Resolução de Sudoku baseado em Grafos e Coloração

Danilo S. B. Aciole, Gabriel I F Paiva

Instituto Metrópole Digital - Universidade Federal do Rio Grande do Norte (UFRN)

danilo25aciole@gmail.com, gabrielpaiva18@gmail.com

Abstract. This work makes a light introduction to the sudoku game, along with presenting the use of graphs in order to solve the game, with the use of colorization modeling.

Resumo. Este trabalho realiza uma leve introdução em relação ao jogo de sudoku, junto de apresentar a utilização de grafos com o objetivo de solucionar o jogo, a partir da utilização da modelagem de colorização.

1. Introdução

A proposta deste artigo é demonstrar como podemos representar um sudoku de tamanho NxN usando grafos e então a partir de uma coloração parcial completarmos as cores e assim acharmos uma possível solução para o quebra-cabeça.

2. O problema

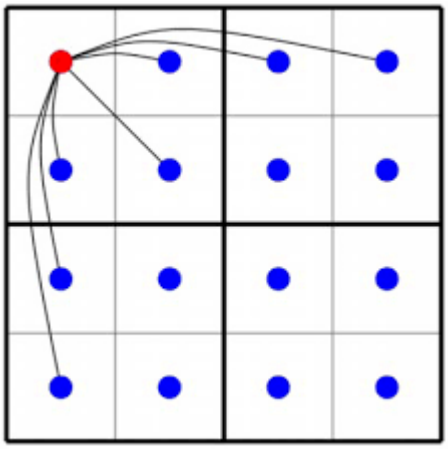
O sudoku é um jogo lógico, onde precisa-se preencher os espaços com dígitos enquanto respeitamos as regras do jogo.

Usando o tamanho mais comum de tabuleiro como exemplo, 9x9 com sub-grades de 3x3, o objetivo do jogo é preencher as células vazias com dígitos que vão de 1 a 9. O quebra-cabeça pode ou não já conter dicas iniciais, que são números já inseridos em alguns espaços, o que diminui a quantidade possível de soluções e facilita uma dedução ou indução dos números que devem estar nas células restantes.

Para uma solução ser considerada válida, cada linha, coluna e sub-grade deve conter apenas um dígito dos possíveis a serem usados no quebra-cabeça, no nosso exemplo 1 a 9.

3. Modelagem

Para a modelagem do problema devemos considerar cada célula do tabuleiro como um vértice diferente. O vértice de cada célula será ligado por arestas aos vértices de células da mesma linha, mesma coluna e/ou mesma sub-grade.



Cada tabuleiro de sudoku possui um número possível de dígitos usados na solução, números indo de 1-N, sendo N a dimensão do sudoku (NxN). Cada número desse será considerado uma cor diferente no nosso grafo, sendo assim um vértice (célula) não pode ter a mesma cor (número) que os vértices aos quais estão ligados (células na mesma linha, coluna e/ou sub-grade)

Por fim o problema em grafos se resume a em um sudoku nxn saber se o grafo é n-colorível e alcançar essa coloração.

4. Estado da arte

Aqui se encontram trabalhos importantes encontrados na literatura com relação a sudoku e a colorização para que possamos contextualizar a proposta deste projeto melhor.

Tabela 4.1 Links de referência ao estado da arte.

Tema	Autor/Ano	Tema de estudo
Sudoku	1) Zong Woo Geem, 2007 2) Rhyd Lewis , 2007 3) T Mantere, J Koljonen, 2006 4) Jaysonne A. , Glaiza Mae, John P. 2009 5) T. Sanders, 2009 6) Peter Norvig, 2016 7) Kyle O., 2016	1) Harmony Search Algorithm for Solving Sudoku 2) Metaheuristics can solve sudoku puzzles 3) Solving and Rating Sudoku Puzzles with Genetic Algorithms

	8) **Tirsa N. , Matheus S. , 2021**	4) **Solving Sudoku Puzzles using Improved Artificial Bee Colony Algorithm** 5) **Sudoku graphs are integral** 6) **Solving Every Sudoku Puzzle** 7) **Math and Sudoku: Exploring Sudoku Boards Through Graph Theory, Group Theory, and Combinatorics** 8) **Comparison Analysis of Breadth First Search and Depth Limited Search Algorithms in Sudoku Game**
Colorismo	1) **Brooks, R. L. ;** Tutte, W. T., 1941 2) Daniel Brélaz, 1979 3) **Richard B., David E. , 2005** 4) **Wichman T. , Öhman E. , 2014** 5) **R.M.R. Lewis. 2016** 6) **T Mostafaie, FM Khiyabani, NJ Navimipour, 2020** 7) **Maciej B. , Armon C. , Kacper J. , Vonarburg-Shmaria, Lukas G. , Torsten H.** 8) **R. M. R. Lewis, 2022** 9) **Alfredo Silveira Araújo Neto, Marcos José Negreiros Gomes. 2014** 10) **Alane Marie de Lima, 2017**	1) **On colouring the nodes of a network** 2) **New methods to color the vertices of a graph** 3) **3-coloring in time O(1.3289 ^{n})** 4) **Comparison of how common sudoku solving techniques perform when adapted and applied to jigsaw sudokus** 5) **A Guide to Graph Colouring** 6) **A systematic study on meta-heuristic approaches for solving the graph coloring problem** 7) **High-Performance Parallel Graph Coloring with Strong Guarantees on Work, Depth, and Quality** 8) **High-Performance Graph Colouring Algorithms** 9) **Problema e Algoritmos de Coloração Em Grafos - Exatos e Heurísticos** 10) **Algoritmos exatos para o problema da coloração de grafos**

Fonte: Autores

Uma boa fonte de casos de teste que iremos utilizar é o sudoku-download, que possui um banco de dados com sudokus, e suas soluções, que variam de 4x4 até 64x64. O sudoku-download também possui modelos com diferentes níveis de dificuldade e de estrutura, dando assim uma base de dados generosa para trabalharmos.

4.1. Sudoku

Os métodos mais simples e intuitivo de resolver um sudoku é através de backtracking, um método que é o refinamento do algoritmo de força bruta. Apesar de fácil implementação e garantir a resolução, desde que o puzzle seja válido, backtracking pode ter um tempo médio muito alto quando comparado com métodos dedutivos.

Outro algoritmo muito utlizado para resolver sudokus é o Rule-based system. Esse sistema usa um conjunto de regras que logicamente provam que uma célula deve conter um número específico ou excluir números impossível para ela. Esse método é muito semelhante a como humanos costumam resolver o sudoku e as regras são de fato originadas de métodos humanos de solução. O problema com esse tipo de algoritmo é que ele é heurístico, o que significa que nem todo puzzle pode ser resolvido por ele.

Temos também os métodos algoritmos stochastic, onde basicamente adivinhamos números para todas as células em branco, calculamos o número de erros e então

“misturamos” os números inseridos até o número de erros serem zero. Alguns algoritmos específicos desses temos o genetic algorithm, tabu search e simmulated annealing.

4.2. Coloração de grafos

A coloração de grafos teve sua origem em 1852, quando Francis Cuthrie em uma carta ao seu irmão Frederick Cuthrie, estudante de matemática da universidade de Londres, propõe que qualquer mapa desenhado em papel poderia ter seus países pintados com no mínimo quatro cores sem que a mesma cor pintasse países que fazem fronteira, essa conjetura ficou conhecida como O Teorema das Quatros Cores que foi provado em 1879 por Alfred Kempe, e reafirmado em 1976 com a ajuda de um computador por Kenneth Appel e Wolfang Haken.

O Teorema das Quatros Cores determina que um grafo planar é 4-colorivel, e a partir dele surge o problema de qual o menor valor de **n** será capaz de colorir um grafo qualquer, dessa forma surge o problema de coloração de grafos dado um grafo não-direcionado G = (V, E), com um conjunto V de vértices e um conjunto E de arestas que incidem sobre os vértices de G, o problema de coloração de grafos consiste em atribuir n-cores para os vértices de G, de modo que dois vértices adjacentes não tenham a mesma cor, ou seja, c:G(V) pertencentes ao inteiros positivos, então, c(x) difere de c(y) se x e y são adjacentes em G.

Para a resolução dessa coloração um algoritmo muito utilizado é o de DSATUR, proposto por Brélaz (1979), onde cada vértice v do grafo está associado a um vetor de cores, em que cada posição é associada a um vértice vizinho, o algoritmo iniciasse colorindo o vértice de maior grau com 1, em seleciona o vértice com maior grau de saturação, vértice com vizinhos já coloridos, e que ainda não foi colorido e atribui a ele a menor cor disponível.

Sendo o melhor algoritmo encontrado para resolução do problema do sudoku, no qual a cor, números, entre os vértices vizinhos, células, não podem ser repetidos, sendo uma pequena adaptação necessária para deixar um algoritmo ótimo, devido a sua forma padrão ser gulosa e integrar um resultado nem sempre otimizado.

4.3. Links importantes

http://sudoku-download.net , site contendo os casos de teste e um gerador para mais caso desejado.

https://ieeexplore.ieee.org/document/5412260/authors#authors , artigo Solving Sudoku Puzzles Using Improved Artificial Bee Colony Algorithm

http://bcsee.org/index.php/bcsee/article/view/1146 , artigo Comparison Analysis of Breadth First Search and Depth Limited Search Algorithms in Sudoku Game

https://dl.acm.org/doi/10.5555/3433701.3433833 , High-performance parallel graph coloring with strong guarantees on work, depth, and quality

5. Algoritmo

O melhor algoritmo encontrado para resolução do problema do sudoku foi o DSATUR, no qual a cor, números, entre os vértices vizinhos, células, não podem ser repetidos, sendo uma pequena adaptação necessária para deixar um algoritmo ótimo, devido a sua forma padrão ser gulosa e integrar um resultado nem sempre otimizado.

6. Implementação

A nossa implementação recebe uma matrix 2d de inteiros representando o tabuleiro do Sudoku, cada coordenada contém o número naquela respectiva célula e zeros representam células vazias.

É importante ressaltar que além da matrix de entrada precisar seguir as regras do Sudoku, ele também tem algumas limitações a quais tamanhos o programa ceita, nós trabalhamos apenas com sudokus de dimensões que possuem raiz quadrada exata (4, 9, 16). Isso se dá pelo motivo destes sudokus possuírem sub-regiões de tamanhos e formatos fixos, dimensões que destoam desta regra podem possuir sub-regiões de diversos tamanhos e formatos.

A criação de vértices é feita a partir da matrix de entrada e para isso precisa-se percorrer cada célula do sudoku individualmente, salvar a cor da célula nas respectivas coordenadas do grafo e caso seja uma celular em branca ele salva também na lista de vértices vazios.

Como cada célula representa um vértice e precisamos percorrer todas elas, temos que o seu tempo pode ser definido como O(n), sendo n o número de células/vértices.

```java
public void makeGraph(int array[][]){
        try {
            g = new SudokuGraph(array.length);
        } catch (Exception e) {
            e.printStackTrace();
        }

        this.expectedtotalColors = array.length;
        this.usedtotalColors = array.length;
        for (int i = 0; i < array.length; i++){
            for (int e = 0; e < array[i].length; e++){

                if (!(usedColors.contains(array[i][e])) && array[i][e]!=0){
                    this.usedColors.add(array[i][e]);
                }

                Vertex v = new Vertex(i, e);
                v.setColor(array[i][e]);
                g.setVertice(i, e, v);
                if (array[i][e] == 0){
                    emptyGraph.add(v);
                }

            }
        }
    }
```

Como nosso grafo representa um sudoku é fácil prever as arestas que teremos através das dimensões. Uma célula apenas se conecta a outras da mesma linha, da mesma coluna ou da sua sub-região, como usamos apenas Sudokus com sub-regiões fixas fica fácil formarmos todas essas ligações apenas pela coordenada de cada vértice.

Sabendo com quais outros vértices cada um vai se ligar o programa percorre cada um deles e insere os adjacentes na lista de adjacência que cada vértice possui formando assim as arestas, isso ainda cria uma exigência de percorrermos cada vértice apenas uma vez o que faz o seu tempo ser O(n).

```java
public void createEdges(){
    int d = g.getDim();

    for (int i = 0; i < d; i++){
        for (int e = 0; e < d; e++){
            this.createEdgesColumn(i, e);
            this.createEdgesRegion(i, e);
            this.createEdgesRow(i, e);
        }
    }
}
```

Na parte da coloração em si foi feita através do Dsatur, onde ele percorre a lista de vértices vazios, definidos durante a criação do grafo, acha o que possui maior saturação e grau e o pinta com a menor cor possível para ele.

O nosso programa trabalha com uma situação especial de coloração, onde alguns vértices podem vir já pintados, isso faz com que o tempo de execução do algoritmo DSatur sofra uma certa alteração. Ao invés de percorrer todos os vértices e comparar cada um com todos os outros, ele apenas precisa percorrer a lista de vértices em branco e comparar com os outros vértices desta mesma lista.

No final ele seria O($(n-i)^2$), onde n é o número total de vértices e i é o número de vértice já pintados. No pior caso, onde i=0, temos o tempo de execução tradicional do Dsatur O(n^2).

```java
public void DSatur(){

    while (!emptyGraph.isEmpty()){
        //Find next vertex to be paint besed first on saturation and if it ties
        //Uses the degree on the sub graph of not colored vertexs.
        Vertex v = Collections.max(this.emptyGraph, new CompareVertex());

        List<Integer> colorsUnavailable = new ArrayList<>();
        for (Vertex i : v.getAdjList()) {
            if (i.getColor() != 0 && !colorsUnavailable.contains(i.getColor())){
                colorsUnavailable.add(i.getColor());
            }
        }

        //Find the smaller color that this vertex can use
        int c;
        for ( c=1; c <= usedtotalColors+1; c++){
            if (!colorsUnavailable.contains(c)){
                break;
            }
        }
        //If it cant find an option inside the already used colors it will take the next one
        //And update the variable so we can see it is not an optimal solution
        if (c <= usedtotalColors){
            v.setColor(c);
        }else{
            v.setColor(c);
            this.usedtotalColors = c;
        }

        //Now we remove our Vertex from the empty list
        emptyGraph.remove(v);

    }

}
```

7. Prova de Corretude

Corretude do algoritmo DSatur implementado por invariante:

Seja G' o grafo de vértices não coloridos.

No começo da primeira interação, se o G' não é vazio, então a vértices a serem coloridos.

Então nós identificamos o vértice v com maior prioridade.

Checamos a lista de adjacência de v para sabermos qual as cores que ele ainda pode ser colorido e utilizamos a menor delas nele, em seguida removemos v de G'.

Se no começo de uma iteração x o G' não é vazio então ainda possuímos vértices a serem coloridos.

Como ao final do loop nós removemos um vértice de G', se ao final desta iteração o programa não se encerrar, G' ainda não é vazio, então ainda possuímos vértices a serem coloridos e passamos para iteração x+1.

E assim o loop se mantém ativo.

Se o loop se encerra, então G' é vazio, se G' é vazio temos que todos os vértices foram coloridos e o algoritmo se encerra.

8. Experimentos

Para realização dos casos de testes foram utilizados sudokus da plataforma sudoku download formatados para arquivos de textos, dividindo em três tipos centrais baseados em seus tamanhos 4x4, 9x9 e 16x16, em seguidas foram subdivididos entre dois a três tipos de dificuldades (fácil, médio e difícil) baseados na quantidade de vértices já coloridos.

A partir dos casos de testes foi aplicado o algoritmo DSatur para solucionar os tabuleiros, como a nossa versão do algoritmo regista a maior cor que ele utilizou usamos isso para comparar com a maior cor esperada, que podemos definir através da dimensão do tabuleiro, assim determinamos se o sudoku foi resolvido corretamente ou não.

O algoritmo respeito as regras da colorização em todos os testes, para definir sua corretude foi considerado o número total de cores utilizadas, se o algoritmo alcançou a solução ideal ou não.

Dos resultados foram obtidos:

Tabela 6.1: Sudokus 4x4

Nível	Tempo Total	Tempo Por Sudoku	Sudokus Corretos	Cores Usadas	Sudokus Incorretos	Cores Usadas
fácil	0.00243	0.000162	15	4	0	0
difícil	0.00284	0.000189	15	4	0	0

Fonte: Autores

Tabela 6.2: Sudokus 9x9

Nível	Tempo Total	Tempo Por Sudoku	Sudokus Corretos	Cores Usadas	Sudokus Incorretos	Cores Usadas
fácil	0.01182	0.001182	10	9	0	0
médio	0.017	0.0017	5	9	5	10
difícil	0.01846	0.001846	0	0	10	10

Fonte: Autores

Tabela 6.3: Sudokus 16x16

Nível	Tempo Total	Tempo Por Sudoku	Sudokus Corretos	Cores Usadas	Sudokus Incorretos	Cores Usadas
fácil	0.08646	0.01729	5	16	0	0
médio	0.1	0.02	5	16	0	0

Fonte: Autores

A partir dos resultados podemos observar que o algoritmo DSatur parece ter uma vantagem no cenário dos Sudokus. Devido a característica do jogo de já possuir algumas células preenchidas isso diminui as possibilidades para as restantes e assim o algoritmo tem uma maior facilidade em colorir o grafo.

Nesse cenário o algoritmo não possui vantagem apenas em grafos menores, mas em grafos que possuem mais células já preenchidas, como podemos observar ele erra mais em grafos 9x9 difíceis (com menor número de dicas) do que em grafos 16x16 fáceis.

9. Conclusão

Usando esse modelo de grafo construído baseado no sudoku, se preenchermos as cores dos vértices de forma a respeitar as regras da coloração e o limite de cores, que é definido pelo número dos possíveis dígitos do sudoku, conseguimos encontrar uma possível resposta para o quebra-cabeça.

Dessa forma sabemos que algoritmos e técnicas que realizem essa coloração em grafos irão funcionar também na resolução do sudoku.

Repositório do projeto no GitHub https://github.com/Arphor/Sudoku_Solver.

10. Referências

Klotz, Walter; Sander, Torsten. Integral Cayley Graphs over Abelian Groups. Electronic Journal of Combinatorics, 17 (1): Research Paper 81, 25 de maio de 2010. Disponível em: https://www.combinatorics.org/ojs/index.php/eljc/article/view/v17i1r81. Acesso em: 12 de setembro de 2022.

Sander, Torsten. Sudoku Graphs are Integral. Electronic Journal of Combinatorics, 16 (1): Note 25, 7pp, 24 de julho de 2009. Disponível em: https://www.combinatorics.org/ojs/index.php/eljc/article/view/v16i1n25. Acesso em: 10 de setembro de 2022.

A Guide to Graph Colouring: Algorithms and Applications. Springer, 2016.

Araújo Neto, Alfredo & Negreiros, Marcos. (2014). Problema e Algoritmos de Coloração Em Grafos - Exatos e Heurísticos. RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao. Dezembro de 2014 Disponivel em: https://www.researchgate.net/publication/332950927_Problema_e_Algoritmos_de_Coloracao_Em_Grafos_-_Exatos_e_Heuristicos. Acesso em: 15 de Outubro de 2022

Lima, Alane Marie de. **Algoritmos exatos para o problema de coloração de grafos**. Dissertação - Faculdade de Ciência da computação, informática, Universidade Federal do Paraná, 2017. Disponível em: https://acervodigital.ufpr.br/handle/1884/49429.

Sudokus 4x4 fácil. Dísponível em: http://sudoku-download.net/files/60_Sudokus_4x4_Easy.pdf

Sudokus 4x4 difícil. Dísponível em: http://sudoku-download.net/files/60_Sudokus_4x4_Difficult.pdf

Sudokus 9x9 fácil. Dísponível em: http://sudoku-download.net/files/60_Sudokus_Pattern_Easy.pdf

Sudokus 9x9 médio. Dísponível em: http://sudoku-download.net/files/60_Sudokus_New_Medium.pdf

Sudokus 9x9 difícil. Dísponível em: http://sudoku-download.net/files/60_Sudokus_New_Difficult.pdf

Sudokus 16x16 fácil. Dísponível em: http://sudoku-download.net/files/24_Sudokus_16x16_Easy.pdf

Sudokus 16x16 médio. Dísponível em: http://sudoku-download.net/files/24_Sudokus_16x16_Medium.pdf