

RESOLVENDO SUDOKUS GENERALIZADOS ATRAVÉS DE ALGORITMOS DE BACKTRACKING E DE DANCING LINKS

SOLVING GENERALIZED SUDOKUS THROUGH BACKTRACKING AND DANCING LINKS ALGORITHMS

RESOLVIENDO SUDOKUS GENERALIZADOS MEDIANTE ALGORITMOS DE BACKTRACKING Y DANCING LINKS

Allan Cruz¹

Programa de Pós-Graduação Doutorado em Ciência Da Computação -
Associação UFMA/UFPI, São Luís, Maranhão

João D. de S. Almeida ²

Programa de Pós-Graduação Doutorado em Ciência Da Computação -
Associação UFMA/UFPI, São Luís, Maranhão

Pamela Cruz³

Faculdade de Letras da Universidade de Coimbra, Coimbra, Portugal

RESUMO

Este trabalho apresenta uma implementação da resolução de sudokus generalizados através de algoritmos de backtracking e Dancing Links. Foram implementados dois algoritmos, um utilizando a técnica de backtracking e outro utilizando a técnica de Dancing Links. Os algoritmos foram testados em diferentes tamanhos de sudoku (9x9, 16x16 e 25x25) e o tempo de execução foi medido. Os resultados mostraram que o algoritmo de Dancing Links é mais eficiente que o algoritmo de backtracking para a resolução de sudokus generalizados.

¹ Doutorando em Ciência da Computação. UFMA/DCCMAPI. allankassio@gmail.com.

² Doutor em Ciência da Computação. UFMA/DCCMAPI. joao.dallyson@ufma.br.

³ Doutoranda em Ciência da Informação. Universidade de Coimbra. pam.tmaia@gmail.com.

Palavras-chave: Sudoku; Backtracking; Dancing Links; Algoritmo.

ABSTRACT

This paper presents an implementation for solving generalized Sudoku puzzles using backtracking and Dancing Links algorithms. Two algorithms were implemented, one using the backtracking technique and the other using the Dancing Links technique. The algorithms were tested on Sudokus of different sizes (9x9, 16x16 and 25x25) and the execution time was measured. The results show that the Dancing Links algorithm is more efficient than the backtracking algorithm for solving generalized Sudoku puzzles.

Keywords: Sudoku; Backtracking; Dancing Links; algorithm.

RESUMEN

Este trabajo presenta una implementación de la resolución generalizada de sudokus a través de algoritmos de backtracking y Dancing Links. Se implementaron dos algoritmos, uno usando la técnica de backtracking y el otro usando la técnica de Dancing Links. Los algoritmos se probaron en diferentes tamaños de sudoku (9x9, 16x16 y 25x25) y se midió el tiempo de ejecución. Los resultados mostraron que el algoritmo de Dancing Links es más eficiente que el algoritmo de retroceso para resolver sudokus generalizados.

Palabras clave: Sudoku; retroceder; Enlaces de baile; algoritmo.

1 INTRODUÇÃO

O Sudoku é um jogo de lógica que consiste em um quadro 9x9 dividido em subquadros 3x3, no qual cada subquadro contém os dígitos de 1 a 9 sem repetição e cada linha e coluna do quadro também contém os dígitos de 1 a 9 sem repetição (Felgenhauer & Jarvis, 2006). O objetivo do jogo é preencher os quadros com os dígitos de 1 a 9 de tal forma que cada dígito apareça apenas uma vez em cada linha, coluna e subquadro. O tabuleiro inicia com alguns

espaços previamente preenchidos (geralmente com inteiros de 1 a n). A Figura 1 mostra uma instância do Sudoku 9x9 e uma possível solução.

Figura 1 – Grid Sudoku 9x9 com solução

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

Fonte: Adaptado de Dreamstime (2018).

A história dos quebra-cabeças de Sudoku provavelmente tem suas raízes no conceito matemático de quadrados latinos (McKay & Wanless, 2005). Eles são um tipo de quadrado mágico, uma tabela quadrada contendo números naturais, tal que a soma dos números em cada linha, coluna, diagonal e quadrado mágico principal é a mesma. A Figura 2 mostra um exemplo de um quadrado latino de ordem 4.

Figura 2 – Quadrado latino de ordem 4

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

Fonte: Adaptado de Nogueira (2015).

Os quadrados latinos foram introduzidos pela primeira vez pelo matemático suíço Leonhard Euler em 1782. Em 1892, Euler apresentou o problema de determinar se existe um quadrado latino de ordem 9 (COTA, 2011). Esse problema permaneceu sem resposta até 1915, quando o matemático estadunidense Sam Loyd publicou um quadrado latino de ordem 9 em sua revista americana de matemática, *Mathematical Puzzles* (COSTA, 2014).

No entanto, Loyd afirmou que esse quadrado latino de ordem 9 era um problema de "brinquedo", e não uma demonstração de um teorema matemático. O tabuleiro de Sudoku que conhecemos hoje, no entanto, é uma prova de um teorema matemático. Em 1979, o matemático japonês Tetsuya Nishio e seus colegas demonstraram que existe um número infinito de tabuleiros de Sudoku que satisfazem as propriedades do jogo (TSAO, 2019).

Assim pode-se dizer que o Sudoku é um jogo relativamente novo. O nome do jogo é uma contração da frase em japonês *suuji wa dokushin ni kagiru*, que significa "os números devem estar sozinhos". O jogo ganhou popularidade nos Estados Unidos nos anos 2000 e, desde então, tem sido um dos jogos de lógica mais populares do mundo (MAIA, 2012).

1.1 Sudoku Generalizado

Um sudoku generalizado é um sudoku em que as dimensões da grade de jogo são ajustáveis (HAYNES, 2008). Isso significa que, em vez de uma grade 9x9, um sudoku generalizado pode ser uma grade de qualquer tamanho, como 16x16 (Figura 3), 25x25 ou 36x36. Além disso, as regras podem ser ajustadas para permitir que os jogadores usem qualquer número de 1 a 9, ou até mesmo letras ou símbolos diferentes.

O Sudoku generalizado é um problema de otimização NP-completo, o que significa que, em princípio, não existe um algoritmo que possa resolver o problema de forma eficiente para todos os possíveis casos (MAJI & PAL, 2014). Isto é devido à natureza de exponencial do problema, uma vez que aumentando o tamanho da grade do Sudoku, aumenta-se exponencialmente o número de possíveis soluções.

Um problema NP-difícil é um problema para o qual não existe um algoritmo de polynomial para resolvê-lo. Isso significa que, se um problema é NP-difícil, então não existe um algoritmo que possa resolvê-lo em tempo polinomial (RAI et. al, 2018). No entanto, existem algoritmos que podem resolver problemas NP-difíceis em tempo subpolinomial, mas esses algoritmos são considerados ineficientes (YILDIRIM et. al., 2008). Para tentar aproximar-se do resultado ideal são utilizadas heurísticas para prover soluções.

Figura 3 – Grid Sudoku 16x16

	6						8	11			15	14			16
15	11				16	14				12			6		
13		9	12					3	16	14		15	11	10	
2		16		11		15	10	1							
	15	11	10			16	2	13	8	9	12				
12	13			4	1	5	6	2	3					11	10
5		6	1	12		9		15	11	10	7	16			3
	2				10		11	6		5			13		9
10	7	15	11	16				12	13						6
9						1			2		16	10			11
1		4	6	9	13			7		11		3	16		
16	14			7		10	15	4	6	1				13	8
11	10		15				16	9	12	13			1	5	4
		12		1	4	6		16				11	10		
		5		8	12	13		10			11	2			14
3	16			10			7			6				12	

Fonte: Adaptado de Schmidl et al. (2014).

Uma heurística é uma técnica que busca uma solução aproximada para um problema de otimização NP-completo, de forma a encontrar uma solução que seja "suficientemente boa". Não existe uma heurística única que seja capaz de

resolver todos os problemas de Sudoku, uma vez que cada problema é único e pode requerer uma abordagem diferente (TAKANO et. al., 2015).

Uma heurística comum para o Sudoku é a de tentar preencher as células vazias com os valores que restam, de forma a minimizar o número de valores que faltam para completar a grade (DITTRICH et. al., 2010). Outra heurística é a de procurar as células que têm o menor número de valores possíveis e tentar preenchê-las com esses valores (TAKANO et. al., 2015).

Existem muitas outras heurísticas que podem ser usadas para resolver problemas de Sudoku, e é importante experimentar várias delas para encontrar aquela que melhor se adapta ao problema em questão.

2 ALGORITMOS DE RESOLUÇÃO DE SUDOKU GENERALIZADO

Existem vários algoritmos diferentes que podem ser usados para resolver um Sudoku Generalizado, mas a maioria dos algoritmos segue os mesmos princípios básicos. O primeiro passo é determinar o tamanho do quadrado e, em seguida, preencher as linhas e colunas com os números de 1 a N. Em seguida, é necessário determinar os blocos e, finalmente, preenchê-los com os números restantes.

Uma vez que o quadrado estiver completamente preenchido, é possível verificar se a solução é correta, comparando-a com as regras do Sudoku Generalizado. Se a solução estiver correta, é possível parar o algoritmo. Se a solução não estiver correta, é necessário verificar se há algum erro na solução e, se houver, corrigi-lo.

No entanto a solução para um Sudoku se enquadra como um problema da cobertura exata. Que é um problema de otimização combinatória que trata de encontrar um subconjunto de um determinado conjunto tal que os elementos do subconjunto cubram todos os elementos do conjunto dado, e dois elementos do subconjunto não cubram o mesmo elemento do dado conjunto (KAPANOWSKI, 2010). O problema de cobertura exata é um exemplo de problema de satisfação de restrição. Na maioria dos casos, quando são instâncias de solução possíveis, são resolvidos por algoritmos recursivos com backtracking. Dentre os algoritmos que resolvem o sudoku e se encaixam na cobertura exata podemos destacar os seguintes:

- **Minimum Remaining Values:** defende que é mais viável selecionar variáveis com menos possibilidades de valores (ABULUAIH et. al., 2018). Ou seja, para o Sudoku, significa selecionar a célula com menos candidatos de inserção para toda vez que for necessário procurar uma nova célula.
- **Forward Checking:** propõe o término quando finaliza a busca caso alguma variável ainda não testada não tenha valores dentro de seu domínio (SIMONIS, 2005). Para o caso do Sudoku, toda vez que o Backtracking encontrar uma célula em que não será possível inserir mais nenhum valor, a busca é finalizada e testada para um próximo estado (SKIENA, 2008).
- **Manipulação de Bits:** técnica utilizada para representação das estruturas de dados do Sudoku e manipulação das unidades (linha, coluna e bloco) com consulta em tempo constante (TAKANO et. al., 2015). Implementa o modelo teórico de coloração em hipergrafos, aplicando-se a ideia de que as unidades são hiperarestas, e assim o acesso para verificação de um dígito nas unidades durante a execução do método é $O(1)$. Para que a Manipulação de Bits ocorra é necessário que se guarde todos os dígitos pre-preenchidos do tabuleiro do Sudoku em vetores. Cada dígito do tabuleiro é representado por um algarismo de um número presente no vetor. A verificação é a remoção de dígitos e feita através dos vetores criados.
- **Constraint Propagation:** uma técnica que remove valores de um domínio de variáveis da qual não irão participar de nenhuma solução (RUSSEL, 2010). Enquanto o algoritmo for executado através da busca em profundidade, a técnica irá remover os candidatos do tabuleiro da qual seriam impossíveis de ser inseridas.
- **Dancing Links:** técnica proposta por Donald Knuth (KNUTH, 2000), também conhecida como DLX, para implementar de forma eficiente seu algoritmo X. O algoritmo X é um tipo de backtracking com podas, uma

busca em profundidade recursiva nao-determinística, que encontra todas as soluções para realizar a cobertura exata do problema.

Assim neste trabalho foi escolhido implementar o algoritmo de Dancing Links e paralelamente, a título de comparação de performance, um algoritmo trivial baseado backtracking.

2.1 Backtracking

Como todos os outros problemas de Backtracking, o Sudoku pode ser resolvido passo a passo, atribuindo números às células vazias. Antes de atribuir um número, devemos verificar se podemos atribuir. Verifica-se se o mesmo número não está presente na linha atual, coluna atual e subgrade atual. Depois de verificar a segurança, devemos atribuir um número e verificar recursivamente se essa atribuição leva a uma solução ou não. Se a atribuição não levar a uma solução, devemos tentar o próximo número para a célula vazia atual. E se nenhum dos números (1 a 9) levar a uma solução, devemos retornar falso e imprimir que não existe solução (nesse caso, provavelmente há um problema com a grade original gerada).

É necessário criar uma função que verifique, após atribuir o índice atual, se a grade se torna insegura ou não. Devemos manter um hashmap para uma linha, coluna e caixas. Se algum número tiver uma frequência maior que 1 no hashmap, retornamos falso, caso contrário, retornamos verdadeiro (o hashmap pode ser evitado usando loops).

Em resumo, o algoritmo segue essa ordem:

1. Crie uma função recursiva que recebe uma grade.
2. Verifique se há algum local não atribuído.
3. Se presente, atribua um número de 1 a n^2 , verifique se atribuir o número ao índice atual torna a grade insegura ou não.
4. Se for segura, chame recursivamente a função para todos os casos seguros de 0 a n^2 .
5. Se qualquer chamada recursiva retornar true, finaliza o loop e retornar true.
6. Se nenhuma chamada recursiva retornar true, então retorne false.

7. Se não houver local não atribuído, retorne true.

2.1.1 Análise de Complexidade do Backtracking

Complexidade de tempo: $O((n^2)^{n*n})$. Para cada índice não atribuído, existem n^2 opções possíveis, então a complexidade de tempo é $O((n^2)^{n*n})$. A complexidade de tempo permanece a mesma, mas haverá algumas podas iniciais, de modo que o tempo gasto será muito menor do que o algoritmo ingênuo, mas a complexidade de tempo do limite superior permanece a mesma.

Complexidade espacial: $O(n * n)$, pois para armazenar a saída é necessária uma estrutura do tipo matriz.

2.1 Dancing Links

Programas para resolver o Sudoku, geralmente caem no problema da cobertura total ou cobertura exata. Esse tipo de problema pode ser resolvido com o algoritmo conhecido como “Algoritmo X” (KNUTH, 2000).

Dada uma coleção S de subconjuntos do conjunto X, uma cobertura exata é o subconjunto S^* de S tal que cada elemento de X contido é exatamente um subconjunto de S^* (KNUTH, 2000). Deve satisfazer as seguintes duas condições:

- A interseção de quaisquer dois subconjuntos em S^* deve estar vazia. Ou seja, cada elemento de X deve estar contido em no máximo um subconjunto de S^*
- A união de todos os subconjuntos em S^* é X. Isso significa que a união deve conter todos os elementos do conjunto X. Então podemos dizer que S^* cobre X.

Exemplo (representação padrão):

Seja:

$$S = \{ A, B, C, D, E, F \}$$

e

$$X = \{1, 2, 3, 4, 5, 6, 7\}$$

Tal que:

$$A = \{1, 4, 7\}, B = \{1, 4\}, C = \{4, 5, 7\}, D = \{3, 5, 6\}, E = \{2, 3, 6, 7\}, F = \{2, 7\}.$$

Então $S^* = \{B, D, F\}$ é uma cobertura exata, porque cada elemento em X está contido exatamente uma vez nos subconjuntos $\{B, D, F\}$. Se unirmos subconjuntos, obteremos todos os elementos de X :

$$B \cup D \cup F = \{1, 2, 3, 4, 5, 6, 7\}$$

O problema da cobertura exata é um problema de decisão para determinar se a cobertura exata existe ou não. É considerado um problema NP-Completo. O problema pode ser representado na forma de uma matriz onde a linha representa os subconjuntos de S e as colunas representam o elemento de X .

2.2.1 Algoritmo X

O Algoritmo X foi proposto visando poder encontrar todas as soluções para o problema de cobertura exata. O Algoritmo X pode ser implementado eficientemente pela técnica de Dancing Links (ou links dançantes), proposta pelo mesmo autor, chamada DLX (KNUTH, 2000).

É um algoritmo recursivo, primeiro em profundidade, e um algoritmo que usa o conceito de backtracking. É de natureza não determinística, o que significa que, para a mesma entrada, pode exibir comportamentos diferentes em uma execução diferente.

A seguir está o pseudocódigo para o Algoritmo X:

1. Se a matriz A não tiver colunas, a solução parcial atual é uma solução válida; terminar com sucesso.
2. Caso contrário, escolha uma coluna c (deterministicamente).
3. Escolha uma linha r tal que $A[r] = 1$ (não deterministicamente).
4. Inclua a linha r na solução parcial.
5. Para cada coluna j tal que $A[r][j] = 1$,
 - a. para cada linha i tal que $A[i][j] = 1$,
 - i. exclua a linha i da matriz A .
 - b. exclua a coluna j da matriz A .

6. Repita este algoritmo recursivamente na matriz reduzida A.
7. Escolha não determinística de r significa, o algoritmo copia a si mesmo no subalgoritmo. Cada subalgoritmo herda a matriz original A, mas a reduz em relação ao r escolhido (veremos isso em breve no exemplo)

O subalgoritmo forma uma árvore de busca com o problema original na raiz e cada nível k tem um subalgoritmo que corresponde às linhas escolhidas no nível anterior (assim como o espaço de busca) (KNUTH, 2000).

Se a coluna escolhida C for totalmente zero, então não há subalgoritmos e o processo terminou sem sucesso. Knuth (2000) sugere que devemos escolher a coluna com o número mínimo de 1s. Se não sobrar nenhuma coluna, então sabemos que encontramos nossa solução.

A técnica do Dancing Links baseia-se na ideia de lista encadeada duplamente circular. Conforme discutido por Knuth (2000), transforma-se o problema de cobertura exata em forma de matriz de 0 e 1. Aqui cada “1” na matriz é representado por um nó de lista encadeada e toda a matriz é transformada em uma malha de nós conectados de 4 vias. Cada nó contém os seguintes campos:

- Ponteiro para o nó à esquerda dele
- Ponteiro para o nó direito a ele
- Ponteiro para o nó acima dele
- Ponteiro para o nó abaixo dele
- Ponteiro para listar o nó do cabeçalho ao qual pertence

Cada linha da matriz é, portanto, uma lista circular vinculada entre si com ponteiros para a esquerda e para a direita e cada coluna da matriz também será uma lista circular vinculada a cada uma acima e abaixo com os ponteiros para cima e para baixo. Cada lista de colunas também inclui um nó especial chamado “nó de cabeçalho de lista”. Este nó de cabeçalho é como um nó simples, mas tem poucos campos extras:

- Código da coluna
- Contagem de nós na coluna atual

Podemos ter dois tipos diferentes de nós, um para as colunas que possui o "atributo tamanho" e um para os nós "dançantes", que possui o atributo que identifica o "cabeçalho da coluna".

3 CONCLUSÃO

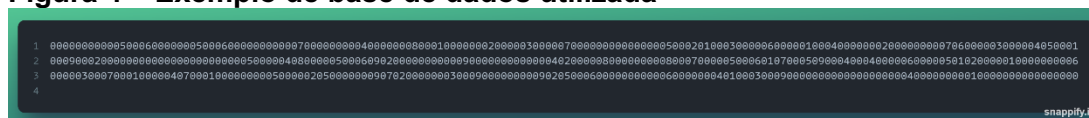
Como dito anteriormente as soluções escolhidas foram a de Backtracking e de Dancing Links. Cada um desses algoritmos foi implementado como uma classe separada. Adicionalmente foram implementadas classes para o Iterador, para o Nó, para representar a Matriz do Sudoku, para gerar o sudoku a partir dos datasets e para validação do sudoku. Os códigos do sistema estão disponíveis no Anexo deste trabalho e no repositório GitHub: <https://github.com/allankassio/generalized-sudoku-solver>.

3.1 Base de dados utilizada

A base de dados utilizada consiste em quatro arquivos CSV que modelam e representam os puzzles do soduko como linhas únicas. Cada coluna é composta por dois dígitos (permitindo assim casas maiores do que 9. Para cada tamanho N do rank representado, os arquivos possuem $2(N^2)$ dígitos por linha.

Para tabuleiros de 9x9 a base de dados possui 3000 entradas. Para 16x16, 25x25 e 36x36 são 10 entradas para cada um. Já 49x49 possui 6 entradas, enquanto 64x64 tem 5 entradas e 81x81 tem apenas 1 entrada. A Figura 1 mostra um exemplo de 3 entradas de dados do tipo 9x9.

Figura 4 – Exemplo de base de dados utilizada

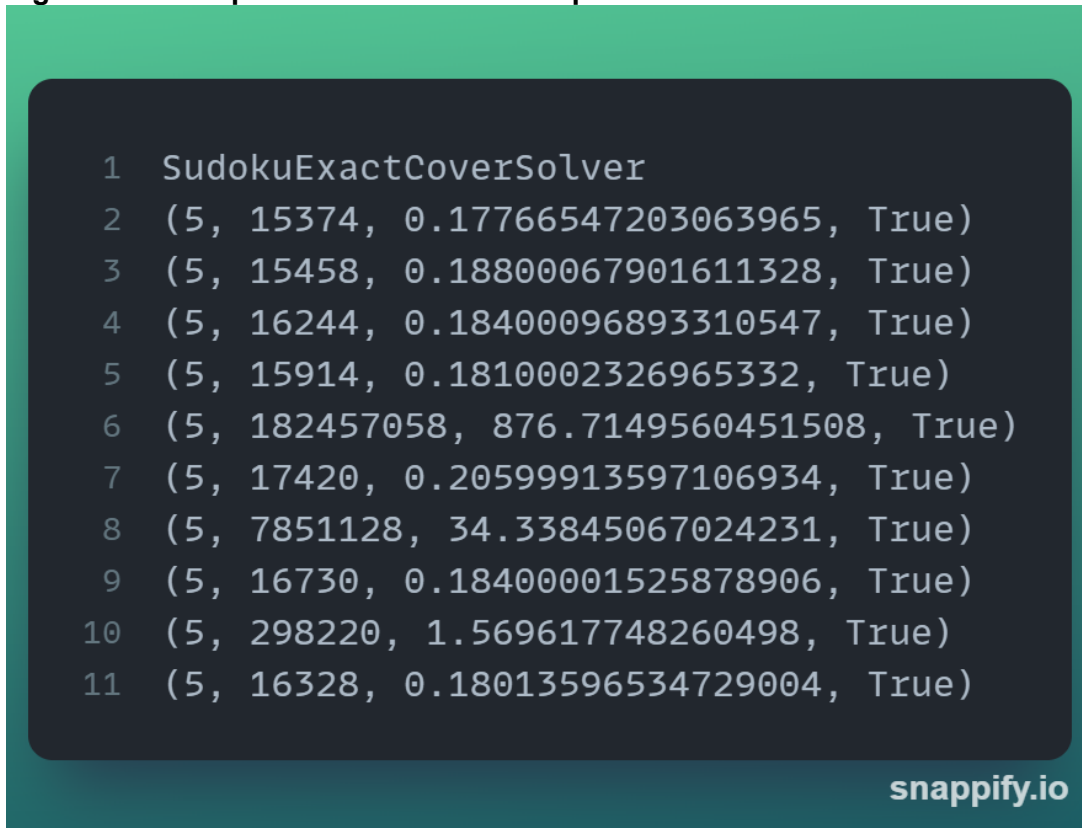


Fonte: Elaborado pelo autor com base no trabalho realizado.

3.2 Exemplos de saídas

As saídas são compostas do número de iterações necessárias para resolver o sudoku, o tempo levado para a resolução e se aquela resposta é válida. A Figura 4 mostra um exemplo de saídas no console.

Figura 5 – Exemplo de saída no console para entradas com n=5

A screenshot of a terminal window with a dark background and light green text. It shows 11 lines of output from a program named 'SudokuExactCoverSolver'. Each line represents a solution for a 5x5 Sudoku puzzle, listing the number of iterations, the solving time in seconds, and a boolean indicating if the solution is valid. The 'snappify.io' logo is visible in the bottom right corner of the terminal window.

```
1  SudokuExactCoverSolver
2  (5, 15374, 0.17766547203063965, True)
3  (5, 15458, 0.18800067901611328, True)
4  (5, 16244, 0.18400096893310547, True)
5  (5, 15914, 0.1810002326965332, True)
6  (5, 182457058, 876.7149560451508, True)
7  (5, 17420, 0.20599913597106934, True)
8  (5, 7851128, 34.33845067024231, True)
9  (5, 16730, 0.18400001525878906, True)
10 (5, 298220, 1.569617748260498, True)
11 (5, 16328, 0.18013596534729004, True)
```

Fonte: Elaborado pelo autor com base no trabalho realizado.

3.3 Comparação dos algoritmos

Para comparar o algoritmo de backtracking com o de Dancing Links foram utilizados os seguintes critérios: comparação da média dos 10 primeiros resultados para matrizes 9x9, 16x16 e 25x25 tanto em termo de iterações quanto em termo de tempo médio.

Ambos algoritmos utilizaram a mesma base de dados como entrada. Na Figura 5 podemos vemos a comparação para n=3. Na Figura 7 podemos vemos a comparação para n=4. Na Figura 8 podemos vemos a comparação para n=5. Os dados do Backtracking para n=4 e n=5 foram inviáveis, e por isso não foram gerados.

Figura 6 – Comparação de desempenho com n=3

Dancing Links			Backtracking		
3	2112	0,005000352859497070	3	170126	10,657027721405000000
3	1992	0,004998445510864250	3	130052	7,676674604415890000
3	2154	0,005249023437500000	3	3079	0,203737020492553000
3	2160	0,003999710083007810	3	47673	3,103725910186760000
3	2010	0,003999471664428710	3	3106	0,192025184631347000
3	2106	0,004999637603759760	3	46140	2,963670492172240000
3	2070	0,003999948501586910	3	16683	1,085922479629510000
3	2244	0,005005598068237300	3	40344	2,709635019302360000
3	2118	0,005000352859497070	3	87861	5,951405286788940000
3	2166	0,006001472473144530	3	329971	22,379093408584500000

N	Iterações	Tempo	N	Iterações	Tempo
3	2113,2	0,004825401306152340	3	87503,5	5,692291712760910000

Fonte: Elaborado pelo autor com base no trabalho realizado.

O algoritmo de Backtracking apresenta um crescimento exponencial muito rápido a medida que aumentamos o valor de n. Em contrapartida, o Dancing Links, mesmo também apresentando crescimento exponencial, apresenta para a grande maioria dos casos tempos abaixo de 1 segundo.

Figura 7 – Comparação de desempenho com n=4

Dancing Links		
4	36114	0,115999698638916000
4	223784	0,669000148773193000
4	35288	0,108999967575073000
4	11212	0,045999765396118100
4	6802	0,035000085830688400
4	6248	0,033999443054199200
4	137750	0,371982574462890000
4	5780	0,034000158309936500
4	25254	0,085000276565551700
4	6440	0,033999919891357400

N	Iterações	Tempo
4	49467,2	0,153398203849792000

Fonte: Elaborado pelo autor com base no trabalho realizado.

Figura 8 – Comparação de desempenho com n=5

Dancing Links		
5	15374	0,1776654720306390
5	15458	0,1880006790161130
5	16244	0,1840009689331050
5	15914	0,1810002326965330
5	182457058	876,7149560451500000
5	17420	0,2059991359710690
5	7851128	34,3384506702423000
5	16730	0,1840000152587890
5	298220	1,5696177482604900
5	16328	0,1801359653472900

N	Iterações	Tempo
5	19071987,4	91,392382693290600000

Fonte: Elaborado pelo autor com base no trabalho realizado.

Com isso é possível perceber que Dancing Links pode ser mais eficiente que backtracking para resolver o sudoku, dependendo da implementação, dependendo do tamanho do sudoku e da entropia da entrada. Dancing links consegue resolver sudoku de forma eficiente porque ao contrário de backtracking, dancing links não precisa fazer muitas buscas caso existam várias soluções. Isso é possível pois ele utiliza listas encadeadas circulares para representar os dados.

REFERÊNCIAS

- ABULUAH, S. et al. Fog of Search Resolver for Minimum Remaining Values Strategic Colouring of Graph. International Conference on Soft Computing in Data Science. Anais...Springer, 2018.
- COSTA, O. DA. A matemática recreativa no ensino Básico. PhD Thesis—[s.l: s.n.].
- COTA, A. C. DA S. Euler e os números pentagonais. Master's Thesis—[s.l.] Universidade Federal do Rio Grande do Norte, 2011.
- DITTRICH, M.; PREUSSER, T. B.; SPALLEK, R. G. Solving Sudokus through an Incidence Matrix on an FPGA. 2010 International Conference on Field-Programmable Technology. Anais...IEEE, 2010.

DREAMSTIME. Sudoku com solução livra para usar se. , 2018. Disponível em:
<<https://pt.dreamstime.com/fotos-de-stock-royalty-free-sudoku-com-solucao-livra-para-usar-se-image9264088>>

FELGENHAUER, B.; JARVIS, F. Mathematics of Sudoku I. 2006.

FISCHER, T. A necessary solution condition for Sudoku. arXiv preprint arXiv:1210.6343, 2012.

GUNTHER, J.; MOON, T. Entropy minimization for solving Sudoku. IEEE transactions on signal processing, v. 60, n. 1, p. 508–513, 2011.

HARRYSSON, M.; LAESTANDER, H. Solving sudoku efficiently with dancing links. , 2014.

HAYNES, I. W. Analysis of generalized sudoku puzzles: A mixture of discrete techniques. PhD Thesis—[s.l.] University of South Carolina, 2008.

KAPANOWSKI, A. Python for education: the exact cover problem. arXiv preprint arXiv:1010.5890, 2010.

KNUTH, D. E. Dancing links. arXiv, , 2000. Disponível em:
<<http://arxiv.org/abs/cs/0011047>>. Acesso em: 10 jul. 2022

MAIA, P. A. DE A. Jogos na aprendizagem matemática: Uma proposta com Sudokus, Malba Tahan e Tangram. 2012.

MAJI, A. K.; PAL, R. K. Sudoku solver using minigrid based backtracking. 2014 IEEE International Advance Computing Conference (IACC). Anais...IEEE, 2014.

MCKAY, B. D.; WANLESS, I. M. On the Number of Latin Squares. Annals of Combinatorics, v. 9, n. 3, p. 335–344, 2005.

NOGUEIRA, R. Quadrados mágicos. Ciência de Garagem, 2015. Disponível em:
<<https://cienciadegaragem.blogspot.com/2015/09/quadrados-magicos.html>>. Acesso em: 10 jul. 2022

RAI, D.; INGLE, M.; CHAUDHARI, N. POLYNOMIAL 3-SAT REDUCTION OF SUDOKU PUZZLE. International Journal of Advanced Research in Computer Science, v. 9, n. 3, 2018.

RUSSELL, P. N. Artificial Intelligence: A Modern Approach by Stuart. Russell and Peter Norvig contributing writers, Ernest Davis...[et al.], 2010.

SCHMIDL, D. et al. Suitability of Performance Tools for OpenMP Task-Parallel Programs. Em: KNÜPFER, A. et al. (Eds.). Tools for High Performance Computing 2013. Cham: Springer International Publishing, 2014. p. 25–37.

SIMONIS, H. Sudoku as a constraint problem. CP Workshop on modeling and reformulating Constraint Satisfaction Problems. Anais...Citeseer, 2005.

SKIENA, S. The Algorithm Design Manual. Springer Publishing Company. Incorporated, , 2008.

TAKANO, K.; DE FREITAS, R.; DE SÁ, V. G. P. O jogo de lógica Sudoku: modelagem teórica, NP-completude, algoritmos e heurísticas. XXXIV Concurso de Trabalhos de Iniciação Científica da Sociedade Brasileira de Computação, 2015.

TSAO, H. EVOLUTIONARY MATHEMATICS AND ARTS FOR TALK ELEGANCY INVESTIGATION. Evolutionary Progress in Science, Technology, Engineering, Arts, and Mathematics (STEAM), p. 1–68, 2019.

VIDEL, M.; PALO, M. Scaling of popular Sudoku solving algorithms. , 2014.

YILDIRIM, H.; KRISHNAMOORTHY, M.; DEO, N. A study of the Sudoku graph family. Proceedings of the Thirty-Ninth Southeastern International Conference on Combinatorics, Graph Theory and Computing. Congr. Numer. Anais...2008.