



UNIVERSIDADE DO MINHO
Licenciatura em Engenharia Informática

INTELIGÊNCIA ARTIFICIAL

Resolução de Problemas - Algoritmos de Procura

Grupo 08

Inês Ferreira - A97040

Inês Castro - A95458

Marta Sá - A97158

José Ferreira - A97642

22 de dezembro de 2022

Conteúdo

1	Introdução	2
2	Objetivo Geral	3
3	Formulação do Problema	4
3.1	Estratégias adotadas	4
3.1.1	Procura Não-Informada	4
3.1.2	Procura Informada	5
4	Circuitos	7
5	Algoritmo de Pesquisa	9
5.1	1 ^o Fase - BFS	9
5.2	2 ^a Fase - Teste e Melhor algoritmo	9
6	Programa	14
6.1	Execução do Programa	14
6.2	Multi-Jogador	14
6.2.1	Possibilidade de dois participantes se dirigirem para a mesma célula da pista	15
6.3	Representação da pista em forma de grafo	15
6.4	Passagem por Paredes	16
6.5	Caminho Não Encontrado	16
7	Conclusão	17

1 Introdução

Este relatório foi redigido com o intuito de suportar o projeto de Inteligência Artificial para o ano letivo 2022/2023. O objetivo deste é aplicar as noções e algoritmos de pesquisa informada e não-informada lecionadas nesta mesma UC. Para tal, foi proposta a concretização de um projeto de grupo para o desenvolvimento em duas fases de um sistema de simulação de corridas de carros (simplificado) usando os respectivos conceitos aprendidos.

2 Objetivo Geral

Como segunda e última fase do processo este constitui-se em simular uma corrida inspirada em *VectorRace* com algoritmos de procura lecionados durante o semestre. Fazemos a leitura de um ficheiro em formato .txt que contenha um circuito e a partir dessa leitura, o mesmo deve ser representado num grafo. Deve ser escolhido um dos algoritmos lecionados na UC (de pesquisa informada ou não-informada) para determinar o melhor caminho entre um ponto de partida (estado inicial) - P - e um ponto final (estado final) - F - para cada jogador, apresentando o custo de percorrer esse mesmo caminho. Caso não haja algum caminho possível deve ser identificada essa situação.

3 Formulação do Problema

O movimento do carro relaciona-se fortemente com as acelerações deste a cada momento (cada estado). Considerando l como uma certa linha e c uma certa coluna para um vetor do espaço R^2 , o carro pode sofrer acelerações dentro deste conjunto - $\{-1,0,+1\}$ - em ambas as direções do seu movimento (linha e coluna). Consequentemente, representamos a aceleração em cada instante da seguinte forma - (a_l, a_c) .

Tendo em conta um p como o tuplo que indica a posição de um carro numa determinada jogada j temos então que $p_j = (p_l, p_c)$, e o tuplo v que indica a velocidade do carro nessa jogada $v_j = (v_l, v_c)$, na seguinte jogada o carro irá estar na posição:

$$\begin{aligned} p_l^{j+1} &= p_l^j + v_l^j + a_l \\ p_c^{j+1} &= p_c^j + v_c^j + a_c \end{aligned}$$

A velocidade do carro a cada instante é calculada pela seguinte expressão:

$$\begin{aligned} v_l^{j+1} &= v_l^j + a_l \\ v_c^{j+1} &= v_c^j + a_c \end{aligned}$$

Sendo uma simulação de corrida de carros, existe a possibilidade do carro sair da pista, sendo que quando essa situação ocorrer o carro terá de voltar para a posição anterior, assumindo um valor de velocidade zero. Cada movimento do carro numa determinada jogada, de uma determinada posição para outra, terá um custo de 1 unidade, sendo que quando o mesmo sair dos limites da pista, o custo é de 25 unidades.

3.1 Estratégias adotadas

Iremos explicar nesta secção as estratégias implementadas pelo grupo.

3.1.1 Procura Não-Informada

Como problema de pesquisa, pretendemos então que seja encontrado o caminho mais curto da partida (P) até à meta (F).

Para as procuras não-informadas, o grupo implementou o algoritmo de procura onde alteramos o problema de pesquisa de acordo com as implicações que esta procura acarreta.

1. **Estado Inicial:** Carro no ponto de Partida (P);
2. **Estado Final:** Carro no ponto de Chegada (F);
3. **Objetivo:** Chegar ao ponto de chegada percorrendo o caminho mais curto;

4. **Pré-Condições:** Aceleração = 0 e $v_c = v_l = \{-1, 0, 1\}$; A velocidade inicial é sempre zero.
5. **Custo:** Depende do circuito e do algoritmo utilizado.

Neste caso, a aceleração é considerada 0, uma vez que estes algoritmos BFS e DFS apenas visam percorrer o grafo com o intuito de encontrar uma das soluções ótimas (com menor custo) existentes no mesmo (é um algoritmo completo). Para o BFS, a solução ótima poderá existir quando o grafo não tem pesos ou estes são uniformes como é esta situação onde cada nodo tem um peso de 1 e evita os locais considerados obstáculos em que o peso é 25. No entanto, o DFS devolve a primeira solução encontrada.

Com o algoritmo BFS, este percorre os nodos nível a nível, procurando sempre através dos seus adjacentes aquele para qual se pode mover, sendo que atribuição da aceleração ao movimento feito pelo carro não coincidiria com a forma com que este algoritmo funciona.

Para o algoritmo de DFS, este percorre o grafo expandido cada nodo para cada nodo que localiza, não sendo importante o uso da aceleração nesta situação.

Ao considerarmos a aceleração nula, isto implica que a procura seja feita sempre nos nodos adjacentes. Consequentemente, isso garante que entre quaisquer duas posições não-adjacentes do caminho considerado correto, todos os nodos pelos quais passamos são possíveis, isto é, não são obstáculos e não estão fora dos limites da pista.

3.1.2 Procura Informada

Tanto para procuras Não Informadas como Informadas o problema de pesquisa é o mesmo, chegar à meta desde o ponto de partida percorrendo o caminho com o menor custo.

1. **Estado Inicial:** Carro no ponto de Partida (P);
2. **Estado Final:** Carro no ponto de Chegada (F);
3. **Objetivo:** Chegar ao ponto de chegada percorrendo o caminho mais curto;
4. **Pré-Condições:** $a_c = a_l = \{-1, 0, 1\}$ e $v_c = v_l = \{-1, 0, 1\}$; A velocidade inicial é sempre zero.
5. **Custo:** Depende do circuito e do algoritmo utilizado.

O algoritmo A* em cada passo, escolhe o nodo de acordo com os valores da distância do nodo atual até à meta e tendo em conta o custo do caminho até ele. Estes dois valores são somados e associados ao nodo como nova heurística. Assim, o algoritmo vai percorrendo o grafo escolhendo o nodo com a menor heurística.

O algoritmo Greedy, por sua vez, para cada nodo do grafo visita o nodo mais "próximo" que não foi visitado, isto é, o nodo cuja distância euclidiana até à meta é menor. Sabemos então que este algoritmo nem sempre encontra a melhor solução, visto que as suas escolhas ou o caminho feito não poderá ser alterado.

4 Circuitos

Relativamente à geração de circuitos, representamos os mesmos em ficheiro de texto. Por exemplo, seguem-se os seguintes 3 ficheiros com diferentes características:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X-----X-----XXXXX-----F
X-XXXX-----XXXXX-----X
X--XXX-----XXXXX-----X
X-PXXX-----XXXXXXXXX-----X
X--XXX-----XXXXXX-----X
X-----XXXXX-----X
X-----XXXXX--XXXXX--X
X-----X
X-----XXXXXXXX--X
X-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Figura 1: Circuito.txt - 12 linhas e 32 colunas

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X-----X
F-----X
F-----X
F-----X
X-----X
X-----X
X-----X
X-----XXXXXXXX--X
X-----X-----X
X-----X-P-----X
X-----X-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Figura 2: Circuito2.txt - 13 linhas e 32 colunas


```

IA > ≡ circuito3.txt
 1  XXXXXXXXXXXXXXXXXXXX
 2  XXX-----XXX-----X
 3  X-----XXX-----X
 4  XP-----XXXXX-----X
 5  X---XXXXX-----X
 6  X-----XXXXXXXXX
 7  X---XXXXXX-----X
 8  X-----F
 9  XXXXXXXX-----XXX
10  XXXXXXXXXXXXXXXXXXXX

```

Figura 3: Circuito3.txt - 10 linhas e 19 colunas

Nos circuitos criados com diversos tamanhos (como podemos conferir pelas legendas das imagens), temos que os o ‘-‘ representa a pista, o ‘X’ representa fora da pista e/ou obstáculos, o P a partida e o F a meta.

No circuito após a corrida apareceram números em células de acordo com o *ranking* de cada jogador (1,2,3,4). Este *ranking* tem em conta o custo final, isto é, o custo do caminho calculado por cada algoritmo escolhido associado ao custo resultante das colisões. Desta forma, na primeira posição encontra-se o jogador com o menor custo final. Caso mais do que um jogador passe na mesma célula, o programa imprime o caracter ‘C’ nessa posição.

Definimos circuitos que têm sempre um caminho possível de se realizar, isto é, consegue-se chegar de um ponto de partida a um ponto de chegada e em que todas as linhas têm igual tamanho entre si, bem como as colunas.

5 Algoritmo de Pesquisa

5.1 1º Fase - BFS

Como maneira de encontrar uma solução para o problema do caminho mais curto, o grupo optou por utilizar um algoritmo de procura não informada, mais especificamente o BFS (*Breadth-First Search*).

O BFS foi escolhido, uma vez que, por ser um algoritmo de procura não informada, a sua implementação é simples. Para além disso, a procura realizada por este algoritmo é mais eficiente pois o caminho mais curto é obtido a partir de uma procura feita nível a nível.

O algoritmo de procura em largura (também conhecido em inglês por Breadth-First Search - BFS) é um algoritmo de procura usado para fazer uma travessia de um grafo. A procura começa pelo nodo raiz (neste caso o nodo que contém as coordenadas da partida) e explora os nodos vizinhos. Então, para cada um desses nodos, exploramos os seus vizinhos, ou seja, os nodos que se encontram no mesmo nível até que encontremos o nodo final.

Este algoritmo é um método de busca não-informada que expande e examina metodicamente os nodos de um grafo. Isto é, podemos dizer que o algoritmo realiza uma procura num grafo passando por todos os seus nodos. O algoritmo garante que nenhum nodo é visitado mais de uma vez e, para isso, utiliza uma estrutura de dados *array*. Desta forma, a visita aos nodos é realizada através da ordem de chegada no *array* e um nodo que já foi visitado não poderá entrar novamente nesta estrutura.

5.2 2ª Fase - Teste e Melhor algoritmo

Para esta última fase, foram testados os quatro algoritmos de procura que foram implementados com o ficheiro "circuito.txt". Em seguida, estão apresentados os algoritmos, com o tempo que levou a procura do caminho mais curto, o circuito com o caminho que cada jogador tomou e o custo.

```

Procura executada em:
0.008228
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
X----X-----XXXXX-----1
X-XXX-----XXXXX-----X
X-XXX-----XXXXX-----1-X
X-PXXX-----XXXXXXXXX-----X
X-1XXX-1-1-----XXXXX-----X
X-1-1-----1-XXXXX-1-----X
X-----XXXXX-XXXXX-X
X-----1-1-----X
X-----XXXXXXX-X
X-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Nome: qwe
Algoritmo: 0
Caminho: (4, 2) (5, 2) (6, 3) (6, 5) (5, 8) (5, 11) (6, 13) (8, 15) (8, 19) (6, 23) (3, 27) (1, 30)
Custo: 11

```

Figura 4: Circuito.txt - Algoritmo de Procura A*

```

Procura executada em:
0.007093000000000002
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
X----X-----XXXXX-----1
X-XXX-----XXXXX-----X
X-XXX-----XXXXX-----1-X
X-PXXX-----XXXXXXXXX-----X
X-1XXX-1-1-----XXXXX-----X
X-1-1-----1-XXXXX-1-----X
X-----XXXXX-XXXXX-X
X-----1-1-----X
X-----XXXXXXX-X
X-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Nome: qwe
Algoritmo: 1
Caminho: (4, 2) (5, 2) (6, 3) (6, 5) (5, 8) (5, 11) (6, 13) (8, 15) (8, 19) (6, 23) (3, 27) (1, 30)
Custo: 11

```

Figura 5: Circuito.txt - Algoritmo de Procura Greedy

```

Procura executada em:
0.006328
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
X----X-----XXXXX-----1
X-XXX-----XXXXX-----1X
X-XXX-1-XXXXX-----1-X
X-PXXX-1-1-XXXXXXX-----11-X
X-1XXX-11-1-----XXXXX1-1-X
X-1111-1-XXXXX11-1-X
X-----1XXXXX1-XXXXX-X
X-----1-111-----X
X-----1-XXXXXXX-X
X-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Nome: qwe
Algoritmo: 2
Caminho: (4, 2) (5, 2) (6, 3) (6, 4) (6, 5) (6, 6) (5, 7) (5, 8) (4, 9) (3, 10) (4, 11) (5, 12) (6, 13) (7, 14) (8, 15) (9, 16) (8, 17) (8, 18) (8, 19) (7, 20) (6, 21) (6, 22) (5, 23) (6, 24) (5, 25) (4, 26) (4, 27) (3, 28) (2, 29) (1, 30)
Custo: 29

```

Figura 6: Circuito.txt - Algoritmo de Procura BFS


```

Procura executada em:
0.026185999999999987
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X-----X
1--1-----1--1-----X
F-----X
F-----1-----1-----X
X-----1-----X
X-----X
X-----X
X-----XXXXXXXXX1-----X
X-----X-----1-----X
X-----X-P1-1-----X
X-----X-11-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Nome: qwe
Algoritmo: 1
Caminho: (10, 15) (10, 16) (11, 16) (11, 17) (10, 19) (9, 22) (8, 23) (5, 23) (4, 21) (2, 18) (2, 14) (4, 9) (2, 4) (2, 0)
Custo: 13

```

Figura 9: Circuito2.txt - Algoritmo de Procura Greedy

```

Procura executada em:
0.013323000000000008
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X-----X
F-----X
F-----X
1-----X
X1-----X
X-1-----11-----1--1-----X
X-1-1-1-1-1111-11-11-1-----X
X-1-1-----XXXXXXXXX1-----X
X-----X-1-----1-----X
X-----X-P1-1-11-----X
X-----X-----1-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Nome: qwe
Algoritmo: 2
Caminho: (10, 15) (10, 16) (9, 17) (10, 18) (11, 19) (10, 20) (10, 21) (9, 22) (8, 23) (7, 22) (6, 21) (7, 20) (7, 19) (7, 18) (6, 17) (7, 16) (7, 15) (6, 14) (7, 13) (7, 12) (7, 11) (7, 10) (6, 9) (6, 8) (7, 7) (8, 6) (7, 5) (8, 4) (7, 3) (6, 2) (5, 1) (4, 0)
Custo: 31

```

Figura 10: Circuito2.txt - Algoritmo de Procura BFS

```

Procura executada em:
0.0018139999999999962
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X-----X
F-----X
F-----X
1-----X
X1-----1-1-----X
X-1-----1-11-----1-----X
X-1-----111-11-1-----X
X-1-----11-XXXXXXX1-----X
X-111111-X-1-----1-----X
X-----1-X-P11-11-----X
X-----X-----111-----X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Nome: qwe
Algoritmo: 3
Caminho: (10, 15) (10, 16) (9, 17) (10, 17) (11, 18) (11, 19) (10, 20) (11, 20) (10, 21) (9, 22) (8, 23) (7, 23) (6, 22) (5, 21) (4, 20) (5, 19) (6, 18) (7, 18) (6, 17) (7, 16) (7, 15) (6, 14) (7, 13) (7, 12) (7, 11) (8, 11) (8, 10) (9, 10) (9, 9) (10, 9) (9, 8) (9, 7) (9, 6) (9, 5) (8, 4) (7, 3) (6, 2) (5, 1) (4, 0)
Custo: 38

```

Figura 11: Circuito2.txt - Algoritmo de Procura DFS

Concluimos, através destes dois exemplos, que de facto os algoritmos de procura informada produzem melhores resultados, e ainda entre estes podemos deduzir que o algoritmo A* é aquele que encontra o caminho com menor custo

para a meta. Apesar de ser aquele que leva mais tempo a ser executado, como estamos a falar de frações mínimas de segundo e de uma situação onde este fator não é prioridade, o algoritmo A^* é o favorito à procura do caminho menos custoso.

6 Programa

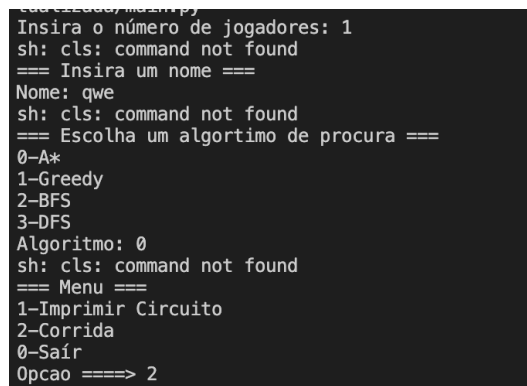
O nosso programa é composto por três ficheiros, nomeadamente *main.py* que contém a classe *main* (classe de teste), *grafo.py* que contém a classe *Graph* e o *jogador.py* que é composto pela classe *Player*. Por sua vez a classe *Player* é constituída pelas variáveis *nome*, *algoritmo*, *grafo*, *cost* e *caminho*.

6.1 Execução do Programa

O programa é executado a partir do comando:

```
py .\main.py {ficheiro do circuito}
```

Após o comando ser efetuado, deverá aparecer um menu como apresentado na figura seguinte:



```
Insira o número de jogadores: 1
sh: cls: command not found
=== Insira um nome ===
Nome: qwe
sh: cls: command not found
=== Escolha um algortimo de procura ===
0-A*
1-Greedy
2-BFS
3-DFS
Algoritmo: 0
sh: cls: command not found
=== Menu ===
1-Imprimir Circuito
2-Corrída
0-Sair
Opcao ==> 2
```

Figura 12: Menu principal

Podemos ver pela figura 4 o menu desenvolvido pelo grupo, onde primeiramente indicamos o número de jogadores que vão participar na corrida, havendo um limite de 4 jogadores. De seguida, para cada jogador indicamos o seu nome e o algoritmo de procura, não poderá ser repetido pelos jogadores na mesma corrida, que este irá utilizar durante a sua prestação. Posteriormente temos 3 opções: Imprimir Circuito, Corrida ou Sair.

6.2 Multi-Jogador

Tal como era pedido no enunciado, o grupo foi capaz de produzir o jogo da 1ª Fase, agora com a funcionalidade de haver mais do que 1 jogador. Neste momento é possível incluir até 4 jogadores cada um com um algoritmo de procura, nunca sendo repetidos entre jogadores na mesma corrida. Esta decisão foi tomada, uma vez que, o objetivo seria concluir qual dos algoritmos de procura produz melhores resultados. Estamos assim a competir entre algoritmos de procura.

6.2.1 Possibilidade de dois participantes se dirigirem para a mesma célula da pista

Com a implementação de multi-jogador foi importante ter em consideração o facto de os jogadores poderem dirigir-se para a mesma célula ao mesmo tempo, seria aplicada uma penalização de cerca de 10% do custo que o jogador tinha. Para tal, o grupo decidiu atribuir prioridades a cada participante. Deste modo, caso dois ou mais jogadores queiram aceder ao mesmo local da pista ao mesmo tempo, aquele cuja velocidade é maior terá prioridade sobre os outros. Caso os jogadores tenham a mesma velocidade máxima determinamos que aqueles que tiverem maior custo serão os penalizados. Para além disso, se os jogadores tiverem o mesmo custo então a penalização é atribuída a um jogador aleatório. Para este efeito temos uma lista com os jogadores por ordem crescente do seu custo.

6.3 Representação da pista em forma de grafo

A representação da pista em forma de grafo começa com a leitura e armazenamento do ficheiro que contém o circuito numa matriz. Para isto, recorremos à função *parse_ficheiro* que, ao receber como *input* o ficheiro, a posição inicial e uma lista, tem como *output* uma matriz de caracteres representativos do circuito, um nodo inicial (assinalado pelo carácter 'P') e uma lista com as coordenadas dos nodos finais (assinalados pelos caracteres 'F'). De seguida, utiliza-se estes dados para construir um objeto da classe *Graph*. Esta classe é, por sua vez, construída da seguinte forma:

```
self.start = partida
self.end = fim
self.circuito = circuito
self.possibilidades = [(-1,-1),(-1,0),(-1,1),(0,-1),(0,0),(0,1),(1,-1),(1,0),(1,1)]
self.grafo = self.criaGrafo(partida,(0,0))
self.largura = len(circuito[0])
self.altura = len(circuito)
```

Nesta definição, consideramos que a variável *possibilidades* corresponde a uma lista de tuplos. Esta, por sua vez, é utilizada para atribuir velocidades a algoritmos de procura não informada e acelerações a algoritmos de procura informada. Para além disso, utilizamos a função *geraGrafo* que atribui um dicionário à classe *Graph*, representativo do circuito.

Esta representação, como foi referido, é realizada a partir do método *geraGrafo* que, ao receber as coordenadas do ponto de partida e uma velocidade inicial (0,0), inicializa uma lista de tuplos com estes valores que representam os nodos a visitar. Desta forma, enquanto esta lista não for vazia, o algoritmo irá calcular para um dado determinado elemento da mesma, a próxima posição e velocidade. De seguida, irá analisar os resultados obtidos e, caso sejam válidos,

estes serão adicionados ao grafo com o custo 1 e, se necessário, à lista de nodos a visitar. Se os resultados não forem válidos então serão adicionados ao grafo com custo 25.

6.4 Passagem por Paredes

Achamos importante mencionar que o grupo tomou especial atenção em garantir que os algoritmos de procura informada têm em conta a existência de paredes no movimento do carro de uma posição para outra não imediatamente adjacente. Conseguimos fazer com que os nossos algoritmos de pesquisa informada prevejam se para chegarem a um determinado nodo vão atravessar paredes e se o próprio nodo final também é uma parede evitando estas situações e navegando para outros nodos. Tal podemos verificar nos resultados obtidos do `circuito2.txt`.

6.5 Caminho Não Encontrado

Na eventualidade de não haver nenhum caminho (exemplo: caso a meta esteja rodeada por paredes) o caminho aparecerá nulo e o custo 0. No caso de o circuito não ser válido (não haver ponto de partida ou ponto de chegada) o programa envia uma mensagem de erro e sai.

7 Conclusão

A elaboração deste trabalho permitiu aplicar conceitos lecionados nas aulas da unidade curricular de uma forma mais prática para o nosso futuro profissional.

O grupo acredita ter conseguido implementar os algoritmos de procura sugeridos de forma a este ser capaz de responder ao que foi pedido (o custo mínimo). Como esperávamos, apesar de alguns desafios, foi possível expandir as funcionalidades do projeto de modo a utilizar mais algoritmos de procura e permitir mais que um jogador. O grupo gostaria de ter implementado situações mais dinâmicas como obstáculos temporários, a criação de uma interface de utilizador mais apelativa, assim como acrescentar mais algoritmos de procura mas, infelizmente, não foi possível.

É de salientar que todo o código se encontra devidamente comentado de forma a esclarecer dúvidas que possam surgir quer sejam de carácter mais funcional (do enunciado) ou mais dedicadas à programação elaborada.

Por fim, o grupo empenhou-se e manteve-se focado no objetivo do trabalho e encontra-se satisfeito com o trabalho realizado.