Grupo 40 Diogo Marante 86409 João Daniel Silva 86445

# Inteligência Artificial

## Projeto 1 - Relatório

2018/2019

| Procura Gananciosa - Greedy | | | | |
|---|---|---|---|---|
| | Número de peças | Tempo de Execução (s) | Número de nós expandidos | Número de nós gerados |
| m4x4 | 14 | 0.006068468094 | 20 | 56 |
| m4x5 | 16 | 0.5789146423 | 5798 | 5852 |
| m5x5 | 11 | 0.002222537994 | 10 | 21 |
| m4x6 | 20 | 0.05686926842 | 172 | 306 |

| Procura em Profundidade Primeiro - DFS | | | | |
|---|---|---|---|---|
| | Número de peças | Tempo de Execução (s) | Número de nós expandidos | Número de nós gerados |
| m4x4 | 14 | 0.1879117489 | 5986 | 6003 |
| m4x5 | 16 | 1.759344339 | 53636 | 53664 |
| m5x5 | 11 | 0.0007915496826 | 13 | 20 |
| m4x6 | 20 | NULL | NULL | NULL |

| Procura A* (A-Star) | | | | |
|---|---|---|---|---|
| | Número de peças | Tempo de Execução (s) | Número de nós expandidos | Número de nós gerados |
| m4x4 | 14 | 0.006244897842 | 16 | 64 |
| m4x5 | 16 | 0.4389920235 | 3445 | 3550 |
| m5x5 | 11 | 0.001917600632 | 10 | 21 |
| m4x6 | 20 | 0.03278660774 | 39 | 179 |

# Análise dos Resultados

Obtivemos os resultados acima executando as funções de procura fornecidas pelo corpo docente, disponíveis no ficheiro search.py. As matrizes executadas foram as disponibilizadas no enunciado. Para obtermos as medições de tempo, utilizámos a biblioteca *time* do Python, e para a informação dos *nodes,* através da dos atributos implementados na classe *InstrumentedProblem.* Os testes foram realizados num computador com cpu i7-6700 HQ quad core.

A nossa heurística consiste no cálculo do número de movimentos não possíveis e do número de peças que não se podem mover, havendo preferência de escolha dos nós que tenham estes fatores mais baixos. Se existirem muitas peças que não se podem mover e/ou poucos movimentos possiveis, esse é um estado que mais dificilmente estará perto da solução final, e por isso a sua escolha deve ser desfavorecida. Assumimos que cada peça que não se pode mover num dado estado do tabuleiro poderá se mover numa jogada seguinte, quando outra peça se mova para uma posição que lhe permita a jogada.

Para calcular o número de movimentos não possíveis, subtraímos a um número que maximiza o número de movimentos que pode haver em função do número de peças (dobro do número de peças) o número de movimentos possíveis *(len(board_moves(node.state.board))*). Para calcular o número de peças que não se podem mover, subtraímos ao número total de peças *(node.state.countPegs())* o número de peças que se conseguem mover pelo menos para um sitio *(filterPegs(board_moves(node.state.board)*).

O tabuleiro 4x6 é o maior e tem o maior número de peças, sendo por isso o que mais demora a ser resolvido por qualquer algoritmo. Apesar do tabuleiro 5x5 ser o segundo maior, é o que tem o menor número de peças, por isso é muito mais rapidamente resolvido que os restantes.

Em relação aos algoritmos, a procura em profundidade é a mais ineficiente de todas, tanto em tempo de execução e em número de nós gerados. Na matrix 4x6 torna-se impraticável a sua aplicação. A procura A* é a mais eficiente.

Nas matrizes 4x4 e 4x5 os algoritmos que usaram a heurística geraram aproxidamente 1% e 9% (respetivamente) dos nós relativamente à DFS que não usa a heurística. No caso da 5x5 os nos foram próximos devido à trivialidade da solução e ao número reduzido de peças e no tabuleiro. Para a 4x6 não temos dados para os nós devido ao elevado tempo de computação do algoritmo por parte da máquina.