

# Instituto Tecnológico de Aeronáutica

Departamento de Engenharia de Software  
Divisão de Ciência da Computação  
**Laboratório de CES-33**



## Relatório da atividade

### **Projeto 1:** Buscas

Lucas França de Oliveira, [lucas.fra.oli18@gmail.com](mailto:lucas.fra.oli18@gmail.com)  
Professor: Prof. Paulo André Castro , [pauloac@ita.br](mailto:pauloac@ita.br)

30 de agosto de 2017

# 1 Introdução

Este trabalho consiste no estudo da modelagem de problemas de Inteligência Artificial em um espaço de estados e na aplicação algoritmos de busca sobre esses estados.

Os códigos podem ser visualizados no repositório do github: [https://github.com/splucs/Lab1\\_CTC17.git](https://github.com/splucs/Lab1_CTC17.git)

## 2 Atividade: distância mínima heurística

Nesta atividade, foi fornecido pelo professor um arquivo que representa um sistema de transporte do Uruguay na forma de um grafo planar. O grafo possui 734 nós e é pedida a distância mínima entre os nós 203 e 600. Foi pedida implementação e teste de algoritmos gulosos e do algoritmo de caminho mínimo heurístico  $A^*$ .

### 2.1 Algoritmo guloso 1: busca em profundidade orientada pela heurística

Neste método, realiza-se uma busca em profundidade sobre o grafo. Cada nó  $u$  só é visitado uma única vez e suas arestas  $(u, v)$  são percorridas por ordem de distância entre a outra extremidade  $v$  e o destino  $dist(v, t) = h[v]$ . Este método executa com complexidade de pior caso  $O(V + E)$ , sendo o mais rápido em termos de complexidade assintótica de pior caso entre os algoritmos apresentados aqui. A ordenação das listas de adjacência teria pior caso  $O(E \log E)$ , mas pela entrada dada, o grau de saída de cada nó é no máximo 9.

### 2.2 Algoritmo guloso 2: busca por fila de prioridade orientada somente pela heurística

Neste método, adicionou-se a estrutura de dados da fila de prioridade, que permite armazenar um conjunto de nós, com tempo de inserção  $O(\log n)$ , acesso do mínimo segundo algum comparador em  $O(1)$  e remoção deste em  $O(\log n)$ . Para este caso, o comparador levará em conta apenas a distância entre o nó e o destino  $dist(u, t) = h[u]$ , ou seja, no topo da fila estará o elemento mais próximo em distância reta até o destino.

O algoritmo inicia colocando a origem  $s$  na fila com  $g[s] = 0$ . A seguir, executa-se o loop até que  $t$  saia da fila ou até que a fila esteja vazia: remove o menor elemento  $u$  pelo comparador da fila; cada nó  $v$  vizinho de  $u$  não adicionado na fila é adicionado; e define-se  $g[v] = g[u] + dist(u, v)$ , ou o menor valor encontrado para  $g[v]$  ao longo de todos os nós que incidem em  $v$ .

Este algoritmo executa com complexidade de pior caso  $O(V \log V + E)$  e não é ótimo.

### 2.3 Algoritmo de Dijkstra: busca por fila de prioridade orientada somente pela distância mínima da origem

Este é um algoritmo conhecido da literatura. É demonstradamente ótimo e funciona com complexidade de pior caso  $O(V \log V + E)$ . Funciona de forma de forma semelhante ao apresentado acima, porém de forma não heurística: o comparador da fila de prioridades leva em consideração apenas a distância mínima encontrada por um caminho da origem até o nó  $u$  ( $g[u]$ ). Isso faz com que um elemento  $u$ , ao ser removido da fila, possui armazenado em seu  $g[u]$  a distância mínima da origem até ele. Quando um elemento é adicionado na fila, já foi encontrado pelo menos um caminho até ele e, portanto, o comparador é válido.

## 2.4 Algoritmo A\*: busca por fila de prioridade orientada pela distância mínima da origem e pela heurística

Este algoritmo é um junção dos últimos dois algoritmos apresentados: ele utiliza a fila de prioridades com um comparador que depende tanto da distância mínima até um ponto  $g[u]$  como sua distância de linha reta até o destino  $h[u] = \text{dist}(u, t)$ . A combinação utilizada é a soma direta, que possui a seguinte propriedade: se a heurística sempre subestima a distância mínima até o destino, ou seja,  $h[u] \leq h'[u]$ , em que  $h'[u]$  é a distância mínima por algum caminho do grafo, então o algoritmo funciona de forma ótima. Tendo apenas uma alteração no comparador em termos de implementação em relação aos dois anteriores, sua complexidade de pior caso também é  $O(V \log V + E)$ .

## 2.5 Implementação e resultados

Os algoritmos apresentados foram implementados em C++ em um único arquivo. O programa é de console e lê o grafo de um arquivo na formatação padrão do arquivo "Uruguay.csv". Nos argumentos de linha de comando, pode-se passar, nesta ordem, o nome do arquivo de casos de teste, o índice do nó de origem e o índice do nó de destino. Caso estes não sejam especificados, eles são definidos por default para "Uruguay.csv 203 600". Para se fazer a medição de tempo, fez-se uma média de 1000 iterações. Foi contado também o número de nós processados.

O programa apresentou a seguinte saída default:

```
Opening "Uruguay.csv" ... done.
Reading start and end cities ... done: 203 600.
Reading graph ... done.
Running Dijkstra ... Result (optimal): 93.561, average time: 0.0003930, nodes
    processed: 599.
Running A* ... Result (sub-optimal): 93.561, average time: 0.0004720, nodes processed:
    594.
Running Greedy Heap ... Result (non-optimal): 183.479, average time: 0.0001560, nodes
    processed: 270.
Running Greedy DFS ... Result (non-optimal): 200.065, average time: 0.0001040, nodes
    processed: 181.
Closing file ... done.
```

## 3 Atividade: Jogo da Velha

Nesta atividade, foi solicitado pelo professor uma inteligência artificial capaz de jogar jogo da velha, assim como a interface para realizar os testes e partidas humano versus máquina. Foi feita a modelagem em espaço de estados: cada estado é definido pela configuração do campo e de quem é a vez. Foram implementadas 3 formas de inteligência para cada jogador, que serão explicadas a seguir.

### 3.1 Jogador-humano

Nesta forma o usuário visualiza o estado atual pela interface e insere via I/O a informação da próxima jogada.

### 3.2 Jogador-máquina com busca heurística

Nesta forma, a máquina decide qual a próxima jogada segundo a seguinte heurística: escolhe-se o estado adjacente que maximiza o número de retas em que o jogador atual pode ganhar menos o números de retas em que o oponente pode ganhar.

### 3.3 Jogador-máquina com busca completa sobre a árvore de decisão

Nesta forma, a máquina executa uma busca completa sobre a árvore de decisão utilizando o método minimax para determinar de forma ótima qual a próxima jogada. Nesta implementação, cada estado guarda se ele é uma posição de vitória, de empate ou de derrota. Como o jogo alterna em movimentos, os estados adjacentes são sempre do oponente. O movimento ótimo é, portanto, um que leva a um estado de derrota, de empate ou de vitória nesta ordem prioridade.

Cada uma das 9 células pode ser: vazia, X ou O, tendo-se  $3^9 = 19683$  estados. Diferenciando-se o turno de X e de O, tem-se  $2 \times 3^9 = 39366$  estados. Como, para o jogo da velha, não existem ciclos no grafo de transições de estados (um estado não pode voltar a ele mesmo), ele se configura como um DAG (Directed Acyclic Graph), e, portanto, pode-se usar o artifício da programação dinâmica para pré-calcular em tempo hábil o movimento ótimo para todos os estados, pois o número de estados é menor que 40000. Como pode haver mais de um movimento ótimo, a máquina escolhe aleatoriamente o próximo estado, de forma a dar a impressão de que os movimentos são mais variados perante um usuário humano. A primeira jogada é completamente aleatória, mas isso não torna a IA não ótima.

### 3.4 Implementação e resultados

Os métodos foram implementados em C++ em um único arquivo. O programa é de console. Ao se iniciar o programa, pergunta-se como funcionará cada jogador: o usuário insere dois números que identificam o tipo do primeiro (X) e do segundo (O) jogador, respectivamente: 0 para jogador humano, 1 para IA heurística, 2 para IA ótima. A seguir, inicia-se a partida. No final, pergunta-se se o usuário quer outra partida.

O input de uma jogada de um jogador humano é feita especificando a linha (0, 1 ou 2) e coluna (0, 1 ou 2), nesta ordem, da próxima jogada.

Segue um exemplo de execução com um duelo entre os dois métodos de IA:

```
Welcome to Tic Tac ToITA
```

```
By Lucas Franca de Oliveira , COMP-18.
```

```
Please input type of game players (two numbers):
```

```
0 = human, 1 = heuristic machine, 2 = complete search machine
```

```
1 2
```

---

---

```
Current game board. Turn: Machine
```

```
    0 1 2
0
1
2
```

---

---

```
Current game board. Turn: Machine
```

```
    0 1 2
0
1    X
2
```

---

---

Current game board. Turn: Machine

	0	1	2
0			
1		X	
2			O

---

---

Current game board. Turn: Machine

	0	1	2
0		X	
1		X	
2			O

---

---

Current game board. Turn: Machine

	0	1	2
0		X	
1		X	
2	O		O

---

---

Current game board. Turn: Machine

	0	1	2
0	X		X
1		X	
2	O		O

---

---

Final game board. Turn: Machine

	0	1	2
0	X		X
1		X	
2	O	O	O

VICTORY for player 2!

Would you like to play again (Y/N)?

n

Thank you for playing.

By Lucas Franca de Oliveira, COMP-18.

### 3.5 Testes de partidas contra o jogador humano

Segue um exemplo de vitória, empate e derrota de um jogador humano contra a máquina funcionando no modo heurístico:

Welcome to Tic Tac ToTA

By Lucas Franca de Oliveira, COMP-18.

Please input type of game players (two numbers):

0 = human, 1 = heuristic machine, 2 = complete search machine

0 1

---

---

Current game board. Turn: Human

0 1 2

0

1

2

You are X. Please insert your next move's row and column:

0 1

---

---

Current game board. Turn: Machine

0 1 2

0 X

1

2

---

---

Current game board. Turn: Human

0 1 2

0 X

1 O

2

You are X. Please insert your next move's row and column:  
2 0

---

---

Current game board. Turn: Machine

	0	1	2
0		X	
1		O	
2	X		

---

---

Current game board. Turn: Human

	0	1	2
0	O	X	
1		O	
2	X		

You are X. Please insert your next move's row and column:  
2 2

---

---

Current game board. Turn: Machine

	0	1	2
0		O	X
1			O
2	X		X

---

---

Current game board. Turn: Human

	0	1	2
0		O	X
1			O
2	X		X

You are X. Please insert your next move's row and column:  
2 1

---

---

Final game board. Turn: Machine

	0	1	2
0	O	X	O

```

1      O
2  X X X

```

VICTORY **for** player 1!

Would you like to play again (Y/N)?

y

Please input type of game players (two numbers):

0 = human, 1 = heuristic machine, 2 = complete search machine

0 1

---

---

Current game board. Turn: Human

```

      0 1 2
0
1
2

```

You are X. Please insert your next move's row and column:

0 0

---

---

Current game board. Turn: Machine

```

0 1 2
0 X
1
2

```

---

---

Current game board. Turn: Human

```

0 1 2
0 X
1 O
2

```

You are X. Please insert your next move's row and column:

2 2

---

---

Current game board. Turn: Machine

```

      0 1 2
0  X

```



```

1      O
2      X

```

---



---

Current game board. Turn: Human

```

      0 1 2
0  X   O
1     O
2     X

```

You are X. Please insert your next move's row and column:  
2 0

---



---

Current game board. Turn: Machine

```

0 1 2
0  X   O
1     O
2  X   X

```

---



---

Current game board. Turn: Human

```

0 1 2
0  X   O
1  O O
2  X   X

```

You are X. Please insert your next move's row and column:  
1 2

---



---

Current game board. Turn: Machine

```

      0 1 2
0  X   O
1  O O X
2  X   X

```

---



---

Current game board. Turn: Human

```

      0 1 2
0  X   O
1  O O X
2  X O X

```

You are X. Please insert your next move's row and column:  
0 1

---

---

Final game board. Turn: Machine

```

0 1 2
0  X X O
1  O O X
2  X O X

```

TIE!

Would you like to play again (Y/N)?

y

Please input type of game players (two numbers):

0=human, 1=heuristic machine, 2=complete search machine

0 1

---

---

Current game board. Turn: Human

```

0 1 2
0
1
2

```

You are X. Please insert your next move's row and column:

1 0

---

---

Current game board. Turn: Machine

```

      0 1 2
0
1  X
2

```

---

---

Current game board. Turn: Human

```

      0 1 2

```

```

0
1  X O
2

You are X. Please insert your next move's row and column:
0 1
=====
Current game board. Turn: Machine

0 1 2
0   X
1  X O
2

=====
Current game board. Turn: Human

0 1 2
0   O X
1  X O
2

You are X. Please insert your next move's row and column:
2 2
=====
Current game board. Turn: Machine

0 1 2
0  O X
1  X O
2      X

=====
Current game board. Turn: Human

0 1 2
0  O X O
1  X O
2      X

```

```
You are X. Please insert your next move's row and column:  
2 1
```

---

---

```
Current_game_board . Turn : Machine
```

```
0 1 2  
0 O X O  
1 X O  
2 X X
```

---

---

```
Final_game_board . Turn : Human
```

```
0 1 2  
0 O X O  
1 X O  
2 O X X
```

```
VICTORY for player 2!
```

```
Would you like to play again (Y/N)?  
n
```

```
Thank you for playing .  
By Lucas Franca de Oliveira , COMP-18.
```

## 4 Conclusão

A atividade foi executada com sucesso, com todos os algoritmos que, teoricamente, deveriam funcionar de forma ótima, demonstrando comportamento ótimo nos testes. A atividade se demonstrou desafiadora, com a necessidade de implementação de *parsing* da descrição de um grafo no formato do arquivo "Uruguay.csv", assim como o desenvolvimento de uma heurística para o jogador de Jogo da Velha.

A heurística usada no Jogo da Velha foi a ensinada em aula pelo professor. Houve a ideia não implementada de, para cada consulta, mergulhar na árvore de decisão até uma profundidade fixa  $h$  e aplicar a heurística apenas nos limites e o método minimax nos estados intermediários.

Os algoritmos estudados são de suma importância para a indústria, sendo utilizados nas mais diversas áreas. No próprio H8, eles são usados pelo grupo de estudos e de robótica ITAndroids nas competições da CBR/LARC, grupo o qual o aluno fez parte e trabalhou com estes algoritmos nos anos de 2015 e 2016, junto com outros da classe. Havia, portanto, conhecimento prévio dos métodos aqui estudados, tornando esta atividade um complementar para o aprendizado do aluno.