Instituto Superior de Engenharia do Porto

Licenciatura em Engenharia Informática

3DI

Relatório de Algoritmia Avançada

Grupo 051

Por Beatriz Seixas (1190424),

Jéssica Alves (1190682)

Pedro Santos (1190967),

Tiago Costa (1191460)

Porto, dezembro de 2021

Índice geral

Índice de Figuras

1. INTRODUÇÃO

Na sequência do projeto referente à unidade curricular de *ALGAV* foi proposto o desenvolvimento de várias *user storys* com fim a criar uma aplicação cuja missão é fornecer aplicações de manipulação e visualização de grafos de redes sociais.

Neste documento iremos apresentar uma breve explicação do problema, assim como o pensamento para cada *user story*, e exemplos demonstrativos. Iremos também mostrar o estudo da Complexidade do Problema da Determinação de Caminhos.

As competências requisitadas para este sprint, foram as seguintes:

- Determinação do Tamanho da Rede de um utilizador até um determinado Nível
- Obtenção dos Utilizadores que tenham em Comum *X Tags*
- Sugestão das Conexões com outros Utilizadores tendo por base as *Tags* e Conexões Partilhadas até um determinado Nível
- Determinação do Caminho Mais Forte
- Determinação do Caminho Mais Curto
- Determinação do Caminho Mais Seguro

2. REPRESENTAÇÃO DO CONHECIMENTO DO DOMÍNIO

2.1. Base de Conhecimento

No Desenvolvimento deste projeto, foram utilizadas duas bases de conhecimento locais, e uma base de conhecimento integrada no projeto integrador através do protocolo *HTTP*. Na Figura 1 e Figura 3, podemos verificar a base de conhecimento completa, fornecida pelos docentes de *ALGAV*. Esta apresenta dois factos: *node*/3 e *connection*/4. O *node*/3 é referente a um utilizador da aplicação, denominado por jogador e apresenta, respetivamente, o seu id, nome e lista de *tags*. Já *connection*/4, é referente às ligações de amizade entre jogadores, conectando assim a nossa rede, como podemos verificar na Figura 1. Para dos factos mencionados previamente, também são utilizados alguns factos

dinâmicos, como por exemplo: *dynamic safest_currentRoute*/2 que serve de auxílio para o caso de uso referente ao caminho mais seguro, onde é guardado o caminho mais seguro atual, durante a execução do programa, par determinar qual seria esse caminho (Figura 2). Desta base de conhecimento completa resultará uma rede representada graficamente pela Figura 4. Seguidamente, a base de conhecimento parcial apenas tem alguns nós para ser mais prático o teste e apresentação das funcionalidades. Por último, a versão integrada no projeto através de *HTTP*, constrói a sua base de conhecimento e vai a atualizando ao longo dos *requests* que recebe.

```
connection(1,11,10,8).
```

Figura 1 - Base de Conhecimento Completa (Connection)

```
:- dynamic safest_currentRoute/2.
```

Figura 2 - Facto dinâmico referente ao caminho mais seguro

```
node(1,ana,[natureza,pintura,musica,sw,porto]).
node(11,antonio,[natureza,pintura,carros,futebol,lisboa]).
node(12,beatriz,[natureza,musica,carros,porto,moda]).
node(13,carlos,[natureza,musica,sw,futebol,coimbra]).
node(14,daniel,[natureza,cinema,jogos,sw,moda]).
node(21,eduardo,[natureza,cinema,teatro,carros,coimbra]).
node(22,isabel,[natureza,musica,porto,lisboa,cinema]).
node(23,jose,[natureza,pintura,sw,musica,carros,lisboa]).
node(24,luisa,[natureza,cinema,jogos,moda,porto]).
node(31,maria,[natureza,pintura,musica,moda,porto]).
node(32,anabela,[natureza,cinema,musica,tecnologia,porto]).
node(33,andre,[natureza,carros,futebol,coimbra]).
node(34,catia,[natureza,musica,cinema,lisboa,moda]).
node(41,cesar,[natureza,teatro,tecnologia,futebol,porto]).
node(42,diogo,[natureza,futebol,sw,jogos,porto]).
node(43,ernesto,[natureza,teatro,carros,porto]).
node(44,isaura,[natureza,moda,tecnologia,cinema]).
node(200,sara,[natureza,moda,musica,sw,coimbra]).

node(51,rodolfo,[natureza,musica,sw]).
node(61,rita,[moda,tecnologia,cinema]).
```

Figura 3 - Base de Conhecimento Completa (Node)

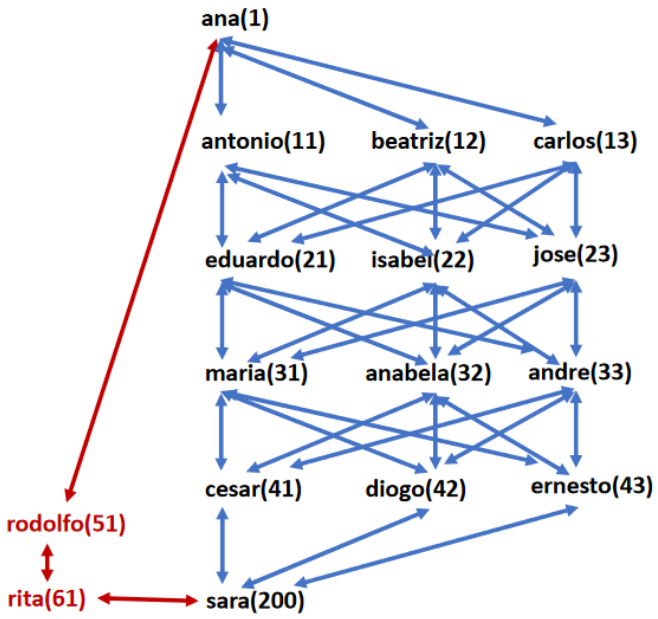

Figura 4 - Representação Gráfica da Base de Conhecimento Completa

3. DETERMINAÇÃO DO TAMANHO DA REDE DE UM UTILIZADOR ATÉ UM DETERMINADO NÍVEL

3.1. Funcionalidade desenvolvida por Beatriz Seixas (1190424)

Nesta funcionalidade, o predicado chamado irá ser o *network_getNetworkByLevel*/4 (Figura 5). Os parâmetros para este predicado serão, em primeiro lugar, *Orig*(nó de origem), em segundo lugar, *Level*(nível pretendido), em terceiro o *L*(Lista) e, por fim o *Total*(tamanho da rede).Primeiramente com o *retractall*(*userVisited*(_)) removemos as regras e factos da base de conhecimento relativos ao *userVisited*/1. Depois utilizamos duas vezes o *findall*, a primeira vez recorre ao *dfs*/2 de modo a obtermos os vários caminhos possíveis que vão ser guardados no *userVisited*/1. O segundo *findall* tem como objetivo copiar os nós para a lista(*L*).

Seguidamente usamos novamente o *retractall*(*userVisited*(_)) de modo a limpar os dados, e por último obtemos o tamanho da lista.

Na segunda parte definimos o dfs2/3(figura6) que tem como parâmetros o nó origem (*Act*), o nível (*Level*) e a lista auxiliar (*LA*), que guarda os nós.

Primeiro é feita uma validação para que o nível seja maior que zero. E seguidamente determinamos as conexões que podem ser obtidas. Verificamos depois que a conexão já não esta no *userVisited* e também não esta na lista auxiliar.

Depois, se não for membro vamos decrementar o nível e recorrendo ao asserta(*userVisited*(*X*)), guardamos o valor na base de conhecimento.

Por último chamamos o predicado com o elemento já adicionado, tanto a lista como nível atual.

Para a base de conhecimento completa (Figura 7), com os parâmetros: “1” como *Orig*, “2” como *Level*, “L” de lista e “T” de total, este método retorna uma lista resultado assim como o tamanho da rede.

```
network_getNetworkByLevel(Orig,Level,L, Total):-
    retractall(userVisited(_)),
    findall(Orig,dfs(Orig,Level),_),
    findall(User,userVisited(User),L),
    retractall(userVisited(_)),
    length(L,Total).
```

Figura 5 – Predicado network_getNetworkByLevel/4 para determinação do Tamanho da Rede

```
dfs2(Act,Level,LA):-
    Level > 0,
    (connection(Act,X,_,_);connection(X,Act,_,_)),
    \+userVisited(X),
    \+ member(X,LA),
    Level1 is Level-1,
    asserta(userVisited(X)),
    dfs2(X,Level1,[X|LA]).
```

Figura 6-Predicado dfs2/3 para determinação do Tamanho da Rede

```
?- network_getNetworkByLevel(1,2,L,T).
L = [61, 51, 14, 13, 12, 24, 23, 22, 21|...],
T = 10.
```

Figura 7-exemplo

4. OBTENÇÃO DOS UTILIZADORES QUE TENHAM EM COMUM X TAGS

4.1. Funcionalidade desenvolvida por Tiago Costa (1191460) e Beatriz Seixas (1190424)

Nesta funcionalidade, o predicado chamado irá ser o *common_tags*/2 (Figura 8). Os parâmetros para este predicado serão, em primeiro lugar, *x* (número de *tags* em comum) e em segundo lugar, *List_Result*, uma lista de pares de listas, as quais contêm, de um lado, as combinações de x *tags* (ou sinónimos) possíveis, e, do outro lado, os utilizadores aos quais essas *tags* correspondem. É de notar que logo no início deste predicado, todas as *tags* são alteradas pelo seu sinónimo. Após, com base num *findall*, este predicado chama o predicado *common_tags_combinations*/3, que irá calcular todas as possíveis permutações de X tags. Logo após, será chamado o método *common_tags_users_combination/3*, que, para além de servir como um iterador recursivo para as combinações, também remove as combinações de *tags* possuídas por menos de 2 jogadores. Este, por sua vez, chama o predicado *common_tags_users_combination_aux*/4 (Figura 9), que tem como objetivo encontrar utilizadores com *X* número de *tags* iguais às *tags* existentes na totalidade através de uma simples interseção. É também de notar que as *tags* dos utilizadores foram trocadas pelos seus sinónimos, dessa forma, ficando em coerência com as *tags* existentes já anteriormente trocadas. O método que permite que isto aconteça tem de nome *common_tags_change_to_synonyms* (Figura 10). Este é um simples método, que ao percorrer as *tags* dadas pelo parâmetro *All_Tags*, verifica se essa *tag* tem um sinónimo associado, sendo que se tiver, executa uma *union* entre o sinónimo dessa *tag* e uma lista inicialmente vazia, e se não tiver um sinónimo associado, executa uma uma *union* entre a própria *tag* e a já falada lista.

Para a base de conhecimento completa (Figura 11) com sinónimos como tecnologia-jogos e teatro-musica, com os parâmetros: “2” como x e “L” como lista de retorno, este método retorna uma lista extensiva de combinações possíveis de *tags* com os jogadores que as contêm em 0.0s.

```
common_tags(X,List_Result):-
    get_time(T1),
    common_tags_get_all_tags(All_TagsT),
    common_tags_change_to_synonyms(All_TagsT, All_Tags),
    findall(Combination,common_tags_combination(X,All_Tags,Combination),Combinations),
    findall(UserId,node(UserId,_,_),Users),
    common_tags_users_combination(X,Users,Combinations),
    findall([Comb,ListUsers],common_tags_users(Comb,ListUsers),List_Result),
    retractall(common_tags_users(_,_)),
    write('Solution found in '),
    get_time(T2),
    T is T2-T1,write(T),write(' seconds'),nl.
```

Figura 8 - Common Tags

```
common_tags_users_combination_aux(_,_,[],[]):-!.
common_tags_users_combination_aux(X,Tags,[U|Users],Result):-
    node(U,_,User_TagsT),
    common_tags_change_to_synonyms(User_TagsT, User_Tags),
    intersection(Tags, User_Tags,Commun),
    length(Commun, Size),
    Size >= X, !,
    common_tags_users_combination_aux(X,Tags,Users,Result1),
    append([U], Result1, Result).
common_tags_users_combination_aux(X,Tags,[_|Users],Result):-
    !,
    common_tags_users_combination_aux(X,Tags,Users,Result).
```

Figura 9 - Common_tags_users_combination_aux

```
common_tags_combination(0,_,[]).
common_tags_combination(N,[X|T],[X|Comb]):-N>0,N1 is N-1,common_tags_combination(N1,T,Comb).
common_tags_combination(N,[_|T],Comb):-N>0,common_tags_combination(N,T,Comb).
```

Figura 10 - Common_tags_combination

```
common_tags_change_to_synonyms([],[]).
common_tags_change_to_synonyms([Tag|All_Tags],Tags):-
    common_tags_change_to_synonyms(All_Tags,Tags1),!,
    (synonym(Tag, Sign) ->
        union([Sign], Tags1, Tags);
        union([Tag], Tags1, Tags)).
```

Figura 11 - Common_tags_change_to_synonyms

```
?- common_tags(2, L).
Solution found in 0.0 seconds
L = [[[jogos, cinema], [14, 24, 32, 44, 61]], [[moda, cinema], [14, 24, 34, 44, 61]], [[moda, jogos], [
14, 24, 44, 61]], [[sw, jogos], [14, 42]], [[sw, moda], [14, 200]], [[musica, cinema], [21|...]], [[mus
ica|...], [...|...]], [[...|...]|...], [...|...]|...].
```

Figura 12 - Common_tags example

5. SUGESTÃO DAS CONEXÕES COM OUTROS UTILIZADORES TEND POR BASE AS TAGS E CONEXÕES PARTILHADAS ATÉ UM DETERMINADO NÍVEL

5.1. Funcionalidade Desenvolvida por Jéssica Alves (1190682) e Pedro Santos (1190967)

O predicado de sugerir utilizadores (*suggest_players*/3) tem como parâmetros: o id do jogar atual, o nível pretendido e a lista de utilizadores sugeridos, respetivamente. Este obtém as sugestões através dos seguintes passos: em primeiro lugar, ele irá buscar a rede utilizando o método desenvolvido noutra funcionalidade; de seguida, irá chamar um predicado que remove os amigos do jogador atual da lista retornada no método anterior; após a remoção, este irá chamar um predicado que irá percorrer recursivamente a lista retornada e identificar os jogadores que tem *tags* em comum com o utilizador e filtrar a lista conforme; por último, irá chamar um predicado que irá verificar para cada utilizador com *tags* em comum da sua rede, quais apresenta um caminho que contém uma *tag* em comum. Por exemplo, se uma das *tags* em comum for "música", este vai tentar encontrar um caminho na rede em que todos os nós intermédios tenham essa *tag*, no caso de haver várias *tags* em comum, este irá pesquisar separadamente para cada *tag*, não aceitando caminhos com mistura de *tags*, a pedido do cliente (como podemos verificar na Figura 13).

Na Figura 14, temos o resultado da pesquisa feita na base de conhecimento completa e fornecida pelo cliente, com o "1" (ana) como jogador atual, "2" como nível e "L" como lista de retorno com as sugestões. Nesta rede todos os utilizadores do nível enviado têm a *tag* "natureza", resultando a lista da figura e encontrando caminhos para os utilizadores lá representados. Para melhor teste desta funcionalidade (Figura 15), os autores recomendam a utilização da base de conhecimento parcial onde, para o segundo nível, ele não aceita um dos utilizadores que não tem *tags* em comum e, para outro utilizador, procura um caminho alternativo apesar deste ter caminhos sem *tags* em comum.

```
suggest_players(Player, Level, SuggestedPlayersList):-
    network_getNetworkByLevel(Player, Level, NetworkList),
    suggest_removeFriends(Player, NetworkList, CandidateList),
    suggest_getRelatedPlayers(Player, CandidateList, RelatedPlayersList),
    suggest_checkSuggestedPaths(Player, RelatedPlayersList, SuggestedPlayersList).
```

Figura 13 - Sugerir Utilizadores

```
?- suggest_players(1, 2, L).
L = [24, 23, 22, 21]
```

Figura 14 - Sugestão de Utilizadores Resultado

```
?- suggest_players(1, 2, L).
L = [5, 4]
```

Figura 15 - Sugestão (Base de Conhecimento Parcial)

6. DETERMINAÇÃO DO CAMINHO MAIS FORTE

6.1. Funcionalidade desenvolvida por Tiago Costa (1191460)

Nesta funcionalidade, o predicado chamado irá ser o *strongest_route*/3. Os parâmetros para este predicado serão, em primeiro lugar, o nome do jogador atual (a origem do caminho); em segundo lugar, o nome do jogador objetivo (o destino do caminho retornado); por último, o terceiro parâmetro será referente ao caminho retornado e preenchido pelo predicado, sendo este caminho o caminho com maior número de força de ligação total e bidirecional. Este predicado chama o predicado *strongest_findRoute*/2, que tem como objetivo encontrar (através da chamada a outros predicados) o caminho mais forte e atualizá-lo no facto dinâmico *strongest_currentRoute*/2, bem como após determinar um caminho, através do *fail*, fazer *backtracking* (Figura 22) e ir buscar caminhos alternativos que ainda não tinha determinado, comparando os com o melhor caminho atual no facto dinâmico através do predicado *strongest_updateRoute*/2. Este método irá chamar um *dfs* modificado especificamente para o cálculo do caminho mais forte. Este predicado irá ter outro predicado auxiliar recursivo para determinar o caminho mais seguro, representado na Figura 23.

Para a base de conhecimento completa (Figura 24), com os parâmetros: "ana" como jogador atual, "eduardo" como jogador objetivo e "L" como lista de retorno, este método retorna um caminho com 118 de força de ligação total e que irá percorrer o caminho retornado na Lista "L", com um tempo de execução de 52.59s. Com este caminho podemos verificar que este método não seria bastante prático no mundo real pois para uma ligação direta como "ana" e "eduardo" (representação gráfica na Figura 4), ele irá retornar um caminho bastante longo, que num contexto real, não seria prático, mas foi este o requisito do cliente para este projeto. Em suma, o método retorna no quatro parâmetro o caminho com maior força de ligação total que tem forças de ligação bidirecionais.

```
strongest_findRoute(Orig, Dest):-
    asserta(strongest_currentRoute(_, 0)),
    strongest_dfs(Orig, Dest, Strength, PathList),
    strongest_updateRoute(Strength, PathList),
    fail.
```

Figura 16 - Backtracking e fail (Caminho mais forte)

```
strongest_dfsAux(Dest, Dest, AuxList, 0, Path):-!,reverse(AuxList, Path).
strongest_dfsAux(Current, Dest, AuxList, Strength, Path):-
    node(CurrentID, Current, _),
    (connection(CurrentID, FriendID, StrengthA, StrengthB);
    connection(FriendID, CurrentID, StrengthA, StrengthB)),
    node(FriendID, Friend, _),
    \+ member(Friend, AuxList),
    strongest_dfsAux(Friend, Dest, [Friend | AuxList], SX, Path),
    Strength is (SX + StrengthA + StrengthB).
```

Figura 17 - Dfs modificado para o caminho mais forte

```
?- strongest_route(ana, eduardo, L).
Time:52.58835411071777
Strength:118
Solution path:[ana,rodolfo,rita,sara,diogo,maria,cesar,anabela,isaura,andre,ernesto,catia,isabel,
daniel,jose,carlos,luisa,antonio,eduardo]
L = [ana, rodolfo, rita, sara, diogo, maria, cesar, anabela, isaura|...].
```

Figura 18 – Resultado do caminho mais forte na base de conhecimento completa

7. DETERMINAÇÃO DO CAMINHO MAIS CURTO

7.1. Funcionalidade Desenvolvida por Jéssica Alves (1190682)

A funcionalidade de determinar o caminho mais curto, apresenta duas opções de escolha: a primeira, que utiliza o predicado *findall*/3, sendo este mais ineficiente, estando este algoritmo presente no predicado *shortest_allDfs*/3 (Figura 19); e a segunda, recomendada pelos autores, que é chamada no predicado *shortest_route*/3 que não utiliza o *findall*, sendo bastante mais eficiente do que o previamente descrito. Ambas as soluções recebem como parâmetros o *id* do jogador atual (origem), o *id* do jogador objetivo (destino) e a lista de retorno do caminho calculado, respetivamente. Para calcular o caminho, os predicados utilizam um predicado shortest_*dfs*/3 que determina o caminho com base no menor número de ligações, percorrendo cada nó e obtendo um caminho até a origem, com uma condição de paragem quando o nó atual for o nó referente ao jogador objetivo. Depois irá, através do *fail* e de *backtracking*, comparar os caminhos obtidos e verificar qual deles tem menor número de ligações, ou seja, que lista tem menor tamanho e guardar no facto dinâmico *shortest_currentRoute*/2, sendo esta verificação e atualização feita no predicado *shortest_updateRoute*/2. A diferença de eficiência entre as duas soluções é estudada no capítulo da complexidade e aí pode-se verificar a diferença entre os dois predicados.

Utilizando a base de conhecimento completa, com os parâmetros: "1" (ana) como jogador atual, "200" (sara) como objetivo e "L" como lista de retorno do caminho, obtém-se como seria de esperar, o caminho [ana (1), rodolfo (51), rita (61), sara (200)], como podemos verificar na Figura 20. Em contrapartida, na Figura 21, podemos verificar que utilizando o predicado que usa *findall*/3, para a mesma rede, a *stack* ultrapassa o limite.

```
shortest_allDfs(Player1, Player2, PathList):- get_time(T1),
    findall(Path, shortest_dfs(Player1, Player2, Path), PathList),
    length(PathList, PathLength),
    get_time(T2),
    write(PathLength),write(' paths found in '),
    T is T2-T1,write(T),write(' seconds'),nl,
    write('Possible Path List: '),write(PathList),nl,nl.
```

Figura 19 - Caminho Mais Curto Com Findall

```
?- shortest_route(1, 200, L).
Solution generation time:50.54819107055664
L = [1, 51, 61, 200].
```

Figura 20 - Caminho Mais Curto Resultado

```
?- shortest_allDfs(1, 200, L).
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 5Kb, global: 13Kb, trail: 0Kb
ERROR:   Stack depth: 16, last-call: 25%, Choice points: 25
ERROR:   In:
ERROR:     [16] system:'$add_findall_bag'([length:16])
ERROR:     [15] '$bags':findall_loop([length:16], <compound (:)/2>, _3506, [])
ERROR:     [14] system:setup_call_catcher_cleanup(<compound (:)/2>, <compound (:)/2>, _3542, <compound (:)/2>)
ERROR:     [10] user:shortest_allDfs(1, 200, _3584)
ERROR:     [9] '$toplevel':toplevel_call(<compound (:)/2>)
ERROR:
ERROR: Use the --stack_limit=size[KMG] command line option or
ERROR: ?- set_prolog_flag(stack_limit, 2_147_483_648). to double the limit.
```

Figura 21 - Caminho Mais Curto Resultado com Findall

8. DETERMINAÇÃO DO CAMINHO MAIS SEGURO

8.1. Funcionalidade Desenvolvida por Pedro Santos (1190967)

Nesta funcionalidade, o predicado chamado irá ser o *safest_route*/4. Os parâmetros para este predicado serão, em primeiro lugar, o *id* do jogador atual (a origem do caminho); em segundo lugar, o *id* do jogador objetivo (o destino do caminho retornado); depois é requerido o limite mínimo para a força de ligação em ambas as direções (bidirecional), descartando caminho com valor inferior a este; por último, o quarto parâmetro será referente ao caminho retornado e preenchido pelo predicado, sendo este caminho o caminho com maior número de força de ligação total e biderecional que cumpre os requisitos de força mínimos. Este predicado chama o predicado *safest_findRoute*/3, que tem como objetivo encontrar (através da chamada a outros predicados) o caminho mais seguro e atualizá-lo no facto dinâmico *safest_currentRoute*/2, bem como após determinar um caminho, através do *fail*, fazer *backtracking* (Figura 22) e ir buscar caminhos alternativos que ainda não tinha determinado, comparando os com o melhor caminho atual no facto dinâmico através do predicado *safest_updateRoute*/2. Este método irá chamar um *dfs* modificado especificamente para o caminho mais seguro, tendo em conta o limite de força de conexão. Este predicado irá ter outro predicado auxiliar recursivo para determinar o caminho mais seguro, representado na Figura 23. Como podemos verificar, este tem em conta as conexões bidirecionais requisitadas pelo enunciado e verifica, para cada uma delas verifica se a força de ligação é superior ao limite.

Para a base de conhecimento completa (Figura 24), com os parâmetros: “1” (ana) como jogador atual, “11” (antonio) como jogador objetivo, “2” como limite minimio de forca de ligação e “L” como lista de retorno, este método retorna um caminho com 67 de força de ligação total e que irá percorrer o caminho retornado na Lista “L”, com um tempo de execução de 0.0s. Com este caminho podemos verificar que este método não seria bastante prático no mundo real pois para uma ligação direta como “ana” e “antonio” (representação gráfica na Figura 4), ele irá retornar um caminho bastante longo, que num contexto real, não seria prático, mas foi este o requisito do cliente para este projeto. Outro exemplo, será a execução do caminho entre “1” (ana) e “200” (sara) com limite mínimo de 4, pois este não encontra caminho devido ao facto de existir uma ligação para chegar

ao nó de destino que tem uma força menor que o limite fornecido (Figura 26), não retornando caminho e informando o utilizador disso (Figura 25). Em suma, o método retorna no quatro parâmetro o caminho com maior força de ligação total que tem forças de ligação bidirecionais acima do limite fornecido no terceiro parâmetro.

```
safest_findRoute(Orig, Dest, Threshold):-
    asserta(safest_currentRoute(_,-10000)),
    safest_dfs(Orig, Dest, Threshold, Strength, PathList),
    safest_updateRoute(Strength, PathList),
    fail.
```

Figura 22 - Backtracking e Fail (Caminho Mais Seguro)

```
safest_dfsAux(Dest, Dest, AuxList, _ , Strength, Strength, Path):-!, reverse(AuxList,Path).
safest_dfsAux(Current, Dest, AuxList, Threshold, Strength, ReturnStrength, Path):-
    node(CurrentID,Current,_),
    (connection(CurrentID, FriendID, StrengthA, StrengthB);
    connection(FriendID, CurrentID, StrengthA, StrengthB)),
    node(FriendID, Friend, _),
    \+ member(Friend, AuxList),
    StrengthA >= Threshold,
    StrengthB >= Threshold,
    CurrentStrength is Strength + StrengthA + StrengthB,
    safest_dfsAux(Friend, Dest, [Friend | AuxList], Threshold, CurrentStrength, ReturnStrength, Path).
```

Figura 23 - DFS modificado para o caminho mais seguro

```
?- safest_route(1, 11, 2, L).
Time:0.0
Strength: 67
L = [1, 12, 23, 32, 41, 34, 43, 33, 21|...].
```

Figura 24 - Resultado do caminho mais seguro na base de conhecimento completa

```
?- safest_route(1, 200, 4, L).
No path found.
false.
```

Figura 25 - Exemplo de caminho não encontrado devido à falta de forças de ligação inferiores ao limite

```
connection(1,51,6,2).
connection(51,61,7,3).
connection(61,200,2,4).
```

Figura 26 - Conexões entre "ana" e "sara" com força menor que 4

9. ESTUDO DA COMPLEXIDADE DO PROBLEMA DA DETERMINAÇÃO DE CAMINHOS

9.1. Caminho Mais Curto Bidirecional (sem Findall)

Nº de Camadas Intermédias (sem nós de origem e destino)	**Nº de Nós por camada**	**Nº de soluções**	**Tempo para gerar a solução com menor nº de ligações**
1	1	1	0.0s
2	2	8	0.0s
3	3	891	0.06989s
4	4	8184064	134.873291s
5	5	-	Mais de 7h

Resultados acima de cinco camadas intermédias ultrapassam as 7h de execução, tendo o grupo terminado este estudo neste número.

9.2. Caminho Mais Curto Bidirecional (com findall)

Nº de Camadas Intermédias (sem nós de origem e destino)	**Nº de Nós por camada**	**Nº de soluções**	**Tempo para gerar a solução com menor nº de ligações**
1	1	1	0.0s
2	2	8	0.0s
3	3	891	0.011994s
4	-	-	*Stack Limit*

Resultados acima de quatro camadas intermédias chega ao *stack limit*, tendo o grupo terminado este estudo neste número.

9.3. Caminho Mais Seguro (Limite Menos Restrito)

Nº de Camadas Intermédias (sem nós de origem e destino)	Nº de Nós por camada	Nº de soluções	Tempo para gerar a solução com menor nº de ligações
1	1	1	0.0s
2	2	8	0.0s
3	3	891	0.007994s
4	4	8184064	147.586855s
5	5	-	Mais de 10h

Para este estudo foi utilizado um limite mínimo de -10, ou seja, para testar a componente de encontrar os caminhos todos. Resultados acima de cinco camadas intermédias ultrapassam as 10h de execução, tendo o grupo terminado este estudo neste número.

9.4. Caminho Mais Seguro (Forças de Ligação Positivas)

Nº de Camadas Intermédias (sem nós de origem e destino)	Nº de Nós por camada	Nº de soluções	Tempo para gerar a solução com menor nº de ligações
1	1	0	0.0s
2	2	0	0.0s
3	3	4	0.0s
4	4	734	0.061983s
5	5	-	Mais de 10h

Para este estudo foi utilizado um limite mínimo de 0, ou seja, apenas os caminhos com forças positivas seriam considerados, assim testando a componente do limite. Os resultados com nenhuma solução existem devido a existirem caminhos, mas com forças inferiores ao limite, assim podemos verificar o algoritmo em ação. Resultados acima de cinco camadas intermédias ultrapassam as 10h de execução pois as conexões adicionadas tinham forças positivas, tendo o grupo terminado este estudo neste número.

9.5. Caminho Mais Forte Bidirecional

Nº de Camadas Intermédias (sem nós de origem e destino)	**Nº de Nós por camada**	**Nº de soluções**	**Tempo para gerar a solução com menor nº de ligações**
1	1	1	0.0s
2	2	8	0.0s
3	3	891	0.01s
4	4	8184064	101.18s
5	5	-	Mais de 10h

10. CONCLUSÕES

Primeiramente, ao comparar o estudo da complexidade com e sem a utilização do *findAll*, podemos concluir que os resultados mais eficientes são conseguidos sem a aplicação do mesmo. Neste estudo, também é possível demonstrar que quanto maior as camadas intermedias, o número de soluções e o tempo de as gerar aumenta exponencialmente.

Por outro lado, podemos afirmar que num contexto real, os caminhos mais forte e mais seguro não são práticos. Isto deve-se à existência de uma maior possibilidade de não aceitação.

Em suma, todos os métodos foram implementados no âmbito da unidade curricular de *ALGAV*. Estes têm a finalidade de suportar a aplicação desenvolvida no âmbito do projeto integrador, servindo como servidor de *HTTP* para a mesma. Todos os predicados e funcionalidades descritas no presente relatório foram também implementadas completamente utilizando o protocolo *HTTP*, bem como a importação da base de conhecimento através de uma base de dados remota.