Relatório de ALGAV

Turma 3DL _ Grupo 67

1191430 - Fábio Fernandes

1190447 - Bruno Silva

1210308 - Alejandro Jorge

1190633 - Gonçalo Jordão

1190523 - Diogo Domingues

Data: 28/11/2021

Índice

1. Criação de uma rede à parte com os utilizadores que podem ser alcançados até N ligações a partir de um dado utilizador

```
no(1,ana,[natureza,pintura,musica,sw,porto]).

no(11,antonio,[natureza,pintura,carros,futebol,lisboa]).
no(12,beatriz,[natureza,musica,carros,porto,moda]).
no(13,carlos,[natureza,musica,sw,futebol,coimbra]).
no(14,daniel,[natureza,cinema,jogos,sw,moda]).
no(15,afonso,[natureza,pintura,carros,futebol,lisboa]).
no(16,artur,[natureza,musica,carros,porto,moda]).

no(21,eduardo,[natureza,cinema,teatro,carros,coimbra]).
no(22,isabel,[natureza,musica,porto,lisboa,cinema]).
no(23,jose,[natureza,pintura,sw,musica,carros,lisboa]).
no(24,luisa,[natureza,cinema,jogos,moda,porto]).
no(25,bento,[natureza,pintura,carros,futebol,lisboa]).
no(26,benjamim,[natureza,musica,carros,porto,moda]).

no(31,maria,[natureza,pintura,musica,moda,porto]).
no(32,anabela,[natureza,cinema,musica,tecnologia,porto]).
no(33,andre,[natureza,carros,futebol,coimbra]).
no(34,catia,[natureza,musica,cinema,lisboa,moda]).
no(35,ethan,[natureza,pintura,carros,futebol,lisboa]).
no(36,erik,[natureza,musica,carros,porto,moda]).

no(41,cesar,[natureza,teatro,tecnologia,futebol,porto]).
no(42,diogo,[natureza,futebol,sw,jogos,porto]).
no(43,ernesto,[natureza,teatro,carros,porto]).
no(44,isaura,[natureza,moda,tecnologia,cinema]).
no(45,fabricio,[natureza,pintura,carros,futebol,lisboa]).
no(46,gregorio,[natureza,musica,carros,porto,moda]).

no(51,rodolfo,[natureza,musica,sw]).
no(52,aurora,[natureza,futebol,sw,jogos,porto]).
no(53,anastacia,[natureza,teatro,carros,porto]).
no(54,bianca,[natureza,moda,tecnologia,cinema]).
no(55,cecilia,[natureza,pintura,carros,futebol,lisboa]).
no(56,flora,[natureza,musica,carros,porto,moda]).

%no(61,rita,[moda,tecnologia,cinema]).

%no(200,sara,[natureza,moda,musica,sw,coimbra]).
```

```
ligacao(1,11,10,8,12,4).
ligacao(1,12,2,6,4,5).
ligacao(1,13,-3,-2,1,-3).
ligacao(1,17,24,8,9,4).
ligacao(1,14,1,-5,1,4).
ligacao(11,21,5,7,2,-5).
ligacao(11,22,2,-4,1,4).
ligacao(11,23,-2,8,-5,3).
ligacao(11,24,6,0,4,-5).
ligacao(12,21,4,9,4,5).
ligacao(12,22,-3,-8,1,7).
ligacao(12,23,2,4,3,-6).
ligacao(12,24,-2,4,4,1).
ligacao(13,21,3,2,4,-6).
ligacao(13,22,0,-3,2,-6).
ligacao(13,23,5,9,4,-8).
ligacao(13,24,-2, 4,2,4).
ligacao(14,21,2,6,5,7).
ligacao(14,22,6,-3,2,-5).
ligacao(14,23,7,0,2,8).
ligacao(14,24,2,2,2,5).
ligacao(17,21,8,6,6,9).
ligacao(17,23,2,2,4,5).
ligacao(17,22,20,2,4,5).
ligacao(17,24,17,2,5,4).
ligacao(21,31,2,1,-4,5).
ligacao(21,32,-2,3,6,2).
ligacao(21,34,4,2,1,-5).
ligacao(21,33,3,5,5,2).
ligacao(22,31,5,-4,3,5).
ligacao(22,32,-1,6,3,-5).
ligacao(22,33,2,1,4,6).
ligacao(22,34,2,3,2,-5).
ligacao(23,31,4,-3,4,6).
ligacao(23,32,3,5,2,1).
ligacao(23,33,4,1,2,4).
ligacao(23,34,-2,-3,2,4).
ligacao(24,31,1,-5,2,-4).
ligacao(24,32,1,0,2,4).
ligacao(24,33,3,-1,2,1).
ligacao(24,34,-1,5,1,6).
ligacao(31,41,2,4,4,2).
ligacao(31,42,6,3,3,8).
ligacao(31,43,2,1,-1,3).
ligacao(31,44,2,1,1,6).
ligacao(32,41,2,3,-7,2).
ligacao(32,42,-1,0,2,-4).
```

```
ligacao(32,43,0,1,1,4).
ligacao(32,44,1,2,1,1).
ligacao(33,41,4,-1,6,4).
ligacao(33,42,-1,3,4,-4).
ligacao(33,43,7,2,2,6).
ligacao(33,44,5,-3,-5,3).
ligacao(34,41,3,2,4,6).
ligacao(34,42,1,-1,2,-4).
ligacao(34,43,2,4,4,-5).
```

2. Adaptação do A* ao problema da determinação do caminho mais forte (máximo de N ligações).

Perante o algoritmo fornecido houve algumas alterações, respetivamente na implementação de N, para o controlo das ligações necessárias para construir o caminho pretendido, limitando o número de ligações. Neste caso N será o valor inicial que será introduzido pelo utilizador para restringir até que número máximo de ligações serão necessárias para construir o caminho entre 2 utilizadores. E não só, foram adicionados alguns predicados que explicarem em detalhe de seguida.

```
aStar(Orig,Dest,Cam,Custo,N):-
   forcas_ligacao_decrescente(Orig,N,RedeUtilizadores),
   aStar2(Dest,[(_,0,[Orig])],RedeUtilizadores,Cam,Custo,N),!.
```

A primeira alteração ao algoritmo A* foi a inserção de um predicado chamado forcas_ligacao_decrescente.

```
forcas_ligacao_decrescente(Orig,N,Lista):-
      tamanho_rede_utilizadores(Orig,N,RedeUtilizadores),
      forcas_ligacao_decrescente_2(RedeUtilizadores,ListaInt),
      flatten(ListaInt,ListaInt2),
      sort(0,@>=,ListaInt2,Lista).

forcas_ligacao_decrescente_2([_],[]):-!.
forcas_ligacao_decrescente_2([User1|Y],[RedeUtilizadores|Z]):-
      findall([F1,F2],
      (member(User2,Y),
      (ligacao(User1,User2,F1,F2,_,_);ligacao(User2,User1,F2,F1,_,_))),RedeUtilizadores),
      forcas_ligacao_decrescente_2(Y,Z).
```

Com o algoritmo que aqui fica descrito, acedemos a um predicado tamanho_rede_utilizadores, que foi criado no Sprint B, com uma pequena alteração, agora não dá o número de utilizadores por nível, mas sim a lista de utilizadores até determinado nível. Com a lista sabemos quantos utilizadores podem estar intervenientes até determinado nível.

Com o acesso a uma lista de todos os utilizadores intervenientes até determinado nível, nós podemos agora determinar quais são as forças que todos os utilizadores têm com as suas ligações diretas. Isto é o propósito do predicado forcas_ligacao_decrescente. Que quando recebe depois todas as forças de ligação daquela rede de utilizadores, vai fazer o *flatten*, de maneira a evitar ligações repetidas entre utilizadores (porque vão ser consideradas as forças em ambos os sentidos da ligação) e recorre ao *sort* para as ordenar por ordem decrescente, lista que vai ser mais tarde utilizada no cálculo da estimativa.

3. Estimativa Implementada

```
estimativa(_,Dest,Dest,_,_,_,0):- !.
estimativa(ForcaX,_,_,Atual,Destino,ListaInt,Estimativa):-
        Index is Destino - Atual,
        apaga_primeiro(ForcaX,ListaInt,Resultado),
        somar_Forca_N(0,Index,Resultado,Estimativa).
```

Para a estimativa pedida para o algoritmo A* uma alternativa que nós chegamos engloba o uso do de 2 novos predicados que vão se limitar a apagar e a somar os valores que estão a ser chegados da lista que anteriormente foi concebida.

A estimativa irá acabar com o primeiro nó origem chegar ao nó destino, e enquanto ele processa a recursividade, ele tem o ato de ir verificar primeiro o Index, que é o resultante da subtração entre nível do destino e o nível atual, com isto, ele irá a lista que recebe do predicada anterior (contem as forças de ligação que vão ser utilizadas para formar o caminho), retira a força que necessita (utiliza o predicado apaga_primeiro) ser adicionada e adiciona (usando o predicado somar_força_N, pois esta força é referente ao nível em que ele está).

```
somar_Forca_N(N,N,_,0):-!.
somar_Forca_N(C,N,[H|T],S):-
    C1 is C+1,
    somar_Forca_N(C1,N,T,S1),
    S is H + S1.
```

Como se pode ver no algoritmo demonstrado, a soma de todas as forças irá parar só quando o valor de C for igual ao valor de N, ou seja, no momento em que o valor 0 chegar ao valor do Index, todas as somas foram feitas e o valor a apresentar pode ser enviado.

No caso da estimativa ser utilizada com a função multicritério, atuará da mesma forma no entanto a força que lhe será chegada não será apenas a de ligação, mas sim uma junção da força de ligação com a força de relação, para ambas terem o mesmo peso de contribuição.

4. Adaptação do Best First ao problema da determinação do caminho mais forte (máximo de N ligações)

Algoritmo Best First (Sem Função Multicritério):

```
bestfs1(Orig,Dest,Cam,Custo,N):-
        bestfs12(Dest,[[Orig]],Cam,Custo,N),
        write('Caminho='),write(Cam),nl.

bestfs12(Dest,[[Dest|T]|_],Cam,Custo,_):-
        reverse([Dest|T],Cam), calcula_custo(Cam,Custo).

bestfs12(Dest,[[Dest|_]|LLA2],Cam,Custo,N):-
        !,bestfs12(Dest,LLA2,Cam,Custo,N).

bestfs12(Dest,LLA,Cam,Custo,N):-
        member1(LA,LLA,LLA1),LA=[Act|_],
        length(LA,Tamanho),
        Tamanho =< N,
        ((Act==Dest,!,bestfs12(Dest,[LA|LLA1],Cam,Custo)) ; (findall((CX,[X|LA]),
        (no(NAct,Act,_),no(NX,X,_),(ligacao(NAct,NX,FL,_,FR,_);ligacao(NX,NAct,_,FL,_,FR)),CX is FL+FR,
        \+member(X,LA)),Novos),
        Novos\==[],!,
        sort(0,@>=,Novos,NovosOrd),
        retira_custos(NovosOrd,NovosOrd1),
        append(NovosOrd1,LLA1,LLA2),
        %write('LLA2='),write(LLA2),nl,
        bestfs12(Dest,LLA2,Cam,Custo))).
```

```
member1(LA,[LA|LAA],LAA).
member1(LA,[_|LAA],LAA1):- member1(LA,LAA,LAA1).

retira_custos([],[]).
retira_custos([(_,LA)|L],[LA|L1]):- retira_custos(L,L1).

calcula_custo([Act,X],C):- !, no(NAct,Act,_), no(NX,X,_), (ligacao(NAct,NX,S,_,_,_) ;
ligacao(NX,NAct,_,S,_,_)), C is S.
calcula_custo([Act,X|L],P):-
        calcula_custo([X|L],P1),
        no(NAct,Act,_), no(NX,X,_),
        (ligacao(NAct,NX,S,_,_,_) ; ligacao(NX,NAct,_,S,_,_)),
        P is P1 + S.
```

Explicação Geral:

Neste algoritmo usou-se a lógica de ir para o vizinho pelo qual se percorre a maior distância (escolher a ligação que podemos seguir que tem maior CX).

É uma implementação do Best First com retrocesso e no limite pode dar todas as soluções (apenas para vermos o valor das diversas soluções). Não garante que a melhor solução seja encontrada em primeiro lugar, apenas garante que a primeira solução é encontrada rapidamente. No 2º argumento de bestfs12 está uma lista de listas com os vários caminhos em construção.

Explicação dos Predicados:

```
bestfs12(Dest,[[Dest|T]|_],Cam,Custo,_):- reverse([Dest|T],Cam), calcula_custo(Cam,Custo).
```

Quando termina e encontra uma solução.

```
bestfs12(Dest,[[Dest|_]|LLA2],Cam,Custo,N):- !, bestfs12(Dest,LLA2,Cam,Custo,N).
```

Necessário para poder gerar mais soluções depois de encontrar uma.

```
bestfs12(Dest,LLA,Cam,Custo,N):- member1(LA,LLA,LLA1), LA=[Act|_],
length(LA,Tamanho),Tamanho =< N,
((Act==Dest,!,bestfs12(Dest,[LA|LLA1],Cam,Custo)) ;
 ( findall((CX,[X|LA]),(edge(Act,X,CX), \+member(X,LA)),Novos), Novos\==[],!,
sort(0,@>=,Novos,NovosOrd), retira_custos(NovosOrd,NovosOrd1), append(NovosOrd1,LLA1,LLA2),
%write('LLA2='),write(LLA2),nl, bestfs12(Dest,LLA2,Cam,Custo) )).
```

As variáveis N e Tamanho servirão para verificar se a lista já chegou ou não ao limite de número de ligações.
É necessário verificar se Act é igual a Dest (por causa do ! na regra anterior), uma vez que devido ao funcionamento do member1 se não houver solução com a lista à cabeça de LLA iremos passar à lista seguinte, que pode ter a particularidade de começar pelo destino Dest. É importante referir que, ao gerar Novos no findall pode aparecer um caminho que vá para o destino mas o seu CX pode não ser o maior.

Em relação ao "!", só faz essa chamada para poder ir para a 1ª regra do bestfs pois já chegamos a uma solução.

Em relação a "Novos\==[],!,", se Novos==[] é porque chegamos a um "beco sem saída", se isso acontecer deve falhar e em member1 é escolhida nova lista.

Em relação a "sort(0,@>=,Novos,NovosOrd),", esta chamada serve para ordenar em ordem decrescente, o que define são os valores de CX nos pares (CX,[X|LA]).

```
member1(LA,[LA|LAA],LAA). member1(LA,[_|LAA],LAA1):-member1(LA,LAA,LAA1).
```

member1/3 tem no 2º argumento uma lista de listas e coloca no primeiro argumento uma dessas listas e no 3º argumento as listas depois da colocada no primeiro argumento. A finalidade do seu uso é permitir que no retrocesso do Prolog se possa optar por um novo caminho para encontrar uma nova solução. É importante referir que a primeira lista da lista de listas até pode levar a que não se encontre uma solução.

```
retira_custos([],[]). retira_custos([(_,LA)|L],[LA|L1]):-retira_custos(L,L1).
```

O uso dos valores de CX só é necessário para ordenar os caminhos a partir de um dado nó, através do findall que cria uma lista com termos do tipo (CX,[X|LA]), mas a nossa lista de listas com os caminhos alternativos não envolve custos pelo que há que removê-los de NovosOrd antes de juntar essa lista com LLA1.

```
calcula_custo([Act,X],C):- !, no(NAct,Act,_), no(NX,X,_), (ligacao(NAct,NX,S,_,_,_) ; ligacao(NX,NAct,_,S,_,_)), C is S.
calcula_custo([Act,X|L],P):-
        calcula_custo([X|L],P1),
        no(NAct,Act,_), no(NX,X,_),
        (ligacao(NAct,NX,S,_,_,_) ; ligacao(NX,NAct,_,S,_,_)),
        P is P1 + S.
```

Neste predicado, é efetuada a soma das forças de ligação e apenas estas, uma vez que esta se trata da implementação sem recorrer à função multicritério.

A versão aqui explicada obtém várias soluções. No nosso ficheiro ALGAV.pl temos presente uma versão deste algoritmo obtendo uma única solução.

5. Adaptação do Primeiro em Profundidade para gerar a melhor solução (já implementado no Sprint anterior) para o máximo de N ligações

Depth First com Função multicritério com força de ligaçao

```
all_dfs(Nome1,Nome2,Cam):-
   get_time(T1),
   dfs(Nome1,Nome2,Cam),!,
   length(Cam,NLCam),
   get_time(T2),
   write(NLCam),
   write('solutions found in'),
   T is T2-T1,write(T),write('seconds'),nl, write('List of all the paths: '),
   write(Cam),nl,nl.

dfs(Orig,Dest,Cam):-dfs2(Orig,Dest,[Orig],Cam).
dfs2(Dest,Dest,LA,Cam):-!,reverse(LA,Cam).
dfs2(Act,Dest,LA,Cam):-
    no(NAct,Act,_),
  (ligacao(NAct,NX,_,_,_,_);ligacao(NX,NAct,_,_,_,_)),
    no(NX,X,_),\+ member(X,LA),dfs2(X,Dest,[X|LA],Cam).

compute_value(U1,U2,V):-!,
    ligacao(U1,U2,ForcaU1_U2,ForcaU2_U1,Likes,Deslikes),
  F is ForcaU1_U2 + ForcaU2_U1,
  LD is Likes - Deslikes,
  V is F * 1 + LD * 0.5,
  multicriteria_function(F,LD,V).
```

Explicação Geral:

Neste algoritmo usou-se a lógica de ir para o vizinho hasta chegar abajo

Não garante que a melhor solução seja encontrada em primeiro lugar, apenas garante que a primeira solução é encontrada rapidamente.

Explicação dos Predicados:

```
all_dfs(Nome1,Nome2,LCam):-
all_dfs(Nome1,Nome2,Cam):-
   get_time(T1),
   dfs(Nome1,Nome2,Cam),!,
   length(Cam,NLCam),
   get_time(T2),
   write(NLCam),
   write('solutions found in'),
   T is T2-T1,write(T),write('seconds'),nl, write('List of all the paths: '),
   write(Cam),nl,nl.

dfs(Orig,Dest,Cam):-dfs2(Orig,Dest,[Orig],Cam).
dfs2(Dest,Dest,LA,Cam):-!,reverse(LA,Cam).

dfs2(Act,Dest,LA,Cam):-
   no(NAct,Act,_),
  (ligacao(NAct,NX,_,_,_,_);ligacao(NX,NAct,_,_,_,_)),
   no(NX,X,_),\+ member(X,LA),dfs2(X,Dest,[X|LA],Cam).
```

Aquí implementamos o algoritmo Depth First donde recorremos um camino hasta o final e volvemos hacia arriba.

```
compute_value(U1,U2,V):-
   ligacao(U1,U2,ForcaU1_U2,ForcaU2_U1,Likes,Deslikes),
  F is ForcaU1_U2 + ForcaU2_U1,
  LD is Likes - Deslikes,
  V is F * 1 + LD * 0.5,
  multicriteria_function(F,LD,V).
```

Aquí implementamos a força de ligaçao com os likes criando uma escala para uma de as formas de a funçao multicriterio e asi obtemos a solução.

Depth First com outra Função multicriterio

```
all_dfs(Nome1,Nome2,LCam):-
   get_time(T1),
   findall(Cam,dfs(Nome1,Nome2,Cam),LCam),
   length(LCam,NLCam),
   get_time(T2),
   write(NLCam),
   write('solutions found in'),
   T is T2-T1,write(T),write('seconds'),nl, write('List of all the paths: '),
   write(LCam),nl,nl.

dfs(Orig,Dest,Cam):-dfs2(Orig,Dest,[Orig],Cam).
dfs2(Dest,Dest,LA,Cam):-!,reverse(LA,Cam).

dfs2(Act,Dest,LA,Cam):-
   no(NAct,Act,_),
  (ligacao(NAct,NX,_,_,_,_);ligacao(NX,NAct,_,_,_,_)),
   no(NX,X,_),\+ member(X,LA),dfs2(X,Dest,[X|LA],Cam).
```

Depth First com força de ligacao

```
compute_value(U1,U2,V):-
   ligacao(U1,U2,ForcaU1_U2,ForcaU2_U1),
  F is ForcaU1_U2 + ForcaU2_U1.
```

6. Comparação dos 3 métodos com vários exemplos, comparando tempos de geração da solução e valor da solução gerada

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo de geração – Best First sem Função Multicritério (Sprint C)	Tempo de geração (Sprint B)
1	1	0.0000 s	0.0000 s
2	2	0.0069 s	0.0000 s
3	3	0.0010 s	0.0139 s
4	4	0.0079 s	312.5943 s

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo de geração – A* com Função Multicritério (Sprint C)	Tempo de geração (Sprint B)
1	1	0.0009 s	0.0000 s
2	2	0.0010 s	0.0000 s
3	3	0.0020 s	0.0139 s
4	4	0.0010 s	312.5943 s

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo de geração – Depth First sem Função Multicritério (Sprint C)	Tempo de geração (Sprint B)
1	1	0.0000 s	0.0000 s
2	2	0.0080 s	0.0000 s
3	3	0.0000 s	0.0139 s
4	4	0.0000 s	312.5943 s

7. Implementação da função multicritério que contemple forças de ligação e diferença entre likes e dislikes

```
multicriterio(FL, FR, R):-
   ((FL < 50, with_FL_0(FL,FR,R));
   (FL < 100, with_FL_50(FL,FR,R));
   (FL = 100, with_FL_100(FL,FR,R))),!.

with_FL_0(FL, FR, R):-
   ((FR =< -200, with_FL_0_FR_smaller(FL,FR,R));
   (FR >= 200, with_FL_0_FR_upper(FL,FR,R));
   with_FL_0_FR_equal(FL,FR,R)).

with_FL_50(FL,FR,R):-
   ((FR =< -200, with_FL_50_FR_smaller(FL,FR,R));
   (FR >= 200, with_FL_50_FR_upper(FL,FR,R));
   with_FL_50_FR_equal(FL,FR,R)).

with_FL_100(FL,FR,R):-
   ((FR =< -200, with_FL_100_FR_smaller(FL,FR,R));
   (FR >= 200, with_FL_100_FR_upper(FL,FR,R));
   with_FL_100_FR_equal(FL,FR,R)).

with_FL_0_FR_smaller(_,_,R):- R is 0.
with_FL_0_FR_upper(_,_,R):- R is 50.
with_FL_0_FR_equal(_,_,R):- R is 25.

with_FL_50_FR_smaller(_,_,R):- R is 25.
with_FL_50_FR_upper(_,_,R):- R is 75.
with_FL_50_FR_equal(_,_,R):- R is 50.
```

with_FL_100_FR_smaller(_,_,R):- R is 50.

with_FL_100_FR_upper(_,_,R):- R is 100.

with_FL_100_FR_equal(_,_,R):- R is 75.

Este algoritmo foi desenvolvido tendo por base a seguinte tabela:

Força de Ligação	Força de Relação Likes-Dislikes	Resultado da Função Multicritério
0	≤-200	0
0	0	25
0	≥+200	50
50	≤-200	25
50	0	50
50	≥+200	75
100	≤-200	50
100	0	75
100	≥+200	100

```
multicriterio(FL, FR, R):-
   ((FL < 50, with_FL_0(FL,FR,R));
   (FL < 100, with_FL_50(FL,FR,R));
   (FL = 100, with_FL_100(FL,FR,R))),!.
```

Aqui são verificados os valores da primeira coluna (Forças de ligação).

```
with_FL_0(FL, FR, R):-
   ((FR =< -200, with_FL_0_FR_smaller(FL,FR,R));
   (FR >= 200, with_FL_0_FR_upper(FL,FR,R));
   with_FL_0_FR_equal(FL,FR,R)).
```

```
with_FL_50(FL,FR,R):-
   ((FR =< -200, with_FL_50_FR_smaller(FL,FR,R));
   (FR >= 200, with_FL_50_FR_upper(FL,FR,R));
   with_FL_50_FR_equal(FL,FR,R)).

with_FL_100(FL,FR,R):-
   ((FR =< -200, with_FL_100_FR_smaller(FL,FR,R));
   (FR >= 200, with_FL_100_FR_upper(FL,FR,R));
   with_FL_100_FR_equal(FL,FR,R)).
```

Aqui, após a verificação das forças de ligação, procedemos à verificação das forças de relação.

```
with_FL_0_FR_smaller(_,_,R):- R is 0.
with_FL_0_FR_upper(_,_,R):- R is 50.
with_FL_0_FR_equal(_,_,R):- R is 25.

with_FL_50_FR_smaller(_,_,R):- R is 25.
with_FL_50_FR_upper(_,_,R):- R is 75.
with_FL_50_FR_equal(_,_,R):- R is 50.

with_FL_100_FR_smaller(_,_,R):- R is 50.
with_FL_100_FR_upper(_,_,R):- R is 100.
with_FL_100_FR_equal(_,_,R):- R is 75.
```

Aqui, consoante as verificação efetuadas anteriormente, será atribuído um resultado a R de acordo com a terceira coluna da tabela.

8. Adaptação dos 3 métodos (Primeiro em Profundidade, Best First e A*) para considerar a função multicritério do ponto anterior

Algoritmo Best First (Com recurso à função multicritério):

Todos os predicados utilizados para este algoritmo já foram explicados em cima à exceção deste calcula_custo_mc:

```
calcula_custo_mc([Act,X],R):-!,
        no(NAct,Act,_),
        no(NX,X,_),
        (ligacao(NAct,NX,FL,_,FR,_);ligacao(NX,NAct,_,FL,_,FR)),
        multicriterio(FL, FR, R).

calcula_custo_mc([Act,X|L],P):-calcula_custo_mc([X|L],P1),
        no(NAct,Act,_),
        no(NX,X,_),
        (ligacao(NAct,NX,FL,_,FR,_);ligacao(NX,NAct,_,FL,_,FR)),
        multicriterio(FL, FR, R),
        P is P1 + R.
```

Neste predicado, não é efetuada apenas a soma das forças de ligação, uma vez que aqui efetuamos a chamada à função multicritério, que já foi explicado anteriormente.

Mais uma vez, no nosso ficheiro ALGAV.pl temos uma versão do algoritmo que obtém uma única solução.

Algoritmo A* (Com recurso à função de multicritério):

```
aStar2_cmc(Dest,[(_,Ca,LA)|Outros],RedeUtilizadores,Cam,Custo,N):-
    LA=[Act|_],
    length(LA,C),
    findall((CEX,CaX,[X|LA]),
    (C=<N,
    Dest\==Act,
    (ligacao(Act,X,ForcaX,_,ForcaY,_);ligacao(X,Act,_,ForcaX,_,ForcaY)),
    \+ member(X,LA),multicriterio(ForcaX,ForcaY,Resultado),
    CaX is Resultado + Ca,
    estimativa(Resultado,X,Dest,C,N,RedeUtilizadores,EstX),
```

Com o recurso à função de multicritério tivemos, como demonstra o excerto do código gerado, fazer uma pequena alteração ao algoritmo. Neste caso, utilizamos mais do que apenas a força de ligação, mas também a força de relação, que foi previamente calculada pela diferença entre likes e dislikes.

No entanto, também haverá uma pequena diferença no predicado que irá fornecer a lista "RedeUtilizadores", isto porque, no A* sem recorrer à função multicritério, o predicado "forcas_ligacao_decrescente" que recorria posteriormente ao "forcas_ligacao_decrescente_2", iria apenas recorrer a força de ligação do utilizador A e à força de ligação do utilizador B, mas neste caso como temos de considerar as duas novas forças de relação (utilizador A e B), foi criado um novo predicado, chamado "forcas_ligacao_decrescente_cmc", que recorre a um novo predicado "forcas_ligacao_descrescente_2_cmc", como demonstrado de seguida.

```
forcas_ligacao_decrescente_2_cmc([_],[]):-!.
forcas_ligacao_decrescente_2_cmc([User1|Y],[RedeUtilizadores|Z]):-
    findall([F1,F2,F3,F4],
    (member(User2,Y),
    (ligacao(User1,User2,F1,F2,F3,F4);ligacao(User2,User1,F2,F1,F4,F3))),RedeUtilizadores),
    forcas_ligacao_decrescente_2(Y,Z).
```

Para recorrer à função de multicritério, ela irá utilizar as duas forças de ligação e relação. A explicação da função multicritério já foi apresentado no tópico anterior.

9. Comparação dos 3 métodos com vários exemplos e usando a função multicritério

Determinar o caminho mais forte entre dois utilizadores

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo de geração – Best First com Função Multicritério (Sprint C)	Tempo de geração (Sprint B)
1	1	0.0000 s	0.0000 s
2	2	0.0000 s	0.0000 s
3	3	0.0000 s	0.0139 s
4	4	0.0000 s	312.5943 s

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo de geração – A* com Função Multicritério (Sprint C)	Tempo de geração (Sprint B)
1	1	0.0009 s	0.0000 s
2	2	0.0019 s	0.0000 s
3	3	0.0020 s	0.0139 s
4	4	0.0010 s	312.5943 s

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo de geração – Depth First com Função Multicritério (Sprint C)	Tempo de geração (Sprint B)
1	1	0.0000 s	0.0000 s
2	2	0.0080 s	0.0000 s
3	3	0.0000 s	0.0139 s
4	4	0.0000 s	312.5943 s

Alteração efetuada para verificar o tempo de geração no Algoritmo Best First:

```
bestfs12_mc_one(Dest,[[Dest|T]|_],Cam,Custo,_):-
                get_time(Ti),
                reverse([Dest|T],Cam),
                calcula_custo_mc(Cam,Custo),
                get_time(Tf),
                Temp is Tf-Ti,
                write('Tempo de geracao da solucao obtida: '), write(Temp),
write(' segundos'), nl,
                write('Caminho='),write(Cam),nl.
```

10. Conclusões

Perante os algoritmos que foram feitos por nós, conseguimos perceber que nem sempre é obtido os caminhos com mais força, mas sim os caminhos mais rápidos a obter o tamanho mais forte, contudo, isso acontece raramente.

No ponto 6 e 9 deste relatório, conseguimos verificar que o algoritmo A* tem os tempos de pesquisa maiores, pelo simples facto da demora a executar a estimativa pedida, tornando-o mais lento que os restante algoritmos. Comparando com os algoritmos pedidos no sprint anterior (B), conseguimos verificar que temos tempos muito melhores, para situações onde o número de camadas e de nodes por camada, sejam maiores.

Mencionando o Algoritmo A*, a estimativa pedida, foi um pouco difícil de calcular, tornando este algoritmo difícil de conceber, isto porque, perante o enunciado pedido para LAPR5, foi bastante difícil obter uma estimativa de valores (força de ligação e de relação) que não conseguimos tratar tão facilmente como os exemplos dados.

Também conseguimos concluir que perante os mesmos utilizadores, os algoritmos conseguem chegar ao mesmo destino, ou seja, obtendo o mesmo valor tanto para o custo como para o caminho obtido.