,Custo) :-
    estimativa(Nodo, Estima),
    aestrela([[Nodo]/0/Estima], InvCaminho/Custo/_),
    inverso(InvCaminho, Caminho).

aestrela(Caminhos, Caminho) :-
    obtem_melhor(Caminhos, Caminho),
    Caminho = [Nodo|_]/_/_,
    destino(Nodo).

aestrela(Caminhos, SolucaoCaminho) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela(MelhorCaminho, ExpCaminhos),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela(NovoCaminhos, SolucaoCaminho).

obtem_melhor([Caminho], Caminho) :- !.

obtem_melhor([Caminho1/Custo1/Est1,
_/Custo2/Est2|Caminhos],MelhorCaminho) :-
    Custo1 + Est1 =< Custo2 + Est2, !,
    obtem_melhor([Caminho1/Custo1/Est1|Caminhos],
    MelhorCaminho).
```

```
obtem_melhor([_|Caminhos], MelhorCaminho) :-
    obtem_melhor(Caminhos, MelhorCaminho).

expande_aestrela(Caminho, ExpCaminhos) :-
    findall(NovoCaminho,
    adjacente(Caminho,NovoCaminho),
    ExpCaminhos).

adjacente([Nodo|Caminho]/Custo/_,
[ProxNodo,Nodo|Caminho]/NovoCusto/Est) :-
    percurso(Nodo, ProxNodo, Carreira, PassoCusto),
    \+ member(ProxNodo, Caminho),
    NovoCusto is Custo + PassoCusto,
    estimativa(ProxNodo, Est).
```

6.2 Pesquisa RBFS

O algoritmo de pesquisa RBFS (Recursive best-first search) é um algoritmo relativamente simples. A sua estrutura é algo semelhante a uma pesquisa dept-firsr recursiva, mas em vez de continuar indefinidamente por um caminho, é usado um valor **f-limit** para poder controlar o **f-value** do melhor caminho alternativo a partir de qualquer antecessor do nodo atual. Se o nodo atual exceder este limite, retrocede-se até ao caminho alternativo. Consoante se vai retrocedendo, o algoritmo substitui o f-value de cada nodo com um valor de reserva (o melhor f-value dos seus filhos). Deste modo, o algoritmo "sabe"sempre qual o f-value do melhor filho. A pesquisa RBFS é **robusta**, **ótima** e usa um **valor limitado de memória**. Embora a pesquisa A* consiga seja uma pesquisa mais eficiente em certos aspetos, a pesquisa RBFS consegue resolver os casos em que a pesquisa A* fica limitado por causa da memória, dado o tempo suficiente. Assim sendo, segue-se a implementação da pesquisa RBFS:

6.2.1 Implementação

```
% ---------- Pesquisa RBFS ----------

pesquisarbfs(NodoInicial,NodoFinal) :-
        evolucao(destino(NodoFinal)),
        rbfs([],[(NodoInicial, 0/0/0)],99999,_,yes,S),
        involucao(destino(NodoFinal)),
        reverse(S,Solucao),
        duracao(Solucao,D),
        escreve_resultado(Solucao,D).

rbfs(Caminho,[(Nodo, G/F/FF)|Nodos],Limite,FF,no,_) :-
        FF > Limite, !.

rbfs(Caminho,[(Nodo, G/F/FF)|_],_,_,yes,[Nodo|Caminho]) :-
        F = FF,
        destino(Nodo).

rbfs(_,[],_,_,never,_) :- !.

rbfs(Caminho,[(Nodo,G/F/FF)|Nodos],Limite,NovoFF,S,Solucao) :-
        FF =< Limite,
        findall(Filho/Custo, (percurso(Nodo,Filho,Carreira,Custo),
        \+ member(Filho,Caminho)),Filhos),
        herda(F,FF,FFHerdado),
        succlist(G,FFHerdado,Filhos,NodosSuc),
        bestff(Nodos,ProxMelhorFF),
        min(Limite,ProxMelhorFF,NLimite),!,
        rbfs([Nodo|Caminho],NodosSuc,NLimite,NovoFF2,S2,Solucao),
        continua(Caminho,[(Nodo, G/F/NovoFF2)|Nodos],
        Limite,NovoFF,S2,S,Solucao).
```

```
continua(Caminho,[N|Ns],Limite,NovoFF,never,S,Solucao) :- !,
        rbfs(Caminho,Ns,Limite,NovoFF,S,Solucao).

continua(_,_,_,_,yes,yes,Solucao).

continua(Caminho,[N|Ns],Limite,NovoFF,no,S,Solucao) :-
        insert(N,Ns,NovoNs), !,
        rbfs(Caminho,NovoNs,Limite,NovoFF,S,Solucao).

succlist(_,_,[],[]).

succlist(G0,FFHerdado,[Nodo/C|NCs],Nodos) :-
        G is G0 + C,
        estimativa(Nodo,H),
        F is G + H,
        max(F,FFHerdado,FF),
        succlist(G0,FFHerdado,NCs,Nodos2),
        insert((Nodo,G/F/FF),Nodos2,Nodos).

herda(F,FF,FF) :- FF > F,!.
herda(F,FF,0).

insert((N,G/F/FF),Nodos,[(N,G/F/FF)|Nodos]) :-
       bestff(Nodos,FF2),
       FF =< FF2, !.

insert(N,[N1|Ns],[N1|Ns1]) :-
       insert(N,Ns,Ns1).

bestff([(N,F/G/FF)|Ns],FF).
bestff([],99999).
```

7 Comparação dos Algoritmos de Pesquisa

Após vermos alguns dos algoritmos de pesquisa apropriados ao problema em questão, e de analisarmos quais são os mais eficientes para o problema em questão, podemos efetuar algumas comparações de modo a perceber como se comparam entre si. Para esse efeito, construiu-se a seguinte tabela:

Algoritmo	Completo	Ótimo	Complexidade Tempo	Complexidade Espaço
Depth-First	Não	Não	$O(b^m)$	$O(b^m)$
Breadth-First	Sim	Sim	$O(b^d)$	$O(b^d)$
A*	Sim	Sim	Número nodo com g(h)+h(n) $\leq C*$	
RBFS	Robusto	Sim	$O(b^d)$	O(bd)

Relativamente a esta tabela, m representa a profundidade máxima da àrvore de procura, d a profundidade da solução, e b o fator de ramificação.

8 Requisitos do Sistema

Como parte do enunciado, foi proposto um conjunto de requisitos que o sistema deveria permitir responder. De seguida irão ser enumerados esses requisitos bem como algumas funcionalidades suplementares incluídas.

Calcular um trajeto entre dois pontos

Para resolver este requisito aplicamos o algoritmo Breadth-First, dado que este nos retorna o percurso com o menor número de adjacências, cumprindo deste modo aquilo que foi considerado inicialmente o nosso critério ideal de procura de caminhos. Assim sendo para obter este requisito basta correr o seguinte comando:

```
viagembf(Paragem Inicial, Paragem Final).
```

Sendo que o resultado final é algo como o seguinte exemplo:

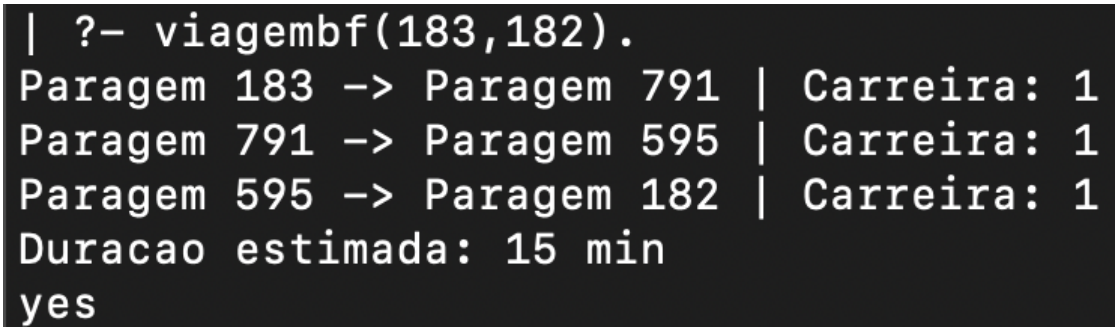

Figura 6: Exemplo do predicado viagembf

Selecionar apenas algumas das operadoras de transporte para um determinado percurso

Para cumprir este requisito utilizou-se na mesma o predicado viagembf, no entanto, acrescentaram-se dois parâmetros: o primeiro, que pode tomar o valor 'y' ou 'n', e o segundo, que é uma lista de operadoras. O valor 'y' serve para incluir as operadoras da lista na pesquisa, e o valor 'n' para excluir as operadoras da lista. Para isso foi feita uma adaptação no predicado bfsr, sendo que quando efetuamos o findall, basta adicionar 2 restrições:

```
bfsr(Y,[n(S,P1)|Ns],C,'y',Ops,P) :-
    findall(n(S1,[[S,S1,A]|P1]),
    (percurso(S,S1,A,T), operadoras(S,Ops),
    operadoras(S1,Ops),\+ member(n(S1,_),C)),
    Es),append(Ns,Es,O),
    append(C,Es,C1),
    bfsr(Y,O,C1,'y',Ops,P).
```

Deste modo, ao invocar o predicado viagembf com os 2 parâmetros novos, é aplicada a restrição à pesquisa.

```
viagembf(Paragem Inicial, Paragem Final,'y',Lista Operadoras).
```

Excluir um ou mais operadores de transporte para o percurso

Este requisito segue o mesmo funcionamento do anterior, necessitando apenas do valor 'n' em vez do valor 'y'. O predicado bfsr apenas sofre uma ligeira alteração, que passa por aplicar o predicado não:

```
bfsr(Y,[n(S,P1)|Ns],C,'y',Ops,P) :-
    findall(n(S1,[[S,S1,A]|P1]),
    (percurso(S,S1,A,T), nao(operadoras(S,Ops)),
    nao(operadoras(S1,Ops)),\+ member(n(S1,_),C)), Es),
    append(Ns,Es,O),
    append(C,Es,C1),
    bfsr(Y,O,C1,'y',Ops,P).
```

Sendo que para efetuar a pesquisa excluindo operadoras basta correr o seguinte predicado:

```
viagembf(Paragem Inicial, Paragem Final,'n',Lista Operadoras).
```

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

Para resolver este requisito utilizamos a mesma pesquisa que temos vindo a utilizar até agora. Apenas necessitamos de uma variável condição com o valor 'maiscar-

reiras', sendo que deste modo, no final da pesquisa, corremos um predicado **mais-carreiras** que recolhe todas as paragens do percurso e conta as carreiras que por lá passam, imprimindo o resultado no final. Para poder ver este resultado basta invocar o seguinte predicado:

```
viagembf(Paragem Inicial, Paragem Final, 'maiscarreiras').
```

Sendo que o resultado de correr o comando é o seguinte:

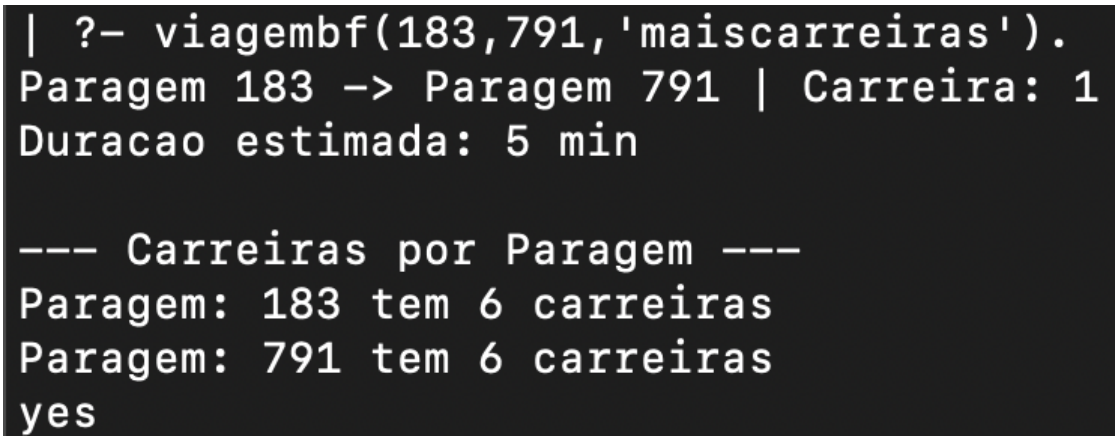

Figura 7: Exemplo do predicado viagembf exibindo as carreiras por paragem.

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

O facto de recorrermos à pesquisa Breadth-First garante-nos que o percurso encontrado entre duas paragens tem sempre o menor número de paragens. Assim sendo o predicado viagembf cumpre este requisito, sendo que é considerado o nosso critério ótimo.

Escolher o percurso mais rápido (usando critério da distância)

Para conseguir obter este resultado, a pesquisa Breadth-First não nos consegue garantir que a distância seja a menor, visto que as distâncias não são tidas em conta durante a procura. Para isso, necessitámos de recorrer a pesquisas informadas, sendo que a pesquisa A* e a pesquisa RBFS nos garantem o percurso mais curto entre 2 paragens. Dependendo da nossa preferência em termos de eficiência, como vimos anteriormente, podemos optar por um dos algoritmos.

Escolher o percurso que passe apenas por abrigos com publicidade

Para cumprir este requisito, recorremos também a uma variável condição, sendo que o seu funcionamento é semelhante ao requisito de incluir ou excluir operadoras. A diferença passa por ter o predicado **publicidade** no findall, sendo que este nos informa se a paragem em questão é ou não abrigada. Para isso, utilizamos o predicado viagembf do seguinte modo:

```
viagembf(Paragem Inicial, Paragem Final,'publicidade').
```

Escolher o percurso que passe apenas por paragens abrigadas

Para conseguir cumprir este requisito seguimos exatamente o mesmo método do requisito anterior, no entanto, no findall, utilizamos os predicado **abrigado**, que nos informa se uma paragem é ou não abrigada. Uma paragem não é abrigada em todos os casos que não seja fechada dos lados ou aberta dos lados. O predicado usado para este efeito é o seguinte:

```
viagembf(Paragem Inicial, Paragem Final,'abrigado').
```

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

Para conseguir efetuar este tipo de percurso, fornecemos uma lista com todas as paragens por onde queremos passar. O percurso é feito calculando todos os percursos entre cada 2 paragens e retornando todos esse percursos. O predicado viagem passa então a ter o seguinte aspeto:

```
viagembf([Lista Paragens]).
```

O resultado final terá o seguinte aspeto:

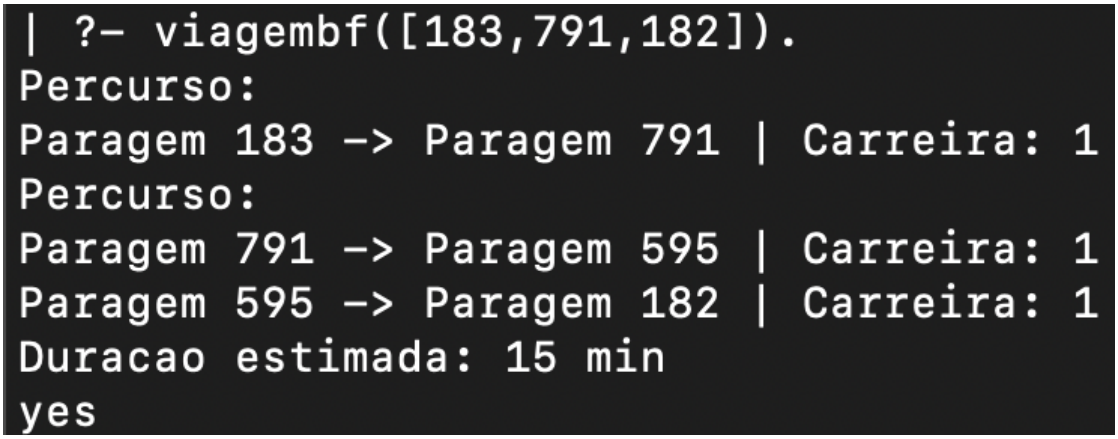

Figura 8: Exemplo do predicado viagembf com uma paragem intermédia.

8.1 Extras

De modo a completar o sistema e adicionar algumas funcionalidades relevantes, foram adicionadas alguns requisitos extra.

A primeira funcionalidade extra que se implementou foi a estimativa do tempo que poderá demorar, sendo que consideramos que cada viagem entre 2 paragens consecutivas demora aproximadamente 5 minutos. Apenas se considerou esta abordagem pois a distância euclidiana não pareceu muito correta num contexto urbano.

De seguida também se acrescentou a possibilidade de procurar percursos sem abrigo e sem publicidade. Esses 2 predicados extra foram relativamente simples de adicionar dado que são iguais aos seus inversos, bastando considerar o predicado não dentro do findall.

Por sua vez, apenas para a pesquisa informada, implentaram-se predicados para tornar o output da pesquisa legível e de fácil compreensão.

O último extra feito foi relativo à pesquisa informada, sendo que se implementaram dois algoritmos de pesquisa: o A* e o RBFS. Esta decisão foi tomada com o simples intuito de estudar algoritmos que contrastam entre si, continuando sempre com o mesmo objetivo.

9 Conclusões e Trabalho Futuro

Com este trabalho foi possível aprender um pouco sobre grafos bem como algoritmos de procura sobre grafos. Também ficámos a ter uma noção do impacto que a eficiência de cada algoritmo tem sobre a rapidez e eficácia da procura. Outro aspeto que serviu para efetuar algumas reflexões foi o aspeto dos dados e as estruturas. Com este trabalho ficámos com uma ideia do impacto que a qualidade dos nossos dados têm na solução final bem como o impacto que a decisão da nossa estrutura de dados têm. Para trabalho futuro, a principal proposta passa por melhorar a nossa estrutura de dados. Algo que se chegou à conclusão ser uma má decisão foi ter um arco entre dua paragens para cada carreira. Isso gera alguma confusão e complexidade desnecessária no processo de decisão. A melhoria passaria por ter uma lista de paragens, em vez de uma paragem, no predicado percurso. Além disso os algoritmos também necessitam por sua vez de algumas afinações ligeiras. Outra melhoria que seria interessante realizar, seria alterar o calculo das estimativas. De momento temos que reimportar todos os dados sempre que quisermos alterar o destino na pesquisa informada. Seria interessante e prático conseguir fazer isso durante a execução do programa.

Referências

Breadth-first search (2020). URL: https://en.wikipedia.org/wiki/Breadth-first_search (acedido em 01/06/2020).

Depth-first search (2020). URL: https://en.wikipedia.org/wiki/Depth-first_search (acedido em 01/06/2020).