Inteligência Artificial

Resolução de problemas através de buscas

# Introdução

O objetivo desse trabalho é resolver o problema intitulado de Régua-puzzle, o qual é descrito na próxima seção, utilizando-se os algoritmos de busca estudados em sala, a saber:

* Busca em largura
* Busca em profundidade
* Busca em profundidade iterativa
* Busca em profundidade limitada
* Busca de custo uniforme
* Busca gulosa com utilização de heurística
* Busca A\*

A fim de fornecer uma análise comparativa de cada um deles.

# O Jogo

Considere um jogo onde 2N blocos são alinhados em uma régua com 2N+1 posições. Existem N blocos brancos (B), N blocos azuis (A) e uma posição vazia.

O objetivo do jogo é movimentar os blocos (de acordo com as regras descritas a seguir) de forma a colocar todos os blocos brancos do lado esquerdo dos blocos azuis, ou seja, para todo i, se R[i] = B então R[j] 6= A, para todo 0 < j < i. Uma solução ótima para esse jogo é:

* A solução de menor número de movimentos, considerando custo 1 para todo movimento ou
* A solução de custo mínimo, considerando o custo de movimentos conforme definido na próxima seção.

Para esse trabalho, consideraremos como critérios:

1. Solução de menor número de movimentos
2. Solução de menor custo em termos de caminho

# Regras e Custo de Movimentação de blocos

Definimos a distância entre duas posições i e j, como j – i, sendo 0 < i < j ≤ 2N + 1. Blocos podem pular para a posição vazia quando a posição vazia estiver distante de no máximo N casas da posição do bloco. Desta maneira, existem no máximo 2N movimentos legais (no caso de o espaço vazio estar localizado exatamente no meio da régua). O custo de um pulo é igual a distância entre a posição do bloco e a posição vazia.

# Análise

Considerando os critérios de aceitação de soluções descritos na seção anterior, pudemos constatar que:

* Para o critério 1, a busca em largura se encaixa perfeitamente, pois a estratégia de expansão de cada nó nesse jogo, ou seja, a regra de como geramos descendentes de um certo nó n, dá-se mudando uma peça de lugar com o espaço vazio, dessa forma, cada peça movida estará em um nível da árvore construída pela busca em largura. Exemplo: Consideremos S = AB-BA como estado inicial, então:

Nível K = 0

Nível K = 1

Podemos ver que os nós obtidos em k = 0, foram obtidos pela movimentação de uma e somente uma peça em relação ao estado inicial, da mesma forma, os nós obtidos no nível k = 2, são gerados a partir de duas movimentações em relação ao estado inicial. Portanto, se estamos interessados no menor número de movimentos, dado a propriedade da busca em largura (varrer por níveis), encontraremos com certeza a solução segundo o critério 1.

Da mesma forma, a busca em profundidade limitada ajustada a um limite d, cujo d é a profundidade da solução encontrada pela busca em largura, nos devolve uma solução que atente ao critério 1, pois estaremos “podando” a árvore de busca na altura d, logo se a solução não está dentro dos limites em um certo ramo da árvore, então a busca em profundidade limitada troca de ramo até encontrar o ramo onde a solução está dentro do limite d. A busca em profundidade em geral, usa menos memória do que a busca em largura, portanto, se d é conhecido, ou pelo menos estimado, então a busca em profundidade limitada torna-se uma solução a considerar.

* Para o critério 2, percebemos que a busca de custo uniforme satisfaz e que a busca gulosa com uma heurística admissível, nesse caso a heurística escolhida foi:
  + H2: Dado um nó n qualquer, sabemos calcular h(n), tal que seja o número de peças em posições erradas em um certo estado.

Para o algoritmo de busca gulosa usando H2, o resultado foi satisfatório, pois a heurística mostrou-se admissível e eficaz, encurtando a complexidade do algoritmo em termos de tempo e espaço, e ainda assim, retornando uma solução que satisfaz o critério 2. O mesmo resultado não foi percebido no algoritmo de Busca A\*, onde a utilização da heurística, somada ao custo do caminho, não nos guiou para a solução de forma eficaz, ao menos não para os dados testados.

# Dados de testes

Foram feitos alguns testes com tamanhos de problemas variados. Usaremos as tabelas abaixo para sintetizar os resultados.

Tabela : Legenda

|  |  |
| --- | --- |
| Símbolo | Significado |
| N | Tamanho do problema |
| N(g) | Número de Nós Gerados |
| N(e) | Número de Nós Explorados |
| F(b) | Fator de Ramificação Médio |
| I(s) | Estado Inicial |
| F(s) | Solução encontrada |
| C(s) | Custo da Solução |
| D(s) | Profundidade da Solução |
| A(t) | Tipo de Agente |
| BFS | Busca em Largura |
| DFS | Busca em Profundidade |
| UCFS | Busca de Custo Uniforme |
| LDFS | Busca em Profundidade Limitada |
| IDFS(l) | Busca em Profundidade Iterativa com limite máximo l |
| GBFS(h) | Busca Guiada Gulosa usando a heurística h |
| A\*(h) | Busca A\* usando a heurística h |

Tabela : Resultado para n = 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Buscas em um espaço de N = 2 | | | | | | |
| I(s) = B-ABA | | | | | | |
| A(t) | N(G) | N(E) | F(b) | C(s) | D(s) | F(s) |
| BL | 9 | 3 | 3 | 2 | 1 | BBA-A |
| BP | 3 | 1 | 3 | 2 | 1 | BBA-A |
| BPI | 3 | 1 | 3 | 2 | 1 | BBA-A |
| BCU | 9 | 3 | 3 | 2 | 1 | BBA-A |
| A\*(h1) | 5 | 2 | 2 | 2 | 1 | BBA-A |
| A\*(h2) | 5 | 2 | 2 | 2 | 1 | BBA-A |
| GBFS(h1) | 5 | 2 | 2 | 2 | 1 | BBA-A |
| GBFS(h2) | 3 | 1 | 3 | 2 | 1 | BBA-A |
| BPL(5) | 3 | 1 | 3 | 2 | 1 | BBA-A |
| BPL(10) | 3 | 1 | 3 | 2 | 1 | BBA-A |
| BPL(8) | 3 | 1 | 3 | 2 | 1 | BBA-A |

Tabela : Resultado para n = 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Buscas em um espaço de N = 4 | | | | | | |
| I(s) = BAABABAB- | | | | | | |
| A(t) | N(G) | N(E) | F(b) | C(s) | D(s) | F(s) |
| BL | 2288 | 395 | 5 | 19 | 6 | BBBBAAA-A |
| BP | 1557 | 285 | 5 | 393 | 184 | B-BBBAAAA |
| BPI | 24887 | 4137 | 6 | 19 | 6 | BBBBAAA-A |
| BCU | 3540 | 613 | 5 | 19 | 7 | BBBBAAA-A |
| A\*(h1) | 3344 | 586 | 5 | 19 | 6 | BBBBAAA-A |
| A\*(h2) | 3364 | 575 | 5 | 19 | 7 | B-BBBAAAA |
| GBFS(h1) | 1022 | 200 | 5 | 38 | 16 | BBBBAA-AA |
| GBFS(h2) | 48 | 8 | 6 | 19 | 6 | B-BBBAAAA |
| BPL(16) | 16884 | 2809 | 6 | 29 | 16 | BBBBAAA-A |
| BPL(11) | 27646 | 4594 | 6 | 25 | 11 | B-BBBAAAA |
| BPL(14) | 16875 | 2807 | 6 | 27 | 14 | BBBBAAA-A |

Tabela : Resultado para n = 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Buscas em um espaço de N = 8 | | | | | | |
| I(s) = BBAAA-BBAAAABBBAB | | | | | | |
| A(t) | N(G) | N(E) | F(b) | C(s) | D(s) | F(s) |
| BL | 2468503 | 209821 | 11 | 74 | 12 | BBB-BBBBBAAAAAAAA |
| BP | 257960 | 26709 | 9 | 61744 | 25321 | BBBBBBBBA-AAAAAAA |
| BPI | - | - | - | - | - | - |
| BCU | - | - | - | - | - | - |
| A\*(h1) | - | - | - | - | - | - |
| A\*(h2) | - | - | - | - | - | - |
| GBFS(h1) | 49218 | 4851 | 10 | 141 | 34 | BBBBBBBBAAAA-AAAA |
| GBFS(h2) | 386 | 31 | 12 | 74 | 13 | BBB-BBBBBAAAAAAAA |
| BPL(16) | - | - | - | - | - | - |
| BPL(13) | - | - | - | - | - | - |
| BPL(14) | - | - | - | - | - | - |

Tabela : Resultado para n = 100

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Buscas em um espaço de N = 100 | | | | | | |
|  |  |  |  |  |  |  |
| A(t) | N(G) | N(E) | F(b) | C(s) | D(s) | F(s) |
| BFS | - | - | - | - | - | - |
| DFS | - | - | - | - | - | - |
| UCFS | - | - | - | - | - | - |
| LDFS | - | - | - | - | - | - |
| IDFS(5) | - | - | - | - | - | - |
| IDFS(10) | - | - | - | - | - | - |
| IDFS(15) | - | - | - | - | - | - |
| GBFS(h1) | - | - | - | - | - | - |
| GBFS(h2) | 118279 | 756 | 156 | 11457 | 133 |  |
| A\*(h1) | - | - | - | - | - | - |
| A\*(h2) |  |  |  |  |  |  |

Podemos notar que, a partir de n = 8, alguns algoritmos não completaram a solução, talvez apenas demoraram demais (não tivemos paciência de esperar) e eventualmente retornariam a solução.[[1]](#footnote-1)

Fica fácil perceber o motivo da demora do algoritmo de busca em profundidade limitada, expande os nós segundo o critério de expansão em árvore, ou seja, continua a expandir nós que já foram expandidos antes, o que faria entrar em loop infinito se não dispuséssemos de um mecanismo de “poda”. Essa forma de expandir os nós faz com que o algoritmo demore muito mais, aliás, quanto maior o limite, mais demorado fica, pois, mais repetições teremos em nossa árvore. No entanto, se o limite escolhido L for:

* L > d, onde d é a profundidade da solução encontrada pela busca em largura, então o algoritmo poderá demorar e muito
* L < d, nesse caso, o algoritmo não encontrará uma solução
* L = d, então, nesse caso o algoritmo encontrará uma solução tão boa quanto a busca em largura de forma mais eficaz em termos de espaço

Para a busca em profundidade iterativa, o mesmo argumento se aplica, com agravante de que esse tipo de busca, executa várias buscas, aumentando o limite gradativamente, o que faz com que o algoritmo fique bem mais lento, no entanto é indicado em casos onde o limite d, não é conhecido.

Quanto ao algoritmo A\*, nos dados descritos nas tabelas, em especial na tabela 5, podemos notar que o algoritmo não chegou a terminar (demorou muito), analisando a execução do algoritmo, percebemos que quando o algoritmo levava apenas o valor de H2(n) para ordenar a borda, como é o caso do algoritmo guloso com uso de heurística, os nós retirados seguiam exatamente a semântica de H2(n), ou seja, nós com menos peças em locais errados eram retirados da borda primeiro, porém, ao somar o custo do caminho até n ao valor obtido por H2(n), essa ordem era alterada, fazendo com que o algoritmo não fosse guiado muito bem.

# Conclusão

Esse trabalho foi bem interessante para analisar a estrutura dos algoritmos e como implementá-los, em especial, de forma que pudéssemos reutilizar código, afinal pouca coisa difere de um tipo de busca para outro. Além disso, também acrescenta muito a parte de análise, pois pudemos quantificar e ao menos tentar entender alguns resultados que aparentemente não estão conforme.

1. Para n = 100, não exibiremos a entrada (pois não cabe na tabela), no entanto, é irrelevante, pois o intuito dessa tabela é mostrar que apenas um algoritmo conseguiu resolver o problema. [↑](#footnote-ref-1)