Análise da implementação do algoritmo do A*

Diogo Alves de Almeida RA: 95108

Centro de Tecnologia
Departamento de Informática – Universidade Estadual de Maringá (UEM)
CEP 87020-900 – Maringá – Paraná – Brasil

1. Proposta

Foi proposto como trabalho 1 para a disciplina de Modelagem e Otimização Algorítmica (6903-2020-T1), no qual é lecionada pelo professor Dr. Ademir Aparecido Constantino, a implementação do algoritmo A* para a resolução do jogo do tabuleiro de 15 peças. A implementação deveria permitir entrar com qualquer estado inicial com o objetivo de encontrar a menor sequência de movimentos que leve ao estado final do jogo. Como requisito do trabalho foi solicitado a implementação de 5 heurísticas para que o algoritmo tenha como base para a tomada de decisão. Essas heurísticas deveriam ser implementadas e analisadas seu comportamento diante dos casos apresentados pelo professor.

Estado final desejado:

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	

Estado final desejado.

Além da implementação das heurísticas foi proposto a implementação da versão II do algoritmo A* para analisar se as linhas 9 e 10 do algoritmo teve alguma influência nos resultados. Para esse trabalho foram testadas todas as heurísticas propostas pelo professor para a versão I do algoritmo A* com a finalidade de verificar qual heurística teve o melhor desempenho, logo em seguida foi testado, com a heurística de melhor desempenho, a versão II do algoritmo A*. A análise de desempenho foi feita levando em consideração o consumo de memória e o tempo de processamento, a pedido do professor, como também a quantidade de nós verificados, que foi uma escolha pessoal para aprimorar o relatório.

2. Heurísticas

Heurísticas a serem implementadas e analisadas:

1. h'1(n) = número de peças foras de seu lugar na configuração final;
2. h’2(n) = número de peças fora de ordem na sequência numérica das 15 peças, seguindo a ordem das posições no tabuleiro;
3. h’3(n) = para cada peça fora de seu lugar somar a distância Manhattan; (quantidade de deslocamentos) para colocar em seu devido lugar. Neste caso considera-se que o caminho esteja livre para fazer o menor número de movimentos;
4. h’4(n) = p1 * h'1(n) + p2 * h’2(n) + p3 * h’3(n), sendo p1 + p2 + p3 são pesos (número real) tais que p1 + p2 + p3 = 1. A escolha desses pesos deverá ser escolhida conforme os resultados dos experimentos.
5. h’5(n) = max(h'1(n), h’2(n), h’3(n)).

3. Especificações Técnicas

As especificações do computador utilizado para o trabalho são:

Memória RAM: 15,5 GiB
Processador: Intel® Core™ i5-8400 CPU @ 2.80GHz × 6
Gráficos: Mesa Intel® UHD Graphics 630 (CFL GT2)
Capacidade de Disco: 240,1GB SSD
SO: Ubuntu 20.04.1 LTS
Tipo do SO: 64 bits
Versão do GNOME: 3.36.8

Para a implementação foi utilizado a linguagem Java na versão a seguir:

```
-> ~ java -version
openjdk version "1.8.0_282"
OpenJDK Runtime Environment Corretto-8.282.08.1 (build 1.8.0_282-b08)
OpenJDK 64-Bit Server VM Corretto-8.282.08.1 (build 25.282-b08, mixed mode)
```

Para representar cada estado do tabuleiro foi criada uma classe chamada *Node* onde nela é guardada os dados do estado atual incluindo o valor de *g(n)* e *f(n)* calculados, além da referência para seus estados filhos como também do estado pai no qual ele foi gerado. Caso a referência do estado pai seja igual a *NULL* é porque esse estado é o inicial do tabuleiro. A representação da classe veremos a seguir:

```
public static class Node {
        int[] state = new int[16];
        String stateStr;
        int emptyIndex;
        int level = 0;      // g(n)
        int hCalculated;    // h(n)
        String movement = "";
        Node root = null;
        Node up = null;
        Node down = null;
```

```
        Node left = null;
        Node right = null;

        public void printState() {
            System.out.println(" ---------------");
            for (int line = 0; line < 4; line++) {
                System.out.print("| ");
                for (int column = 0; column < 4; column++) {
                    printNumber(state[4 * line + column]);
                }
                System.out.print("\n");
            }
            System.out.print(" ---------------\n\n");
        }
}
```

Para representar o conjunto A (conjunto dos estados abertos) foi utilizada uma lista de *Nodes* no qual no final de cada iteração do algoritmo essa lista é preenchida com os novos *Nodes* sucessores do atual.

O conjunto F (conjunto dos estados fechados) foi utilizado uma variável *int* no qual é incrementado com 1 representando o *Node* atual da iteração que foi verificado. Nesse caso, foi considerado que não era relevante criar uma lista guardando esses *Nodes* visitados porque tornaria o algoritmo custoso em relação ao uso de memória, uma vez que esses dados não seriam utilizados para análise.

4. Testes

Para os testes, foi disponibilizado, na plataforma *Moodle*, 10 casos onde o algoritmo teria como entrada (estado de partida).

Os 10 casos disponibilizados foram:

1. *0 2 9 13 3 1 5 14 4 7 6 10 8 11 12 15*
2. *3 2 1 9 0 5 6 13 4 7 10 14 8 12 15 11*
3. *2 1 9 13 3 5 10 14 4 6 11 15 7 8 12 0*
4. *9 13 10 0 5 2 6 14 1 7 11 15 3 4 8 12*
5. *4 3 2 1 8 10 11 5 12 6 0 9 15 7 14 13*
6. *9 13 14 15 5 6 10 8 0 1 11 12 7 2 3 4*
7. *10 6 2 1 7 13 9 5 0 15 14 12 11 3 4 8*
8. *6 2 1 5 4 10 13 9 0 8 3 7 12 15 11 14*
9. *10 13 15 0 5 9 14 11 1 2 6 7 3 4 8 12*
10. *5 9 13 14 1 6 7 10 11 15 12 0 8 2 3 4*

A finalidade desses testes, como dito anteriormente, é de analisar o comportamento das heurísticas como também a diferença de comportamento do algoritmo para a versão I e II tendo como base o consumo de memória, tempo de processamento além também da quantidade de estados verificados como escolha pessoal.

Em relação às heurísticas foi observado que a de melhor desempenho foi a heurística 5 tendo ela o menor tempo de processamento e também o menor consumo de memória. A de pior desempenho foi a heurística 2 tendo o maior tempo de processamento e o maior consumo de memória em todos os casos.

O consumo de memória não teve tanta disparidade da heurística 2 para as outras, mas ainda assim ela sempre obteve os piores resultados. Porém em relação ao tempo de processamento teve casos em que ela foi 58.633 vezes pior que a segunda mais lenta. Como por exemplo o caso 2 que para a heurística 1 (segunda pior em relação ao tempo) levou 16 milissegundos para verificar que a resposta correta era 19 movimentos, enquanto para a heurística 2 demorou 15 minutos, 38 segundos e 128 milissegundos. Em relação à quantidade de estados verificados, também houve uma disparidade parecida com a do tempo de processamento.

Vale ressaltar que para a heurística 4 foi utilizado os pesos da seguinte maneira:

- p1 = 0,20
- p2 = 0,20
- p3 = 0,60

A nível de performance da melhor para a pior temos a seguinte lista de heurísticas:

1. Heurística 5
2. Heurística 3
3. Heurística 4
4. Heurística 1
5. Heurística 2

Portanto, para a verificação da versão I e II foi utilizado a heurística 5. Para cada tipo de análise (estados verificados, memória consumida e tempo de processamento) foi realizada a variação da diferença entre versão II para a versão I. Para os estados verificados, foi analisado que a versão II verificou (visitou) menos estados em todos os casos, ou seja, ela conseguiu reduzir a quantidade de nós necessários para conseguir encontrar o resultado.

Mesmo reduzindo essa quantidade, não houve uma melhora de performance para o que foi proposto de análise. Em relação a memória consumida ela conseguiu se sobressair em 50% dos casos, enquanto 3 casos ficaram com a mesma quantidade de megabytes. Em contrapartida em relação ao tempo de processamento a versão II se saiu muito pior do que a versão I tendo casos no qual ela foi 488% mais lenta, como o caso 9. A única exceção para esta métrica foi o caso 1 onde a versão II foi 11% mais rápida.

Essa perda de performance acontece devido a verificação da linha 9 onde ela procura o estado atual na lista de *Nodes* verificados, isso traz um tempo maior tendo em vista que quanto maior for a quantidade de Nodes visitados mais demorado será essa verificação.

Como a diferença de memória foi pequena e quase irrelevante para os padrões atuais de poder computacional, temos que considerar o tempo como maior fator para provar que a versão I é melhor que a versão II. Para a entrega do trabalho será escolhida a versão I junto com a heurística 5 para ser avaliada.

O resultados de movimentos necessários para cada caso foram:

1. *0 2 9 13 3 1 5 14 4 7 6 10 8 11 12 15 = 18 movimentos*
2. *3 2 1 9 0 5 6 13 4 7 10 14 8 12 15 11 = 19 movimentos*
3. *2 1 9 13 3 5 10 14 4 6 11 15 7 8 12 0 = 12 movimentos*
4. *9 13 10 0 5 2 6 14 1 7 11 15 3 4 8 12 = 21 movimentos*
5. *4 3 2 1 8 10 11 5 12 6 0 9 15 7 14 13 = 38 movimentos*
6. *9 13 14 15 5 6 10 8 0 1 11 12 7 2 3 4 = 32 movimentos*
7. *10 6 2 1 7 13 9 5 0 15 14 12 11 3 4 8 = 38 movimentos*
8. *6 2 1 5 4 10 13 9 0 8 3 7 12 15 11 14 = 34 movimentos*
9. *10 13 15 0 5 9 14 11 1 2 6 7 3 4 8 12 = 27 movimentos*
10. *5 9 13 14 1 6 7 10 11 15 12 0 8 2 3 4 = 29 movimentos*

A seguir as tabelas e os gráficos criados para melhor entendimento da análise feita das versões:

H'5	ESTADOS VERIFICADOS		
	VERSÃO I	VERSÃO II	VARIAÇÃO DE VII PARA VI
CASO 01	3.082,00	2.179,00	-29,29915639
CASO 02	643,00	570,00	-11,35303266
CASO 03	685,00	564,00	-17,66423358
CASO 04	19.764,00	13.860,00	-29,87249545
CASO 05	3.373,00	2.742,00	-18,70738215
CASO 06	24.462,00	19.105,00	-21,89927234
CASO 07	307.377,00	212.257,00	-30,94571162
CASO 08	60.082,00	45.221,00	-24,73452948
CASO 09	2.560,00	2.330,00	-8,984375
CASO 10	243.878,00	179.718,00	-26,30823609

Tabela de estados verificados.

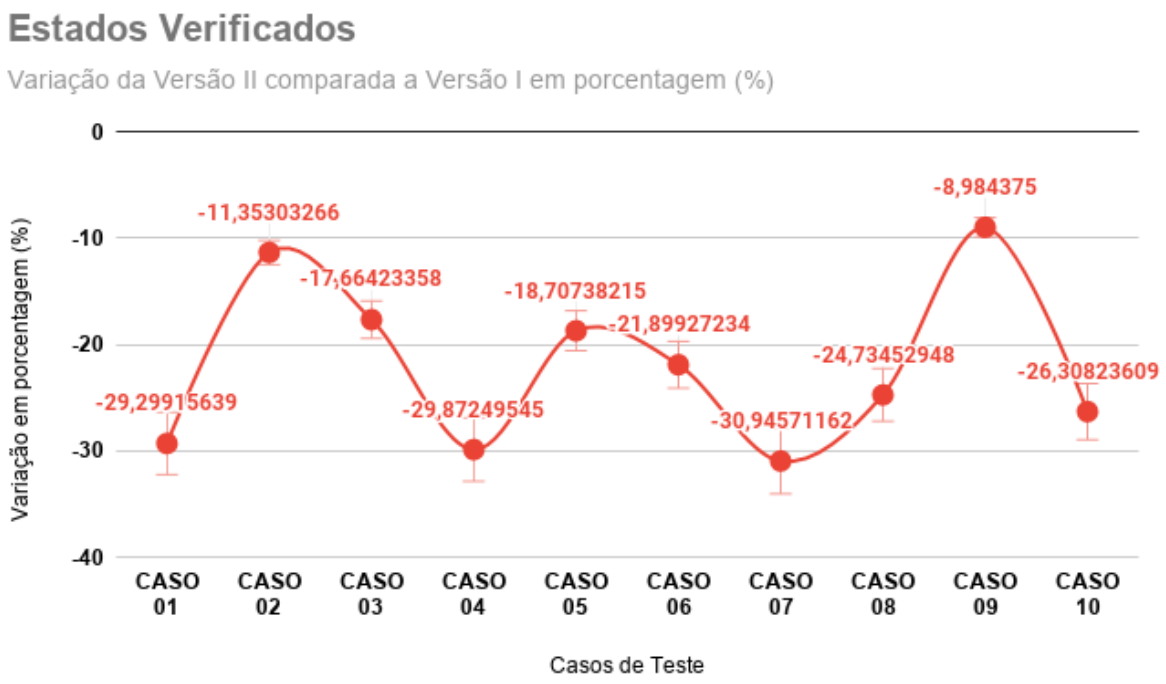

Gráfico de variação da versão II comparada com a versão I para os estados verificados.

H'5	MEMÓRIA CONSUMIDA (MB)		
	VERSÃO I	VERSÃO II	VARIAÇÃO DE VII PARA VI
CASO 01	20,00	17,00	-15
CASO 02	4,00	4,00	0
CASO 03	5,00	4,00	-20
CASO 04	61,00	34,00	-44,26229508
CASO 05	18,00	18,00	0
CASO 06	19,00	60,00	215,7894737
CASO 07	244,00	165,00	-32,37704918
CASO 08	47,00	50,00	6,382978723
CASO 09	15,00	15,00	0
CASO 10	188,00	142,00	-24,46808511

Tabela de memória consumida.

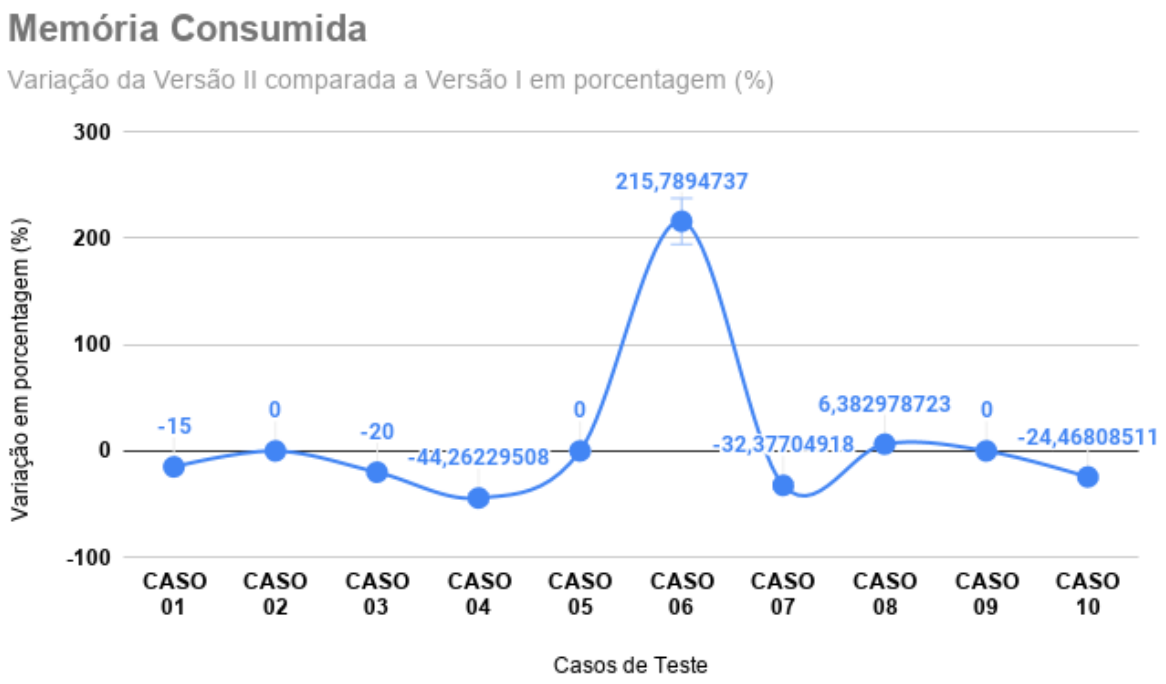

Gráfico de variação da versão II comparada com a versão I para a memória consumida.

H'5	TEMPO DE PROCESSAMENTO (MS)		
	VERSÃO I	VERSÃO II	VARIAÇÃO DE VII PARA VI
CASO 01	118,00	105,00	-11,01694915
CASO 02	4,00	8,00	100
CASO 03	3,00	5,00	66,66666667
CASO 04	1.403,00	4.147,00	195,5808981
CASO 05	39,00	178,00	356,4102564
CASO 06	2.175,00	10.037,00	361,4712644
CASO 07	997.985,00	2.043.148,00	104,7273256
CASO 08	24.599,00	68.667,00	179,1454937
CASO 09	26,00	153,00	488,4615385
CASO 10	595.705,00	1.373.027,00	130,4877414

Tabela do tempo de processamento.

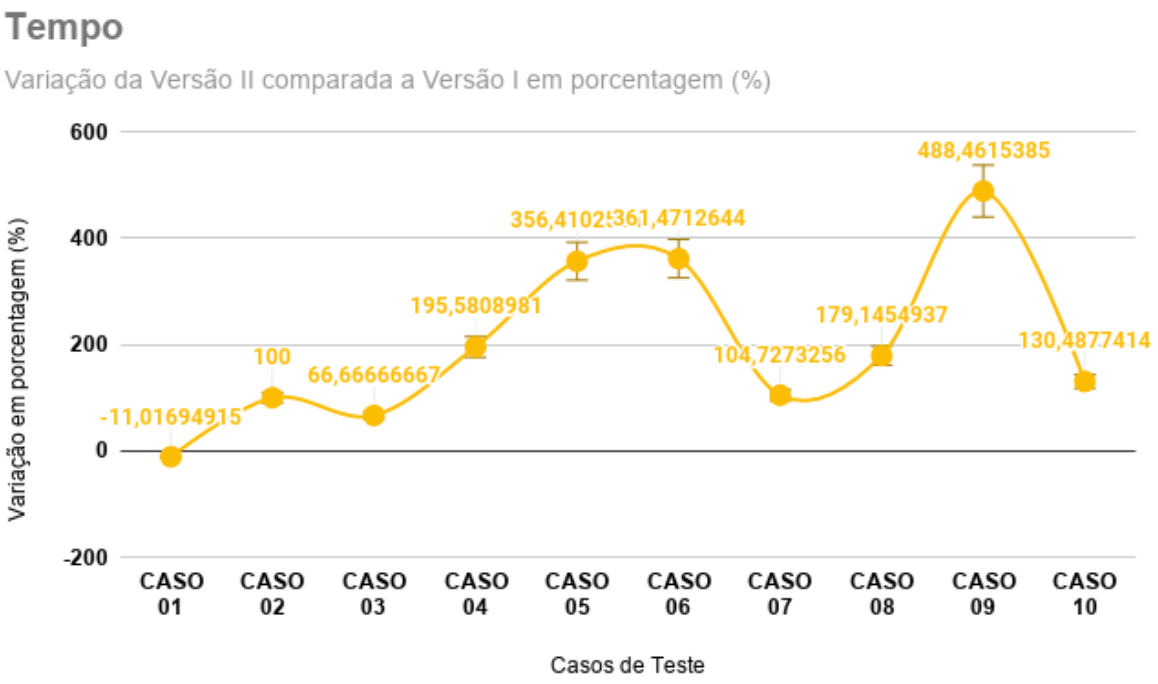

Gráfico de variação da versão II comparada com a versão I para o tempo de processamento.

5. Apêndice

Prints analisados para realização da análise da melhor versão do algoritmo A*:

```
Caso[1] 0 2 9 13 3 1 5 14 4 7 6 10 8 11 12 15
Version[I] H'[5] Movements[18] Nodes[3082] Memory[20mb] Time[118ms] TimeFormatted[0d:0h:0m:0s:118ms]
Version[II] H'[5] Movements[18] Nodes[2179] Memory[17mb] Time[105ms] TimeFormatted[0d:0h:0m:0s:105ms]

Caso[2] 3 2 1 9 0 5 6 13 4 7 10 14 8 12 15 11
Version[I] H'[5] Movements[19] Nodes[643] Memory[4mb] Time[4ms] TimeFormatted[0d:0h:0m:0s:4ms]
Version[II] H'[5] Movements[19] Nodes[570] Memory[4mb] Time[8ms] TimeFormatted[0d:0h:0m:0s:8ms]

Caso[3] 2 1 9 13 3 5 10 14 4 6 11 15 7 8 12 0
Version[I] H'[5] Movements[12] Nodes[685] Memory[5mb] Time[3ms] TimeFormatted[0d:0h:0m:0s:3ms]
Version[II] H'[5] Movements[12] Nodes[564] Memory[4mb] Time[5ms] TimeFormatted[0d:0h:0m:0s:5ms]

Caso[4] 9 13 10 0 5 2 6 14 1 7 11 15 3 4 8 12
Version[I] H'[5] Movements[21] Nodes[19764] Memory[61mb] Time[1403ms] TimeFormatted[0d:0h:0m:1s:403ms]
Version[II] H'[5] Movements[21] Nodes[13860] Memory[34mb] Time[4147ms] TimeFormatted[0d:0h:0m:4s:147ms]

Caso[5] 4 3 2 1 8 10 11 5 12 6 0 9 15 7 14 13
Version[I] H'[5] Movements[38] Nodes[3373] Memory[18mb] Time[39ms] TimeFormatted[0d:0h:0m:0s:39ms]
Version[II] H'[5] Movements[38] Nodes[2742] Memory[18mb] Time[178ms] TimeFormatted[0d:0h:0m:0s:178ms]

Caso[6] 9 13 14 15 5 6 10 8 0 1 11 12 7 2 3 4
Version[I] H'[5] Movements[32] Nodes[24462] Memory[19mb] Time[2175ms] TimeFormatted[0d:0h:0m:2s:175ms]
Version[II] H'[5] Movements[32] Nodes[19105] Memory[60mb] Time[10037ms] TimeFormatted[0d:0h:0m:10s:37ms]

Caso[7] 10 6 2 1 7 13 9 5 0 15 14 12 11 3 4 8
Version[I] H'[5] Movements[38] Nodes[307377] Memory[244mb] Time[997985ms] TimeFormatted[0d:0h:16m:37s:985ms]
Version[II] H'[5] Movements[38] Nodes[212257] Memory[165mb] Time[2043148ms] TimeFormatted[0d:0h:34m:3s:148ms]

Caso[8] 6 2 1 5 4 10 13 9 0 8 3 7 12 15 11 14
Version[I] H'[5] Movements[34] Nodes[60082] Memory[47mb] Time[24599ms] TimeFormatted[0d:0h:0m:24s:599ms]
Version[II] H'[5] Movements[34] Nodes[45221] Memory[50mb] Time[68667ms] TimeFormatted[0d:0h:1m:8s:667ms]

Caso[9] 10 13 15 0 5 9 14 11 1 2 6 7 3 4 8 12
Version[I] H'[5] Movements[27] Nodes[2560] Memory[15mb] Time[26ms] TimeFormatted[0d:0h:0m:0s:26ms]
Version[II] H'[5] Movements[27] Nodes[2330] Memory[15mb] Time[153ms] TimeFormatted[0d:0h:0m:0s:153ms]

Caso[10] 5 9 13 14 1 6 7 10 11 15 12 0 8 2 3 4
Version[I] H'[5] Movements[29] Nodes[243878] Memory[188mb] Time[595705ms] TimeFormatted[0d:0h:9m:55s:705ms]
Version[II] H'[5] Movements[29] Nodes[179718] Memory[142mb] Time[1373027ms] TimeFormatted[0d:0h:22m:53s:27ms]
```