Coloração de Grafos
Aplicado na resolução do Sudoku

Samuel Borges[1] ,Thales Lima[1], Vítor Marques[1]

[1]Instituto Metrópole Digital
Universidade Federal do Rio Grande do Norte (UFRN)
Av. Senador Salgado Filho, 3000 – 59078-970 – Natal – RN – Brasil

thalesaguiar21@gmail.com, franca_borges@hotmail.com,
vitorgodeirom@gmail.com

Abstract. *This paper makes a brief historical and conceptual introduction about Graph Theory and Graph Coloring, we introduce the problem of the Sudoku puzzle and show how to solve it through Graph Coloring. For this, we have molded the problem in a Graph, using an adjacency matrix, and from a small art state review of the algorithms responsible for solving this problem, we also introduce a solution using two exact algorithms, one using Backtracking and other using the Degree Saturation for choosing the visitation order of the vertex. In the end, we show the results of the algorithms after applying them to resolutions of instances of the Sudoku and Graphs from the library of benchmarks for Graph Coloring.*

Resumo. *Este artigo faz uma breve introdução histórica e conceitual sobre a teoria de Grafos e coloração de Grafos, introduzimos o problema do jogo Sudoku e mostramos como resolvê-lo através da coloração de grafos. Para isso, modelamos o problema em Grafo, usando matriz de adjacências, e a partir de uma pequena revisão do estado da arte dos algoritmos responsáveis por resolver esse problema, introduzimos uma solução usando dois algoritmos exatos, um usando backtracking e outro usando o Degree Saturation para escolha da ordem de visitação dos vértices. Ao final, exibimos os resultados dos algoritmos após aplicá-los a resolução de instâncias de Sudoku e Grafos de uma biblioteca de benchmarks para Coloração de Grafos.*

1. Introdução

A teoria de grafos teve início com o problema das pontes de *Königsberg*, resolvido por Euler [Euler 1736], já a coloração de grafos surge apenas em 1853 quando Francis Guthrie tentou colorir o mapa da Inglaterra de tal maneira que regiões com fronteiras em comum não recebessem a mesma cor, Guthrie percebeu que seriam necessárias quatro cores para isso, dando origem a conjectura das *quatro cores*, ele estende essa questão para toda a comunidade científica com o objetivo de saber se isso era possível a qualquer mapa ou grafo planar. O problema de confirmar a conjectura das *quatro cores* passou por Appel & Haken [Appel e Haken 1997] e Arthur Cayley [Cayley 1879], mas a sua demonstração só foi aceita por toda a comunidade científica quando Gonthier [Gonthier 2001] a realizou. A [Figura 1] mostra um exemplo de coloração de um mapa.

Neste artigo abordaremos a coloração de grafos aplicado ao problema do Sudoku. Este trabalho está dividido em blocos, inicialmente introduziremos alguns conceitos preliminares referentes ao Sudoku, os quais serão importantes para um melhor

acompanhamento deste texto. Posteriormente modelamos o jogo Sudoku em um problema de coloração de grafos, em seguida apresentaremos os algoritmos implementados para a resolução do referido problema, após introduzir os algoritmos usados apresentaremos os testes que realizamos e traremos os resultados obtidos nesses testes, por fim concluiremos o trabalho.

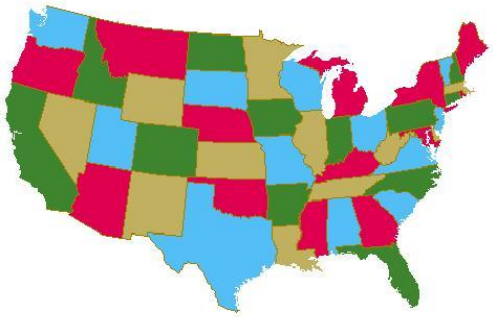

Figura 1: O mapa dos Estados Unidos da América colorido com quatro cores.

2. Sudoku

O Sudoku é um jogo do gênero *puzzle* (quebra-cabeça), projetado por Howard Garns [Wikipédia], um arquiteto e construtor independente de *puzzles*. O jogo aparece pela primeira vez na edição de maio de 1979 da revista Dell Pencil Puzzles and Word Games [Wikipédia]. Sendo o Sudoku um caso particular do quadrado latino, uma construção matemática criada pelo suíço Leonhard Euler no século XVIII [Wikipédia]. O jogo consiste em uma grade de tamanho $n \times n$e $n$blocos de tamanho $n^{\frac{1}{2}} \times n^{\frac{1}{2}}$, onde $n = i^2 ei \in \mathbb{N}^+$. A grade vem com alguns quadrados já preenchidos, e o objetivo é preencher os demais com números de 1a $n$sem repetir números nas linhas, colunas ou blocos. A [Figura 2] ilustra um jogo não resolvido e uma solução.

5						2	3	
				4	8	9	7	
	8				3			
1			3	8	9	7	4	
				5	6			
8	9							
	4						6	7
6	2		8	3				1
	5	1		6			2	

➡

5	1	4	6	9	7	2	3	8
2	3	6	1	4	8	9	7	5
7	8	9	5	2	3	6	1	4
1	6	5	3	8	9	7	4	2
4	7	3	2	5	6	1	8	9
8	9	2	4	7	1	3	5	6
3	4	8	9	1	2	5	6	7
6	2	7	8	3	5	4	9	1
9	5	1	7	6	4	8	2	3

Figura 2. Exemplo de Sudoku 9 x 9 resolvido.

2.1. Modelando o problema de resolução com grafos

Para modelar o Sudoku em um grafo podemos pensar em cada quadrado da grade como um vértice, onde dois vértices ue $v$são adjacentes se $(i)uev$ pertencem a mesma linha, $(ii)uev$ pertencem a mesma coluna ou $(iii)uev$ pertencem ao mesmo bloco. Portanto,

como os vértices são adjacentes, existe uma aresta ligando cada vértice aos demais contidos no bloco, na coluna e na linha. Veja que aplicamos a coloração de vértices, ou seja, dois vértices adjacentes não possuem a mesma cor, porém agora as cores serão representadas por números, assim chegamos no problema da $k - coloração$própria. A [Figura 3] ilustra um exemplo de uma modelagem em grafo para um Sudoku 4×4.

Note que podemos modelar através de grafos simples ou hipergrafos, neste trabalho optamos pela modelagem através de grafos simples. Por exemplo, considere um Sudoku 9×9, cada vértice está ligado a outros 20vértices, pois cada vértice precisa ter cor diferente dos 20vértices ligados a ele (vértices adjacentes). Nessa modelagem teremos no total 810arestas, como podemos ver na [Figura 4], temos a representação em grafos simples de um vértice ligado a todos os seus adjacentes em um Sudoku 9×9.

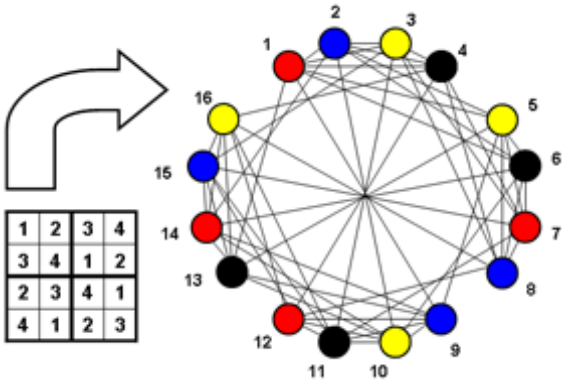

Figura 3. Modelagem de um Sudoku 4×4em um grafo.

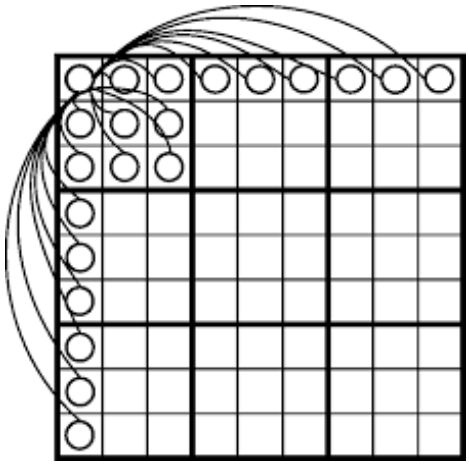

Figura 4. Modelagem de um Sudoku 9×9em um grafo simples.

2.2. Estado da arte

Apresentado e modelado o problema, introduziremos algoritmos presentes na literatura para o referido problema, sendo estes:

- Coloração em Largura [dos Santos Cunha et al. 2013];
- Coloração em Profundidade [dos Santos Cunha et al. 2013];
- Algoritmo heurístico The Hybrid Evolutionary Algorithm (HEA) [Galinier e Hao 1998];

- Algoritmos de coloração de vértices usando enumeração implícita segundo modificação do algoritmo de Sewell (1996) [Segundo 2011];
- Algoritmo que implementa o DSATUR em $O(m\log n)$[Turner 1988] usando o algoritmo de Welsh e Powell [Welsh e Powell 1967].

2.3. Algoritmo proposto

Para este trabalho implementamos um algoritmo usando a técnica de *backtracking* e o algoritmo de *DSatur* com *backtracking* [Korman 1979], que pode ser acessado no livro do Lewis [Lewis 2016].

Os códigos a seguir exibem os algoritmos desenvolvidos para solucionar o problema de Coloração de grafos, eles foram desenvolvidos baseados em estudos dos algoritmos encontrados na literatura e descritos anteriormente nesta seção. Note que os códigos nessa seção podem ser obtidos a partir do github [de França Borges et al. 2016].

O primeiro código, exibido no arquivo **GerarMatriz.cpp** [de França Borges et al. 2016], mostra como foi gerada a estrutura de dados utilizada para modelagem do Sudoku em grafos, como apresentada em uma seção anterior. A representação de matriz de adjacências foi escolhida, pois essa modelagem apresenta maior facilidade de manipulação dos dados.

O segundo, **Colorir.cpp** [de França Borges et al. 2016], é um algoritmo de coloração de grafos usando *backtracking*. Este algoritmo busca essencialmente por uma $k-coloração$, onde$k$ neste caso será a ordem do Sudoku ao qual o algoritmo está sendo aplicado. A entrada é composta de um conjunto de cores e um contador. A cada iteração o algoritmo verifica se é possível colorir aquele vértice com uma cor existente, caso contrário ele cria uma nova cor, e assim sucessivamente até que o número de cores seja igual a ordem do Sudoku o qual o algoritmo foi aplicado.

Por último, temos o **ColorirInteligente.cpp** [de França Borges et al. 2016]. Nesse algoritmo usamos *backtracking* com o auxílio de uma heurística parecida com a *Degree Saturarion* (DSatur), anteriormente citada, e que usa a saturação de vértices para escolha da ordem de $k-coloração$. Entretanto, diferentemente do DSatur, em caso de empate no grau de saturação o **ColorirInteligente.cpp** não desempata com base no grau do vértice, ele apenas escolhe o primeiro vértice visitado.

```
1.  void Grafo::geraMatrizAdjacencia(){
2.      for (unsigned int i = 0; i <_numeroVertices; i++){
3.          for(unsigned int j = 0; j < _numeroVertices; j++){
4.              if(j >= (i/_ordem)*_ordem && j < (i/_ordem)*_ordem + _ordem)
5.                  _matrizAdjacencia[i][j] = true;
6.              else
7.                  if(j%_ordem == i%_ordem)
8.                      _matrizAdjacencia[i][j] = true;
```

```
9.                  else
10.                     _matrizAdjacencia[i][j] = false;
11.             }
12.         }
13.         for (unsigned int i = 0; i < _numeroVertices; i++){
14.             for(unsigned int f = 0; f < _ordem0;f++){
15.                 for(unsigned int k = 0; k < _ordem0; k++){
16.                     if((i/_ordem)%_ordem0 == 0 )
17.                         for(unsigned int j =0; j < _ordem0; j ++){
18.                             _matrizAdjacencia[i+_ordem*f][(i/_ordem0)*_ordem0+j + _ordem*k] =
true;
19.                         }
20.                 }
21.            }
22.         }
23.     }
```

GerarMatriz.cpp

```
1.      bool Grafo::grafoColorindoAuxiliar(unsigned int color[], unsigned int contador){
2.          if(contador == _numeroVertices)
3.              return true;
4.          if(color[contador]) {
5.              if(grafoColorindoAuxiliar(color, contador +1))
6.                  return true;
7.              else
8.                  return false;
9.          }
10.         for(unsigned int i = 1; i <= _ordem; i++) {
11.             if(verificaCor(contador,color,i)){
12.                 if(color[contador] == 0)
13.                     color[contador] = i;
14.                 if(grafoColorindoAuxiliar(color, contador +1))
15.                     return true;
16.                 color[contador] = 0;
17.             }
18.         }
19.         return false;
20.     }
```

Colorir.cpp

```
1.  bool Grafo::grafoColorindoAuxiliar(unsigned int color[], unsigned int contador, int saturacao[]){
2.      if(!saturacaoCheia(saturacao))
3.          return true;
4.      int cor = getSaturacaoMaior(saturacao, color);
5.      if(color[cor]){
6.          saturacao[cor] = -1;
7.          if(grafoColorindoAuxiliar(color, contador +1, saturacao))
8.              return true;
9.          else
10.             return false;
11.     }
12.
13.     for(unsigned int i = 1; i <= _ordem; i++){
14.         if(verificaCor(cor,color,i)){
15.             if(color[cor] == 0){
16.                 color[cor] = i;
17.                 saturacao[cor]    = -1;
18.
19.             }
20.             if(grafoColorindoAuxiliar(color, contador +1, saturacao))
21.                 return true;
22.             color[cor] = 0;
23.         }
24.     }
25.     saturacao[cor] = 0;
26.     return false;
27. }
```

ColorirInteligente.cpp

3. Testes e resultados computacionais

Realizamos vários experimentos com os algoritmos implementados, todos os exemplares foram executados no mesmo computador possuindo como especificação técnica 8 GB de memória RAM e processador i5 4420 x64. Inicialmente vamos mostrar os resultados referentes aos algoritmos aplicados em exemplos de Sudoku,

posteriormente em problemas de coloração de grafos em geral. Vale notar que Algoritmo 1 corresponde ao **Colorindo.cpp** e o Algoritmo2 corresponde ao **ColorindoInteligente.cpp**.

O experimento com exemplos do jogo foi realizado para diversos níveis do *puzzle*: 9×9, 16×16e 25×25. Todos os exemplares de Sudokus foram retirados do [sudoku download].

Para o Sudoku 9×9, realizamos experimentos para 4 (quatro) níveis de dificuldades, sendo eles fácil, médio, difícil e muito difícil. Para cada nível de dificuldade realizamos experimentos com 5 (cinco) Sudokus diferentes, sendo sempre as cinco primeiras configurações obtidas a partir do [sudoku download(fácil 9x9)] para o fácil, [sudoku download(médio 9x9)] para o médio , [sudoku download(difícil 9x9)] para o difícil e [sudoku download(muito difícil 9x9)] para o muito difícil.

Tabela 1. Sudoku 9×9

Dificuldade	Algoritmo 1 - Tempo médio(s)	Algoritmo 2 - Tempo médio(s)
Muito difícil	0,0354726	0,01467900
Difícil	0,0187330	0,00789740
Médio	0,0008682	0,00407220
Fácil	0,0003140	0,00061856

Para o Sudoku 16×16, realizamos experimentos para dois níveis de dificuldades, sendo eles fácil e médio. Em cada nível de dificuldade realizamos experimentos com 5 (cinco) Sudokus diferentes, sendo sempre as cinco primeiras configurações obtidas à partir de [sudoku-download (fácil 16x16)] para o fácil e de [sudoku-download (médio 16x16)] para o médio.

Tabela 2. Sudoku 16×16

Dificuldade	Algoritmo 1 - Tempo médio(s)	Algoritmo 2 - Tempo médio(s)
Médio	0,2715000	0,1072125
Fácil	0,0618132	1,3481394

Para o nível de dificuldade 25×25, realizamos um experimento com 5 (cinco) configurações diferentes do jogo, sendo sempre as cinco primeiras configurações obtidas à partir de [sudoku-download (25x25)].

Tabela 3. Sudoku 25×25

Algoritmo 1 - Tempo médio(s)	Algoritmo 2 - Tempo médio(s)
9702	46,3352

Realizamos 24 testes com a biblioteca de *benchmark* [d'Angers a] [Harrisburg a] [Trick's a], cada arquivo de testes seguiu o padrão do DIMACS [library]. Todos os resultados obtidos estão representados na Tabela 4 e cada caso de teste pode ser obtido

nas referências.

Tabela 4. Biblioteca e benchmark

Teste N°	Vértices	Arestas	Algoritmo1: Tempo - N° Cores	Algoritmo2: Tempo - N° Cores
1	87	812	0,00004800 - 12 cores	0,001083 - 11 cores
2	95	755	0,00004200 - 7 cores	0,001544 - 7 cores
3	100	2940	0,00012000 - 16 cores	0,002385 - 18 cores
4	120	1276	0,00009780 - 9 cores	0,002918 - 9 cores
5	138	986	0,00008721 - 12 cores	0,003521 - 11 cores
6	191	2360	0,00013100 - 8 cores	0,011264 - 8 cores
7	300	21695	0,00258300 - 46 cores	0,126372 - 47 cores
8	450	9757	0,00157900 - 18 cores	0,157006 - 14 cores
9	450	16680	0,00264300 - 30 cores	0,194018 - 26 cores
10	450	16750	0,00256700 - 31 cores	0,193605 - 27 cores
11	450	17343	0,00321900 - 37 cores	0,191473 - 31 cores
12	905	43081	0,02045700 - 64 cores	1,236080 - 51 cores
13	947	43571	0,02097800 - 64 cores	1.361720 - 52 cores
14	1000	49629	0,01199100 - 31 cores	1,751020 - 30 cores
15	1000	245000	0,09307500 - 126 cores	4,596730 - 124 cores
16	1000	245830	0,09129700 - 125 cores	4,633350 - 124 cores
17	1000	246708	0,09105600 - 125 cores	4,662440 - 127 cores
18	1000	249826	0,09766900 - 127 cores	4,611470 - 125 cores
19	1000	449449	0,13832000 - 321 cores	2,952710 - 322 cores
20	1216	7844	0,00620100 - 9 cores	2,116580 - 8 cores
21	1406	9695	0,00577200 - 6 cores	3,255740 - 6 cores
22	1809	103368	0,06986000 - 71 cores	8,548540 - 49 cores
23	1916	12506	0,01480600 - 10 cores	8,121110 - 10 cores
24	3600	212400	0,34072400 - 64 cores	56,42770 - 64 cores

4. Conclusão

Neste trabalho, introduzimos de forma simples o conceito e história dos grafos e coloração de grafos, abordamos o problema do jogo Sudoku, fizemos uma sucinta revisão da literatura e exibimos algoritmos que resolvem o problema, tornando esse trabalho uma boa fonte de estudo para quem deseja iniciar o estudo de grafos voltado para o Sudoku, reproduzir experimentos já realizados e comparar resultados.

Além disso, realizamos vários testes de verificação para os algoritmos apresentados, resultando em análises desses códigos com relação aos tempos de resolução do problema. A partir dos dados coletados podemos concluir que, ao comparar o Algoritmo1 ao Algoritmo2 eles apresentaram resultados diferentes em alguns casos. Note que, o Algoritmo2 apresentou melhores resultados para resolução de Sudokus mais complexos, enquanto o Algoritmo1 acabou sendo mais rápido para problemas de coloração em Grafos Simples os quais não foram modelados a partir de Sudokus e para Sudokus mais simples.

Referências

Figura 1.http://conteudo.icmc.usp.br/pessoas/sandra/2/MapaEUA.jpg. Accessed: 2016-04-17.

Figura 2. http://jogadamais.blogspot.com.br/2013_11_01_archive.html. Accessed: 2016-04-17.

Figura 3. http://www.rhydlewis.eu/images/sudoku.gif. Accessed: 2016-04-17.

Figura 4. http://vigusmao.github.io/manuscripts/sudoku.pdf. Accessed: 2016-04-17.

Appel, K. e Haken, W. (1997). Every planar map is four-colorable.

Cayley, A. (1879). On the colourings of maps.

da Costa e Thiago de Melo, P. P. (2008). Dissertação: Teoria de grafos e suas aplicações.

d'Angers, D. I. U. Benchmark graph coloring. http://www.info.univ-angers.fr/pub/porumbel/graphs/. Accessed: 2016-05-17.

d'Angers, D. I. U. Teste de número 14º . http://www.info.univ-angers.fr/pub/porumbel/graphs/dsjc1000.1.col. Accessed: 2016-05-17.

d'Angers, D. I. U. Teste de número 15ª . http://www.info.univ-angers.fr/pub/porumbel/graphs/flat1000_50_0.col. Accessed: 2016-05-17.

d'Angers, D. I. U. Teste de número 16ª . http://www.info.univ-angers.fr/pub/porumbel/graphs/flat1000_60_0.col. Accessed: 2016-05-17.

d'Angers, D. I. U. Teste de número 17ª . http://www.info.univ-angers.fr/pub/porumbel/graphs/flat1000_76_0.col. Accessed: 2016-05-17.

d'Angers, D. I. U. Teste de número 18ª. http://www.info.univ-angers.fr/pub/porumbel/graphs/dsjc1000.5.col. Accessed: 2016-05-17.

d'Angers, D. I. U. Teste de número 19ª. http://www.info.univ-angers.fr/pub/porumbel/graphs/dsjc1000.9.col. Accessed: 2016-05-17.

d'Angers, D. I. U. Teste de número 7ª . http://www.info.univ-angers.fr/pub/porumbel/graphs/flat300_28_0.col. Accessed: 2016-05-17.

de França Borges, S. N., de Lima, T. A., e de Godeiro Marques, V. (2016). Implementação de algoritmos de coloração de grafos aplicados ao sudoku. https://github.com/thalesaguiar21/GraphColouring. Accessed: 2016-05-30.

dos Santos Cunha, N., da Silva, T. G., e Jr, P. D. M. (2013). Técnicas de coloração de grafos aplicadas à resolução de quebra-cabeças do tipo sudoku.

Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis.

Galinier, P. e Hao, J.-K. (1998). Hybrid evolutionary algorithms for graph coloring.

Goldbarg, M. e Goldbarg, E. (2011). Grafos Conceitos, algoritmos e aplicações. Elsevier.

Gonthier, G. (2011). Formal proof—the four-color theorem.

Harrisburg, P. S. Benchmark graph coloring. https://turing.cs.hbg.psu.edu/txn131/file_instances/coloring/graph_color/. Accessed: 2016-05-17.

Harrisburg, P. S. Teste de número 12ª. https://turing.cs.hbg.psu.edu/txn131 /file_instances/coloring/graph_color/wap05a.col. Accessed: 2016-05-17.

Harrisburg, P. S. Teste de número 13ª. https://turing.cs.hbg.psu.edu/txn131/ file_instances /coloring/graph_color/wap06a.col. Accessed: 2016-05-17.

Harrisburg, P. S. Teste de número 20ª . https://turing.cs.hbg.psu.edu/txn131 /file_instances/coloring/graph_color/ash608GPIA.col. Accessed: 2016-05-17.

Harrisburg, P. S. Teste de número 21º. https://turing.cs.hbg.psu.edu/txn131/ file_instances/ coloring/graph_color/3-Insertions_5.col. Accessed: 2016-05-17.

Harrisburg, P. S. Teste de número 22º. https://turing.cs.hbg.psu.edu/txn131/ file_instances/ coloring/graph_color/wap07a.col. Accessed: 2016-05-17.

Harrisburg, P. S. Teste de número 23º . https://turing.cs.hbg.psu.edu/txn131/ file_instances/ coloring/graph_color/ash958GPIA.col. Accessed: 2016-05-17.

Harrisburg, P. S. Teste de número 24º . https://turing.cs.hbg.psu.edu/txn131/ file_instances/ coloring/graph_color/qg.order60.col. Accessed: 2016-05-17.

Korman, S. (1979). Combinatorial optimization. chapter 8, pages 211–235. Wiley, New York.

Lewis, R. M. R. (2016). A Guide to Graph Colouring, algorithms and aplications. Springer.

library, B. https://sites.google.com/site/graphcoloring/. Accessed: 2016-05-30.

Segundo, P. S. (2011). A new dsatur-based algorithm for exact vértex coloring.

Sloane, N. J. A. (2005). Number of inequivalent (completed) n^2 x n^2 sudokus (or sudokus).

sudoku download. http://www.sudoku-download.net/. Accessed: 2016-05-30.

sudoku-download (25x25). Sudoku 25x25. http://www.sudoku-download.net/files/ 12_Sudokus_25x25_Easy.pdf. Accessed: 2016-05-30.

sudoku-download (difícil 9x9). Sudoku 9x9, dificuldade difícil. http://www.sudoku-download.net/ files/60_Sudokus_New_Difficult.pdf. Accessed: 2016-05-30.

sudoku-download (fácil 16x16). Sudoku 16x16, dificuldade fácil. http://www.sudoku-download.net/files/24_Sudokus_16x16_Easy.pdf. Accessed: 2016-05-30.

sudoku-download (fácil 9x9). Sudoku 9x9, dificuldade fácil. http://www.sudoku-download.net/files/60_Sudokus_New_Easy.pdf. Accessed: 2016-05-30.

sudoku-download (muito difícil 9x9). Sudoku 9x9, dificuldade muito difícil. http://www.sudoku-download.net/files/30_Sudokus_Very_Difficult.pdf. Accessed: 2016-05-30.

sudoku-download (médio 16x16). Sudoku 16x16, dificuldade média. http://www.sudoku-download.net/files/24_Sudokus_16x16_Medium.pdf. Accessed: 2016-05-30.

sudoku-download (médio 9x9). Sudoku 9x9, dificuldade média. http://www.sudoku-download.net/files/60_Sudokus_New_Medium.pdf. Accessed: 2016-05-30.

Szwarcfiter, J. L. (1986). Grafos e Algoritmos Computacionais. Campus, 2nd edition.

Trick's, M. Benchmark graph coloring. http://mat.gsia.cmu.edu/COLOR/instances/. Accessed: 2016-05-17.

Trick's, M. Teste de número 10ª. http://mat.gsia.cmu.edu/COLOR/instances/le450_15d.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 11ª . http://mat.gsia.cmu.edu/COLOR/instances/le450_25c.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 1ª . http://mat.gsia.cmu.edu/COLOR/instances/david.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 2ª . http://mat.gsia.cmu.edu/COLOR/instances/myciel6.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 3ª . http://mat.gsia.cmu.edu/COLOR/instances/queen10_10.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 4ª. http://mat.gsia.cmu.edu/COLOR/instances/games120.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 5ª . http://mat.gsia.cmu.edu/COLOR/instances/anna.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 6ª. http://mat.gsia.cmu.edu/COLOR/instances/myciel7.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 8ª. http://mat.gsia.cmu.edu/COLOR/instances/le450_5d.col. Accessed: 2016-05-30.

Trick's, M. Teste de número 9ª. http://mat.gsia.cmu.edu/COLOR/instances/le450_15c.col. Accessed: 2016-05-30.

Turner, J. S. (1988). Almost all k-colorable graphs are easy to color.

Welsh, J. A. e Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems.

Wikipedia. Sudoku. https://pt.wikipedia.org/wiki/Sudoku. Accessed: 2016-04-17.

YATO, T. e SETA, T. (2002). Complexity and completeness of finding another solution and its application to puzzles.