Roteamento e o Cubo Mágico

Gustavo Ribeiro Monteiro e Daniel Sadoc Menasché

[1]Dept de Ciência da Computação, Universidade Federal do Rio de Janeiro (UFRJ)

Abstract. *The magic cube, also known as Rubik cube, is a popular game that has recently attracted attention from the scientific community as a stylized problem to illustrate the applicability of machine learning techniques. In this paper, we show novel results that leverage symmetry properties of the Rubik cube for routing purposes. Given two reachable states of the cube, we show that we can efficiently route from one state to the other given a solution to the standard Rubik cube problem. Then, we indicate how the proposed efficient algorithm can be used to refine existing suboptimal solutions to the shortest path routing between states of the Rubik cube.*

Resumo. *O cubo mágico, também conhecido como cubo de Rubik, é um jogo popular que recentemente atraiu a atenção da comunidade científica como um problema estilizado para ilustrar a aplicabilidade das técnicas de aprendizado de máquina. Neste artigo, mostramos novos resultados que aproveitam as propriedades de simetria do cubo Rubik para fins de roteamento. Dados dois estados alcançáveis do cubo, mostramos que podemos rotear eficientemente de um estado para outro, dada uma solução para o problema padrão do cubo Rubik. Em seguida, indicamos como o algoritmo eficiente proposto pode ser usado para refinar as soluções abaixo do ideal para o roteamento de caminho mais curto entre os estados do cubo Rubik.*

1. Introdução

O Rubik's Cube ou Cubo Mágico foi criado em 1974 pelo professor de arquitetura Erno Rubik e teve grande popularidade e reconhecimento nos anos 80. O fascínio do quebra cabeça é a sua simplicidade aparente, que mascara uma grande complexidade para atingir seu estado resolvido.[1]

Motivação. Uma equipe de pesquisadores da Califórnia publicou na Nature, em 2019, uma rede neural profunda chamada DeepCubeA. Essa rede é capaz de levar qualquer estado do cubo para o estado resolvido. Em cerca de 60% dos casos, segundo os dados de teste, a rede gera o caminho mais curto [Agostinelli et al. 2019].

A DeepCubeA se baseia na Busca A*, que por sua vez demanda uma função heurística admissível. Como seria impraticável gerar uma tabela de custo de transição para cada estado do cubo, a estratégia foi criar uma rede neural artificial que oferecesse o valor da heurística para cada estado.

A DeepCubeA resolve o problema de múltiplas origens para um destino. Ou seja, leva qualquer um dos cerca de 4.10^{19} estados possíveis para o único estado resolvido. Porém, a rede não resolve o problema mais geral de levar um estado qualquer do cubo para

[1]Este trabalho foi parcialmente financiado pela FAPERJ, CNPq e CAPES.

outro estado qualquer do cubo. *Algoritmos de roteamento de uma origem para múltiplos destinos, de múltiplas origens para um destino, e de roteamento de múltiplas origens para múltiplos destinos, são fundamentais em redes de computadores. Neste trabalho, indicamos propriedades particulares da solução destes problemas que se aproveitam da simetria da topologia induzida pelo cubo de Rubik.*

Contribuição. O principal objetivo do presente trabalho é expandir o problema original do cubo de múltiplas origens e um destino, para um problema geral de múltiplas origens e múltiplos destinos sem *overhead* computacional significativo. *Para tal, propomos um método para rotear de qualquer fonte a qualquer destino na topologia do cubo de Rubik com custo computacional semelhante ao de resolver o problema de roteamento original de múltiplas fontes para um destino.*

A ideia de nossa solução se baseia na semelhança/simetria entre estados do cubo. Como exemplo, considere o estado resolvido **S** (com uma cor para cada face) e 2 estados do cubo **A** e **B**. Se formos capazes de criar um estado **X** tal que o caminho (sequência de movimentos) de **X** a **S** seja o mesmo de **A** a **B** e tivermos um oráculo (DeepCubeA) que nos forneça um caminho de **X** a **S**, também teremos um caminho de **A** para **B**. *Neste trabalho, indicamos a viabilidade de se gerar de forma eficiente o estado **X** acima referido.*

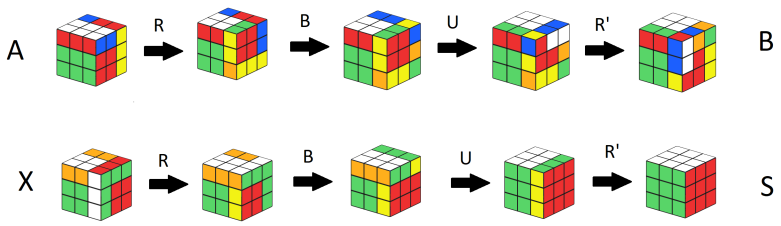


Figura 1. Exemplo simples em que aplicar a mesma série de movimentos leva o cubo de A para B e também de X para S (vide notação na Seção 3.2).

2. O cubo mágico e suas aplicações

2.1. O cubo mágico

Intuitivamente, o Cubo Mágico (cubo) é feito por 27 cubinhos que se organizam em formato de um cubo com dimensões 3x3x3, e por isso apenas 26 são visíveis. Cada face aparente dos cubinhos é adesivada com uma de 6 cores de tal forma que seja possível levar o cubo ao seu *estado resolvido com uma cor por face*. Para embaralhar ou resolver o cubo, é possível aplicar movimentos que são rotações em sentido horário ou anti horário de 90º ou 180º em uma face do cubo. Com esse simples conjunto de regras, o cubo possui apenas 1 estado resolvido entre cerca de $4,3.10^{19}$ estados possíveis. A topologia do espaço de estado do cubo é completamente simétrica, onde todos os pares de estado origem e estado destino são alcançáveis entre si em até 20 movimentos [Rokicki et al. 2014]. Sendo assim, o cubo mágico é um exemplo canônico para avaliação de técnicas de exploração de amplos espaços de estados, e.g., para problemas de busca e roteamento.

2.2. Aplicações do cubo mágico em redes de computadores

Embora o cubo mágico tenha recentemente atraído muita atenção da comunidade científica (vide [Agostinelli et al. 2019] e referências), não é de nosso conhecimento nenhuma aplicação concreta útil do cubo mágico na área de redes. Mesmo levando-se em

conta todas as áreas do conhecimento, a aplicabilidade do cubo de Rubik ainda é muito limitada [Hassan and Abdulmuim 2019, Lai et al. 2019], sendo algumas das aplicações na área de criptografia [Zeng et al. 2018] e artes [Mannone et al. 2019]. A seguir, indicamos potenciais aplicações do cubo mágico em redes, que se aproveitam dos resultados apresentados no restante deste artigo. Em todos os casos, elas fazem uso da simetria do cubo mágico para fins de roteamento.

Cabe destacar que em 1994, os autores de [Fortin et al. 1994] fizeram uma conjectura de que o cubo de Rubik poderia ser usado para fins de roteamento. Entretanto, por limitações tecnológicas não foi possível seguir em frente com tal projeto. Em 2019, um resultado importante publicado na Nature sobre a solução do cubo de Rubik [Agostinelli et al. 2019] sugere que o momento é adequado para rever conjecturas como aquelas indicadas por [Fortin et al. 1994], sendo este um dos propósitos do presente trabalho.[2]

Endereçamento de conteúdos e nós em redes de cache chave-valor (*key-value* caches): Nossa proposta consiste em endereçar nós de uma rede p2p usando estados do cubo mágico. Alternativamente, em uma rede orientada por conteúdos podemos endereçar conteúdos usando estados do cubo mágico, e cada nó passa a ser responsável por um conjunto de conteúdos. Tais abordagens são clássicas em redes chave-valor (*key-value*) [Huq et al. 2017] (vide também Figuras 1 e 2 em [Zhao et al. 2004]). Redes overlay com simetrias podem ser exploradas para fins de roteamento, e o presente trabalho apresenta uma nova proposta nesta linha. Vários dos desafios subjacentes, como desbalanceamento de nós e assimetrias nos atrasos nas redes foram fruto de trabalhos relacionados, e vislumbramos que para contornar tais desafios dentro do ambiente considerado podemos usar ideias similares, sendo tais questões assunto para trabalhos futuros.

Roteamento pela fonte – source routing: Em muitas aplicações de rede deseja-se que a fonte seja capaz de estabelecer o caminho completo da fonte ao destino. Tais soluções, conhecidas como *source routing*, podem beneficiar-se de simetrias na topologia da rede [Johnson and Maltz 1996]. Neste trabalho, indicamos que topologias inspiradas pelo cubo mágico são promissoras para realizar roteamento pela fonte de forma eficiente.

Roteamento topológico – topology-aware routing (Chord, Kelips e Tapestry): Chord, Kelips e Tapestry são alguns dos famosos algoritmos para roteamento em redes P2P que tomam proveito da topologia da rede [Chan et al. 2008, Castro et al. 2003, Zhao et al. 2004]. Tais algoritmos enquadram-se na classe de *toplogy-aware routing*, e são baseados na construção de uma topologia com propriedades de simetria especiais que garantem um eficiente roteamento. Neste trabalho, sugerimos que usando os estados do cubo mágico como vértices da topologia, e caracterizando as adjacências entre estados a partir dos possíveis movimentos do cubo, podemos abrir margem para uma nova ampla gama de soluções de roteamento ainda não exploradas. Em particular, indicamos que nesta topologia o problema de rotear entre qualquer par origem-destino pode ser resolvido de forma eficiente se tivermos a solução de roteamento para um dado destino específico.

[2]O trabalho [Delorme and Panaite 1996], embora refira-se a "Rubik routing" no título, trata de tema não diretamente relacionado ao presente artigo.

3. Roteamento no cubo mágico

3.1. Modelagem do cubo

Existem três principais estratégias para modelar o cubo mágico. A primeira é pela representação dos adesivos e suas cores. A segunda é por identificadores únicos e a terceira é pela representação das peças levando em conta posição e orientação.

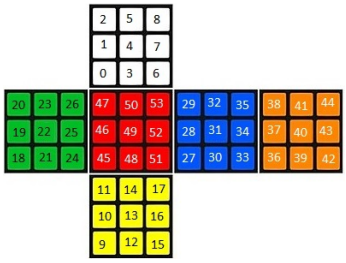


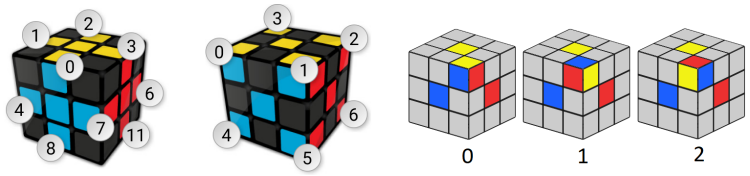
Figura 2. Indexação dos adesivos utilizados neste trabalho.

Representação por cores: Cada valor de um vetor de 54 posições, indexadas de 0 a 53, representa a cor do adesivo presente no cubo na posição indexada (vide Figura 2). Para o presente trabalho, o mapa de cores é: 0 = branco, 1 = amarelo, 2 = verde, 3 = azul, 4 = laranja, 5 = vermelho. A representação por cores do estado resolvido é o seguinte vetor: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5]

Representação com identificadores único: Segundo esta representação, enumera-se unicamente cada adesivo do cubo. Cada valor de um vetor de 54 posições, indexadas de 0 a 53, representa o adesivo presente no cubo na posição indexada. Seguindo a indexação definida para o trabalho (Figura 2), o estado resolvido é um vetor de 54 posições, contendo a sequência de números naturais ordenados de forma crescente, de 0 a 53 (vide novamente Figura 2).

Representação por peças: Para essa modelagem, focamos no fato de que o cubo possui ao todo 20 peças, sendo 12 meios (peças com 2 cores visíveis) e 8 quinas (peças com 3 cores visíveis). Note que as peças de centro (com 1 cor visível) não são explicitamente representadas nesse modelo. Toda peça possui 3 atributos:

- **Posição:** É o espaço do cubo em que a peça se encontra. Para ligarmos cada espaço a um número, utilizamos a indexação sugerida pelo site CUBO VELOCIDADES (Figuras 3(a) e 3(b)). Como uma peça de meio não pode ocupar o espaço de uma peça de quina e vice-versa, podemos indexar a posição dos 2 tipos de peça de forma independente.
- **Orientação:** As orientações de uma peça são as diferentes formas em que a peça pode se apresentar em uma determinada posição do cubo. Por exemplo, na Figura 3(c) ilustramos as 3 orientações possíveis para uma peça de quina em uma determinada posição.
- **Identificação:** Cada peça é identificada por um número. Este número é determinado pela posição da peça no estado resolvido. Por exemplo, na Figura 3(b) a peça identificada pelo número 1 é uma peça azul-amarelo-vermelho, que recebe o número 1 já que no cubo resolvido ela se encontra na posição 1.



(a) (b) (c)

Figura 3. (a) e (b): indexação das posições das peças. Note que temos um total de 20 posições, mas na figura apenas 16 estão representadas. As 4 restantes do fundo são denotadas por 7 (quina) e 5, 9 e 10 (dos meios). Fonte: [Cerpe 2007]; (c): três orientações possíveis de uma peça de quina em uma dada posição.

Para representar orientações de forma numérica, utilizamos as seguintes regras. No caso das peças de quina, uma peça terá (i) orientação 0 se o adesivo branco ou amarelo estiver voltado para face branca ou amarela (sendo as cores das faces aquelas correspondentes ao estado resolvido do cubo); (ii) orientação 1 se a peça tiver rotacionada no sentido horário em relação a sua orientação 0 e (iii) orientação 2 se tiver rotacionada no sentido anti-horário em relação a sua orientação 0 (vide exemplo na Figura 3(c)). No caso das peças de meios, uma peça terá (i) orientação 0 se tiver as cores branco ou amarelo voltadas para uma das regiões cinza da Figura 4 ou, caso a peça não tenha nem branco nem amarelo, terá orientação 0 se tiver as cores vermelho e laranja voltadas para uma das regiões já citadas; e (ii) terá orientação 1 caso contrário.

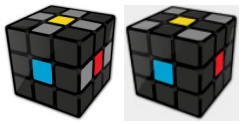


Figura 4. Adesivos de referência de meios e de quinas [Cerpe 2007].

Neste trabalho utilizamos um vetor de tamanho 8 (resp., 12) para representar a posição das peças de quinas (resp., meios). O i-ésimo elemento deste vetor contém o identificador da peça que encontra-se na posição i, e.g., se a peça 6 estiver na posição de quina 0, o vetor de posição das peças de quina conterá o valor 6 na posição 0. Para representar a orientação das quinas (resp., meios), utilizamos um vetor de tamanho 8 (resp., 12) que armazena as orientações das 8 (resp., 12) peças que se encontram em cada uma das 8 (resp., 12) posições de quina (resp., meio). Cada elemento do vetor de orientação de peças de quinas (resp., peças de meios) é um número entre 0 e 2 (resp., entre 0 e 1), inclusive, seguindo as regras explicadas acima.

Nessa modelagem o estado resolvido é:

$$cornerPosition = [0, 1, 2, 3, 4, 5, 6, 7] \quad (1)$$

$$cornerOrientetion = [0, 0, 0, 0, 0, 0, 0, 0] \quad (2)$$

$$edgePosition = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] \quad (3)$$

$$edgeOrientetion = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] \quad (4)$$

Uso das diferentes representações: Conforme indicado acima, adotamos três representações distintas do cubo mágico. A representação por cores é a entrada da rede neural artificial (ANN), a representação por peças é usada para calcular o estado auxiliar e a por identificadores únicos é usada em todo o restante do processamento.

3.2. Modelagem dos movimentos e caminhos

O cubo possui 6 faces que podem ser rotacionada em sentido horário e anti-horário. Para este trabalho usaremos a notação proposta no capítulo 3 do livro [Singmaster 1981]. Cada face é representada por uma letra: F = frente, R = direita, L = esquerda, U = topo, B = costas, D = baixo. Um giro de 90º no sentido horário é representado pela letra da face (Exemplo: [F, R, L, U, B, D]). Um giro 90º no sentido anti-horário é representado pela letra da face seguindo de aspas simples (') (Exemplo: [F', R', L', U', B', D']). O movimento composto por dois giros seguidos, na mesma face, é representado pela letra da face seguido do número 2 (Exemplo: [F2, R2, L2, U2, B2, D2]).

Um caminho é um sequência de movimentos. Aplicar um caminho em um estado significa efetuar a sequência de movimentos a partir do estado em que o caminho está sendo aplicado.

3.3. Apresentação da DeepCubeA

A já citada DeepCubeA é uma abordagem que busca a solução do cubo através de uma variação da busca A* com a função de heurística sendo implementado por uma *Artificial Neural Network* (ANN). A busca A* procura o menor caminho entre dois estados, expandindo os estados de menor custo. O custo é calculado por $f(x) = g(x) + h(x)$, onde $g(x)$ é o custo total do caminho até o estado x, e $h(x)$ é uma heurística que estima o custo de x até o estado desejado.

Para criar a função $h(x)$, foi criada uma Rede Neural Artificial (ou *Artificial Neural Network*, ANN) que é treinada por aprendizado profundo por reforço. A estratégia consiste em, partindo de um cubo resolvido, criar dados de treino efetuando embaralhamentos aleatórios e dando reforço para a ANN quando ela retornar a estimativa correto para cada estado.

A rede é formada por 1 camada de entrada com 5.000 nós e 1 camada com 1.000 nós completamente conectadas. Em seguida há 4 blocos residuais cada um com 2 camadas com 1.000 nós. A saída consiste um único valor que é a $h(x)$ [Agostinelli et al. 2019].

4. Resolvendo o cubo mágico estendido: caminho mais curto de múltiplas fontes para múltiplos destinos via geração do estado auxiliar

Nesta seção, descrevemos a solução do problema do cubo mágico. Iniciamos com duas definições.

Definição 1 *A solução do cubo mágico tradicional consiste num caminho de um estado A, dado, para o estado final no qual cada face do cubo tem apenas uma cor.*

Definição 2 *A solução do cubo mágico estendido consiste num caminho de um estado A, dado, para um estado final qualquer.*

4.1. Abordagem ingênua

A abordagem ingênua para resolver o problema de múltiplas fontes para múltiplos destinos consiste em resolver duas vezes o problema de múltiplas fontes para único destino. Ou seja, resolvendo duas vezes o problema do cubo mágico tradicional, podemos

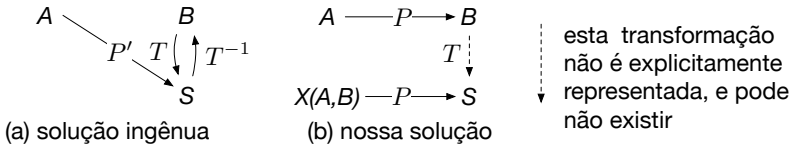


Figura 5. Transformação de estados.

resolver o problema do cubo mágico estendido (vide Figura 5). Na Figura 5(a) indicamos que a solução $A \rightarrow S$ combinada com o caminho inverso ao da solução $B \rightarrow S$ (representado por T^{-1}) gera a solução $A \rightarrow B$ para qualquer par de estados A e B. Entretanto, tal solução apresenta alguns problemas:

1. **alto número de movimentos do cubo**: o número de operações será igual a soma do número de operações para ir do estado inicial para o estado final do cubo padrão, e de tal estado para o estado final desejado. É possível encontrar soluções muito melhores adotando-se a nossa abordagem.

2. **incapacidade de melhorar soluções existentes**: A proposta ingenua não permite melhorar as soluções geradas. Nossa solução, em contrapartida, tem a vantagem de poder ser usada para refinar caminhos já pre-existentes.

3. **custo computacional**: a abordagem ingênua tem custo computacional igual ao dobro do custo da solução padrão. Já a nossa abordagem terá um custo tipicamente menor que a abordagem ingênua.

4. **lidando com estados inalcançáveis**: a abordagem ingênua não é capaz de lidar com estados A ou B que não alcancem o estado resolvido padrão S. Já a abordagem alternativa é capaz de fazê-lo, porque em nenhum momento se calcula explicitamente o caminho nem de A nem B para S.

4.2. Abordagem por ANN

A DeepCubeA, depois de treinada, resolve o problema tradicional em 60% das vezes de forma ótima, sem consumo significativo de memória e em tempo relativamente curto [Agostinelli et al. 2019]. Porém para levar qualquer estado do cubo para qualquer outro estado, seguindo a mesma estratégia, precisaríamos de uma ANN que recebesse como *entrada* tanto o estado de origem quanto o estado de destino, e retornasse a estimativa do custo de um para o outro. Para isso precisaríamos re-treinar a rede não mais a partir de embaralhamentos do caso resolvido, mas sim a partir de embaralhamentos de diversos casos adicionais para novos pares origem-destino. Esta abordagem possivelmente iria requerer uma arquitetura maior, seja em profundidade ou em nós por camada.

4.3. Nossa abordagem: estados auxiliares e caminhos equivalentes

Utilizando simetrias entre cubos podemos criar um estado auxiliar para cada caso do problema estendido de tal forma que a solução tradicional do estado auxiliar é a mesma solução do caso estendido. Assim, resolvemos o problema estendido sem precisar retreinar a rede ou fazer qualquer alteração no algoritmo que resolve o problema tradicional.

4.3.1. Independência entre caminho e estado do cubo

Todos os movimentos disponíveis podem ser aplicados em qualquer estado do cubo (vide Seção 3.2). Assim, não há restrições entre estados e movimentos. Portanto, todo e qualquer caminho pode ser aplicado a todo e qualquer estado. Dado um estado qualquer **A** e outro estado qualquer **B** achar um caminho **P** que leve de **A** para **B** se resume a achar um caminho que rearranje e rotacione as peças de tal forma que quando aplicado em **A** resulte em **B**. Porém, não precisamos efetivamente procurar **P** partindo de **A**. Ou seja, não precisamos efetuar uma busca partindo do estado **A** e aplicando movimentos até chegar no estado **B**.

Podemos achar o caminho **P** aplicando movimentos a partir de qualquer estado inicial **X** até chegar a um estado **S** tal que o rearranjo e a rotação de **X** para **S** seja o mesmo rearranjo e a rotação necessários para levar de **A** para **B**. Assim, podemos usar qualquer método que resolve o problema tradicional para descobrir o caminho **P** desejado. Na próxima seção mostraremos como construir um estado auxiliar **X** no qual o caminho de **X** para **S** rearranje e rotacione as peças da mesma forma que o caminho **P** que leva de **A** para **B**.

4.3.2. Estado auxiliar

O estado auxiliar **X** é um estado do cubo a partir do qual, aplicando **P**, temos o estado resolvido do problema tradicional, denominado **S**. O algoritmo para criarmos o estado **X** recebe como entrada as representações do estados origem e destino, **A** e **B**, respectivamente, e retorna a representação do estado **X**. Utilizando a modelagem por peças apresentada na Seção 3.1, um estado **A** é o conjunto de 4 vetores, denotados por $ACornerPosition, ACornerOrientation, AEdgePosition$ e $AEdgeOrientation$. Os primeiros dois vetores possuem dimensão 1×8 e os dois últimos 1×12. A i-ésima posição de cada um desses vetores é denotada por $a_{cp}(i), a_{co}(i), a_{ep}(i)$ e $a_{eo}(i)$, respectivamente. O estado **B** é o conjunto dos 4 vetores correspondentes, cuja j-ésima posição é denotada por $b_{cp}(j), b_{co}(j), b_{ep}(j)$ e $b_{eo}(j)$. A criação do estado **X** é baseada em duas estratégias.

Posição: Tomemos uma peça de quina como exemplo. Ao aplicar **P** em **A** a peça identificada por $a_{cp}(i)$ sai da posição i para a posição j tal que $a_{cp}(i) = b_{cp}(j)$. Para criar **X**, colocaremos a peça cuja identificação é igual a j na posição i do vetor de posições do estado **X**. Assim, ao aplicar **P** em **X** teremos a peça identificada por j na posição j, que é a configuração do estado resolvido mostrado na Seção 3.1. Para as peças de meios, utilizamos a mesma estratégia.

Orientação: Tomemos uma peça de quina como exemplo. Ao aplicar **P** em **A** a peça $a_{cp}(i)$ é levada da orientação $a_{co}(i)$ para $b_{co}(j)$ tal que $a_{cp}(i) = b_{cp}(j)$. Ou seja, **P** rotaciona $(b_{co}(j) - a_{co}(i)) \mod 3$ vezes a peça $a_{cp}(i)$. Para criamos o estado **X** a peça de quina que ocupa a posição i deve estar na orientação $(a_{co}(i) - b_{co}(j)) \mod 3$. Ao aplicar **P** em **X** a orientação da peça será 0 que é a orientação de todas as peças do estado resolvido como mostrado na Seção 3.1. Para a orientação das peças de meio, seguimos o mesmo princípio. A orientação da peça que ocupa o espaço i do estado **X** deve ser $(a_{eo}(i) - b_{eo}(j)) \mod 2$, sendo $a_{ep}(i) = b_{ep}(j)$.

4.3.3. O algoritmo

O algoritmo para gerar o estado auxiliar **X** está listado no Código 1 em linguagem Pytohn. De acordo com essa implementação, a criação do estado auxiliar leva exatas 120 consultas indexadas a vetores, 60 atribuições a vetores, 20 subtrações e 20 operações de modulo. Totalizamos assim 220 operações.[3]

Código 1. Cria estado auxiliar

```
1  import numpy as np
2
3  def createX(Acp, Aep, Aco, Aeo, Bcp, Bep, Bco, Beo):
4
5      # Declarando as estruturas de dados do estado auxiliar
6      Xcp = np.zeros_like(Acp)
7      Xep = np.zeros_like(Aep)
8      Xco = np.zeros_like(Aco)
9      Xeo = np.zeros_like(Aeo)
10
11     # Pre computando o index das quinas.
12     Bcp_index = np.zeros_like(Bcp)
13     for i in range(len(Bcp)):
14         Bcp_index[Bcp[i]]=i
15
16     # Criando o vetor de posicao das quinas do estado auxiliar
17     for i in range(len(Acp)):
18         Xcp[i] = Bcp_index[Acp[i]]
19
20     # Pre computando o index dos meios.
21     Bep_index = np.zeros_like(Bep)
22     for i in range(len(Bep)):
23         Bep_index[Bep[i]]=i
24
25     # Criando o vetor de posicao dos meios do estado auxiliar
26     for i in range(len(Aep)):
27         Xep[i] = Bep_index[Aep[i]]
28
29     # Criando o vetor de orientacao das quinas do estado auxiliar
30     for i in range(len(Aco)):
31         posicao_final = Bcp_index[Acp[i]]
32
33         orientacao_inicial = Aco[i]
34         orientacao_final = Bco[posicao_final]
35         Xco[i] = (orientacao_inicial - orientacao_final) %3
36
37     # Criando o vetor de orientacao dos meios do estado auxiliar
38     for i in range(len(Aeo)):
39         posicao_final = Bep_index[Aep[i]]
40
41         orientacao_inicial = Aeo[i]
42         orientacao_final = Beo[posicao_final]
43         Xeo[i] = (orientacao_inicial - orientacao_final) %2
44
45     return (Xcp, Xep, Xco, Xeo)
```

A seguir, descrevemos o código linha a linha.

Declaração de variáveis: As linhas 6-9 declaram os vetores da representação por peças do estado **X** (vide Seção 3.1).

Pré-calculando a posição das peças: Os loops das linhas 13-14 e 22-23 preenchem as estruturas Bcp_index e Bep_index, respectivamente, que são vetores das posi-

[3] O código fonte está também disponível no site do projeto: `https://github.com/GustavoMonteiroUFRJ/Rubik_Cube_Routing`

ções das peças indexadas pelo identificador de peça. Ou seja, dado um identificador de peça, o Bcp_index retorna sua posição no vetor Bcp. Tais estruturas são utilizadas para otimização do código evitando múltiplas buscas nos vetores Bcp e Bep.

Preenchendo o vetor de posição das quinas e meios: O loop das linhas 17-18 preenche o vetor de posição das peças de quina. $Xcp[i]$ é o identificador da peça que ocupa a posição i do estado **X**. $Acp[i]$ é o identificador da peça que ocupa a posição i do estado **A** e $Bcp\_index[< id >]$ é a posição que a peça identificada por $< id >$ ocupa no estado **B**. A linha 18 pode ser lida da seguinte maneira: O identificador da peça de quina na posição i do estado **X** é a posição da peça, no estado **B**, que ocupa a posição i do estado **A**. O loop das linhas 26-27 é similar ao das linhas 17-18, aplicando-se às peças dos meios.

Preenchendo os vetores de orientação das quinas e dos meios: O loop das linhas 30-35 preenche o vetor de orientação das peças de quina. $posicao\_final$ guarda o valor da posição no estado B da peça que ocupa a posição i do estado **A**. $orientacao\_inicial$ é a orientação que a peça que ocupa a posição i do estado **A** possui no estado **A**. $orientacao\_final$ é a orientação no estado **B** da peça que possui o mesmo identificador que a peça que o ocupa a posição i do estado **A**. $Xco[i]$ é a orientação da peça que ocupa o espaço i no estado **X**. A conta em módulo 3 é explicada no item **Orientação** da Seção 4.3.2. O loop das linhas 38-43 é similar ao das linhas 30-35, aplicando-se às peças dos meios.

5. Resultados formais

Definição 3 *Uma instrução computacional básica é um acesso a um vetor para leitura ou escrita.*

Teorema 4 *Dado um oráculo que resolve o cubo mágico clássico, o algoritmo apresentado na última seção para gerar o estado auxiliar* ***X*** *pode ser usado para resolver o cubo mágico estendido.*

Demonstração: A seguir, mostramos que a partir do estado auxiliar **X**, os movimentos para se chegar no estado final **S** são os mesmos que aqueles necessários para se chegar em **B** a partir de **A**.

Seja $p_A(i)$ a posição da peça de quina i no estado **A**, e seja $f_A(i)$ a posição final de tal peça no estado **B** após aplicar-se as transformações correspondentes ao caminho P (vide relação (5) abaixo). A função $\varphi(i) : i \mapsto j$ denota que a peça i no estado **A** deve ser trocada pela peça $j = \varphi(i)$ para construir-se o estado **X**. Queremos mostrar que o estado **X** é tal que, aplicando-se as transformações em P, atinge-se o estado **S** (vide relação (6)). Recordando, o estado **S** é caracterizado pelo fato de que a posição de cada peça em tal estado é igual ao identificador da peça (vide lado direito de (6) e Eqs. (1)-(4)).

$$p_A(i) \xrightarrow{P} f_A(i) \tag{5}$$

$$p_X(\varphi(i)) \xrightarrow{P} f_X(\varphi(i)) = \varphi(i) \tag{6}$$

Igualando o lado direito de (5) e (6), seja $\varphi(i) = f_A(i)$ (correspondente às linhas 17-18 do Algoritmo 1). Assim, ao aplicar-se P em **A** leva-se a peça i para a posição $f_A(i)$. Logo, trocando-se a peça i por $\varphi(i)$, constrói-se **X**, e ao aplicar-se P em **X** leva-se a peça $\varphi(i) = f_A(i)$ também para a posição $f_A(i)$.

O argumento acima foi aplicado ao vetor de posições de peças de quinas: as posições das peças de quina decorrentes da aplicação de P em **X** são condizentes com o estado **S**. O mesmo argumento vale para os vetores de posição de peças do meio, e orientação de quinas e meios. Concluímos, então, que aplicando-se P em **X** leva-se o sistema para o estado **S**, conforme desejado.

Corolário 5 *Dado um oráculo que resolve o cubo mágico clássico, de forma possivelmente subótima, ao custo de T instruções computacionais básicas, o cubo mágico estendido pode ser resolvido, também de forma subótima, em T + 220 instruções computacionais básicas.*

Demonstração: Segue imediatamente da implementação do algoritmo.

Corolário 6 *A solução do cubo mágico estendido pode ser usada para refinar a qualidade solução do cubo mágico clássico.*

Demonstração: O caminho ótimo de **A** para **S** é o inverso do caminho ótimo de **S** para **A**. Então, para refinar a solução de **A** para **S** é possível executar o problema estendido de **S** para **A** e selecionar o melhor caminho.

6. Resultados numéricos

6.1. Exemplo ilustrativo preliminar

A seguir, tomamos como exemplo dois padrões propostos no site [Ruwix 2019]. Chamaremos de **A** o padrão "Displaced Motif" e de **B** o padrão "Cube in the cube". **A** é atingido quando, partindo do estado resolvido, aplicamos o caminho: L2 B2 D' B2 D L2 U R2 D R2 B U R' F2 R U' B' U'. E **B** é atingido quando partindo do estado resolvido aplicamos o caminho: F L F U' R U F2 L2 U' L' B D' B' L2 U. A visualização dos estados está expressa na Figura 6.

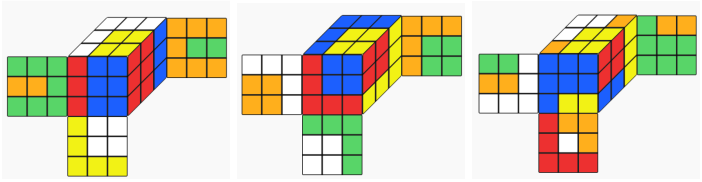


Figura 6. Visualização dos estados A e B, e do estado X ao final.

A representação por peças dos estados **A** e **B** é: $ACornerPosition = [6, 1, 4, 7, 2, 5, 0, 3]$, $ACornerOrientation = [0, 0, 0, 0, 0, 0, 0, 0]$, $AEdgePosition = [0, 10, 9, 3, 6, 5, 4, 7, 8, 2, 1, 11]$, $AEdgeOrientetion = [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]$, $BCornerPosition = [5, 1, 0, 4, 6, 2, 3, 7]$, $BCornerOrientetion = [2, 0, 2, 1, 1, 2, 1, 0]$, $BEdgePosition = [0, 8, 4, 3, 11, 5, 1, 7, 6, 9, 10, 2]$, $BEdgeOrientetion = [0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1]$. Aplicando o algoritmo que cria o estado auxiliar temos: $XCornerPosition = [4, 1, 3, 7, 5, 0, 2, 6]$, $XCornerOrientetion = [2, 0, 2, 0, 1, 1, 1, 2]$, $XEdgePosition = [0, 10, 9, 3, 8, 5, 2, 7, 1, 11, 6, 4]$, $XEdgeOrientetion = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0]$. Utilizando a DeepcubeA para resolver o estado **X** temos um caminho **P** com 15 movimentos em contagem quarter-turn, que levou 5.122068s para ser gerado e visitou 16957 estados antes de ser encontrado: $P = [L, B, D', B, R, R, F', R', D, F, D, R', B', L', D']$.

6.2. Refinando o estado da arte e avaliação de desempenho de nossa proposta

Para criar um subgrafo dos estados do cubo, escolhemos como 5 padrões, denotados por [E_1,E_2,E_3,E_4,E_5], e listados no site [Ruwix 2019].Dados esses estados, um de nossos propósitos é medir as distâncias entre os cinco estados, tomados dois a dois, ou seja, as distância de E_i para E_j, para todo par (i, j). Para tal, geramos 25 estados auxiliares para serem resolvidos por meio da metodologia proposta na Seção 4.3. Dos 25 estados, 5 representavam a distância de E_i para E_j com $i = j$, i.e., o estado auxiliar era o próprio estado resolvido, e sua solução tem tamanho zero (diagonal das matrizes abaixo). Notamos também que o comprimento do caminho mínimo de E_i para E_j é igual à do caminho mínimo de E_j para E_i, o que nos permite refinar a DeepCubeA (vide Corolário 6). Os resultados obtidos depois de resolver os 25 casos estão expressos na matriz abaixo. O valor na posição (i, j) corresponde ao comprimento do caminho retornado pela DeepCubeA para o caminho que leva de E_i para E_j. Note que a matriz à esquerda não é simétrica, indicando a oportunidade que temos para refinar a solução estado-da-arte aproximada pre-existente do problema de caminho ótimo usando o refinamento aqui proposto, que dá origem à matriz simétrica à direita.

$$\begin{pmatrix} 0 & 28 & 25 & 24 & 28 \\ 30 & 0 & 19 & 20 & 26 \\ 27 & 15 & 0 & 27 & 33 \\ 28 & 24 & 25 & 0 & 24 \\ 30 & 26 & 27 & 30 & 0 \end{pmatrix} \rightsquigarrow \begin{pmatrix} 0 & 28 & 25 & 24 & 28 \\ 28 & 0 & 15 & 20 & 26 \\ 25 & 15 & 0 & 25 & 27 \\ 24 & 20 & 25 & 0 & 24 \\ 28 & 26 & 27 & 24 & 0 \end{pmatrix} \tag{7}$$

A máquina utilizada em nossos experimentos foi uma g4dn.xlarge da AWS que possui uma GPU NVIDIA T4 com 2.560 CUDA Cores e 320 Tensor Cores [Nvidia 2019]. O tempo médio de execução para cada resolução foi de 14,68449s. A quantidade média de estados visitados por solução foi de 49136. O tamanho médio das soluções foi de 25,8 movimentos. *Dos 20 casos não triviais analisados, em 9 deles conseguimos encurtar o caminho entre nós usando a abordagem proposta (vide* (7)*).* Descartando os caminhos de tamanho zero, o tamanho médio dos caminhos na matriz final com as melhores soluções é de 24,2 movimentos.

Para medir o desempenho do algoritmo de criação do estado auxiliar, foi feita uma amostra com 400 estados aleatórios, e foi criado o estado auxiliar para cada uma das combinações 2 a 2 dos estados. O experimento foi executado em um computador com processador Intel Core I5-3550 de 3,30 GHz, e o algoritmo foi escrito em Python 3.7 rodando na plataforma Jupyter Notebook. Foram executadas 79.800 rodadas que levaram o total de 393,75094223s, representando uma média de 0,0049342s por execução. Esse valor corresponde a um *ovehead* de 0,0336% com relação ao tempo médio para resolver o cubo usando a DeepCubeA.

7. Trabalhos relacionados

Não é de nosso conhecimento nenhum trabalho que tenha feito uso do cubo de Rubik para resolver problemas em redes. Mesmo em geral, o uso prático do cubo de Rubik é extremamente limitado, sendo considerado primordialmente para fins artísticos [Hassan and Abdulmuim 2019] ou de criptografia [Hassan and Abdulmuim 2019]. Cabe destacar, no entanto, que intuitivamente tenha sido aceito em 2019 que a solução do cubo de Rubik tenha aplicações práticas, haja vista uma patente concedida a proponentes de uma solução para o problema do cubo de Rubik [Garrett and Hatamian 2019].

Neste trabalho, indicamos direções no sentido de aplicar o cubo de Rubik para geração de topologias em redes de computadores.

No trabalho de [Korf 1997] foi pela primeira vez proposta a utilização uma variação da busca A* para atingir a solução mais curta do cubo mágico. Para criar a heurística, o estudo propõe executar uma busca em largura para calcular todos os casos ótimos de um subconjunto de estados, e guardar os resultados em tabelas. Por exemplo, pode-se guardar qual é o menor caminho para resolver qualquer estado das 8 quinas. Depois de calcular diversas tabelas para diversos subconjuntos, a heurística se baseia em buscar o estado atual nas tabelas, e pegar o máximo entre os menores caminhos de cada subgrupo, assim mostrando-se como uma heurística admissível. *Neste trabalho, fazemos uso da DeepCubeA, que substitui tabelas por redes neurais profundas, seguindo assim as tendências recentes na área de representação de funções de custo.*

Em 2010 um grupo de pesquisadores provaram que o "God's Number" (como é chamado o número mínimo de movimentos para resolver qualquer estado do cubo, ou seja, *o diâmetro da rede de estados*) é exatamente 20. O estudo considera a contagem por *half-turn metric* onde rotações de 180º são contados como apenas um movimento. O limite inferior do God's Number já tinha sido estabelecido como 20 em 1995 por Michael Reid ao provar que um estado em especial precisava de exatos 20 movimentos para ser alcançado. Em 2010, foi possível reduzir por simetrias os $4,3.10^{19}$ estados a cerca de $5,5.10^{7}$, e o limite superior de também 20 movimentos foi estabelecido. A maioria dos estados do cubo está a 17 ou 18 movimentos de distância do cubo resolvido [Rokicki et al. 2014].

Em 2019 pesquisadores da Áustria publicaram um trabalho sobre a visualização dos estados do cubo a fim de analisar os métodos humanos de resolução de cubo. O trabalho propõe um método de representação em 2D dos estados do cubo. Para isso eles utilizaram uma representação dos estados com 54 cores via T-distributed Stochastic Neighbor Embedding (t-SNE), que garante que estados com distâncias próximas fiquem em regiões próximas [Steinparz et al. 2019]. *Neste trabalho, indicamos que a topologia do cubo mágico pode ser instrumental em redes p2p, e.g., do tipo* key-value*, nas quais seja necessário endereçar uma enorme quantidade de conteúdos ou hosts.*

8. Conclusão

O cubo de Rubik é universalmente conhecido como o brinquedo mais vendido no mundo [Agostinelli et al. 2019]. Neste trabalho, indicamos que as simetrias envolvidas no cubo de Rubik podem ser usadas para fins de endereçamento de nós de uma rede. O endereçamento inspirado pelos estados do cubo permite que se roteie mensagens em um amplo espaço de endereçamento, tomando-se proveito de soluções recentes para resolver o cubo de Rubik. Em particular, mostramos que tais soluções recentes podem ser facilmente estendidas para resolver o problema de roteamento de múltiplas fontes para múltiplos destinos. Numericamente, mostramos que um dos subprodutos de nossa abordagem consiste em uma eficiente forma de melhorar as soluções estado-da-arte do cubo de Rubik, indicando como diminuir os caminhos entre pares origem-destino tomando proveito de simetrias até então inexploradas. Acreditamos que este trabalho abra muitos caminhos para trabalhos futuros na intereseção entre redes de computadores (roteamento) e inteligência artificial (como o uso de "deep learning" para resolver o cubo de Rubik).

Referências

[Agostinelli et al. 2019] Agostinelli, F., McAleer, S., Shmakov, A., and Baldi, P. (2019). Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363.

[Castro et al. 2003] Castro, M., Druschel, P., Hu, Y. C., and Rowstron, A. (2003). Topology-aware routing in peer-to-peer networks. In *Future directions in distributed computing*, pages 103–107. Springer.

[Cerpe 2007] Cerpe, R. (2007). Cubo 3x3x3. `http://www.cubovelocidade.com.br/tutoriais/cubo-magico-blindfolded.html`.

[Chan et al. 2008] Chan, H. N., Van, K. N., and Hoang, G. N. (2008). Characterizing chord, kelips and tapestry algorithms in p2p streaming applications over wireless network. In *Int. Conference on Communications and Electronics*, pages 126–131. IEEE.

[Delorme and Panaite 1996] Delorme, C. and Panaite, P. (1996). Rubik routing permutations on graphs. In *European Conference on Parallel Processing*, pages 283–286. Springer.

[Fortin et al. 1994] Fortin, D., Kirchner, C., and Strogova, P. (1994). Routing in regular networks using rewriting. In *Proceedings of the CADE international workshop on automated reasoning in algebra (ARIA)*, pages 5–8. Citeseer.

[Garrett and Hatamian 2019] Garrett, D. and Hatamian, M. (2019). Cube puzzle solver. US Patent App. 16/099,833.

[Hassan and Abdulmuim 2019] Hassan, R. M. and Abdulmuim, M. E. (2019). Using Rubik's cube in fragile audio watermark encryption. *Diyala Journal For Pure Science*, 15(03):103–124.

[Huq et al. 2017] Huq, S., Shafiq, Z., Ghosh, S., Khakpour, A., and Bedi, H. (2017). Distributed load balancing in key-value networked caches. In *ICDCS*, pages 583–593. IEEE.

[Johnson and Maltz 1996] Johnson, D. B. and Maltz, D. A. (1996). Dynamic source routing in ad hoc wireless networks. In *Mobile computing*, pages 153–181. Springer.

[Korf 1997] Korf, R. E. (1997). Finding optimal solutions to Rubik's cube using pattern databases. In *AAAI/IAAI*, pages 700–705.

[Lai et al. 2019] Lai, X., Zhang, P., Zhang, X., Wu, H., Pu, Z., Yu, H., and Li, D. (2019). Reconfigurable microfluidics in a Rubik's cube. In *Micro Electro Mechanical Systems*, pages 110–113. IEEE.

[Mannone et al. 2019] Mannone, M., Kitamura, E., Huang, J., Sugawara, R., Chiu, P., and Kitamura, Y. (2019). Cubeharmonic: a new musical instrument based on Rubik's cube with embedded motion sensor. In *ACM SIGGRAPH 2019 Posters*, page 53. ACM.

[Nvidia 2019] Nvidia (2019). Datasheet nvidia t4 tensor core gpu.

[Rokicki et al. 2014] Rokicki, T., Kociemba, H., Davidson, M., and Dethridge, J. (2014). The diameter of the Rubik's cube group is twenty. *SIAM Review*, 56(4):645–670.

[Ruwix 2019] Ruwix (2019). Pretty Rubik's cube patterns with algorithms.

[Singmaster 1981] Singmaster, D. (1981). *Notes on Rubik's magic cube*. Enslow Publishers Hillside, NJ.

[Steinparz et al. 2019] Steinparz, C. A., Hinterreiter, A., Stitz, H., and Streit, M. (2019). Visualization of Rubik's cube solution algorithms. *EuroVis Workshop on Visual Analytics*.

[Zeng et al. 2018] Zeng, D.-X., Li, M., Wang, J.-J., and Hou, Y.-L. (2018). Overview of Rubik's cube and reflections on its application in mechanism. *Chinese Journal of Mechanical Engineering*, 31(1):77.

[Zhao et al. 2004] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., and Kubiatowicz, J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications*, 22(1):41–53.