

Trabalho prático Sudoku

SCC0216 - Modelagem Computacional em Grafos

QUEIROZ, Breno Cunha
11218991

NÚÑEZ, Henrique Hiram Libutti
11275300

01 de Julho de 2020

1 Introdução

O sudoku é um problema extremamente difícil de ser resolvido de forma randômica, visto que possui um teto de $1.5 \cdot 10^{31}$ possibilidades de preenchimento[3]. Neste trabalho modelamos o problema de solução do sudoku utilizando diferentes estratégias de coloração de grafos. Testamos cada algoritmos com casos encontrados em sites e avaliamos a eficiência de cada um para resolver o problema. O programa foi criado com o framework para interfaces gráficas GTK, permitindo assim um uso mais amigável. A inicialização é feita a partir da leitura de um arquivo de texto, onde cada linha do arquivo representa uma linha do sudoku, os 9 valores estão separados por espaço.

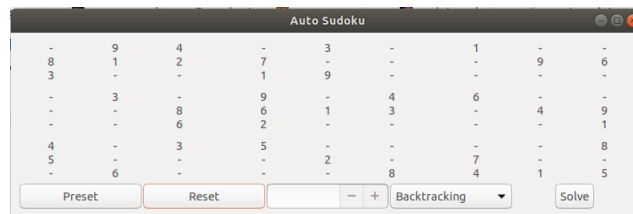


Figura 1: Janela principal do programa.

O código fonte está disponível também em <https://github.com/henriquenunez/sudoku-solver>

2 Modelagem com grafos

A partir do entendimento de como o sudoku funciona, é possível concluir que cada 'casa' interpreta-se como um vértice em nosso grafo, e a partir da estratégia de coloração, onde vértices adjacentes não podem ter a mesma cor, pode se criar as conexões com o seguinte critério: “uma ‘casa’ é adjacente a outra se e somente se encontra-se na mesma célula (cada bloco 3x3), linha ou coluna desta”.

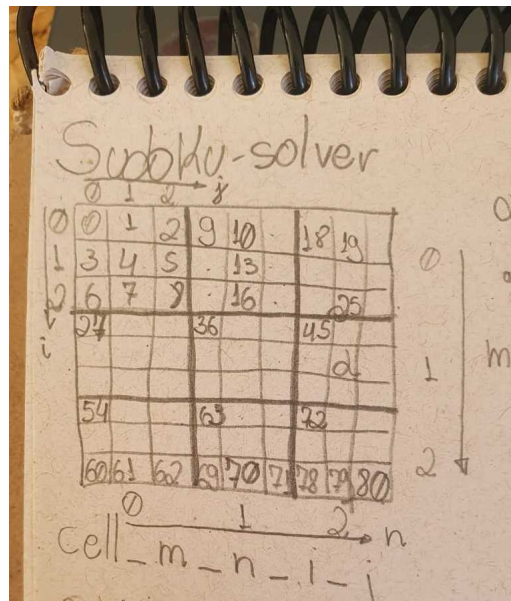


Figura 2: Estrutura de indexação dos vértices.

A partir desta imagem é possível entender como foi feita a indexação das ‘casas’ no programa. Cada célula(3x3) pode ser indexada por 2 variáveis m e n , variando de 0 a 2, e cada subcélula(casa) também pode ser indexada dentro de uma célula por duas variáveis i e j , variando de 0 a 2.

3 Algoritmos utilizados

Neste trabalho testamos 4 algoritmos para resolver o sudoku: Backtracking, Graph Coloring, Genetic Algorithm e Welsh Powell.

Backtracking é um algoritmo famoso de solução do sudoku pela sua eficiência e facilidade de implementação

4 Casos de teste

Nossos casos de teste foram coletados a partir de [2]. Temos um total de 5 exemplos para cada um das modalidades disponíveis: fácil, médio, difícil, expert. Ao testar a eficiência de cada algoritmo executamos os mesmos sobre cada um dos exemplos 3 vezes. O maior e o menor valores foram descartados.

1	0	9	4	0	3	0	1	0	0
1	8	1	2	7	0	0	0	9	6
2	3	0	0	1	9	0	0	0	0
3	0	3	0	9	0	4	6	0	0
4	0	0	8	6	1	3	0	4	9
5	0	0	6	2	0	0	0	0	1
6	4	0	3	5	0	0	0	0	8
7	5	0	0	0	2	0	7	0	0
8	0	6	0	0	0	8	4	1	5

Figura 3: Caso de teste fácil 1 (easy/1.txt).

5 Resultados

5.1 Welsh Powell

Entre os algoritmos testados, dois não conseguiram resolver nenhum dos exemplos. O primeiro foi o welsh powell, acreditamos que isso tenha acontecido pois neste algoritmo não existe uma troca dos valores que já foram inseridos, e, como existe o limite de resolver com 9 cores, este não conseguiu completar, deixando espaços em branco.

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

Figura 4: Tentativa de resolver Sudoku com backtracking com Welsh Powell.

5.2 Algoritmo genético

O segundo que não conseguiu completar o problema foi o algoritmo genético. Como o algoritmo genético faz uma espécie de busca randômica, é necessário um alto fine tuning para conseguir fazer com que este algoritmo resolva o problema. Nos nossos testes foram executadas 1000 gerações com 200 indivíduos, onde cada indivíduo tem 18 genes (um para cada célula no sudoku). Acreditamos que isto esteja acontecendo porque os indivíduos ficam presos em mínimos locais devida a baixa taxa de mutação. Também tentamos aumentar a taxa de mutação, mas como se aproxima de uma busca randômica, passou a demorar muito tempo.

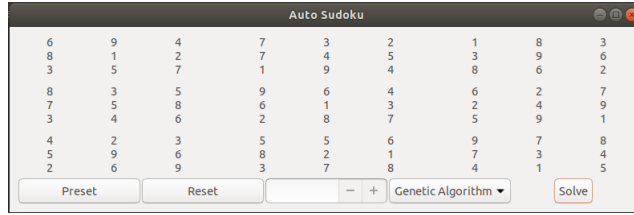


Figura 5: Tentativa de resolver Sudoku com algoritmo genético. Ainda foram detectadas 80 colisões, do ideal seria 0.

5.3 Backtracking

O backtracking conseguiu resolver com sucesso todos os exemplos do dataset. O tempo que o algoritmo demorou para resolver cada exemplo estão nas imagens abaixo.

Para cada experimento foram feitas 3 amostras, o maior e o menor foram descartados					
Backtracking	1.txt	2.txt	3.txt	4.txt	5.txt
easy	0,000089	0,000047	0,000058	0,00031	0,000123
medium	0,003402	0,004	0,003404	0,001941	0,000911
hard	0,003184	0,053876	0,008869	0,059466	0,017834
expert	0,006282	0,035596	0,02198	0,008364	0,04942

Figura 6: Tempo médio de solução de cada exemplo em segundos, 5 exemplos por nível.

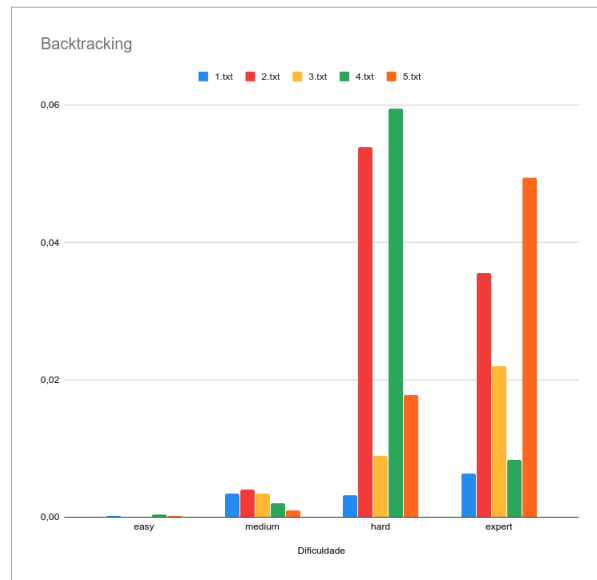


Figura 7: Da esquerda para a direita temos o tempo do dataset fácil, médio, difícil, expert.

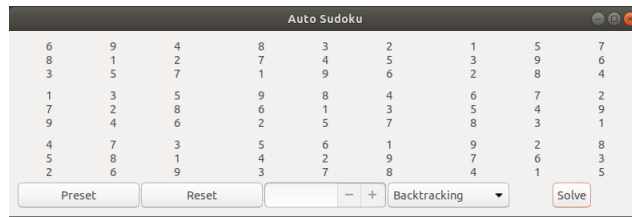


Figura 8: Solução do sudoku com backtracking.

5.4 Coloração de grafos com certas otimizações

A ideia desse algoritmo pensado pelos desenvolvedores é a dos seguintes passos:

1. Encontrar ‘casas’ que possuem apenas uma possibilidade de cor e colori-las.
2. Caso acabarem essas casas, ele faz uma busca de saturação, isto é, quantas casas preenchidas um nó tem ao redor. Com o nó menos saturado, ele atribui de sua lista de cores possíveis uma delas, e volta a encontrar casas que tem apenas uma cor possível.

Essa concepção foi adquirida através da leitura de algumas soluções feitas por outros estudantes e programadores, logo a opção por incluí-la no programa. Este algoritmo conseguiu resolver alguns dos casos de teste, sendo esses os marcados como fáceis. Os estão apresentados nas imagens abaixo.

Graph coloring	1.txt	2.txt	3.txt	4.txt	5.txt
easy	0,000295	0,000108	0,000239	0,000467	0,000185
medium		0,001026			
hard					
expert					

Figura 9: Solução do sudoku com coloração de grafos com otimizações.

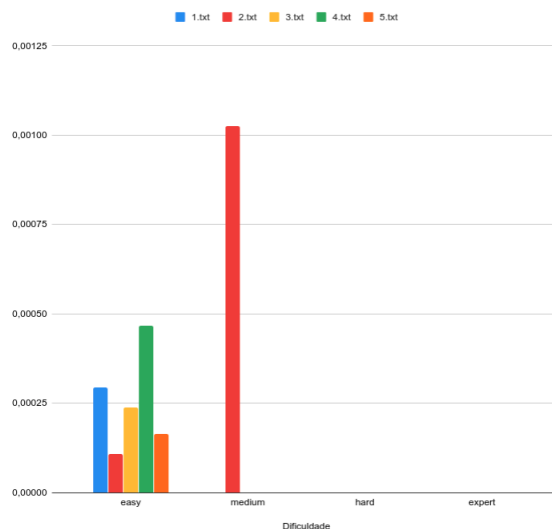


Figura 10: Gráfico dos tempos da coloração de grafos com otimizações.

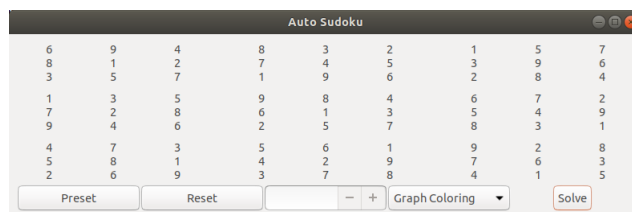


Figura 11: Solução do sudoku com coloração de grafos com otimizações.

6 Conclusão

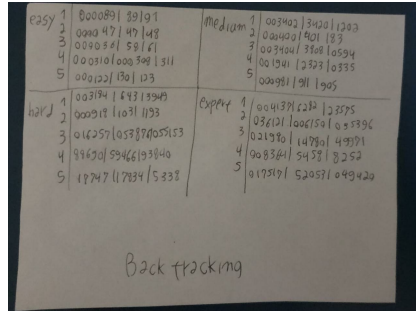
Dos quatro algoritmos testados para solucionar o sudoku, somente dois conseguiram de fato solucionar.

Acreditamos que o Welsh Powell não conseguiu completar a coloração com 9 cores pois em nenhum momento ele desfaz o que já foi colorido, o que o faz depender de iterar da forma certa logo na primeira vez para conseguir completar o grafo. O algoritmo genético não conseguiu completar os casos de teste devido à demora para testar muitas configurações diferentes e por ficar preso em mínimos locais. Tivemos dois algoritmos que conseguiram resolver pelo menos os casos fáceis. O backtracking conseguiu resolver todos, como esperado visto que é o mais comumente utilizado para resolver sudokus. Nosso algoritmo de coloração de grafos com otimizações conseguiu com sucesso resolver todos os casos fáceis e um médio, mas teve problemas para resolver os demais mais complicados.

7 Extras

7.1 Dados coletados

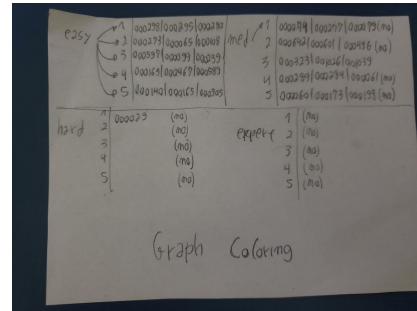
Nas imagens abaixo mostramos os dados coletados durante a execução dos testes.



Handwritten data for the Backtracking algorithm. The table is divided into three sections: 'C25y', 'Med', and 'C3p4y'. Each section contains a list of 5 items, each with a number and a string of digits.

C25y	Med	C3p4y
1 0000891 10101	1 003402 13420 1303	1 0041371 2322 13175
2 0000471 47148	2 000403 1401 183	2 036101 100150 1001596
3 0000841 84141	3 000404 1401 10594	3 021970 14791 499791
4 000101000 501 111	4 001501 13721 10555	4 008361 54581 3252
5 0001221 1201 123	5 000981 191 1905	5 0175171 520571 049420

(a) Dados coletados do backtracking.



Handwritten data for the Graph Coloring algorithm. The table is divided into three sections: 'C25y', 'Med', and 'C3p4y'. Each section contains a list of 5 items, each with a number and a string of digits.

C25y	Med	C3p4y
1 000271 100275 100270	1 000271 100275 100270	1 000271 100275 100270
2 000271 100275 100270	2 000271 100275 100270	2 000271 100275 100270
3 000271 100275 100270	3 000271 100275 100270	3 000271 100275 100270
4 000271 100275 100270	4 000271 100275 100270	4 000271 100275 100270
5 000271 100275 100270	5 000271 100275 100270	5 000271 100275 100270

(b) Dados coletados no algoritmo de coloração de grafos com otimizações.

Figura 12: Dados coletados nos 20 casos de teste.

7.2 Execução do programa

Primeiro é necessário instalar o GTK, no ubuntu é o seguinte comando:

```
sudo apt - get install libgtk - 3 - dev
```

Após isso:

```
git clone https : //github.com/henriquenunez/sudoku - solver
```

```
cd sudoku - solver/src
```

```
make make run
```

Após isso para carregar um exemplo clique em *Preset* e selecione um dos exemplos na pasta *sudoku - solver/presets*.

Referências

- [1] *Explicação do backtracking.*
<https://pt.wikipedia.org/wiki/Backtracking>
- [2] *Site de resolução do sudoku com diversos níveis.*
sudoku.com
- [3] *Modelagem computacional do sudoku.*
<http://pi.math.cornell.edu/mec/Summer2009/meerkamp/Site/CountingSudokus2.html>