Cubo Mágico $2 \times 2 \times 2$

Gabriel Santos Barbosa[1], **Savio Lopes Rabelo**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE),
Eixo Tecnológico de Computação, Campus Maracanaú – CE – Brasil

{gabrielsantos.ifce, saviorabelo.ti}@gmail.com

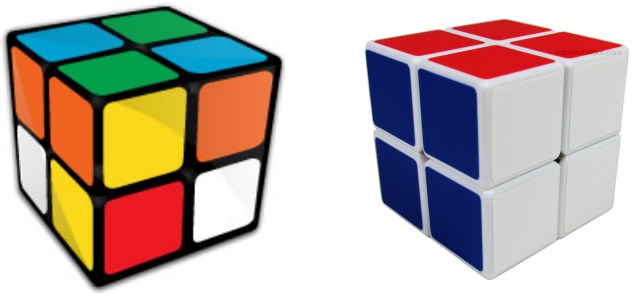
1. Introdução

O cubo mágico, inventado em 1974 por um jovem professor de arquitetura de Budapeste chamado Erno Rubik, na Figura 1, é um dos *puzzles* combinatórios mais famosos na literatura, tendo sido objeto de estudo por diversos pesquisadores desde a sua criação. A versão padrão, consiste em um cubo $3 \times 3 \times 3$ com diferentes adesivos coloridos em cada um dos quadrados expostos dos subcubos. Qualquer plano $3 \times 3 \times 1$ do cubo pode ser rotacionado 90, 180, ou 270 graus em relação ao resto do cubo. No estado objetivo, todos os quadrados em cada lado do cubo devem ter a mesma cor. O cubo é embaralhado através da realização de movimentos possíveis no cubo e o objetivo é restaurar o cubo para o seu estado objetivo original [Rubiks 2017].

O problema é bastante difícil considerando que existem 4.3252×10^{19} estados diferentes que podem ser alcançados a partir de qualquer configuração do cubo. Dessa forma, seria praticamente impossível resolver o cubo mágico sem uma estratégia geral. Essas estratégias geralmente consistem em um conjunto de sequências de movimentos que corretamente posicionam subcubos individuais sem violar outros subcubos posicionados anteriormente. A maioria dos métodos usados para resolver um cubo embaralhado tipicamente requer cerca de 50 e 100 movimentos. No entanto, foi provado que o número máximo de movimentos necessários para resolver qualquer instância do cubo mágico é menor que 20 [Cube20 2017].

Figura 1. Erno Rubik e seu cubo $3 \times 3 \times 3$

Uma outra versão do cubo mágico consiste em um cubo $2 \times 2 \times 2$, Figura 2. O problema é o mesmo da versão original, a diferença está na quantidade de subcubos que o compõem e, consequentemente, no seu espaço de estados. À primeira vista pode parecer fácil resolver essa instância do problema, contudo é importante ressaltar que o cubo mágico $2 \times 2 \times 2$ possui mais de 3.6 milhões de permutações possíveis, tornando inviável a sua resolução através de movimentos aleatórios.



(a) Embaralhado (b) Solucionado

Figura 2. Cubo mágico $2 \times 2 \times 2$**.**

O objetivo do presente trabalho consiste em desenvolver algoritmos baseados em métodos de busca com informação para solucionar o problema do cubo mágico $2 \times 2 \times 2$. Entre esses métodos, destacam-se o algoritmo $A^\star$ [Russel and Norvig 2004] e o *Iterative deepening* $A^\star$ ($IDA^\star$) [Russel and Norvig 2004]; eles são capazes de resolver problemas de busca de forma completa e ótima quando algumas restrições são satisfeitas. Assim, desejamos implementar ambos os algoritmos para o problema do cubo mágico $2 \times 2 \times 2$ e comparar os resultados de cada método.

2. Buscas

O processo de procurar uma sequência de ações que alcançam um dado objetivo é chamado de busca. Um algoritmo de busca recebe um problema como entrada e devolve uma solução sob a forma de uma sequência de ações. Depois que uma solução é encontrada, as ações recomendadas podem ser executadas. Isso se chama fase de execução [Russel and Norvig 2004].

2.1. $A^\star$

$A^\star$ (A-star) é um dos principais algoritmos de bucsa de caminhos. Esse algoritmo é usado tipicamente em jogos, como por exemplo Warcraft III. $A^\star$ não é um algortimo de busca em largura (BFS) nem de busca em profundidade (DFS). Na verdade, ele é uma combinação do algoritmo de Dijkstra e busca pela melhor escolha.

Ao programar os algoritmos DFS e BFS, é intuitivo nos questionarmos o que aconteceria se ao invés de expandirmos os nós de "forma cega", escolhêssemos àqueles que parecem mais promissores? A busca $A^\star$ faz exatamente isso. Em resumo, o algoritmo gera as possibilidades (nós) e escolhe aquela com o menor custo. Esse custo é calculado por meio de uma função de avaliação. Essa função é definida da seguinte forma

$$f(n) = g(n) + h(n) \tag{1}$$

Em que g(n) é o custo de sair do estado inicial até o estado atual na busca, e h(n) é uma função heurística que fornece o custo de se alcançar o objetivo a partir do estado atual.

Os custos de cada possibilidade são armazenados em uma lista de prioridades de sorte que os nós com menores custos são escolhidos primeiro.

$A^\star$ consegue achar o menor caminho a partir do nó inicial para qualquer nó objetivo se a função heurística for admissível, ou seja

$$h(n) \leq h^\star(n) \tag{2}$$

para todos os nós, onde h^* é o custo verdadeiro do menor caminho de n até o objetivo mais próximo.

2.2. Iterative deepening $A^\star$ ($IDA^\star$)

Iterative deepening $A^\star$ ($IDA^\star$), é um algoritmo de busca em caminhos que pode encontrar o menor caminho entre um determinado nó inicial e qualquer membro de um conjunto de nós objetivos em um grafo ponderado[Korf 1985]. Ele é uma variação da busca de aprodundamento iterativo[Korf 1985] que toma a ideia de usar uma função heurística para avaliar o custo restante de se alcançar um objetivo, a partir do algoritmo de busca $A^\star$. Visto que ele é um algoritmo de busca em profundidade, o seu uso de memória é menor do que em $A^\star$. Contudo, diferentemente da busca de aprofundamento iterativo, ele se concentra em explorar os nós mais promissores e, portanto, utiliza profundidades variadas, ao invés de profundidades pré-definidas. Em contraste com o algoritmo $A^\star$, $IDA^\star$ não utiliza programação dinânima (lista de visitados) e, por conta disso, acaba visitando os mesmos nós várias vezes.

Enquanto a versão padrão da busca de aprofundamento iterativo usa um limite em cada iteração, $IDA^\star$ utiliza uma abordagem mais inteligente para determinar os limites de aprofundamento na busca. Nesse caso, utiliza a função de avaliação , Equação 1, descrita na seção anterior.

$IDA^\star$ funciona da seguinte forma: em cada iteração, realiza uma busca em profundidade, podando os ramos em que o custo total da função de avaliação excede um determinado limiar. Esse limiar começa com o valor estimado do custo do nó inicial, e aumenta o seu valor em cada iteração do algoritmo. Em cada iteração, o limiar usado para a próxima iteração é o custo mínimo entre todos os valores que excederam o limiar.

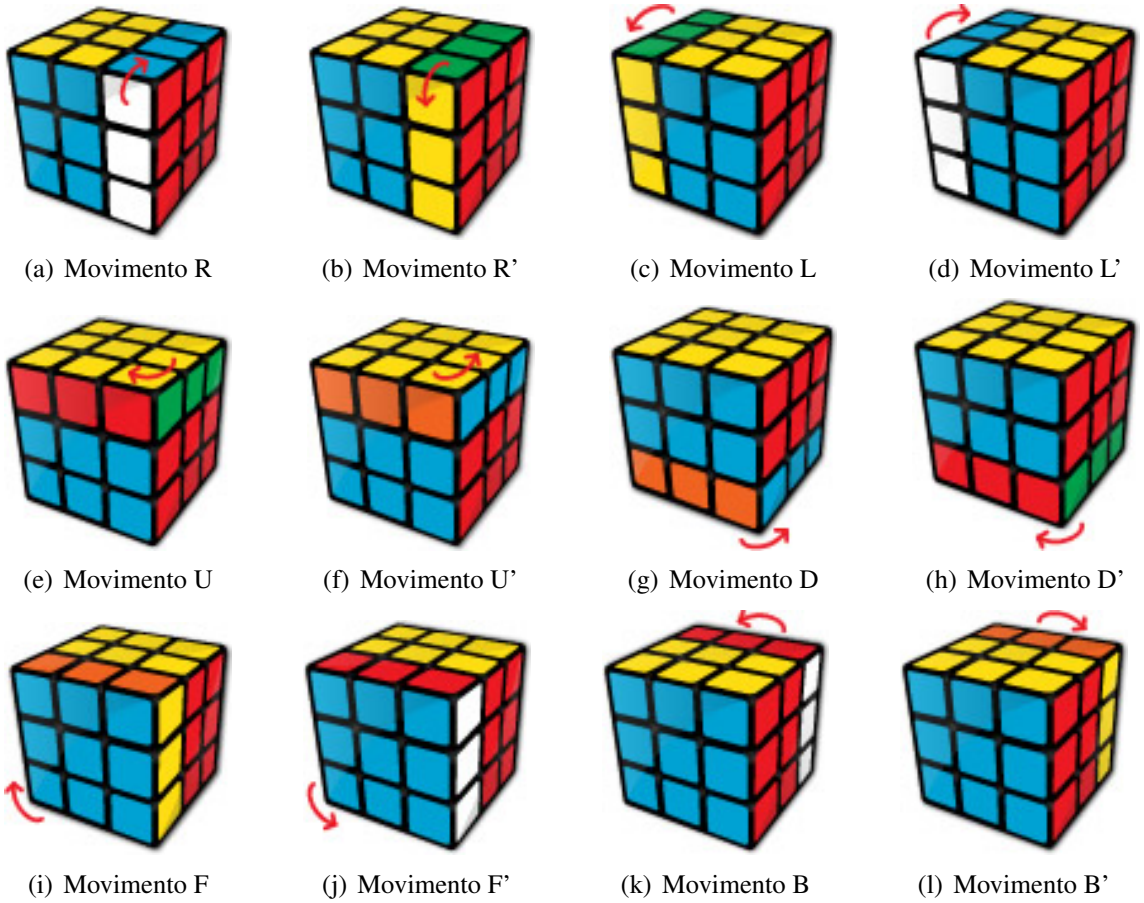
3. Movimentos

Cada uma das 6 faces do cubo é representada por uma letra que significa um movimento em inglês, são eles: R (*right*), L (*left*), U (*up*), D (*down*), F (*front*) e B (*back*) . Se a letra for seguida de um apóstrofo, significa que o movimento é no sentido anti-horário, por exemplo: U', e se for seguida do número 2, significa que o movimento é duplo e em qualquer sentido, por exemplo: L2.

A Figura 3, ilustra os todos os movimentos possíveis do cubo mágico, os movimentos do cubo $2 \times 2 \times 2$ são análogos aos movimentos do $3 \times 3 \times 3$.

4. Metodologia

O primeiro passo para resolver o problema do cubo mágico computacionalmente, é transformar a representação física do *puzzle* em uma forma que os espaços de estados possam ser manipulados por um computador. Sabe-se que o cubo $2 \times 2 \times 2$ possui 6 faces, cada qual com 4 quadrados com um adesivo colorido de uma mesma cor. Assim, se transformamos o cubo em um plano, temos uma figura geométrica conforme ilustrado na figura



(a) Movimento R (b) Movimento R' (c) Movimento L (d) Movimento L'

(e) Movimento U (f) Movimento U' (g) Movimento D (h) Movimento D'

(i) Movimento F (j) Movimento F' (k) Movimento B (l) Movimento B'

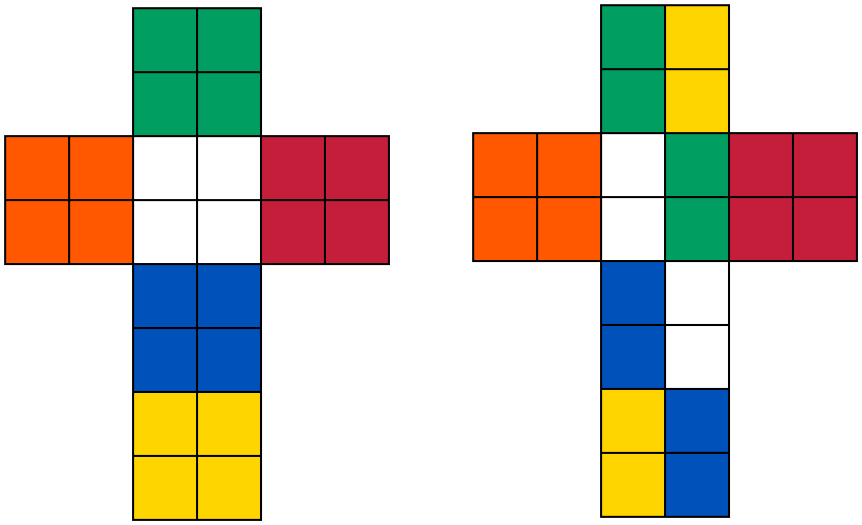
Figura 3. Movimentos do cubo mágico

5, com 24 quadrados, cada qual representando uma das 4 faces dos subcubos do cubo $2 \times 2 \times 2$. Uma forma de representar esse plano de uma maneira que o computador possa interpretar, é por utilizar uma matriz de dimensão 8×6 que armazena os valores dos 24 quadrados do plano.

Dessa forma, a matriz representa os estados do nosso problema. Uma das partes principais desse problema, é a função sucessora (FS). Para definir a FS, utilizamos o conjunto de ações e de estados do problema. Nesse caso, as ações consistiram dos movimentos apresentados na seção anterior e os estados foram considerados como as matrizes. A FS recebe portanto uma matriz e um movimento, e retorna a matriz resultante de se aplicar esse movimento. Nas Figuras 4 e 5 exemplificamos o resultado de se aplicar o movimento R na matriz com estado inicial. Todos os outros movimentos e estados, que somam 24, foram modelados na FS.

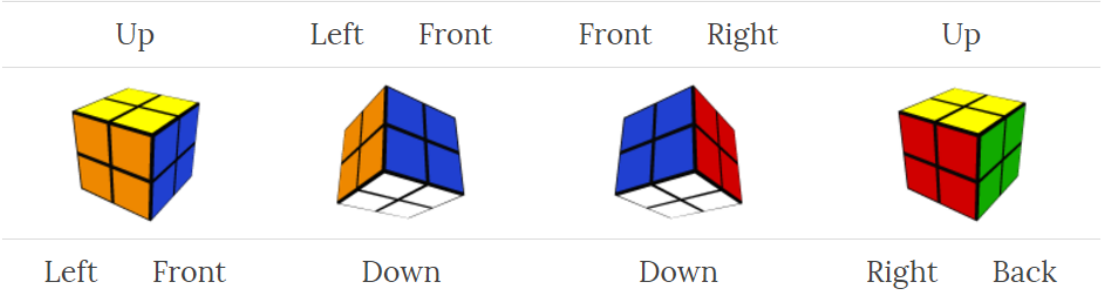
Além da criação da FS, outro fator fundamental implementado nos métodos de buscas com informação empregados neste trabalho, foi a criação da funçao de avaliação. No nosso estudo, utilizamos uma função heurística inspirada na heurística de distância Manhattan. Na heurística implementada, o seu valor consistiou na contabilização de subcubos posicionados corretamente em cada face do cubo $2x2x2$. Essa heurística é viável devido a sua condição de admissibilidade; sabemos que a quantidade de peças posicionadas corretamente nunca será maior do que a quantidade de peças corretas do cubo

solucionado. Já para o caso da função de custo $g(n)$, consideramos valores constantes, visto que no problema cada movimento possui o mesmo custo. No entanto, variações dos valores dessa função foram alterados, devido ao fato de influenciarem a função de avaliação no momento em que são somados com a função heurística.

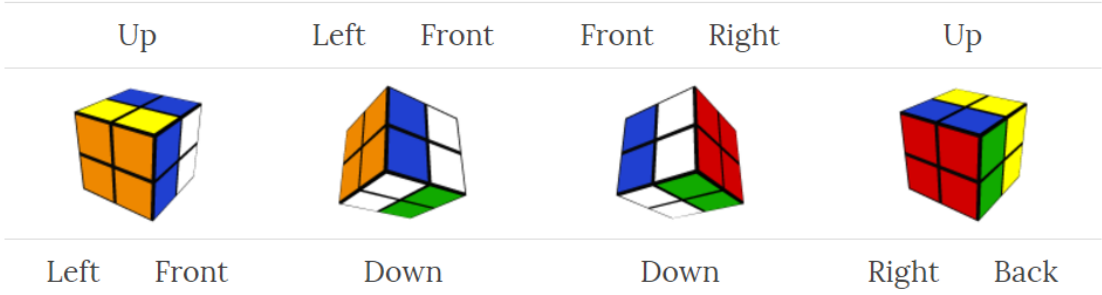


(a) Solucionado (b) Com o movimento R

Figura 4. Representação como um plano



(a) Solucionado



(b) Com o movimento R

Figura 5. Representação 3D

5. Resultados

Para realizar os experimentos e atingir os resultados mostrados, utilizamos um computador com a seguinte configuração: processador Intel(R) Core(TM) i7-6500U a 2.5 GHz com 8 GB de RAM e executando Windowns 10. Além disso, usamos a linguagem de programação *Python* para codificar o problema. Os resultados dos experimentos são mostrados na Tabela 5.

Movimentos Aleatórios	**Tempo (em segundos)**			
	IDA*	**A***	**BFS**	**DFS**
1	0.00	0.00	0.01	-
2	0.00	0.00	0.20	-
3	0.01	0.00	0.11	-
4	0.11	0.02	2.94	-
5	0.83	0.09	-	-
6	7.08	201.87	-	-
7	66.1	425.48	-	-
8	9.09	-	-	-
9	69.5	-	-	-
10	920.95	-	-	-

Tabela 1. Resultados das Buscas

6. Conclusão

Neste trabalho utilizamos métodos be busca com informação, a saber, os algoritmos $A^\star$ e $IDA^\star$, e métodos de busca sem informação, especificamente os algoritmos BFS e DFS, para resolver o problema do cubo mágico $2 \times 2 \times 2$. Uma comparação foi realizada para identificar as vantagens e desvantagens entre esses métodos. Percebemos que o algoritmo $IDA^\star$ conseguiu resolver o problema em menos tempo do que o $A^\star$ nos casos em que a quantidade de movimentos aleatórios inicias foram maiores que quatro, ou seja, nas situações mais difíceis testadas, em contrapartida o $A^\star$ mostrou-se melhor nos casos com poucos movimentos aleatórios. Além disso, percebemos que as buscas sem informações mostraram-se intratáveis para resolver o problema.

Para investigações posteriores, pretende-se fazer uma comparação com variações de heurísticas de escolha dos nós a serem expandidos, tais como a heurística *Pattern Databases*, e comparações com outras abordagens, a exemplo de algoritmos genéticos [Holland 1992], problemas de satisfação de restrições [Kumar 1992] e *simulated annealing* [Kirkpatrick et al. 1983].

Referências

Cube20 (2017). God's number is 20. Disponível em `http://www.cube20.org/`. Setembro.

Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66–73.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109.

Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32.

Rubiks (2017). The history of the rubiks cube. Disponível em `https://www.rubiks.com/about/the-history-of-the-rubiks-cube/`. Setembro.

Russel, S. and Norvig, P. (2004). Inteligência artificial. *Editora Campus*.