SaturBFS: Um Algoritmo de Coloração de Grafos para Resolução do Sudoku

Luís Eduardo Costa Laurindo[1], **Francisco José da Silva e Silva**[1]

[1]Programa de Pós-Graduação em Ciência da Computação – PPGCC
Universidade Federal do Maranhão (UFMA)
São Luís – MA – Brasil

Abstract. *This study proposes a graph coloring algorithm called Saturated Breadth-First Search (SaturBFS) to determine valid Sudoku solutions. The algorithm was conceived by combining the Breadth-First Search (BFS) and DSatur algorithms. The objective is to perform a comparative analysis of SaturBFS with unsaturated BFS and Depth-First Search (DFS) algorithms for different levels (easy, medium and hard) and Sudoku instances (4x4, 6x6, 8x8, 9x9, 10x10 and 12x12). For the evaluation, the time that the algorithms take to determine a valid solution and the percentages of valid solutions found were considered. The SaturBFS algorithm showed a higher percentage of valid solutions found in a shorter time.*

Resumo. *Este estudo propõe um algoritmo de coloração de grafos intitulado Saturated Breadth-First Search (SaturBFS) para determinar soluções válidas do Sudoku. Concebeu-se o algoritmo por meio da combinação dos algoritmos de Busca em Largura (BFS) e DSatur. O objetivo é realizar uma análise comparativa do SaturBFS com os algoritmos BFS não saturado e Busca em Profundidade (DFS) para diferentes níveis (fácil, médio e difícil) e instâncias do Sudoku (4x4, 6x6, 8x8, 9x9, 10x10 e 12x12). Para a avaliação, considerou-se o tempo que os algoritmos levam para determinar uma solução válida e as porcentagens de soluções válidas encontradas. O algoritmo SaturBFS apresentou maior porcentagem de soluções válidas encontradas em um menor tempo.*

1. Introdução

Os problemas da classe NP são caracterizados por não se conhecer soluções ótimas em tempo polinomial [Sipser 2012], mas por meio de um algoritmo de tempo polinomial é possível verificar se uma determinada solução é válida para o problema. Por meio de uma redução polinomial do Quadrado Latino para o Sudoku, [Yato and Seta 2003] provaram que ele está contido na classe de problemas NP-difíceis. Os problemas NP-difíceis representam uma classe de problemas que são tão difíceis quanto os problemas mais difíceis em NP.

O Sudoku é um jogo de raciocínio lógico que tem como objetivo distribuir algarismos de 1 a n em uma matriz *n x n* obedecendo a seguinte regra: Não é permitido repetir números na mesma linha e coluna ou no mesmo bloco [Bailey et al. 2008]. Vários estudos foram concebidos com a finalidade de propor algoritmos eficientes por meio de várias técnicas. Os estudos [Chel et al. 2016] e [Thirer 2012] abordaram o problema por meio de algoritmos genéticos. Já [De Souza and Romero 2013] abordaram o problema por meio das metaheurísticas *Simulated Annealing* e Busca Tabu. O problema também

pode ser modelado como um grafo, logo pode-se utilizar técnicas de coloração para resolução [Gouveia and Maciel Jr 2013] e [Borges et al. 2016].

Portanto, este artigo tem como objetivo conceber um algoritmo de coloração de grafos intitulado *Saturated Breadth-First Search* (SaturBFS) que tem como finalidade encontrar soluções válidas do Sudoku. O algoritmo proposto foi concebido com base nas abordagens dos algoritmos de Busca em Largura (BFS) e DSatur.

O artigo está organizado da seguinte forma: a Seção 2 apresenta a metodologia utilizada para o desenvolvimento deste artigo; a Seção 3 expõe os conceitos fundamentais sobre o Sudoku e Teoria dos Grafos; a Seção 4 apresenta os trabalhos relacionados; a Seção 5 apresenta a descrição do algoritmo proposto; a Seção 6 apresenta os procedimentos da avaliação avaliação e resultados. Por fim, a Seção 7 apresenta a conclusão e os trabalhos futuros.

2. Metodologia

A Figura 1 apresenta o passo a passo realizado para a concepção do estudo. Primeiramente realizou-se uma pesquisa bibliográfica exploratória em algumas bases de dados, tais como: Google Scholar e IEEEXplorer, na qual proporcionou uma visão geral sobre o Sudoku com objetivo de conhecer os conceitos fundamentais do jogo e quais técnicas, estruturas e algoritmos eram utilizados para modelá-lo e solucioná-lo.

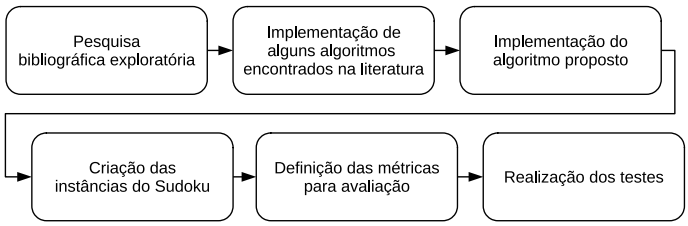


Figura 1. Metodologia da pesquisa.

Após a revisão bibliográfica, foi possível identificar como o Sudoku era modelado e as principais técnicas utilizadas para resolvê-lo. Com isso, alguns algoritmos encontrados na literatura foram implementados para realizar comparações com o algoritmo proposto. Com relação aos experimentos realizados, foram criados várias instâncias do Sudoku de diversos tamanhos (4x4, 6x6, 8x8, 9x9, 10x10 e 12x12) com vários níveis (fácil, médio e difícil). Considerou-se como métrica para avaliar o algoritmo proposto a porcentagem média de soluções válidas que ele é capaz de determinar e o tempo médio que cada algoritmo leva para encontrar uma solução válida. Em seguida, realizamos a comparação da nossa proposta com os demais algoritmos.

3. Fundamentação Teórica

Nessa seção são apresentados os conceitos fundamentais sobre o Sudoku e Teoria dos Grafos.

3.1. Jogo Sudoku

O Sudoku é um jogo de raciocínio lógico que possui uma correlação com o Quadrado Latino proposto por Euler no século XVII [Delahaye 2006]. O Quadrado Latino é formado por uma matriz n x n, preenchida com n símbolos, no qual não é permitido que um

símbolo ocorra mais de vez em cada linha ou coluna. Já o Sudoku tem como objetivo distribuir algarismos de 1 a n em uma matriz *n x n* obedecendo a seguinte regra: Não é permitido repetir o números na mesma linha e coluna ou no mesmo bloco da matriz [Bailey et al. 2008]. Os blocos de uma matriz do *pluzze* Sudoku representam submatrizes de ordem inferior a n.

A matriz do Sudoku é preenchida previamente com alguns valores. Posteriormente, ao inserir um valor na matriz, é verificado se ele atende ou não as restrições. Devido ao conjunto de restrições definidos para o problema, [Yato and Seta 2003] classificou-o como NP-difícil, por meio da redução do Quadrado Latino, que é NP-Completo, para o Sudoku. Os problemas NP-difíceis representam uma classe de problemas que são tão difíceis quanto os problemas mais difíceis em NP. Os problemas da classe NP são caracterizados por não ter ou não se conhecer soluções ótimas em tempo polinomial [Sipser 2012]. Mas, existe um algoritmo de tempo polinomial que verifica se uma solução é válida ou não.

Partindo do conceito das regras do Sudoku, na qual há relacionamentos e restrições entre as células da matriz, podemos representá-lo como um grafo, no qual é possível modelar suas relações e restrições por meio dos vértices e suas arestas.

3.2. Teoria dos Grafos

Um Grafo é definido como um conjunto de par ordenado (V, A) onde V representa os vértices $\{v_1, v_2, ..., v_n\}$ e A o conjunto de arestas $\{a_1, a2, ..., a_n\}$, onde os vértices são definidos como pontos interligados por meio de arestas [Rabuske 1992]. Os grafos podem ser orientados e/ou ponderados. Um grafo orientado é aquele onde suas arestas possuem direções. Já um grafo ponderado possui pesos em suas arestas.

Ainda segundo [Rabuske 1992], podemos representá-los por meio de estruturas de dados, tais como: matrizes e listas. Dado um grafo G com n vértices definimos uma matriz de adjacência nxn onde cada célula da matriz $A(G) = [a_{ij}]$ representa se os vértices v_i e v_j possuem uma aresta de ligação. Caso os vértices v_i e v_j forem adjacentes, $[a_{ij}] = 1$, caso contrário, $[a_{ij}] = 0$. No caso de grafos ponderados, podemos definir uma matriz de custo mínimo $C(G) = [c_{ij}]$, onde $[c_{ij}]$ recebe o valor associado a aresta que ligam o vértice v_i e v_j. Por meio da representação de matrizes e listas, podemos processá-lo de forma eficiente em um computador.

O Sudoku pode ser modelado como um grafo não orientado, onde os vértices representam as células da matriz e as arestas representam as relações entre cada célula. Por meio de técnicas de coloração de grafos podemos encontrar soluções válidas de forma eficiente para os Sudokus [Borges et al. 2016] e [Gouveia and Maciel Jr 2013]. A técnica aborda, na sua forma mais habitual, a atribuição de cores válidas para cada vértice v_i de um grafo G. Uma cor é dita válida para um vértice v_i apenas se a cor do seu vértice adjacente v_j for diferente.

4. Trabalhos Relacionados

Após o levantamento bibliográfico exploratório realizado obteve-se alguns estudos que abordam o problema de encontrar soluções válidas para o Sudoku por meio de várias técnicas e algoritmos. Algumas metaheurísticas são utilizadas para abordar o problema, tais

como: *Simulated Annealing* e Busca Tabu [De Souza and Romero 2013]. Algoritmos genéticos também são utilizados para abordar o problema [Chel et al. 2016] e [Thirer 2012].

Embora o Sudoku não seja um problema de otimização, ele pode ser solucionado modelando-o em Programação Linear Inteira e também como um problema que satisfaça restrições por meio de programação por restrições, onde cada regra é uma restrição para um conjunto de variáveis [Takano et al. 2015]

O problema pode ser modelado também como um grafo e abordado como um problema de coloração [Borges et al. 2016] e [Gouveia and Maciel Jr 2013], no qual é possível obter bons resultados para diversas instâncias e diversos níveis de complexidade. Em [Borges et al. 2016] os autores apresentam uma comparação entre um algoritmo *Backtracking* sem poda e um algoritmo *Backtracking* combinado com o algoritmo DSatur que utiliza como base a saturação (vértice com maior quantidade vizinhos coloridos) para a escolha do vértice a colorir. Já o estudo [Gouveia and Maciel Jr 2013] apresenta também resultados satisfatórios ao comparar dois algoritmos, um BFS e o outro um algoritmo de Busca em Profundidade (DFS).

Com base nas observações dos principais trabalhos encontrados, identificou-se que o algoritmo BFS e o conceito de saturação de vértice podem resolver de forma eficiente as instâncias do problema. Portanto, este artigo propõe um algoritmo que baseia-se no conceito do algoritmo BFS com escolha dos vértices com base no nível de saturação do vértice.

5. Algoritmo SaturBFS

Para a realização do estudo, modelamos o Sudoku por meio de um grafo (Figura 2 (a)). No Sudoku, dois vértices (v, u) são adjacentes se pertencem a mesma linha, coluna ou ao mesmo bloco. Como abordagem para encontrar soluções válidas para o Sudoku, utilizamos a abordagem de coloração (Figura 2 (b)), onde para cada vértice era atribuída uma cor (número válido) com base no grau do Sudoku.

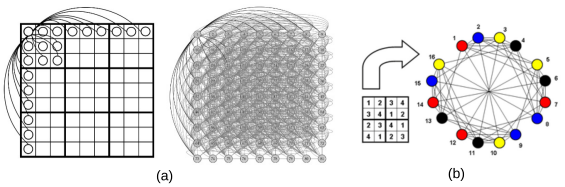


Figura 2. Modelagem do sudoku em forma de um grafo não orientado.

Utilizou-se como base o algoritmo BFS e o algoritmo DSatur para conceber o algoritmo proposto. Primeiramente, escolhe-se um vértice do grafo de maior saturação para iniciar a coloração. O vértice de maior saturação é aquele na qual possui a maior quantidade de adjacentes coloridos. Posteriormente, o vértice é colorido, colocado em uma fila e os seus adjacentes são recuperados por ordem de saturação, em seguida são coloridos e inseridos na fila. O algoritmo executa até colorir todos os vértices ou encontrar um vértice que não possui possibilidade de cores. O método **colorVertex(v)** seleciona as cores disponíveis para um determinado vértice e escolhe uma cor para colorir o vértice por meio de um processo aleatório. O pseudocódigo do algoritmo proposto é apresentado em Algoritmo 1.

Algorithm 1: Heuristic Saturated Breadth-First Search (SaturBFS)

```
Input: Graph G
Output: True or False
queue ← φ;
beginVertex ← highestSaturationVertex(G);
colorVertex(beginVertex);
insert(queue, beginVertex);
while queue ≠ φ do
    vertex ← remove(queue);
    adjacentVertex ← highestSaturationAdjacent(G, vertex);
    while adjacentVertex ≠ φ do
        color ← colorVertex(adjacentVertex);
        if color = φ then
            return False;
        else
            insert(queue, adjacentVertex);
            adjacentVertex ← highestSaturationAdjacent(G, vertex);
        end
    end
end
return True;
```

A busca por um vértice de maior saturação **highestSaturationVertex(G)** possui ordem de $O(V * Adj)$. Onde V representa a quantidade de vértices do grafo e Adj a quantidade de vértices adjacentes a um vértice v. Pois, verifica-se o grau de saturação de cada vértice por meio dos seus adjacentes. Já a busca por um vértice adjacente de maior saturação **highestSaturationAdjacent(G, v)** possui ordem de $O(Adj^2)$.

O método **colorVertex(v)** é de ordem de $O(K * Adj)$, pois é necessário verificar para cada cor k se ela já foi utilizada em algum vértice adjacente de v. A variável K representa a quantidade máxima de cores do grafo do Sudoku. Atribuições e comparações possuem ordem de $O(1)$. A iteração da fila possui ordem de $O(V)$ já que todos os vértices precisam ser inseridos nela. Portanto, concluímos que a complexidade do SaturBFS é $O(V * Adj) + O(K * Adj) + O(V) * [O(Adj^2) + O(K * Adj)]$.

6. Avaliação

Esta seção apresenta os procedimentos para a realização da avaliação do algoritmo proposto e em seguida os resultados obtidos são apresentados.

6.1. Procedimentos

Neste estudo foram considerados Sudokus (4x4, 6x6, 8x8, 9x9, 10x10 e 12x12) para níveis (fácil, médio e difícil). Para cada Sudoku em cada nível foram selecionados 10 tabuleiros distintos para computar uma média do percentual de soluções. Os Sudokus utilizados no estudo foram retirados do Sudoku-download.net[1], Sudoku.cool[2] e dos livros do Nick Snels[3]. Cada Sudoku foi colocado em um arquivo *.txt*.

Para comparar a solução proposta, foram implementados dois algoritmos encontrados em [Gouveia and Maciel Jr 2013]. O primeiro algoritmo (BFS) é uma abordagem de Busca em Largura. Já o segundo (DFS) utiliza uma abordagem de Busca em Profundidade. Cada algoritmo executa 10 vezes para cada instância do Sudoku para computar a porcentagem de soluções válidas encontradas. Devido ao comportamento aleatório

[1]http://www.sudoku-download.net/

[2]https://sudoku.cool/

[3]https://www.puzzlebooks.net/

dos algoritmos, a cada iteração, o algoritmo realiza uma quantidade máxima de tentativas. Esse valor máximo de tentativas é definido em uma variável chamada de ***max_iter***. Esse procedimento de avaliação dos algoritmos foi realizado com base na proposta de [Martí et al. 2010].

Os algoritmos foram implementados por meio da linguagem de programação Java. Os testes foram realizados em um notebook da marca Lenovo com sistema operacional Ubuntu 20.04 LTS, 20GB de memória RAM DDR4 2666MHz, processador Intel® Core™ i7-8565U CPU @ 1.80GHz × 8 e uma placa de NVIDIA GeForce MX110. Os resultados obtidos por meio do experimento foram apresentados em tabelas e gráficos.

6.2. Resultados

Para a execução dos testes, considerou-se o valor da variável ***max_iter*** de acordo com o nível de complexidade dos Sudokus. Portanto, os valores 10.000, 20.000 e 30.000 foram considerados para os níveis fácil, médio e difícil respectivamente.

Notamos que para pequenos Sudoku (4x4 e 6x6) os algoritmos conseguiram um excelente desempenho com relação às soluções válidas encontradas e o tempo para determinar uma solução independente da complexidade. Quando trata-se de Sudoku fáceis (Tabela 1) maiores que 6x6 o desempenho dos algoritmos BFS e DFS reduz bastante com relação à porcentagem de soluções encontradas. Exceto para os Sudoku 12x12, no qual os algoritmos obtiveram um resultado satisfatório. Isso pode ser explicado pelo fato de que os Sudoku 12x12 foram retirados dos livros do Nick Snels, diferente dos demais. Já o algoritmo SaturBFS continua com bons resultados, onde obteve uma média de 93,5% de soluções válidas encontradas em um tempo médio de 0,196 segundos.

Tabela 1. Nível fácil (max_iter = 10.000)

Sudoku	Nº vértices	BFS		DFS		SaturBFS	
		Soluções válidas(%)	Tempo(s)	Soluções válidas(%)	Tempo(s)	Soluções válidas(%)	Tempo(s)
4x4	16	100%	0,0003	100%	0,0002	100%	0,0007
6x6	36	100%	0,0011	100%	0,0019	100%	0,0007
8x8	64	19%	0,8178	8%	2,3603	83%	0,5584
9x9	81	76%	0,3400	54%	0,9308	98%	0,1109
10x10	100	13%	1,2337	31%	1,2256	82%	0,4586
12x12	144	95%	0,0594	94%	0,3756	98%	0,0309
Média		**67,16%**	**0,409**	**64,5%**	**0,815**	**93,5%**	**0,193**

A Tabela 2 apresenta os resultados para os Sudoku de nível médio. No nível médio já notamos uma queda com relação nas porcentagens de soluções válidas encontradas. Mas, ainda, o algoritmo SaturBFS apresenta resultados excelentes. Ele obteve resultado ruim para um tabuleiro 12x12, onde encontrou apenas 8% de soluções válidas. Os demais não conseguiram obter soluções.

Já a Tabela 3 apresenta os resultados para os Sudoku de nível difícil. Os algoritmos BFS e DFS apresentaram resultados muito baixos com relação a soluções válidas encontradas para Sudoku acima de 6x6. Já o SaturBFS não obteve um bom resultado para tabuleiros 12x12 de nível difícil. Mas, obteve bons resultados para os demais com uma média de 60,5% e com um tempo bem aceitável de 4,94 segundos. Os gráficos da Figura 3 mostram um resumo das médias obtidas na realização do experimento com relação

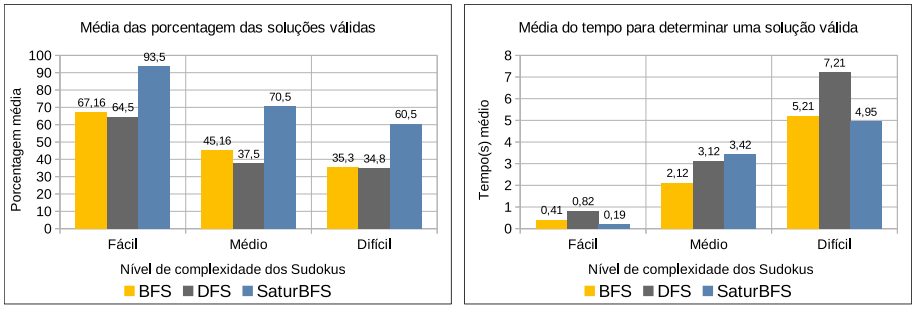
Tabela 2. Nível médio (max_iter = 20.000)

Sudoku	Nº vértices	BFS		DFS		SaturBFS	
		Soluções válidas(%)	Tempo(s)	Soluções válidas(%)	Tempo(s)	Soluções válidas(%)	Tempo(s)
4x4	16	100%	0,0001	100%	0,001	100%	0,0001
6x6	36	100%	0,0038	100%	0,0048	100%	0,0002
8x8	64	20%	2,6195	5%	4,8340	78%	0,7727
9x9	81	18%	3,1310	3%	6,3073	77%	0,8469
10x10	100	15%	4,8443	17%	4,4622	60%	2,5421
12x12	144	0	0	0	0	8%	16,3662
Média		**45,16%**	**2,119**	**37,5%**	**3,121**	**70,5%**	**3,421**

às porcentagens de soluções válidas encontradas pelos algoritmos, e o tempo médio que cada um dele leva para encontrá-las.

Tabela 3. Nível difícil (max_iter = 30.000)

Sudoku	Nº vértices	BFS		DFS		SaturBFS	
		Soluções válidas(%)	Tempo(s)	Soluções válidas(%)	Tempo(s)	Soluções válidas(%)	Tempo(s)
4x4	16	100%	0,0001	100%	0,0001	100%	0,0002
6x6	36	100%	0,0230	100%	0,0157	100%	0,0027
8x8	64	9%	4,5653	2%	9,3520	67%	3,2272
9x9	81	0	0	1%	12,0830	45%	7,1257
10x10	100	3%	16,251	6%	14,6047	50%	11,328
12x12	144	0	0	0	0	1%	7,99
Média		**35,3%**	**5,209**	**34,8%**	**7,211**	**60,5%**	**4,945**



(a) (b)

Figura 3. (a) Porcentagem média de soluções válidas encontradas com relação aos níveis de dificuldade. (b) Tempo médio para determinar uma solução considerando os níveis de dificuldade.

Com esse experimento, notamos que o algoritmo concebido no estudo apresenta melhorias significativas com relação às demais abordagens utilizadas na avaliação. No gráfico (b) da Figura 3, o algoritmo SaturBFS obteve um tempo maior para obter uma solução, isso ocorreu pelo fato de que apenas o algoritmo SaturBFS conseguiu encontrar soluções para um tabuleiro 12x12 de nível médio com um média de 16,3 segundos. Isso influenciou na média do tempo que o algoritmo leva para encontrar uma solução.

7. Conclusão e Trabalhos Futuros

Este estudo concebeu um algoritmo intitulado SaturBFS que tem como objetivo encontrar soluções válidas para tabuleiros de Sudoku para diferentes níveis e tamanhos. O estudo

também teve como objetivo realizar uma avaliação comparativa do algoritmo proposto com outras abordagens encontradas na literatura.

A análise dos resultados apontaram que o SaturBFS apresenta melhorias significativas, tanto com relação ao tempo para determinar uma solução válida quanto para o número de soluções encontradas, quando comparado com os algoritmos BFS não saturada e o DFS.

Como trabalhos futuros propomos a realização de uma nova avaliação com outros algoritmos, tais como: *Simulated Annealing*, Busca Tabu, *Backtraking* e também com algoritmos que se utilizam da modelagem do Sudoku por meio de programação linear inteira. Pretende-se também incluir na nova avaliação a aplicação do algoritmo SaturBFS para instâncias maiores do Sudoku.

Referências

Bailey, R. A., Cameron, P. J., and Connelly, R. (2008). Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and hamming codes. *The American Mathematical Monthly*, 115(5):383–404.

Borges, S., Lima, T., and Marques, V. (2016). Coloração de grafos aplicado na resolução do sudoku.

Chel, H., Mylavarapu, D., and Sharma, D. (2016). A novel multistage genetic algorithm approach for solving sudoku puzzle. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 808–813. IEEE.

De Souza, S. S. and Romero, R. (2013). Metaheurísticas simulated annealing e busca tabu aplicadas na resolução do quebra-cabeça sudoku. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, 1(1).

Delahaye, J.-P. (2006). The science behind sudoku. *Scientific American*, 294(6):80–87.

Gouveia, T. and Maciel Jr, P. D. (2013). Técnicas de coloração de grafos aplicadas à resolução de quebra-cabeças do tipo sudoku. In *Congresso Norte Nordeste de Pesquisa e Inovação*, page 1.

Martí, R., Moreno-Vega, J. M., and Duarte, A. (2010). Advanced multi-start methods. In *Handbook of metaheuristics*, pages 265–281. Springer.

Rabuske, M. (1992). *Introdução a Teoria dos Grafos*. Série didática. Editora da UFSC.

Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage learning.

Takano, K., de Freitas, R., and de Sá, V. (2015). O jogo de lógica sudoku: modelagem teórica, np-completude e estratégias algorítmicas exatas e heurísticas. In *Anais do XXXIV Concurso de Trabalhos de Iniciação Científica da SBC*, pages 71–80. SBC.

Thirer, N. (2012). About the fpga implementation of a genetic algorithm for solving sudoku puzzles. In *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*, pages 1–3. IEEE.

Yato, T. and Seta, T. (2003). Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060.