INTELIGÊNCIA ARTIFICIAL

Relatório Projeto (2021/2022)

Grupo 38 (Alameda) | Margarida Bezerra 99270 | MªSofia Pinho 99272

1. Introdução

O problema que foi proposto tem como objetivo desenvolver um programa em Python que resolva o problema *Takuzu* utilizando técnicas de procura de IA. Este jogo é para um único agente e tem como objetivo completar um tabuleiro com o mesmo número de 0s e 1s em todas as linhas e colunas, sendo todas as linhas e colunas diferentes.

2. Avaliação Experimental

Os critérios usados para comparar os algoritmos de procura foram o **tempo de execução**, os **nós gerados** e os **nós expandidos**.

Procedendo à análise do programa realizaram-se vários testes, dos quais foram extraídos os critérios pedidos para cada algoritmo de procura. Os valores resultantes estão representados nos seguintes gráficos presentes no **Anexo A** na página 3.

3. Análise Experimental – considerações sobre eficiência, penetrância

Tomando como objeto de **análise experimental** os testes aos tabuleiros de tamanho 6x6, 10x10, 21x21 e 31x31 (comparações em gráficos de barras anexadas no final do relatório, Anexo A, página 3) mas também os gráficos (gráfico da evolução do tempo de cada algoritmo de procura anexado), podemos tirar algumas conclusões sobre como estes algoritmos se comportam em valores de N maiores e menores.

Para **N=6** (e também alguns outros valores de N), tanto o número de nós gerados como o número de nós expandidos é igual nos 4 algoritmos (DFS, BFS, A* e Greedy), como podemos ver na **Figura 5**. No entanto, o valor do tempo que demorou a executar cada um e a chegar a uma solução varia um pouco. É mais rápido na DFS e mais lento no A*.

Para **N=10**, o número de nós gerados e o número de nós expandidos no algoritmo Greedy são menores do que nos outros algoritmos e maiores na BFS e A*, como podemos observar na **Figura 6**. No entanto, o tempo de execução é igual em todos (0,17s) exceto na DFS.

Para **N=21**, o número de nós gerados e expandidos é menor utilizando o algoritmo DFS e a seguir com o Greedy. Quanto aos valores de tempo de execução, a DFS é, outra vez, mais rápida, enquanto os outros 3 algoritmos têm tempos muito semelhantes, como vemos na **Figura 7**.

Para **N=31**, ao contrário do que se esperava, os valores de tempo e nós gerados e expandidos não varia quase nada entre os 4 algoritmos, segundo a **Figura 8**. Na verdade, o Greedy é *ligeiramente* melhor quanto à quantidade de nós. E no que toca ao tempo de execução, os valores são iguais (0,48s) exceto na BFS que é um pouco superior.

Concluindo, utilizando os nós gerados e expandidos como medida, podemos concluir que, no geral, o algoritmo Greedy é favorável. Se utilizarmos o tempo de execução dos algoritmos como medida, o algoritmo DFS é favorável. Assim, para um sistema em que memória é algo que se queria poupar, utilizar-se-á o Greedy e num sistema onde o propósito for a rapidez, utilizar-se-á o DFS.

4. Conclusões – completude, otimalidade e função heurística

As quatro procuras garantem a **completude**, pois em cada ação, a não ser quando é verificada alguma invalidade já presente no tabuleiro (caso em que não retorna nenhuma ação para passar para um próximo ramo da procura), é escolhida um valor para uma célula, logo é impossível a procura entrar num loop, uma vez que esse valor não volta a ser alterado e, como o tabuleiro é finito, o número de estados também o é. A **otimalidade**, que se traduz em encontrar a solução de maior qualidade quando há várias distintas, não é relevante para este projeto visto que todas as ações custam o mesmo e todas soluções possíveis estarão ao mesmo nível de profundidade dado que envolvem prencher sempre o mesmo número de células. Sendo assim, a heurística escolhida apenas ajuda a resolver o problema de forma mais rápida.

A **função heurística** é:

$h(n) = nº\_de\_células\_livres$

Uma heurística é **consistente** quando cumpre:

$h(n) \leq h(n') + c(n, a, n')$

No caso da escolhida todas as ações têm o mesmo custo (consistem em jogar 0 ou 1 numa única posição livre), pelo que podemos considerar um custo unitário para cada uma, isto é, dado um nó *n* e um descendente *n'* de *n*:

$c(n, a, n') = 1, \ \forall_a \in actions(n)$

$h(n') = nº\_de\_células\_livres\_em\_n' = nº\_de\_células\_livres\_em\_n - 1 = h(n) - 1$

$h(n') + c(n,a,n') = h(n) - 1 + 1 \geq h(n)$ *c. q. d.*

Logo, h(n) é **consistente** e, consequentemente, é **admissível**.

Anexo A

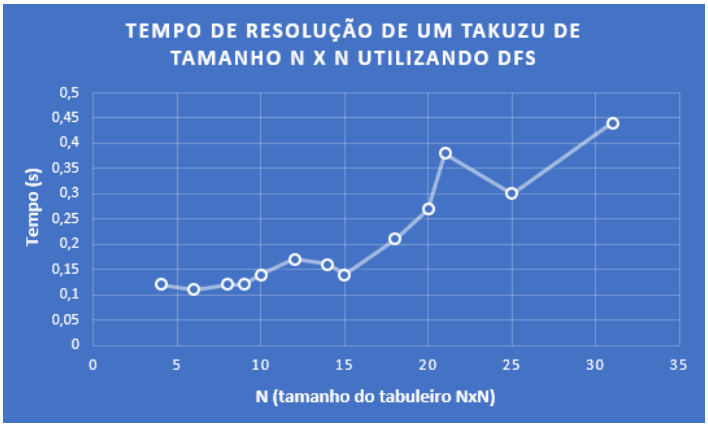

Figure 1: Depth First Search

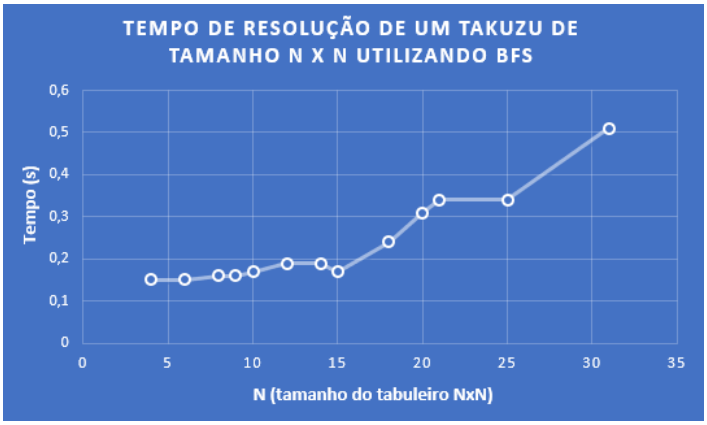

Figure 2: Breadth First Search

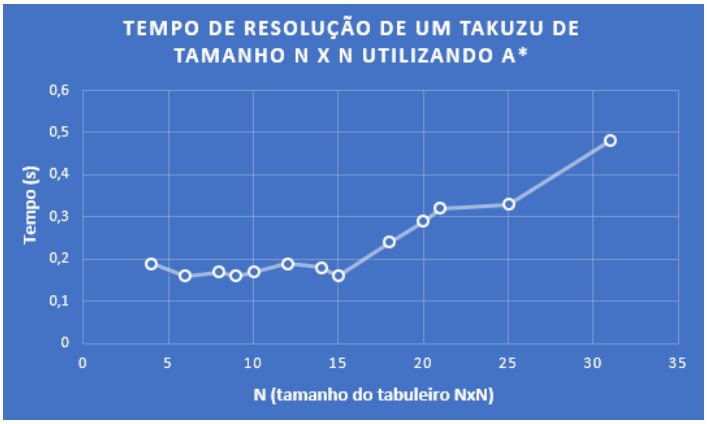

Figure 3: A Search*

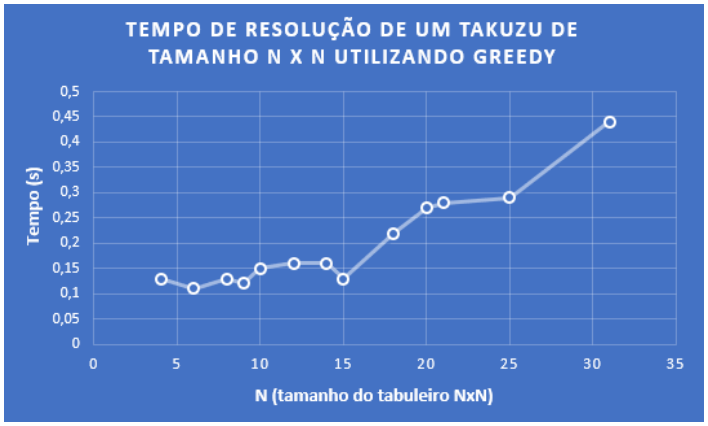

Figure 4: Greedy Search

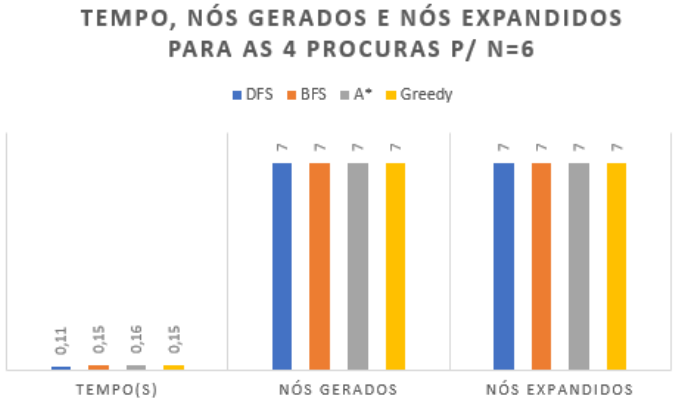

Figure 5: N = 6

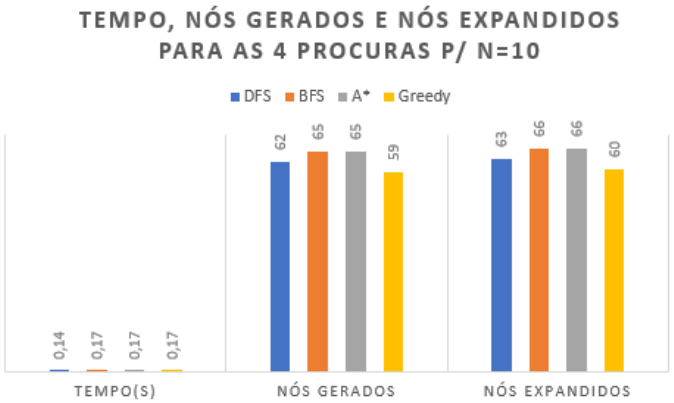

Figure 6: N = 10

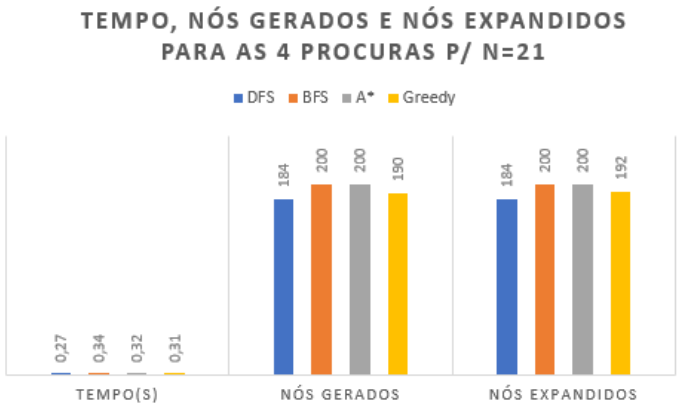

Figure 7: N = 21

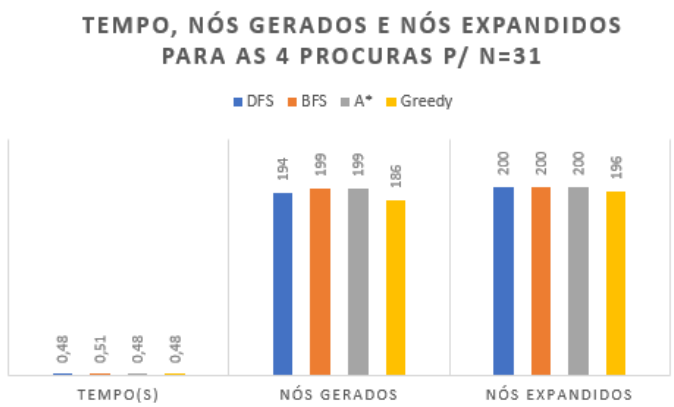

Figure 8: N = 31