

FUNDAÇÃO GETÚLIO VARGAS  
ESCOLA DE MATEMÁTICA APLICADA  
CURSO DE GRADUAÇÃO EM MATEMÁTICA APLICADA

JULIANA HUTHER ALBERNAZ CRESPO

**SUDOKU DO PONTO DE VISTA  
DO JOGADOR: ESTRATÉGIAS  
PARA SOLUÇÃO AUTOMÁTICA E  
SEMI AUTOMÁTICA**

Rio de Janeiro  
2015

FUNDAÇÃO GETÚLIO VARGAS  
ESCOLA DE MATEMÁTICA APLICADA  
CURSO DE GRADUAÇÃO EM MATEMÁTICA APLICADA

JULIANA HUTHER ALBERNAZ CRESPO

**SUDOKU DO PONTO DE VISTA  
DO JOGADOR: ESTRATÉGIAS  
PARA SOLUÇÃO AUTOMÁTICA E  
SEMI AUTOMÁTICA**

“Declaro ser o único autor da presente monografia, requisito parcial para a obtenção do grau de Bacharel em Matemática Aplicada e ressalto que não recorri a qualquer forma de colaboração ou auxílio de terceiros para realizá-lo a não ser nos casos e para os fins autorizados pelo professor orientador”.

Orientador: Asla Sá

Co-orientador: Paulo Cezar Carvalho

Rio de Janeiro  
2015

JULIANA HUTHER ALBERNAZ CRESPO

**Sudoku do ponto de vista do jogador: estratégias para  
solução automática e semi automática**

“Projeto de Monografia apresentado à Escola de  
Matemática Aplicada – FGV/EMAp como requi-  
sito parcial para obtenção do grau de Bacharel em  
Matemática Aplicada”.

Aprovado em: Rio de Janeiro, \_\_\_\_ de \_\_\_\_\_ de \_\_\_\_\_.

---

Profa. Dra. Asla Sá (Professora Orientadora)

---

Prof. Dr. Paulo Cezar Carvalho (Professor Tutor)

Rio de Janeiro  
2015

# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>1</b>
1.1	O Jogo	1
1.2	O ponto de vista do jogador	2
1.3	Metodologia	4
<b>2</b>	<b>PESQUISA EXPLORATÓRIA DA LITERATURA</b>	<b>5</b>
<b>3</b>	<b>DESENVOLVIMENTO E RESULTADOS ESPERADOS</b>	<b>6</b>
<b>3.1</b>	<b>Sudoku como grafo</b>	<b>6</b>
3.1.1	O problema	6
3.1.2	Modelagem	6
3.1.3	Resolução em Python	8
3.1.4	Visualização	10
<b>3.2</b>	<b>Sudoku como polinômio</b>	<b>11</b>
3.2.1	O problema	11
3.2.2	Modelagem	11
3.2.3	Resolução em Python	12
3.2.4	Visualização	13
<b>3.3</b>	<b>Sudoku como problema de otimização</b>	<b>13</b>
3.3.1	O problema	13
3.3.2	Modelagem	13
3.3.3	Resolução em AMPL	17
3.3.4	Visualização	19
<b>4</b>	<b>CONCLUSÃO</b>	<b>21</b>
<b>5</b>	<b>REFERÊNCIA</b>	<b>22</b>

# 1 INTRODUÇÃO

O tema proposto para o TCC se desenvolve em torno do conhecido quebra-cabeça Sudoku. O trabalho foi direcionado em desenvolvimento de estratégias para a solução desse jogo, como consequência foram estudados diferentes domínios da matemática, como por exemplo, álgebra, otimização e teoria de grafos.

## 1.1 O Jogo

O jogo do Sudoku tem como objetivo preencher cada uma das posições vazias de uma matriz de tamanho 9x9 com números de 1 a 9. Contudo, cada um desses números só pode aparecer apenas uma vez em cada linha, coluna e submatriz 3x3 contida na matriz principal. Além disso, cada posição só pode conter um número. Essas regras de unicidade estão resumidas no nome do próprio jogo, que consiste em uma abreviação japonesa para a frase “suuji wa dokudhin ni kagiru”, que significa que “os dígitos devem permanecer únicos”.

Entende-se por posições vazias o fato de algumas posições já conterem números que consiste em pistas iniciais. Essas pistas ajudam na dedução dos números que deverão ser preenchidos.

Em geral, o jogo é classificado em quatro níveis de dificuldade: fácil, médio, difícil e desafiador. Um jogo difícil não é obrigatoriamente aquele que tem menos pista, a dificuldade de um jogo está mais relacionada ao posicionamento das pistas do que a quantidade delas.

## 1.2 O ponto de vista do jogador

Quando se depara com o quebra-cabeça Sudoku, surgem dois tipos de pensamentos, o de quem irá construir o jogo, ou seja, criar novos desafios, posicionar alguns números de forma a criar dificuldades diferentes e de forma que a solução seja possível. E o de quem vai resolver o jogo, ou seja, descobrir o lugar de cada números e encontrar a solução do mesmo. Esse ultimo será o ponto de vista adotado neste artigo.

Resolvendo o Sudoku no papel, existem algumas técnicas que são adotadas para a sua resolução, que serão listadas abaixo.

Assim que se escolhe qual Sudoku irá resolver, primeiramente deve-se percorrer cada número de 1 a 9, passando por todas as linhas, colunas e submatrizes riscando os lugares onde o número não poderá ser preenchido. Por exemplo, olhando para o número 6 na figura 1.1, as linhas na horizontal representam as linhas que já tem o número 6, então não poderá ter outro, o mesmo acontece com as linhas verticais, as cruzes representam os elementos que não podem conter o o número 6, pois ele já se encontra na submatriz. Com isso podemos ver se alguma, linha, coluna ou submatriz ficou apenas com uma possibilidade para colocar o elemento.

Esse procedimento deverá ser repetido até a hora que passar por todos os números de 1 a 9 e não conseguir preencher mais nenhuma casinha.

Também deve-se olhar se tem alguma linha, coluna ou submatriz na qual só esteja faltando um elemento.

Outra técnica utilizada é marcar na ponta do quadradinho todos os possíveis candidatos, como mostra a figura 1.2.

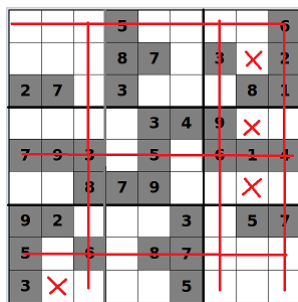


Figura 1.1: técnica de resolução 1 Sudoku

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

Figura 1.2: técnica de resolução 2 Sudoku

A partir daí podemos preencher os quadradinhos que só tem um candidato. E apagar esses candidatos nas linhas, colunas e submatrizes correspondentes. Feito isso, teremos que observar se aparecem exatamente duas possibilidades com os números iguais na mesma linha, coluna ou submatriz (exemplo: numa linha qualquer, o primeiro e o quarto quadrado tenham como possibilidades apenas os números 3 e 7), isso permite que esses dois números sejam removidos das possibilidades de candidatos para os outros quadradinhos da linha. Veja o exemplo na figura 1.3.

O mesmo pode acontecer para 3 posições, quando existem 3 números iguais nas 3 posições.

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

Figura 1.3: técnica de resolução 3 Sudoku

### 1.3 Metodologia

Estudaremos livros que abordem o tema, nos aprofundaremos nele, a partir daí utilizaremos ferramentas computacionais tais como o AMPL e o Python para implementar as diferentes abordagens propostas de forma automática, ou seja, fornecemos para o programa os dados iniciais do Sudoku e o mesmo irá retornar o resultado explícito na tela, já com todas as informações, ou então de forma semi automática, na qual o programa irá fornecendo resultados aos poucos, mostrando o passo a passo e a cada jogada fornecer somente um dos resultados, não o resultado final. Por fim iremos usar o Processing para uma visualização mais dinâmica e lúdica.



## 2 PESQUISA EXPLORATÓRIA DA LITERATURA

A presente proposta tem como principal referencia o livro “Taking Sudoku Seriously: The Math Behind the World’s Most Popular Pencil Puzzle”[1], que aborda estratégias e aponta para teorias úteis no estudo do problema, explorando as conexões entre Sudoku, teoria dos grafos e polinômios. Alguns tópicos do livro serão aprofundados baseando-se em textos adicionais [2][3] e implementações serão feitas para ilustrar o funcionamento dos algoritmos. Em particular, o problema de coloração de grafos surgiu como opção de abordagem para resolver uma dada instância de Sudoku. Adicionalmente estudaremos uma modelagem do jogo Sudoku como um problema de programação inteira.

## 3 DESENVOLVIMENTO E RESULTADOS ESPERADOS

No corpo do trabalho iremos focar em 3 formas de resolução, sendo elas como grafo, como polinômio e como um problema de otimização.

### 3.1 Sudoku como grafo

A ideia básica consiste em transformar o tabuleiro do Sudoku em um problema de coloração de grafos e, desta forma, tentar colorir o grafo gerado utilizando apenas nove cores.

#### 3.1.1 O problema

O problema de coloração de grafo, se baseia em definir cores (valores, no caso do Sudoku) aos vértices de um grafo, de forma que vértices adjacentes, ou seja, que tem uma aresta em comum, não possam ter a mesma cor.

#### 3.1.2 Modelagem

Como vimos anteriormente, o Sudoku possui 3 diferentes tipos de “vizinhos”, sendo eles os vértices que estão na sua mesma linha, coluna ou submatriz. Sendo assim, a melhor forma de solucionar o problema seria dividir esse grande puzzle em 3 grafos interligados, contendo cada um destes grafos os vizinhos referentes ao seu

estilo. Como mostrado na figura 3.1.

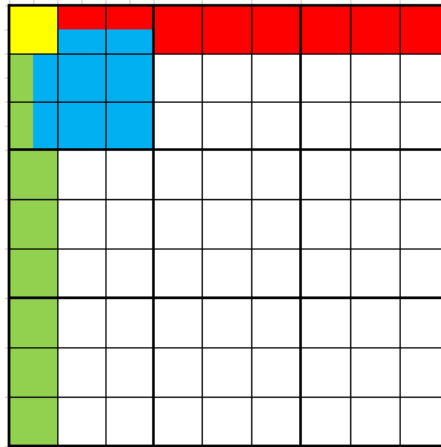


Figura 3.1: Vizinhos

Onde, o amarelo é o vértice analisado, as arestas que o ligam aos vértices vermelhos pertencem ao primeiro grafo, as que ligam aos vértices verdes pertencem ao segundo grafo e as que ligam aos vértices azuis pertencem ao terceiro grafo. Podemos observar que alguns vértices tem duas cores, o que representa que o mesmo se encontra nos dois grafo referentes as cores.

Com as arestas e os vértices criados, resta descobrir aonde colorir cada cor. Para isso são analisados as possíveis cores de cada vértice, que são todas as que não estão sendo utilizadas por nem um de seus vizinhos. Porém, apenas uma delas será a cor referente a esse vértice e os métodos de descobrir qual é são os seguintes:

- primeiro: Analisar se algum dos vértices só tem uma possível cor, nesse caso, não há duvida, essa será a cor referente a ele.
- Segundo: Para cada um dos três grafos, olhar separadamente, se existe um determinado vértice com uma possível cor, na qual nenhum de seus vizinho a tem, ou seja, essa cor só pode ser utilizada nesse vértice.

- Terceiro: E se existirem dois vértices que só podem ser pintados por exatamente as mesmas duas cores? Exato, nenhum outro vértice pode ter essas cores.

Os testes realizados mostram que o algoritmo utilizado não resolve todos os jogos. Por enquanto o código está limitado a resolver um grupo específico de sudoku's. Até o momento acredita-se que com a abordagem de coloração de grafos não é possível resolver todos os tipos de jogos independente da complexidade, como é feito com outros métodos (Ex.: programação linear).

Com isso, um forte interesse em relação a abordagem de escala de dificuldade dos Sudoku's foi despertado, porém aprofundar no tema fugiria do objetivo do trabalho, podendo vir a ser, o mesmo, um futuro tema de projeto.

### 3.1.3 Resolução em Python

O Algoritmo abaixo mostra a implementação do modelo matemático, no qual foi desenvolvido em Python. Os três grafos mencionados anteriormente estão representados como  $G_a$ ,  $G_b$  e  $G_c$  e também foram programados em python para achar as arestas de cada um dele. Os vértices ( $V$ ), foram digitados na mão e passados para o programa. Para o funcionamento do mesmo são utilizadas algumas funções complementares que calculam os possíveis números a serem completados em um vértice (função "faltando"), quando só falta um número é usada a função "faltando" para encontrar e uma outra função, que retira de todos os vizinhos de um vértice o número encontrado.

Listing 3.1: Algoritmo Sudoku

```

1 def color_sudoku(Ga,Gb,Gc,V):
2     G = []
3     G.append(Ga)
4     G.append(Gb)
5     G.append(Gc)
6     len_g = len(Ga)+len(Gb)+len(Gc)
7     distinct_colors = {}
8     for g in G:
9         for node in g.nodes_iter():
10             distinct_colors[node] = set([0])
11     for y in V:
12         if V[y] != 0:
13             distinct_colors[y] = set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
14     for n in V:
15         distinct_colors = vizinhos(V,n,Ga,Gb,Gc,distinct_colors)
16     q = len_g
17     while q > 0:
18         for i in distinct_colors:
19             if len(distinct_colors[i]) == 9:
20                 V[i] = faltando(distinct_colors[i])
21                 distinct_colors[i] = set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
22                 distinct_colors = vizinhos(V,i,Ga,Gb,Gc,distinct_colors)
23         for g in G:
24             for i in distinct_colors:
25                 x = possiveis(distinct_colors[i])
26                 for neighbour in g.neighbors_iter(i):
27                     x = x - possiveis(distinct_colors[neighbour])
28                 if len(x) == 1:
29                     V[i] = x.pop()
30                     distinct_colors[i] = set([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
31                     distinct_colors = vizinhos(V,i,Ga,Gb,Gc,distinct_colors)
32         for g in G:
33             for i in distinct_colors:
34                 for neighbour1 in g.neighbors(i):
35                     if len(distinct_colors[i]) == 8 and distinct_colors[i] == distinct_colors[neighbour1]:
36                         t = possiveis(distinct_colors[neighbour1])
37                         for neighbour2 in g.neighbors(i):
38                             if neighbour2 != neighbour1:
39                                 distinct_colors[neighbour2] = distinct_colors[neighbour2].union(t)
40         q = q - 1
41     return distinct_colors, V

```

### 3.1.4 Visualização

Após a conclusão do algoritmo, foi desenvolvida uma visualização, em Processing, para que a coloração como grafo ficasse mais clara e evidente. O resultado pode ser visto na Figura 3.2, onde observa-se a existência de 9 cores diferentes no lugar dos números.

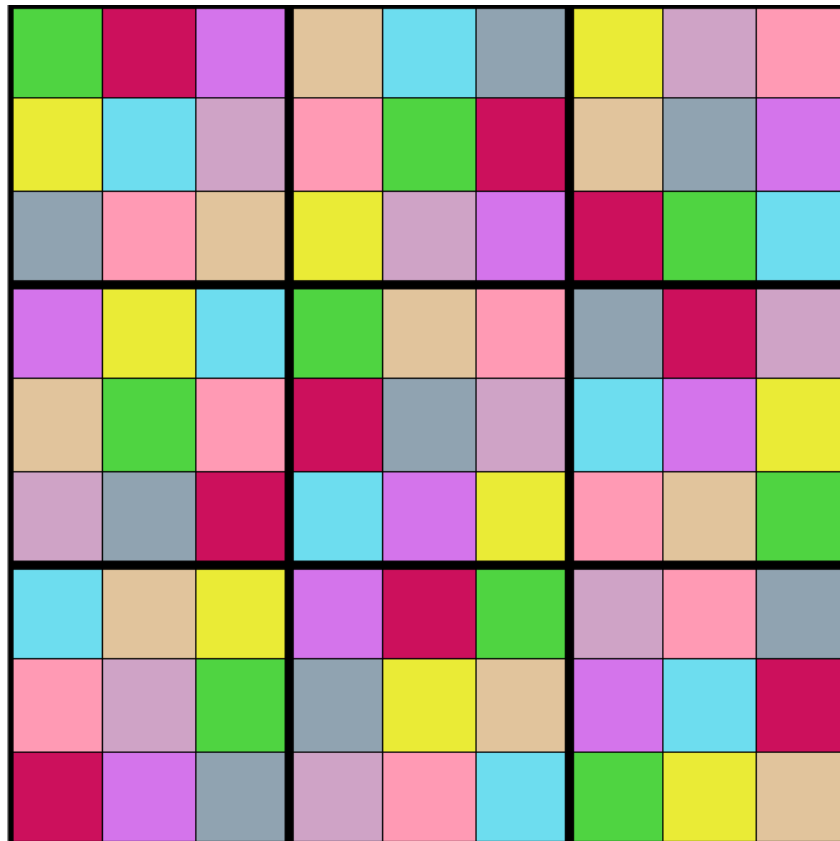


Figura 3.2: Resolução Sudoku

## 3.2 Sudoku como polinômio

A ideia básica consiste em transformar o tabuleiro do Sudoku em um problema de polinômios e, desta forma, tentar encontrar polinômios de primeira ordem.

### 3.2.1 O problema

O problema de polinômios se baseia na junção e fatoração dos mesmos, para simplificar e otimizar as funções.

### 3.2.2 Modelagem

Um Sudoku como polinômio completo estaria na forma  $(x_i - a_j)$  com  $i$  variando de 1 a 81 e  $j$  variando de 1 a 9.

Como nem todas as casa iniciam completas, será definido um polinômio  $F$  que representará casa uma das casa, onde o mesmo será zero quando a casa estiver completa.

$$F(x) = \prod_{j=1}^9 (x - j)$$

Para que nenhum vertice tenha a mesma cor que o seu vizinho, será necessario criar uma função  $G$  que restrinja isso.

$$G(x_i, x_k) = \frac{F(x_i) - F(x_k)}{x_i - x_k}$$

Feito isso, encontradas todas as funções, será aplicada a base de Grobner, que juntando e misturando as funções, encontrará a única solução que fará com que todos os polinômios sejam iguais a zero.

### 3.2.3 Resolução em Python

Onde X é o Sudoku inicial, com algumas casas preenchidas com os seus respectivos números e as outras com zeros. E A é uma lista com o conjunto de arestas, cada vértice ligado aos seus respectivos vizinhos.

A implementação da base de Grobner é bem complexa de implementar, por isso foi usado um pacote do Python, o sympy.

Listing 3.2: Algoritmo Sudoku

```

1 F = []
2 for i in X:
3     fi = (i-1)*(i-2)*(i-3)*(i-4)*(i-5)*(i-6)*(i-7)*(i-8)*(i-9)
4     F.append(fi)
5
6 G = []
7 a = 0
8 for i in x:
9     if i > 0:
10        G.append(X[a]-i)
11        a = a+1
12 for i in A:
13     gi = div((F[i[0]-1]-F[i[1]-1]), (X[i[0]-1]-X[i[1]-1]))[0]
14     G.append(gi)
15
16 t = groebner(G,X,method='buchberger')
```



### 3.2.4 Visualização

Nesta visualização podemos observar cada um dos 81 vértices e seus respectivos polinômios, com os valores do vértice, encontrados pela base de Grobner.

```
( x1 - 7) ( x2 - 1) ( x3 - 6) ( x4 - 3) ( x5 - 5) ( x6 - 8) ( x7 - 4) ( x8 - 2) ( x9 - 9)
(x10 - 8) (x11 - 4) (x12 - 9) (x13 - 2) (x14 - 6) (x15 - 7) (x16 - 3) (x17 - 1) (x18 - 5)
(x19 - 3) (x20 - 5) (x21 - 2) (x22 - 4) (x23 - 1) (x24 - 9) (x25 - 6) (x26 - 8) (x27 - 7)
(x28 - 5) (x29 - 6) (x30 - 7) (x31 - 9) (x32 - 4) (x33 - 1) (x34 - 8) (x35 - 3) (x36 - 2)
(x37 - 4) (x38 - 8) (x39 - 1) (x40 - 5) (x41 - 3) (x42 - 2) (x43 - 7) (x44 - 9) (x45 - 6)
(x46 - 9) (x47 - 2) (x48 - 3) (x49 - 8) (x50 - 7) (x51 - 6) (x52 - 5) (x53 - 4) (x54 - 1)
(x55 - 2) (x56 - 9) (x57 - 4) (x58 - 6) (x59 - 8) (x60 - 5) (x61 - 1) (x62 - 7) (x63 - 3)
(x64 - 1) (x65 - 3) (x66 - 5) (x67 - 7) (x68 - 2) (x69 - 4) (x70 - 9) (x71 - 6) (x72 - 8)
(x73 - 6) (x74 - 7) (x75 - 8) (x76 - 1) (x77 - 9) (x78 - 3) (x79 - 2) (x80 - 5) (x81 - 4)
```

Figura 3.3: Resolução Sudoku Polinômios

## 3.3 Sudoku como problema de otimização

A ideia básica consiste em transformar o tabuleiro do Sudoku em um problema de otimização e, desta forma, tentar encontrar o resultado ótimo.

### 3.3.1 O problema

Embora o Sudoku não seja um problema de otimização, ele pode ser modelado em programação linear inteira 0 – 1 (binária), se a função objetivo for considerada como um vetor de coeficientes nulos e as regras do Sudoku como restrições.

### 3.3.2 Modelagem

Vamos definir um Jogo de Sudoku como uma matriz quadrada  $N * N$  com  $N$  submatrizes  $n * n$ , tal que  $n = \sqrt{N}$ . Cada posição dessa matriz é representada por

um conjunto de variáveis binárias  $x_{ijk}$ , que tem como objetivo decidir se a posição  $(i, j)$  da matriz será preenchida ou não pelo valor  $k$  ( $k = 1, \dots, N$ ).

Sendo assim:

$$x_{ijk} = \begin{cases} 1, & \text{se a posição } (i, j) \text{ possui o valor } k, \forall i, j, k = 1, \dots, N \\ 0, & \text{caso contrário} \end{cases}$$

O objetivo do jogo é encontrar uma solução que satisfaça as restrições; não existe uma função objetivo a ser minimizada ou maximizada. Portanto, esse problema é modelado da seguinte forma:

$$\begin{aligned} \sum_{j=1}^N x_{ijk} &= 1 & \forall i, k = 1, \dots, N \\ \sum_{i=1}^N x_{ijk} &= 1 & \forall j, k = 1, \dots, N \\ \sum_{k=1}^N x_{ijk} &= 1 & \forall i, j = 1, \dots, N \\ \sum_{i=np-(n-1)}^{np} \sum_{j=nq-(n-1)}^{nq} x_{ijk} &= 1 & \forall p, q = 1, \dots, N; \forall k = 1, \dots, N \\ x_{ijk} &= 1 & \forall i, j = 1, \dots, N; \forall k = S_{ij}, S_{ij} \neq 0 \\ x_{ijk} &= 0 & \forall i, j = 1, \dots, N; \forall k = S_{ij}, S_{ij} = 0 \\ x_{ijk} &\in (0, 1) & \forall i, j, k = 1, \dots, N; \end{aligned}$$

Uma linha  $i$  ( $i = 1, \dots, N$ ) é formada por todos os termos ordenados  $(i, j, k)$  em que  $j, k = 1, \dots, N$ . Para cada linha  $i$ , cada número  $k$  aparece uma e somente uma vez. Por exemplo, para  $i = 1$  e  $N = 9$ , temos que:

$$x_{111} + x_{121} + \dots + x_{191} = 1 \quad k = 1$$

$$x_{112} + x_{122} + \dots + x_{192} = 1 \quad k = 2$$

$$\dots \quad \dots$$

$$x_{119} + x_{129} + \dots + x_{199} = 1 \quad k = 9$$

Uma coluna  $j$  ( $j = 1, \dots, N$ ) é formada por todos os termos ordenados  $(i, j, k)$  em que  $i, k = 1, \dots, N$ . Para cada coluna  $j$ , cada número  $k$  aparece uma e somente uma vez. Por exemplo, para  $j = 1$  e  $N = 9$ , temos que:

$$x_{111} + x_{211} + \dots + x_{911} = 1 \quad k = 1$$

$$x_{112} + x_{212} + \dots + x_{912} = 1 \quad k = 2$$

$$\dots \quad \dots$$

$$x_{119} + x_{219} + \dots + x_{919} = 1 \quad k = 9$$

Uma submatriz  $n \times n$  contida na matriz principal  $N \times N$ , ocupa a posição  $(p, q)$  em que  $p, q = 1, \dots, n$ . A Figura 3.4 mostra as posições na matriz de um Sudoku clássico ( $N = 9$  e, portanto,  $n = 3$ ). Por exemplo, a submatriz (1,1) começa em  $i = 1$  até  $i = 3$  e vai de  $j = 1$  até  $j = 3$ ; a submatriz (2,3) começa em  $i = 4$  até  $i = 6$  e vai de  $j = 7$  até  $j = 9$ .

Quando  $p = 1$ , queremos manipular as posições da matriz principal em que  $i = 1, 2, 3$ , quando  $p = 2$ , estamos com  $i = 4, 5, 6$  e quando  $p = 3$ , estamos com  $i = 7, 8, 9$ . Ou seja, estamos sempre começando com os índices da linha avaliados em 1, 4 e 7 e terminando com eles em 3, 6 e 9, respectivamente. Uma forma de

11	12	13	14	15	16	17	18	19
21	22	23	24	25	26	27	28	29
31	32	33	34	35	36	37	38	39
41	42	43	44	45	46	47	48	49
51	52	53	54	55	56	57	58	59
61	62	63	64	65	66	67	68	69
71	72	73	74	75	76	77	78	79
81	82	83	84	85	86	87	88	89
91	92	93	94	95	96	97	98	99

Figura 3.4: Posições na Matriz do Sudoku

relacionarmos os valores de  $p$  com os de  $i$  é da seguinte forma:  $i = 3p - (3 - 1), \dots, 3p$  e de forma mais geral, temos que  $i = np - (n - 1), \dots, np$ .

O mesmo raciocínio é feito para as colunas que abrangem cada submatriz. Sendo assim, uma forma de relacionarmos os valores de  $q$  com  $j$  é da seguinte forma:  $j = nq - (n - 1), \dots, nq$ .

Para cada uma das  $N$  submatrizes  $n * n$ , cada elemento  $k = 1, \dots, N$  deve aparecer uma e somente uma vez. Por exemplo, para  $p = 1, q = 1, N = 9$ , temos que:

$$x_{111} + x_{121} + x_{131} + \dots + x_{311} + x_{321} + x_{331} = 1 \quad k = 1$$

$$x_{112} + x_{122} + x_{132} + \dots + x_{312} + x_{322} + x_{332} = 1 \quad k = 2$$

$$\dots \quad \dots$$

$$x_{119} + x_{129} + x_{139} + \dots + x_{319} + x_{329} + x_{339} = 1 \quad k = 9$$

Uma posição  $(i, j)$  é formada por todos os termos ordenados  $(i, j, k)$  em que  $k = 1, \dots, N$ . Para cada posição  $(i, j)$ , deve ser preenchida com um e somente um número  $k$ . Por exemplo, para  $i = 1, j = 1$  e  $N = 9$ , temos que:

$$x_{111} + x_{112} + \dots + x_{119} = 1$$

Vamos definir uma matriz  $S$  também  $N * N$  em que cada posição  $S_{ij}$  é representada por uma variável binária, que assume valor 1 se  $S_{ij} \neq 0$  e o valor 0 caso contrário. Essa matriz  $S$  armazena as posições do Sudoku que estão inicialmente preenchidas. No Sudoku clássico em que  $N = 9$ , possuímos  $9^3 = 729$  variáveis. Contudo, cada pista dada reduz em 9 o número de variáveis, pois constitui uma restrição que define um valor  $k$  para a posição  $(i, j)$ . Por exemplo, se  $S_{13} = 5$  conclui-se que a posição  $(1,3)$  do Sudoku já está preenchida pelo  $k = 5$ , sendo assim, adicionamos a restrição de que  $x_{135} = 1$  e que  $x_{13k} = 0, \forall k = 1, \dots, 9, k \neq 5$ .

### 3.3.3 Resolução em AMPL

O Algoritmo abaixo mostra a implementação do modelo matemático apresentado em AMPL e o solver utilizado para a resolução foi o CPLEX. Para resolvermos um Sudoku qualquer, preenchemos em "Dados do Modelo" o parâmetro  $S$  (uma matriz  $N * N$ ) com as posições iniciais já fornecidas (as pistas).

Listing 3.3: Algoritmo Sudoku

```

1 #-----SUDOKU-----
2
3 #-----Tamanho do Modelo
4
5 param n := 3; # 2 para um jogo 4x4 ou 3 para um jogo 9x9
6 param N := n * n;
7
8 # Define uma matriz P

```

```

9 | param P{1..N, 1..N}, integer, default 0, >= 0, <= N;
10 |
11 | #-----Variáveis do Modelo
12 |
13 | set Rows := {1..N};
14 | set Cols := {1..N};
15 | set Vals := {1..N};
16 |
17 | var x{Rows, Cols, Vals} binary;
18 |
19 | #-----Restrições do Modelo
20 |
21 | # Cada número de 1,...,N aparece apenas uma vez em cada linha
22 | subject to Limit_Row{i in Rows, k in Vals}:
23 |     sum{j in Cols} x[i,j,k] = 1;
24 |
25 | # Cada número de 1,...,N aparece apenas uma vez em cada coluna
26 | subject to Limit_Cols{j in Cols, k in Vals}:
27 |     sum{i in Rows} x[i,j,k] = 1;
28 |
29 | # Cada posição (i,j) deve ter um e somente um número de 1,...,N
30 | subject to Sum_Vals{i in Rows, j in Cols}:
31 |     sum{k in Vals} x[i,j,k] = 1;
32 |
33 | # Cada número de 1,...,N aparece apenas uma vez em cada N submatriz n*n
34 | subject to Limit_Matrix{k in Vals, p in 1..n, q in 1..n}:
35 |     sum{j in (n*q - (n-1))..(n*q), i in (n*p - (n-1))..(n*p)} x[i,j,k]
36 |         = 1;
37 |
38 | # Lê uma matriz P com as posições da matriz que já estão preenchidas
39 | subject to Pos_ij{i in Rows, j in Cols: P[i,j] > 0}:
40 |     x[i,j,P[i,j]] = 1;
41 |
42 | #-----Dados do Modelo
43 | data;
44 | param P:
45 | 1 2 3 4 5 6 7 8 9 :=
46 | 1 0 0 5 0 0 0 0 0 3
47 | 2 0 0 0 0 4 6 0 0 0
48 | 3 0 0 7 0 0 0 0 0 2
49 | 4 0 1 0 0 0 3 0 6 9
50 | 5 0 4 0 6 0 9 0 5 0
51 | 6 9 8 0 2 0 0 0 7 0
52 |
53 | 7 2 0 0 0 0 0 9 0 0
54 | 8 0 0 0 8 1 0 0 0 0
55 | 9 6 0 0 0 0 0 4 0 0 ;

```

```

56
57 option solver cplex;

```

### 3.3.4 Visualização

Como a variável do modelo é tridimensional, a visualização da resposta em AMPL de  $x_{ijk}$  não é muito intuitiva. Então, foi elaborada alguns comandos que transformam a resposta em uma tabela do formato do Sudoku, o que pode ser visto no algoritmo abaixo.

Listing 3.4: Visualização em AMPL do Sudoku

```

1  #-----Visualizao do Modelo
2
3  for {i in Rows}{
4    for {j in Cols}{
5      for {k in Vals}{
6        if (x[i,j,k] == 1) then printf "%3i", k;
7      };
8      if ((j mod n) == 0) then printf " | ";
9    };
10   printf "\n";
11   if ((i mod n) == 0) then {
12     for {j in 1..n}{
13       for {k in 1..(n-1)}{ printf "—" };
14       if (j < n) then
15         printf "—+";
16       else
17         printf "—+\n";
18     };
19   };
20 };

```

A partir dessa tabela obtida com a execução do AMPL e da matriz  $S$  passada na restrição foi feito um código em Processing que exibe um resultado como o da Figura 3.5. EM vermelho temos as pistas contidas em  $S$  e os demais números são aqueles obtidos executando o modelo proposto.

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

Figura 3.5: Resolução Sudoku



## 4 CONCLUSÃO

Neste trabalho estudei o puzzle Sudoku, utilizando três abordagens diferentes. na primeira delas, o Sudoku foi abordado como um problema em grafo, foi desenvolvido a sua modelagem e uma resolução em Python. No segundo, o Sudoku foi modelado como polinômio, foram criados os polinômios a partir da base de Grobner que instanciavam o mesmo, e por fim foi utilizado um pacote Python para encontrar a solução do polinômio. Por fim, o Sudoku foi abordado como um problema de otimização inteira, foi desenvolvida a sua modelagem e a resolução da mesma em Ampl.

Observamos algumas limitações quando utilizamos a abordagem do Sudoku como problema em grafo, quando os jogos tem um nível de dificuldade maior, a solução chega numa situação de ambiguidade intrínseca do problema, e o algoritmo determinístico de coloração em grafos proposto não resolve tais ambiguidades. Optamos por não apresentar heurística que resolvesse essa ambiguidade o que fugiria do escopo deste trabalho, optando assim por não aprofundar o estudo dessa abordagem. No Sudoku como polinômio o maior desafio foi no desenvolvimento do algoritmo, o que fez com que optássemos pela utilização do pacote do Python pronto. No Sudoku como problema de otimização devido a LIMITAÇÃO do ampl VERSAO de estudantes, os puzzles de Sudoku de maior tamanho não são resolvidos.

Como trabalhos futuros, surgiram algumas opções, como, um estudo aprofundado da base de Grobner, desenvolver um algoritmo que ,mostre a dificuldade dos puzzles, ou ate mesmo partir para o ponto de vista de quem cria o jogo.

## 5 REFERÊNCIA

- [1] jason rosenhouse laura *hemathbehindtheworldsmostpopularpencilpuzzle—oxforduniversitypressusa*(2012)
- [2] O jogo Sudoku, pré-coloração estendida em hipergrafos e algoritmos de enumeração implícita
- [3] Tecnicas de coloração de grafos aplicadas á resolução de quebra-cabeça do tipo sudoku
- [4] [http://www.longwood.edu/assets/mathematics/Team2975\\_problemB.pdf](http://www.longwood.edu/assets/mathematics/Team2975_problemB.pdf)
- [5] <http://arxiv.org/pdf/1210.2584.pdf>
- [6] <http://www.technologyreview.com/view/428729/mathematics-of-sudoku-leads-to-richter-scale-of-puzzle-hardness/>
- [7] <http://planetsudoku.com/how-to/sudoku-solving-techniques.html>
- [8] <http://www.fi.muni.cz/~xpelanek/publications/sudoku-arxiv.pdf>
- [9] <https://idus.us.es/xmlui/bitstream/handle/11441/23605/sudoku-casc-pdflatex-ams.pdf?sequence=1>
- [10] [http://www.scholarpedia.org/article/Groebner\\_basis](http://www.scholarpedia.org/article/Groebner_basis)