Uma imagem com texto

Descrição gerada automaticamente

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**

1. Introdução 7

2. Consideração de estados emocionais para encontrar encontrar caminhos 7

2.1. Funcionalidade desenvolvida por todos os elementos do grupo 7

3. Sugestão de Grupos 8

3.1. Funcionalidade desenvolvida por Tiago Costa (1191460) e Beatriz Seixas (1190424) 8

4. Cálculo de novos valores dos estados emocionais 9

4.1. Funcionalidade Desenvolvida por Jéssica Alves (1190682) e Pedro Santos (1190967) 9

5. Determinação do Caminho Mais Forte 11

5.1. Funcionalidade desenvolvida por Tiago Costa (1191460) 11

6. Determinação do Caminho Mais Curto 13

6.1. Funcionalidade Desenvolvida por Jéssica Alves (1190682) 13

7. Determinação do Caminho Mais Seguro 15

7.1. Funcionalidade Desenvolvida por Pedro Santos (1190967) 15

8. Estudo da Complexidade do Problema da Determinação de Caminhos 17

8.1. Caminho Mais Curto Bidirecional (sem Findall) 17

8.2. Caminho Mais Curto Bidirecional (com findall) 17

8.3. Caminho Mais Seguro (Limite Menos Restrito) 17

8.4. Caminho Mais Seguro (Forças de Ligação Positivas) 18

8.5. Caminho Mais Forte Bidirecional 19

9. Conclusões 21

# Introdução

TODO

# Consideração de estados emocionais para encontrar caminhos

## Funcionalidade desenvolvida por todos os elementos do grupo

Foi adicionado a todos os métodos um parâmetro *boolean* denominado de *EmotionBool*. Caso este seja 1 o predicado emotion\_checkSameEmotion é executado chamando o *emotion\_getMax* que retorna a emoção com o valor mais alto para um *player*. De seguida, as duas emoções são comparadas, retornando o *false* se foram diferentes ou *true* se forem iguais.

# Sugestão de Grupos

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

TODO.

# Cálculo de novos valores dos estados emocionais

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

O predicado para o cálculo de novos valores dos estados emocionais de cada jogador apresenta duas variantes: uma para o cálculo das mesmas na alteração da força de relação (*emotion\_relationChange*/4) e outra para o cálculo de novos valores para as emoções relativo a sugestão de grupos (*emotion\_groupSuggestion*/4).

A primeira variante, recebe como parâmetros: o id do jogar atual, sendo este utilizado para ir pesquisar os valores dos estados emocionais atuais; o valor da força de relação, referente à diferença entre o número de *likes* e dislikes; e dois valores referentes ao retorno das emoções Alegria (*Joy*) e Angústria (*Anguish*) que serão as emoções afetadas por esta funcionalidade. Este obtém, através do id, os valores para a Alegria e Angústia do jogador atual, verifica se este é superior a zero (se este apresenta mais *likes* do que *dislikes*), se sim irá se aumentar a alegria e diminuir a angústia, devido a alteração positiva na força de relação. Caso o contrário aconteça, a angústia será aumentada e a alegria será diminuída. Para os aumentos e diminuições de cada estado emocional, foram utilizadas duas fórmulas. A fórmula de aumento de uma emoção (*emotion\_increase*/4), representada na FIGURA, utiliza o valor anterior de uma determinada emoção, valores de saturação (no nosso caso 200 para a força de relação) e o valor da diferença de *likes* e *dislikes* passado por parâmetro, sendo a sua implementação visível na FIGURA. Já na fórmula para a diminuição (FIGURA) de uma emoção, os mesmos parâmetros serão utilizados, mas de uma forma diferente para obter a diminuição da mesma, sendo a sua implementação representada na FIGURA (*emotion\_decrease*/4). Após o cálculo das mesmas, o predicado irá retirar o facto *occ/7* do jogador atual e atualizá-lo com os novos valores.

De outra forma, a segunda variante, responsável pelo cálculo das mesmas na sugestão de grupos, recebe por parâmetro os seguintes valores: o id do jogador atual (à semelhança do primeiro predicado); o número de *tags* pretendido, utilizado no método da sugestão de grupos; da mesma forma, o número de jogadores, também para passar por parâmetro para a sugestão de grupos; as tags obrigatórias com o mesmo propósito; e, por último, os valores de retorno para as emoções a recalcular neste predicado, sendo estas a Esperança (Hope), Deceção (Deception), Medo (Fear) e Alívio (Relief). Para este predicado foram adicionados dois novos factos, *hope*/2 e fear/2, onde está representado o id do jogador e o jogador que este tem esperança ou medo que seja sugerido, assim cada jogador pode definir que tem medo de que outro seja sugerido, contribuindo para o par de emoções medo/alívio ou mesmo que espera que um jogador seja sugerido, afetando o par de emoções esperança/deceção. Assim, o método em questão chama o predicado *common\_tags*/6, para obter o grupo de utilizadores sugeridos, sendo depois chamados os predicados *emotion\_checkHope*/4 e *emotion\_checkFear*/4 para ser feito o cálculo dos dois pares de emoções afetados. O predicado *emotion\_checkHope*/4 recebe o id do jogador, a sugestão de grupo e os valores antigos para a esperança e deceção para percorrer a lista de jogadores sugeridos, através do predicado *emotion\_countHope*/4, e contar quantos utilizadores que o jogador atual tinha esperança de que fossem sugeridos, identificados no facto *hope*/2. Este valor irá ser utilizado no quociente do novo cálculo da emoção, sendo este feito com número de jogadores desejados a dividir pelo número de jogadores sugeridos relativo à emoção esperança. Este valor é depois transformado, através do seu complementar, para a emoção deceção, por exemplo, se três jogadores dos que o utilizador esperasse que fossem sugeridos, estivessem presentes na lista de sugestão (tendo esta cinco elementos), o quociente do cálculo da emoção esperança seria 3/5, enquanto o da deceção seria 2/5 (3 – 5 = 2). Neste caso, como o numerador do quociente da esperança é maior do que o denominador da deceção, aumentamos a esperança e diminuímos a deceção. Se o contrário acontecesse, a deceção seria aumentada e a esperança diminuída. O mesmo se aplica à execução do predicado *emotion\_checkFear*/4 que segue os mesmos princípios para o par de emoções medo/alívio. Por último, este retira o facto *occ*/7 do jogador atual e atualiza os seus valores com os novos calculados neste predicado.

Um exemplo da utilização destes métodos, seria os exemplos em baixo representados onde calculamos os novos valores para a alegria e angústia relativos a uma diferença de *likes* e *dislikes* de -100, aumentando a angústia para 0.75 e diminuindo a alegria para 0.25. Por outro lado, na sugestão de grupos, para o utilizador com o id de 1 na base de conhecimento parcial, 2 como número de *tags* e jogadores, *a* e *b* como *tags* obrigatórias, obtemos novos valores para os dois pares de emoções relativos à sugestão de grupos. A deceção aumenta devido à nenhum dos jogadores esperados serem sugeridos, levando ao quociente da esperança ser 1/3 (a lista de sugestão apresenta 3 elementos) e, por sua vez, o da deceção ser 2/3, sendo o novo valor da deceção 0.83, e a esperança é diminuída com o quociente 1/3, levando ao seu novo valor de 0.33. O mesmo se aplica ao par de emoções medo/alívio, onde o quociente do medo é 1/3, devido a apenas 1 jogador que o utilizador atual definiu que tinha medo de este ser sugerido estar presente no grupo de sugestão, levando à sua diminuição. Por outro lado, o quociente do alívio é 2/3 devido ao calculo complementar com o medo.

Text

Description automatically generated

Text

Description automatically generated

# Determinação do Caminho Mais Forte

## 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.

Uma imagem com texto, interior

Descrição gerada automaticamente

Figura - Backtracking e fail (Caminho mais forte)

Figura - Dfs modificado para o caminho mais forte

Uma imagem com texto

Descrição gerada automaticamente

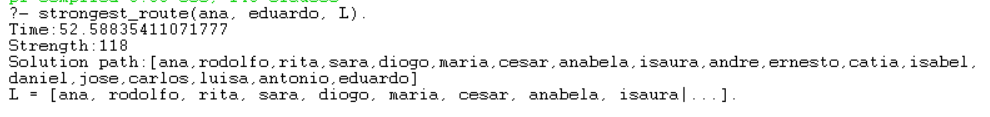


Figura – Resultado do caminho mais forte na base de conhecimento completa

# Determinação do Caminho Mais Curto

## 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.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Caminho Mais Curto Com Findall



Figura - Caminho Mais Curto Resultado

Uma imagem com texto

Descrição gerada automaticamente

Figura - Caminho Mais Curto Resultado com Findall

# Determinação do Caminho Mais Seguro

## 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.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Backtracking e Fail (Caminho Mais Seguro)

Uma imagem com texto

Descrição gerada automaticamente

Figura - DFS modificado para o caminho mais seguro



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



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

Uma imagem com texto, laranja

Descrição gerada automaticamente

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

# Estudo da Complexidade do Problema da Determinação de Caminhos

## 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.

## 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.

## 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.

## 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.

## 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 |

# 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.