Resolução do Atividade 2a

Tiago Pinho Cardoso (201605762)  
Departamento de Engenharia Informática  
Faculdade de Engenharia da Universidade do PortoPorto, Portugal  
201605762@fe.up.pt

*Resumo*— Pretende-se neste trabalho resolver problema proposto escolhido de entre quatro possibilidades, utilizando métodos de pesquisa. Para este relatório será descrito neste artigo o problema, a formulação do mesmo, resultados e conclusões tiradas.

Keywords — Inteligência Artificial, Pesquisa, Pesquisa em Largura, Pesquisa Gulosa, Algoritmo A\*

# Introdução

Para esta entrega foi escolhida a atividade 2ª que consiste num problema do tipo n-puzzle. O objetivo deste exercício é a aplicação de métodos de pesquisa, com ênfase em métodos de pesquisa informada e no Algoritmo A\*, para resolver o conhecido problema N-Puzzle, apresentado nas aulas teóricas da disciplina.

# Formulação do Problema

Representação do estado do jogo: O puzzle irá ser representado por uma matriz de tamanho NxN dependendo nível em que N nunca é inferior a 3. Nesta matriz, todos os elementos irão ser representados por um valor que se encontra no intervalo entre 0 e (N^2) - 1, sendo que 0 representa uma posição sem qualquer elemento e todos os outros valores representam a sua posição na matriz em que o valor 1 deve estar na posição (0,0).

Estado Inicial: O estado inicial irá depender do problema escolhido. Nesta atividade é pedido que comparemos os resultados a nível de tempo e de memória dos seguintes problemas:



Operadores Há quatro ações possíveis: esquerda, direita, cima ou baixo. Cada uma destas ações ocorre ao trocar o elemento 0 que representa um espaço vazio com a peça que se encontra na posição adjacente na direção pretendida, logo, estas ações só são possíveis se existir um elemento na direção pretendida adjacente ao elemento 0.

Teste objetivo: O teste objetivo irá verificar se o estado atual do jogo é um estado final, sendo que este é um estado no qual todas os elementos se encontram nas suas posições na matriz e o elemento 0 se encontra na posição final desta. Por exemplo para uma matriz 3x3 a solução será:

1 2 3

4 5 6

7 8 0

Custo da solução: Neste caso, o custo de cada movimento será um. Assim sendo, o custo da solução que resolve o

problema é o número de trocas necessárias para chegar à tal solução.

# Algoritmos de Pesquisa

Neste actividade foram implementados e utilizados os três algoritmos de pesquisa propostos, sendo estes, pesquisa em largura, gulosa e algoritmo A\*. Todos têm como objetivo obter um estado solução partindo de um estado inicial.

O algoritmo de pesquisa em largura, função *breadthFirst* recebe como argumento o estado inicial do puzzle. Primeiramente, é inicializado um array que apenas possui o estado inicial e, de seguida, é executado um ciclo no qual se percorrem os estados filho do estado que está a ser verificado, ou seja, todos os estados que se encontram a apenas um movimento de distância do pai. Percorrendo os filhos, caso este seja um estado final, já foi encontrada a solução, caso contrário, se este não for um estado que tenha sido verificado, é colocado no array anteriormente falado de forma a ser analisado no futuro.

O método de pesquisa gulosa, função *greedy*, recebe como argumentos um estado inicial “initState” e uma heurística. É declarado um array “state” que é inicializado com “initialState”. De seguida, é inicializado um ciclo While enquanto o boolean “searching” for verdadeiro. Dentro do While são calculados os próximos estados possíveis e guardados numa variável para serem calculadas os valores da heurística em cada um deles e atualizar a melhor hipótese e a variável “heuristicValue” que tem em conta a heurística desejada. Em suma, neste algoritmo, é apenas considerado um próximo estado, sendo este o que possui o melhor valor depois de aplicada a heurística. Como se trata de um algoritmo não ótimo, é normal, por vezes, não encontrar uma solução e entrar num loop. Nestes casos é imprimida a mensagem: “Solution not found”.

Por fim, o algoritmo A\*, função *a\_star,* recebe um estado inicial como argumento, “initialState”. São declarados dois arrays, “front” onde cada elemento corresponde a um valor calculado pela heurística e a um estado, e “expanded” que guarda os estados que já foram expandidos. De seguida é declarado um While onde são comparados dois estados do puzzle (dois seguidos no array front) de acordo com o valor da heurística. A variável path e front são então atualizadas de acordo com o estado que possui melhor heurística. É criada uma variável “endnode” que é igualada ao último elemento de path, se este for um estado final o ciclo termina. Se “endnode” não for um estado final são calculados todos os estados atingíveis a partir de “endnode” e colocados em “newpath”. Após isto é colocado em “front” o “newpath” e em “expanded” o “endnode”.

Todos os algoritmos verificam uma condição no início da função, verificar se o estado inicial recebido é um estado final. Caso a condição seja verdadeira as funções retornam imediatamente.

# Experiências e Resultados

Descrevendo as experiências realizadas com os vários algoritmos para resolver diversos puzzles e os resultados obtidos a nível de tempo e custo da solução obtida em cada nível, assim como o número de movimentos necessários para obter tal solução, por cada um dos métodos experimentados.

**Breadth first**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Prob1 | Prob2 | Prob3 | Prob4 |
| Tempo (seg) | 0.0000 | 0.0041 | 0.0397 | 0.4220 |
| Espaço (nós visitados) | 19 | 65 | 384 | 2324 |
| Movimentos | 4 | 7 | 10 | 10 |

**Greedy – h1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Prob1 | Prob2 | Prob3 | Prob4 |
| Tempo (seg) | 0.0000 | 0.0000 | - | - |
| Espaço (nós visitados) | 4 | 7 | - | - |
| Movimentos | 4 | 7 | - | - |

**Greedy – h2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Prob1 | Prob2 | Prob3 | Prob4 |
| Tempo (seg) | 0.0150 | - | 0.0153 | - |
| Espaço (nós visitados) | 214 | - | 174 | - |
| Movimentos | 214 | - | 174 | - |

**A\* – h1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Prob1 | Prob2 | Prob3 | Prob4 |
| Tempo (seg) | 0.0000 | 0.0000 | 0.5307 | 0.0151 |
| Espaço (nós visitados) | 5 | 8 | 1709 | 70 |
| Movimentos | 4 | 7 | 54 | 14 |

**A\* – h2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Prob1 | Prob2 | Prob3 | Prob4 |
| Tempo (seg) | 0.0000 | 0.0000 | 0.0000 | 0.0619 |
| Espaço (nós visitados) | 5 | 11 | 16 | 222 |
| Movimentos | 4 | 7 | 10 | 14 |

# Conclusões e Perspetivas de Desenvolvimento

A heurística h2 é melhor que a heurística h1 no entanto ambas as heurísticas sobrestimam o custo até à solução final pois nenhuma dela considera sequências de números que estejam prontos a ser colocados no lugar certo e o puzzle que resulta após essa sequência ser percorrida. Exemplo disto é o problema 4 em que um humano consegue ver facilmente a jogada correta, trocar com o número 2, enquanto o algoritmo troca a casa fazia com o número 6.

A h2 é bastante melhor que a h1 porque apesar de não considerar isto é a mais próxima das duas a fazê-lo. Em problemas mais complexos como o problema 3, em que maior parte das peças está fora de posição, nota-se a fragilidade de h1 que é demasiado simplista para este problema.