ALGAV – Sprint C

2021 / 2022

1191454 - Bruno Pereira

1180658 - Rafael Faísca

1191059 - Sérgio Balio

1170385 - Rui Mendes

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

Para este problema utilizamos o caso de uso tamanho da rede de um utilizador realizado na sprint B uma vez que este já cumpria os requisitos para ser utilizado na sprint C. Apenas foi necessário alterar o número de parâmetros da "ligação/6" uma vez que foi adicionado a força de relação.

```
% Quando a queue se encontra vazia, decrementa-se o nível e substitui-se a queue por todos os nós visitados no nível anterior
tamanhoRede2(Nivel,[],ProxQueue,ListaFinal,ListaResultado):-
    NivelAtual is Nivel - 1,
    tamanhoRede2(NivelAtual, ProxQueue, [], ListaFinal, ListaResultado).

tamanhoRede2(0,_,_,ListaFinal,ListaFinal):-
    !.

tamanhoRede2(Nivel, [User|QueueUsers], ProxQueue, ListaFinal, ListaResultado):-
    % X será utilizadores conectados ao primeiro Utilizador na queue e que não se encontre na Rede
    findall(X,(
                ligacao(User,X,_,_,_,_),
                \+member(X, ListaFinal)
            ),
            ListaUserNivel),
    % Adicionar lista de amigos ja visitados à Rede
    append(ListaUserNivel, ListaFinal, ListaCompleta),
    % Criar lista de amigos visitados sem os amigos que já se encontrem em queue
    findall(Y,(
                member(Y, ListaUserNivel),
                \+member(Y, ProxQueue)
            ), UserNivelSemRepetidos),
    % Adicionar os amigos visitados que ainda não estão na queue à proxima queue
    append(UserNivelSemRepetidos, ProxQueue, NovaQueue),
    tamanhoRede2(Nivel, QueueUsers, NovaQueue, ListaCompleta, ListaResultado).
```

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

O AStar foi desenvolvido tendo em conta o método de pesquisa fornecido nas aulas TP. O objetivo deste algoritmo é descobrir o menor caminho com a maior força.

Adaptamos o AStar de forma que este tenha em conta as forças de ligação como custo, um número máximo de ligações passado por parâmetro e também uma estimativa que tenha em conta a força de ligação.

```
?- aStar(1,31,3,Cam,Custo).
0.0020029544830322266 s
Cam = [1, 11, 24, 31],
Custo = 17
```

Utilizamos um "findall/3" para preencher a lista *Novos* com o custo estimado até X, custo do utilizador atual até X, uma lista com todos os utilizadores do caminho até o instante e uma lista de todas as ligações.

Depois de os caminhos serem calculados vamos adicionar a lista *Novos* à lista *Outros* e através do predicado "sort/4" conseguimos ordenar a lista de forma decrescente e mantendo os repetidos. É também verificado se o número de ligações ultrapassa o limite definido por parâmetro.

```
aStar2(Dest,[(_,Custo,[Dest|T],Lista)|_],N,Cam,Custo):-
    length(Lista,NLigacoes),
    NLigacoes =< N,
    reverse([Dest|T],Cam).

aStar2(Dest,[(_,CustoAtual,CamAtual,Lista)|Outros],N,Cam,Custo):-
    CamAtual=[Atual|_],
    findall((CustoEstX,CustoaX,[X|CamAtual],[CustoX|Lista]), (
        Dest\==Atual,
        ligacao(Atual,X,CustoX,_,_,_),
        \+member(X,CamAtual),
        %calculo da estimativa
        estimativaFL(X,Lista,N,EstX),
        CustoaX is CustoX + CustoAtual,
        CustoEstX is CustoaX + EstX
    ), Novos),
    append(Outros,Novos,Todos),
    % ordenar de forma decrescente e mantendo os repetidos
    sort(0,@>=,Todos,TodosOrd),
    aStar2(Dest,TodosOrd,N,Cam,Custo).
```

Estimativa Implementada

Criamos uma estimativa para calcular o custo estimado tendo em conta as forças de ligação("estimativaFL/4"), e uma estimativa que use o multicritério ("estimativaMultiCriterio/4").

```
estimativaFL(X,Lista,N,EstX):-
    length(Lista,NLigacoesAtual),
    Nivel is N-NLigacoesAtual,
    forcasLigacaoUser(X,Nivel,ListaForcas),
    % ordenar de forma decrescente e mantendo os repetidos
    sort(0,@>=,ListaForcas,ListaForcasOrd),
    removeForcas(Lista,ListaForcasOrd,ListaForcasAtual),
    elementosNivel(ListaForcasAtual,Nivel,ListaForcasFinal),
    soma(ListaForcasFinal,EstX).
```

Primeiro calculamos o número e ligações que podem ser realizadas *(Nivel)*. Na função "forcasLigacaoUser/3" vamos calcular as forças de ligação para o utilizador até um nível máximo, para isso utilizamos a função "tamanhoRedeUser/4" de modo a obtermos a rede de utilizadores até o nível máximo pretendido.

```
% forças de ligaçao do user para Nivel N
forcasLigacaoUser(User,Nivel,ListaForcas):-
    %retorna lista de users ate Nivel N
    tamanhoRedeUser(User,Nivel,ListaUserRede,_),
    forcaLigacaoLista(ListaUserRede,[],ListaForcas).
```

Depois de obtermos a rede de utilizadores pretendida chamamos a função "forcaLigacaoLista/3" que através do "findall/3" obtém uma lista das forças de ligação da rede do utilizador *(Lista)*.

```
% forças de ligaçao para lista de users
forcaLigacaoLista([User|ListaUserRede],ListaAtual,ListaFinal):-
    findall(FL, (
        member(X,ListaUserRede),
        X \== User,
        forcaLigacao(User,X,FL)), Lista),
    append(Lista,ListaAtual,ListaCompleta),
    forcaLigacaoLista(ListaUserRede,ListaCompleta,ListaFinal).

forcaLigacao(_,_,_):-
    fail,
    !.

forcaLigacao(X,Y,FL):-
    ligacao(Y,X,_,FL,_,_),
    !.

% força de ligaçao entre 2 users
forcaLigacao(X,Y,FL):-
    ligacao(X,Y,FL,_,_,_),
    !.
```

Para o multicritério a regra é a mesma, a única diferença está no cálculo da força onde chamamos a nossa função multicritério para cada ligação.

```
% forças com multi criterio para lista de users
forcaMultiCriterioLista([User|ListaUserRede],ListaAtual,ListaFinal):-
    findall(Res, (
        member(X,ListaUserRede),
        X \== User,
        forcaMultiCriterio(User,X,Res)), Lista),
    append(Lista,ListaAtual,ListaCompleta),
    forcaMultiCriterioLista(ListaUserRede,ListaCompleta,ListaFinal).

forcaMultiCriterio(_,_,_):-
    fail,
    !.

forcaMultiCriterio(X,Y,Res):-
    ligacao(Y,X,_,FL,_,FR),
    multiCriterio(FL,FR,Res),
    !.

% força com multi criterio entre 2 users
forcaMultiCriterio(X,Y,Res):-
    ligacao(X,Y,FL,_,FR,_),
    multiCriterio(FL,FR,Res),
    !.
```

Voltando à função da estimativa, depois de obtermos a lista com as forças de ligação essa lista vai ser ordenada pela função "sort/4" de forma decrescente e mantendo os repetidos e são removidas as forças de ligação utilizadas através da função "removeForcas/3".

```
removeForcas([],Lista,Lista):-
    !.

% elimina os elementos da primeira lista coincidentes na segunda lista
removeForcas([L|ListaLigacoes],Lista,ResFinal):-
    apaga(L,Lista,Res),
    removeForcas(ListaLigacoes,Res,ResFinal).
```

No final é usada a função "soma/2" para somar forças na lista até o elemento pretendido e assim obtendo o majorante da estimativa para o nó final.

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

O Best First Search também foi desenvolvido tendo em conta o método de pesquisa fornecido nas aulas TP. No entanto, tal como no predicado AStar o valor do custo não é conhecido à *priori* e depende do custo das forças de ligação (ou do multicritério). O objetivo deste algoritmo é também conhecer o menor caminho com a maior força.

```
?- bestfs(1,21,4,Cam,Custo).
0.0 s
Cam = [1, 11, 21],
Custo = 15 .
```

```
bestfs2(Dest,[[Dest|T]|_],N,Cam,Custo):-
    reverse([Dest|T],Cam),
    length(Cam,Temp),
    NLigacoes is Temp - 1,
    NLigacoes =< N,
    calculaCustoForcaLigacao(Cam,Custo).

bestfs2(Dest,[[Dest|_]|Lista],N,Cam,Custo):-
    !,
    bestfs2(Dest,Lista,N,Cam,Custo).

bestfs2(Dest,Lista,N,Cam,Custo):-
    primeiroElemento(CamAtual,Lista,ListaAtual),
    CamAtual = [Atual|_],
    ((Atual == Dest,
        !,
        bestfs2(Dest,[CamAtual|ListaAtual],N,Cam,Custo));
     (findall((FL,[X|CamAtual]), (
            ligacao(Atual,X,FL,_,_,_),
            \+member(X,CamAtual)
        ), Novos),
        Novos \== [],
        !,
        % ordenar de forma decrescente e mantendo os repetidos
        sort(0,@>=,Novos,NovosOrd),
        retiraCusto(NovosOrd,NovosOrd1),
        append(NovosOrd1,ListaAtual,ListaCompleta),
        bestfs2(Dest,ListaCompleta,N,Cam,Custo))).
```

No caso geral, através do "findall/3" obtemos uma lista com cada força de ligação do caminho atual *(FL)* e uma lista com o caminho *([X|CamAtual])*. Se essa lista estiver vazia significa que entramos num beco sem saída, caso contrário ela vai ser ordenada pelo "sort/4" de forma decrescente e mantendo os repetidos de forma a obter a ligação mais forte no início da lista.

No critério de paragem do algoritmo é verificado o limite de ligações passado por parâmetro *(N)*, e usamos a função "calculaCustoForcaLigacao\2" onde são somadas as forças de ligação do caminho.

Adaptação do Primeiro em Profundidade para gerar a melhor solução para o máximo de N ligações

Foi adicionado o número máximo de ligações ao parâmetro dos predicados plan_forteMulti e melhor_caminho_forteMulti e foram adicionadas as linhas 47, 48 e 49 ao predicado abaixo para fazer a tal verificação do número de ligações.

```
?- plan_forteL(ana,antonio,1,Cam,Custo).
122.84694194793701 s
Cam = [ana, antonio],
Custo = 18.
```

```
46  atualiza_melhor_forteL(LCaminho,LF,N):-
47      length(LCaminho,Temp),
48      NLigacoes is Temp - 1,
49      NLigacoes =< N,
50      melhor_sol_forte(_,LCaminho,N),
51      sumlist(LF,SF),
52      SF>N, retract(melhor_sol_forte(_,_,_)),
53      asserta(melhor_sol_forte(LCaminho,_,SF)).
```

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

Origem	Destino	Nº máximo de ligações	Tempo AStar (s)	Tempo BFS (s)	Tempo DFS (s)
1	11	1	0.002006053924560547	0.0	122.84694194793701
1	11	2	0.0009951591491699219	0.0	130.79704904556274
1	11	3	0.0009739398956298828	0.0	117.88814210891724
1	21	2	0.0	0.0	32.7557110786438
1	21	3	0.001021862030029296	0.0	33.16266703605652
1	21	4	0.002000093460083008	0.0	32.836241006851196
1	31	3	0.000999927520751953	0.0	35.82042384147644
1	31	4	0.001996040344238281	0.0	35.97958898544311
1	31	5	0.0019989013671875	0.0	35.4505410194397
1	41	4	0.002009153366088867	0.0	98.70803809165955
1	41	5	0.0029990673065185547	0.0	98.8907070159912
1	41	6	0.004002094268798828	0.0	98.946166992187

Origem	Destino	Nº máximo de ligações	Custo AStar	Custo BFS	Custo DFS
1	11	1	10	10	18
1	11	2	10	10	18
1	11	3	10	10	33
1	21	2	15	15	30
1	21	3	15	15	30
1	21	4	15	15	43
1	31	3	17	17	33
1	31	4	17	17	33
1	31	5	17	17	50
1	41	4	23	23	41
1	41	5	23	23	41
1	41	6	23	23	62

Concluímos com estas tabelas que, para os exemplos dados:

- Os algoritmos AStar e BFS retornam o mesmo caminho com o mesmo custo, apesar de apresentarem tempos de execução um pouco diferentes.
- O AStar apresenta tempos de execução semelhantes em todos os exemplos, todos próximos de 0s.
- O BFS aparenta ser o mais eficiente, visto que em todos os exemplos o tempo de execução é 0.0s.
- O DFS é o algoritmo mais eficaz e o menos eficiente, pois ele retorna os caminhos com maior custo em todos os exemplos, sendo que estes resultados variam dependendo do número máximo de ligações, mas tem um tempo de execução muito maior que os outros algoritmos, também devido à dimensão da base de conhecimento.

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

Para a implementação da função multicritério criamos cinco condições baseadas na tabela dada na tp de apoio ao sprint C. Tendo em conta que a forca de relação varia entre -200 e 200 e a força de ligação entre 0 e 100 a função multicritério obtem o seu resultado de acordo com os valores apresentados nesta 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):- ( FR =< -200 -> R is((-0.5 * FL) + FL);
                         (FR >= 200 -> R is (0.5 * FL) + 50 ;
                         (FR < 0 -> R is ((0.5 * FL) + (0.5*(50 - (FR / 200) * 50)));
                         (FR =:= 0 -> R is ((0.5 * FL) + (0.5 * 50));
                         (FR > 0 -> R is ((0.5 * FL) + (0.5*(50 + (FR / 200) * 50))); true))))).
```

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

- AStar Multicritério

```
?- aStarMulticriterio(1,31,5,Cam,Custo).
0.00776314735412597666 s
Cam = [1, 12, 24, 31],
Custo = 135.5
```

Para acrescentar a função do multicritério no método AStar foi necessário alterar o método de calculo para usar a função multicritério no "findall/3" do predicado principal:

```
aStarMulticriterio2(Dest,[(_,CustoAtual,CamAtual,Lista)|Outros],N,Cam,Custo):-
    CamAtual=[Atual|_],
    findall((CustoEstX,CustoaX,[X|CamAtual],[CustoX|Lista]), (
        Dest\==Atual,
        ligacao(Atual,X,FL,_,FR,_),
        \+member(X,CamAtual),
        %calculo do custo com multi criterio
        multiCriterio(FL,FR,CustoX),
        %calculo da estimativa
        estimativaMultiCriterio(X,Lista,N,EstX),
        CustoaX is CustoX + CustoAtual,
        CustoEstX is CustoaX + EstX
    ), Novos),
    append(Outros,Novos,Todos),
    % ordenar de forma decrescente e mantendo os repetidos
    sort(0,@>=,Todos,TodosOrd),
    aStarMulticriterio2(Dest,TodosOrd,N,Cam,Custo).
```

E adaptamos também o cálculo da estimativa para utilizar a função multicritério:

```
% forças com multi criterio para lista de users
forcaMultiCriterioLista([User|ListaUserRede],ListaAtual,ListaFinal):-
    findall(Res, (
        member(X,ListaUserRede),
        X \== User,
        forcaMultiCriterio(User,X,Res)), Lista),
    append(Lista,ListaAtual,ListaCompleta),
    forcaMultiCriterioLista(ListaUserRede,ListaCompleta,ListaFinal).

forcaMultiCriterio(_,_,_):-
    fail,
    !.

forcaMultiCriterio(X,Y,Res):-
    ligacao(Y,X,_,FL,_,FR),
    multiCriterio(FL,FR,Res),
    !.

% força com multi criterio entre 2 users
forcaMultiCriterio(X,Y,Res):-
    ligacao(X,Y,FL,_,FR,_),
    multiCriterio(FL,FR,Res),
    !.
```

- BFS Multicritério

```
?- bestfsMultiCriterio(1,11,3,Cam,Custo).
0.00099802017211914 06 s
Cam = [1, 11],
Custo = 42.5
```

Para acrescentar a função do multicritério no método BFS foi necessário adicionar a função multicritério no "findall/3" do predicado principal para alterar o cálculo do custo:

```
bestfsMultiCriterio2(Dest,Lista,N,Cam,Custo):-
    primeiroElemento(CamAtual,Lista,ListaAtual),
    CamAtual = [Atual|_],
    ((Atual == Dest,
        !,
        bestfsMultiCriterio2(Dest,[CamAtual|ListaAtual],N,Cam,Custo));
     (findall((Res,[X|CamAtual]), (
            ligacao(Atual,X,FL,_,FR,_),
            \+member(X,CamAtual),
            multiCriterio(FL,FR,Res)
        ), Novos),
        Novos \==[],
        !,
        % ordenar de forma decrescente e mantendo os repetidos
        sort(0,@>=,Novos,NovosOrd),
        retiraCusto(NovosOrd,NovosOrd1),
        append(NovosOrd1,ListaAtual,ListaCompleta),
        bestfsMultiCriterio2(Dest,ListaCompleta,N,Cam,Custo))).
```

E o cálculo do custo utilizado no predicado de critério de paragem:

```
bestfsMultiCriterio2(Dest,[[Dest|T]|_],N,Cam,Custo):-
    reverse([Dest|T],Cam),
    length(Cam,Temp),
    NLigacoes is Temp - 1,
    NLigacoes =< N,
    calculaMultiCriterio(Cam,Custo).
```

- DFS Multicritério

```
?- plan_forteMultiL(ana,eduardo,2,Cam,Custo).
61.93600416183472 s
Cam = [ana, carlos, eduardo],
Custo = 100.0.
```

Para o DFS foi alterada a força de ligação no último parâmetro do predicado abaixo por R, resultado obtido pela função multicritério.

```
dfs_forte2Multi(Act,Dest,LA,Cam,[R|RF]):- no(NAct,Act,_), (ligacao(NAct,NX,F,F1,FR,FR1);ligacao(NX,NAct,F,F1,FR,FR1)),
    multiCriterio(F+F1,FR+FR1,R), no(NX,X,_),
    \+ member(X,LA),dfs_forte2Multi(X,Dest,[X|LA],Cam,RF).
```

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

Origem	Destino	Nº máximo de ligações	Tempo AStar (s)	Tempo BFS (s)	Tempo DFS (s)
1	11	1	0.002019882202148437	0.000995159149169921	225.02525901794434
1	11	2	0.0009989738464355469	0.0010099411010742188	231.45810317993164
1	11	3	0.001013040542602539	0.0009999275207519531	231.06199312210083
1	21	2	0.0009980201721191406	0.0	61.93600416183472
1	21	3	0.0009999275207519531	0.0	62.03260803222656
1	21	4	0.000997781753540039	0.0	62.00197196006775
1	31	3	0.0060100555419921875	0.0	65.76054906845093
1	31	4	0.007019996643066406	0.0	65.54495692253113
1	31	5	0.006997823715209961	0.0	66.78911995887756
1	41	4	0.005014896392822266	0.0	184.5431559085846
1	41	5	0.010000944137573242	0.0	183.5846540927887
1	41	6	0.011020898818969727	0.0	183.25037598609924

Origem	Destino	Nº máximo de ligações	Custo AStar	Custo BFS	Custo DFS
1	11	1	42.5	42.5	36.5
1	11	2	42.5	42.5	36.5
1	11	3	42.5	42.5	156.0
1	21	2	98.0	98.0	100.0
1	21	3	98.0	98.0	100.0
1	21	4	98.0	98.0	191.875
1	31	3	135.5	127.75	134.5
1	31	4	135.5	127.75	134.5
1	31	5	135.5	127.75	254.0
1	41	4	195.25	169.0	193.0
1	41	5	195.25	169.0	193.0
1	41	6	195.25	169.0	300.5

Concluímos com estas tabelas que, para os exemplos dados:

- O BFS aparenta retornar um resultado diferente do AStar onde o custo, em dois dos exemplos, é menor.
- O AStar apresenta tempos de execução semelhantes em todos os exemplos, todos próximos de 0s.

- O BFS aparenta ser o mais eficiente em termos de tempo de execução sendo que grande parte dos resultados apresentam o valor 0.0s.
- O DFS é o algoritmo menos eficiente, pois este tem um tempo de execução muito maior que os outros algoritmos, também devido à dimensão da base de conhecimento.
- O DFS e o AStar são os mais eficazes, pois estes retornam os caminhos com maior custo em todos os exemplos. Estes resultados variam tendo em conta o número máximo de ligações, e podemos ver que em maior parte dos exemplos se o número máximo de ligações for alto o DFS consegue retornar um custo maior.

Conclusões

No final deste sprint o grupo pensa que desenvolveu um bom trabalho e que melhoramos os nossos conhecimentos em relação a esta unidade curricular durante a execução do mesmo.

Podemos também concluir que, tendo em conta os algoritmos que desenvolvemos tanto para a força de ligação como para o multicritério o DFS demonstra ser o mais eficaz, porem é o menos eficiente. Concluímos também que o BFS e o AStar são os algoritmos mais eficientes apresentando em grande parte dos casos um tempo de execução próximo de 0s.

Para que o algoritmo AStar atinja um nível mais elevado de eficácia é necessário aplicar as heurísticas corretas, e aplicáveis ao contexto do seu desenvolvimento.