# O jogo de lógica Sudoku: modelagem teórica, NP-Completude e estratégias algorítmicas exatas e heurísticas

**Kevin Takano**[1], **Rosiane de Freitas**[1], **Vinícius Gusmão de Sá**[2]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
69077-000 – Manaus-AM, Brasil
[2]Instituto de Matemática – Universidade Federal do Rio de Janeiro (UFRJ)
21941-59 – Rio de Janeiro-RJ, Brasil

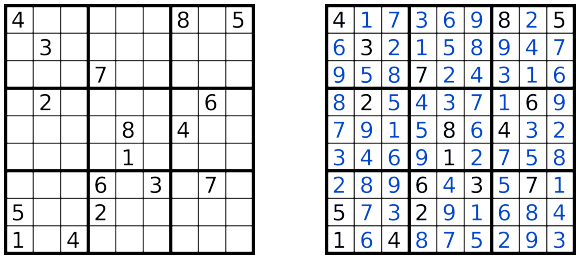
{kkt,rosiane}@icomp.ufam.edu.br, vigusmao@dcc.ufrj.br

***Resumo.*** *Neste artigo, são abordados os aspectos matemático-computacionais do jogo de lógica Sudoku. O Sudoku clássico consiste em um grid* $9 \times 9$*, parcialmente preenchido, onde se deve encontrar o local certo para serem inseridos números de* $1$ *a* $9$*, de modo que nenhum se repita na mesma linha, na mesma coluna ou no mesmo sub-quadrado em que esteja. O Sudoku é um problema computacional NP-completo, apresentado neste estudo sob diversos aspectos - unique SAT, cobertura exata, pré-coloração estendida e quadrado latino. Foi realizado um estudo e implementação das principais técnicas e algoritmos para resolução exata da literatura, envolvendo enumeração implícita como backtracking com podas, manipulação de bits, dancing links, propagação por restrições e programação inteira. Com base neste estudo, dois métodos são propostos, sendo um método exato baseado em propagação por restrições, com resultados de melhor desempenho do que os similares da literatura, bem como um método metaheurístico GRASP para Sudokus genéricos* $n \times n$*. Também, algoritmos para o caso polinomial do grid vazio são propostos, que serviram de base para algoritmos de geração automática de instâncias de nível fácil, médio e difícil. Tal estudo e resultados fazem parte de um projeto de pesquisa do Programa Institucional de Bolsas de Iniciação Científica - PIBIC (UFAM e CAPES).*

***Abstract.*** *In this article, the computational mathematics aspects of the Sudoku puzzle, a logic game, are covered. The classic Sudoku consists of a grid* $9 \times 9$*, partially filled, where to find the right place to be inserted numbers from* $1$ *to* $9$*, so that no repeat on the same line in the same column or in the same sub-square where it is. The Sudoku is an NP-complete computational problem, presented in this study in many ways - as unique SAT, exact cover, precoloring extension, and Latin square. It conducted a study and implementation of the main techniques and algorithms for exact resolution from the literature, involving implicit enumeration as backtracking with pruning, bit manipulation, dancing links, constraint propagation, and integer programming. Based on this study, two methods are proposed, an exact method based on constraint propagation, with better performance than similar one from the literature, as well as a GRASP metaheuristic method for generic Sudokus* $n \times n$*. Also, polynomial algorithms for the case of the empty grid are proposed, which served as the basis for automatic generation algorithms easy level instances, medium and hard. Such study and results are part of a research project of the Institutional Program for Scientific Initiation Scholarships - PIBIC (UFAM and CAPES).*

## 1. Introdução

Sudoku é um passatempo lógico muito conhecido por estar presente nas bancas de jornais e diversos sites pela Internet. O jogo de Sudoku clássico é tradicionalmente encontrado em tabuleiros de tamanho $9 \times 9$, divididos em blocos $m \times m$, tal que $n = m^2$. O objetivo do jogo consiste em preencher todo o tabuleiro já com dígitos pré-preenchidos com valores inteiros entre $1$ e $n$ e divididos em blocos $3 \times 3$. A solução final não deve violar as seguintes regras: cada linha, coluna ou bloco deve conter inteiros entre $1$ e $9$ exatamente uma vez. Na Figura 1 é apresentada uma instância do Sudoku e sua respectiva solução, para um tabuleiro padrão. Entretanto, o jogo existe também em outras versões, podendo ser generalizado para grids de tamanho $n \times n$, como $16 \times 16$.




(a) (b)

**Figura 1. Exemplo de jogo clássico do Sudoku de tamanho $9 \times 9$. a) Instância com 17 dígitos pré-preenchidos. b) Solução para esta instância.**

Existe uma variedade de denominações para os conceitos de elementos do Sudoku. Neste trabalho, define-se a seguinte terminologia. Um **grid** é um tabuleiro $n \times n$ do jogo, sendo um **dígito** todo valor entre $1$ e $n$ que deve ser inserido ou já estar presente no mesmo. Uma **célula** refere-se a cada um dos $n^2$ pequenos quadrados do tabuleiro. Um **bloco** consiste em cada um dos sub-quadrados de tamanho $3 \times 3$ do jogo. Um **candidato** é todo dígito que, possivelmente, poderá ser inserido em uma célula qualquer. Uma **unidade** é uma referência geral para linha, coluna e bloco. E, no caso, uma solução ou **solução válida** é um grid totalmente preenchido que respeita as regras do jogo. Um **grid incompleto** é uma instância válida do Sudoku, com algumas células somente preenchidas. já um **grid completo** é uma solução válida do Sudoku, com todas as células preenchidas.

O restante deste artigo está organizado como segue. Na Seção 2, os aspectos teóricos, modelo em grafos e em programação matemática são apresentados. Na Seção 3, a prova de NP-completude baseando-se em alguns problemas clássicos NP-completos é discutida. Na Seção 4, são apresentadas as principais técnicas e algoritmos da literatura para o Sudoku implementadas, incluindo os métodos exato e metaheurístico GRASP propostos. Na Seção 5, um caso polinomial do Sudoku para o processo de geração automática de instâncias é tratado. Por fim, na Seção 6 são feitas as considerações finais sobre o trabalho e os resultados obtidos.

## 2. Aspectos e modelos teóricos

No jogo Sudoku, deseja-se determinar uma configuração válida para um dado tabuleiro segundo suas regras estabelecidas, ou seja, o retorno da solução é um grid com todas as células sem repetições em cada unidade. A generalização do Sudoku para um tamanho $n = m^2$, como já mencionado, serve de suporte para o estudo de sua com-

plexidade computacional. Em sua versão de decisão, o Sudoku é um problema NP-completo [Yato and Seta 2002], portanto, os algoritmos conhecidos para o problema possuem tempo de execução exponencial em relação a $n$.

Existe uma correlação forte entre o Sudoku e o Quadrado Latino (do inglês, *Latin Square*), proposto pelo matemático suíço Leonhard Euler do século $XVII$. O Sudoku é um caso particular do Quadrado Latino, com a restrição adicional de que cada bloco não pode conter dígitos repetidos. Tal característica torna o número de Quadrados Latinos um limite superior para o número de soluções do Sudoku. Sabe-se que há $12$ Quadrados Latinos de tamanho $3$, $575$ de tamanho $4$ e $5.524.751.496.156.\ 892.842.531.225.600$ de tamanho $9$ [Sloane 2004a], o que demonstra o crescimento exponencial do espaço de soluções do problema. Entretanto, eliminando-se as soluções simétricas, o número de Quadrados Latinos de ordem $9$ passa a ser $377.597.570.964.258.816$, o que apesar de ser apenas $7 \times 10^{-9}\%$ das soluções totais, continua sendo um número muito grande de soluções [Sloane 2004b]. Para o Sudoku clássico $9\times9$, é estimado um número de soluções válidas de $6.670.903.752.021.072.936.960$ [McNair 2005]. Porém, com as eliminações de soluções equivalentes, o número final diminui para $5.472.730.538$ [Sloane 2005]. Tal como para o Sudoku, gerar uma solução válida para um Quadrado Latino parcialmente preenchido é um problema NP-Completo [Colbourn 1984]. Em ambos os casos, se o grid estiver vazio, o problema está em P, possuindo algoritmo de tempo polinomial para a geração de uma solução, como apresentado nas seções seguintes.

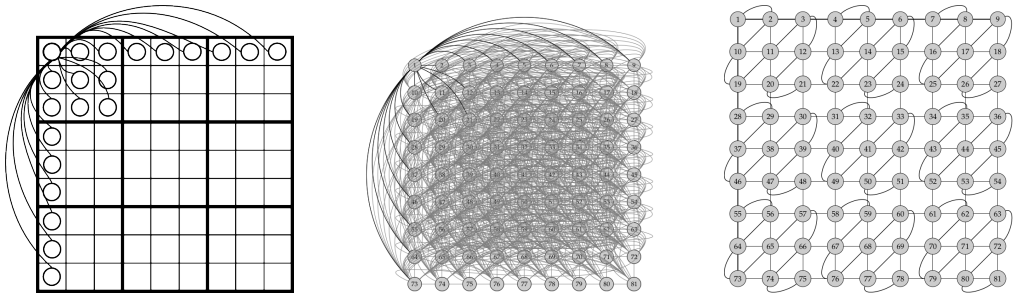
O número máximo de dígitos preenchidos para que um tabuleiro inicial do Sudoku não tenha solução única é $77$, segundo a literatura. Com mais elementos que isso, obviamente a solução será única. Revertendo o raciocínio, o número mínimo conhecido para dígitos preenchidos, é $17$. Uma **conjectura** famosa é de que bastam $16$ dígitos preenchidos para garantir solução única, mas, não se conhece nenhuma instância do Sudoku até hoje que satisfaça tal condição.

Para um melhor entendimento de suas propriedades teóricas e algoritmos desenvolvidos, o Sudoku pode ser modelado de diversas formas. Nas seções seguintes, serão apresentados: o modelo em grafos simples e em hipergrafos; o modelo em programação linear inteira; e, o modelo em programação por restrições.

### 2.1. Modelagem em Grafo Simples e Hipergrafo

O Sudoku pode ser modelado em grafos considerando cada célula do tabuleiro como um vértice e adicionando-se uma aresta para cada célula que estiver na mesma linha, coluna ou bloco. Sendo assim, cada unidade (ou seja, linha, coluna ou bloco) do tabuleiro formará uma clique. Para o caso $3 \times 3$ (isto é, com $9$ linhas e $9$ colunas, tem-se $81$ vértices conectados a $20$ outros $(8 + 8 + 4)$, o que totaliza $1620/2 = 810$ arestas, já que cada aresta relaciona exatamente $2$ vértices. A Figura 2 exemplifica este conceito. O Sudoku pode ainda ser modelado com o uso de **hipergrafos**. Um hipergrafo é uma generalização de um grafo, tal que cada aresta pode relacionar um subconjunto de vértices maior ou igual a $2$. Define-se um hipergrafo $H = (V, E)$ como sendo um conjunto $V$ de elementos unitários chamado de vértices e um conjunto $E$ de subconjuntos não-vazios chamados de **hiperarestas**. Com o modelo de hiperarestas aplicados ao Sudoku, tem-se apenas $9$ hiperarestas para cada unidade do tabuleiro, fazendo com que o número seja reduzido para $27$ hiperarestas no total e não $810$ como no modelo em grafos simples. A Figura 2

exemplifica tal modelagem, e os métodos propostos fazem uso de tal modelo para garantir maior eficiência na manipulação das restrições do problema.



(a) (b) (c)

**Figura 2. a) Representação da relação de adjacências para o vértice que representa a primeira célula de um grid do Sudoku. b) Modelo em grafos simples. c) Modelo em hipergrafo.**

## 2.2. Modelagem em Programação Linear Inteira do Sudoku

Embora o Sudoku não seja um problema de otimização, pode ser modelado em Programação Linear Inteira $0-1$ (binária) se a função objetivo for considerada como um vetor de coeficientes nulo e cada regra do Sudoku como restrições da modelagem [Barllet et al. 2008]. A variável de decisão tri-indexada $x_{ijk}$ será $1$ se um elemento (i,j) contiver um dígito $k$ do tabuleiro do Sudoku e $0$ caso contrário. Assim, tem-se o seguinte modelo PLI:

Minimizar $0^T x$

Sujeito a:

$$\sum_{i=1}^{x_{ijk}} = 1, j, k \in \{1, \ldots, n\} \quad (1)$$

$$\sum_{j=1}^{n} x_{ijk} = 1, i, k = \{1, \ldots, n\} \quad (2)$$

$$\sum_{j=mq-m+1}^{mq} \sum_{i=mp-m+1}^{mp} x_{ijk} = 1, k \in \{1, \ldots, n\}, p, q \in \{1, \ldots, m\} \quad (3)$$

$$\sum_{k=1}^{n} x_{ijk} = 1, i, j \in \{1, \ldots, n\} \quad (4)$$

$$x_{ijk} = 1 \forall (i, j, k) \in T \quad (5)$$

$$x_{ijk} \in \{0, 1\} \quad (6)$$

Em (1), (2) e (3) são definidos as restrições para cada regra, respetivamente: só deve haver um dígito $k$ em cada linha, um dígito $k$ em cada coluna, e um dígito $k$ em cada bloco. A restrição (4) força que todos as células do Sudoku sejam preenchidas. A restrição (5) indica que os dígitos já pré-preenchidos sejam setados para $1$. $T$ é o conjunto de todos os dígitos já pré-preenchidos. A última restrição, é a restrição de integralidade das variáveis de decisão do modelo. O modelo em PLI foi implementado usando o método *branch-and-cut* da ferramenta CPLEX, para realização dos testes e obtenção dos resultados para comparação.

## 2.3. Modelagem em Programação por Restrições

O Sudoku também pode ser modelado como um problema de satisfação de restrições através da combinação de restrições do tipo *All Different*, considerando cada regra como

uma restrição para um conjunto de variáveis de decisão [Hoeve 2001]. Seja o conjunto de variáveis de decisão $X$ para o problema do Sudoku:

$$X = \begin{matrix} \{x_{11}, & x_{12}, & \cdots, & x_{1n}, \\ x_{21}, & x_{22}, & \cdots, & x_{2n} \\ x_{31}, & x_{32}, & \cdots, & x_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1}, & x_{n2}, & \cdots, & x_{nn}, \} \end{matrix}$$

Onde cada variável bi-indexada $x_{ij}$ tem seu respectivo domínio $D_{ij} = \{1, \cdots, n\}$. O Conjunto de Restrições $C = \{C_1, C_2, C_3, C_4\}$ será formado por restrições *All Different*, que força com que toda variável de decisão de um dado grupo tenha que assumir um valor diferente dos valores pertencentes às outras variáveis de decisão:

$$C_1 = Alldif(x_{i1}, x_{i2}, \cdots, x_{in}) \qquad \forall i \in \{1, \cdots, n\} \qquad (1)$$

$$C_2 = Alldif(x_{1j}, x_{2j}, \cdots, x_{nj}) \qquad \forall j \in \{1, \cdots, n\} \qquad (2)$$

$$\begin{aligned} C_3 = Alldif \\ ( \\ & x_{mp-1+1,mq-m+1}, x_{mp-m+1,mq-m+2}, \cdots, x_{mp-m+1,mq-m+m}, \\ & x_{mp-1+2,mq-m+1}, x_{mp-m+2,mq-m+2}, \cdots, x_{mp-m+2,mq-m+m}, \\ & \quad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ & x_{mp-1+m,mq-m+1}, x_{mp-m+m,mq-m+2}, \cdots, x_{mp-m+m,mq-m+m} \\ ), \forall p \in \{1, \cdots, m\}, \quad \forall q \in \{1, \cdots, m\} \end{aligned} \qquad (3)$$

$$C_4 = (x_{ij} == k)\, \forall k \in T \qquad (4)$$

Onde $k$ é todo dígito presente no conjunto $T$ de dígitos pré-preenchidos. A restrição $(1)$, $(2)$ e $(3)$ define que toda linha, coluna e bloco, devem ser diferente, respectivamente. A restrição $(4)$ força todas as variáveis terem seu valor fixo pelos dígitos que já são pré-preenchidos. Este modelo também foi implementado usando o módulo CP-Optimizer, para programação por restrições, do CPLEX, para realização de testes e geração de resultados para comparação.

## 3. Sobre a prova da NP-Completude do Sudoku

Nesta seção, será apresentada uma prova da NP-Completude do Sudoku, provando-se, entãp que este pertence à classe NP e ao mesmo tempo à classe NP-Difícil. Para isto, se considera a versão de decisão do Sudoku, que significa verificar se existe ou não uma solução válida para uma dada instância do Sudoku.

Para a prova de que está em NP, foi elaborado um algoritmo de reconhecimento de uma solução do Sudoku em tempo poliniomial $O(n^2)$ usando estrutura de manipulação de bits, que garante consulta aos dígitos em tempo constante, $O(1)$. Deste modo, mostra-se que problema de decisão do Sudoku está em NP, pois pode ser criado um algoritmo de reconhecimento de sua solução em tempo polinomial (algoritmo omitido em função do espaço limitado).

Para a prova da NP-Dificuldade, foi realizada ma redução polinomial do Quadrado Latino para o Sudoku. Outros problemas computacionais clássicos, já provados serem NP-Completos, também foram estudados e usados para o estudo e entendimento do Sudoku e a realização da redução polinomial ao Sudoku (prova da NP-dificuldade), sendo: Unique SAT, Pré-Coloração Estendida e Cobertura Exata (omitidos por limitação de espaço).

Para a prova de que o Sudoku é NP-difícil [Yato and Seta 2002] fez-se a redução polinomial do Quadrado Latino, que é NP-Completo [Colbourn 1984], para o Sudoku. Assim, dado um Quadrado Latino parcialmente preenchido de ordem $m$, para uma instância

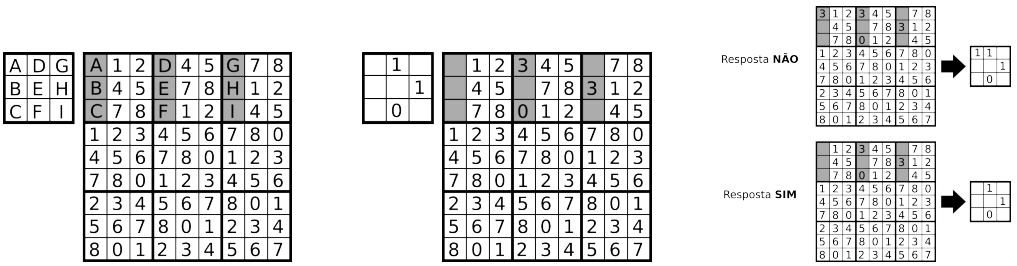
do Sudoku de ordem $n$, tal que $n = m^2$. Seja $S(i, j)$ o dígito de uma célula do Sudoku da linha $i$ e coluna $j$ e $L(p, q)$, a célula do Quadrado Latino da linha $p$ e coluna $q$. Considera-se que o Sudoku seja preenchido por dígitos entre $\{0, .., n\}$ e o Quadrado Latino preenchido por $\{0, ..., m\}$. Desta forma, o grid pré-preenchido do Sudoku pode ser criado seguindo esta equação:

$$S(i,j) = \begin{cases} L(i, j/m) \cdot n & ((i,j) \in B, L(i, j/m) \neq \varnothing) \\ \varnothing & ((i,j) \in B, L(i, j/m) = \varnothing) \\ ((i \bmod n)n + \lfloor i/m \rfloor + j) mod\, n & caso\ contrario \end{cases}$$

*onde,*

$$B = \{(i,j) | \lfloor i/m \rfloor = 0\, e\, (j \bmod n) = 0\}$$

B é o conjunto de todas as posições do grid do Sudoku que serão correlacionadas com o Quadrado Latino. O símbolo $\varnothing$ representa toda célula não preeenchida. A Figura 3 representa o resultado de uma transformação realizada pelo algoritmo implementado, e relação de validade entre as instâncias do Sudoku e Quadrado Latino: se o Sudoku tiver solução válida, então, o quadrado latino também tem; e, caso contrário, se o Sudoku não tiver solução válida, o Quadrado Latino também não terá.




(a) Relação de posições. (b) Transformação de instâncias. (c) Casos SIM e NÃO.

**Figura 3. (a) relaciona as posições do grid do Quadrado Latino com o do Sudoku para o tamanho clássico $m = 3$. (b) é o resultado da transformação de uma instância do Quadrado Latino, para o Sudoku. (c) representa graficamente relação de validade para instâncias SIM e NÃO.**

## 4. Estratégias Algorítmicas de Resolução

Nesta sessão serão apresentadas as técnicas e os algoritmos da literatura para o Sudoku, que foram estudadas e implementadas, algumas com adaptações. A seguir são resumidas as principais técnicas utilizadas em tais métodos:

- ***Minimum Remaining Values*** que defende que é mais viável selecionar variáveis com menos possibilidades de valores. 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*** que propõe o término finaliza a busca caso alguma variável ainda não testada não tenha valores dentro de seu domínio. 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 representãção das estruturas de dados do Sudoku e manipulação das unidades (linha, coluna e bloco) com consulta em tempo constante. 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 pré-preenchidos do tabuleiro do Sudoku em vetores. Cada dígito do tabuleiro é representado por um algarismo de um número presente no vetor. A verificação e a remoção de dígitos é feita através dos vetores criados.

- ***Constraint Propagation*** é uma técnica que remove valores de um domínio de variáveis da qual não irão participar de nenhuma solução. 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 [Norvig 2006].
- ***Dancing Links*** é uma técnica proposta por Donald Knuth, 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 não-determinística, que encontra todas as soluções para realizar a cobertura exata do problema.

Foram implementados os seguintes métodos exatos: algoritmo de Backtracking (puro, sem podas); algoritmo de Backtracking implementado pelo Skiena utilizando técnica de Foward Checking e Heurística de Minimum Remaining Values; algoritmo de backtracking immplementado com técnica de manipulação de bits; algoritmo proposto por Norvig baseado em propagação de restrições (dito de melhor desempenho até então para o Sudoku); algoritmo otimizado do Norvig (primeira versão, com acréscimo de dois novos casos de propagação); implementação do modelo em PLI usando o método branch-and-cut da ferramenta CPLEX; e, implementação do modelo em CP (*Constraint Programming*) do módulo CP-Optimizer da ferramenta CPLEX. Além destes, uma mehloria do algoritmo de propagação do Norvig está sendo proposto (que usa a técnica de manipulação de bits, *heaps* e considera dois novos casos de propagação.

Agora, o Sudoku também pode ser resolvido através de técnicas metaheurísticas se abordado como um problema de otimização. A resolução por heurísticas significa autorizar a construção de grids infactíveis, atribuindo-se para cada grid um custo em que o algoritmo buscará minimizar [Lewis 2007]. A função de custo é calculada de modo que uma solução factível tenha custo zero, quanto maior o custo, maior a distância entre a solução construída e a solução ótima. A geração de vizinhança é feita se escolhendo cada dígito não-fixo de cada bloco e trocando-se dois elementos de uma mesma posição. Cada troca é um novo vizinho, em que o processo é repetido até que todos os pares de todos os blocos já tenham sido trocados. Essa troca é simples e viável, uma vez que ela não viole as regras do Sudoku.

### 4.1. Métodos Propostos

Neste trabalho estão sendo propostos dois novos métodos para o Sudoku: um método exato baseado em propagação por restrições, uso de heaps e manipulação de bits; e, uma metaherística GRASP para resolver o Sudoku com técnica de manipulação de bits.

Quanto ao **algoritmo exato**, otimiza a **propagação por restrições** proposta por um dos melhores algoritmos encontrados na literatura [Norvig 2006]. Neste caso, os candidatos são representados através de vetores de bits, utilizando manipulação de bits para que faça inserção, remoção e verificação de candidatos de forma muito mais rápida. Além

disso, também deseja-se otimizar a Heurística Minimum Remaining Values já implementada, para que a busca da célula menos restritiva seja em $O(1)$ e não $O(n^2)$. Se possível, ainda pretende-se utilizar outras técnicas de otimização de busca, como procurar o candidato mais restritivo dentro de uma unidade. Obteve-se melhor desempenho do que o algoritmo original.

Quanto ao **método metaheurístico**, propõe-se o **GRASP**, baseando em uma metaheurística encontrada na literatura. A representação e custo de cálculo da função objetivo foi otimizada através das técnicas de manipulação de bits, em que a verificação de um dígito será feito em $O(1)$ e não em $O(n)$ e $O(n^2)$, fazendo que o acesso de um valor em uma célula seja $O(1)$. A construção de soluções é feita através da inserção aleatória de candidatos do jogo que estão presentes numa *Lista Restrita de Candidatos* (LRC). Quando um dígito deve ser inserido, cria-se uma lista de candidatos $C$ em que contém todos os candidatos de inserção ordenados pelo custo. E assim, seleciona-se os $\alpha\%$ elementos da lista C (parte gulosa) para inserir na LRC. Após isso, seleciona-se um dígito qualquer da LRC (parte aleatória) para a nova instância. O custo de um candidato qualquer é calculado pelo número de repetições do candidato numa linha ou coluna em que será inserido. A busca local é realizada utilizando-se o algoritmo *Hill Climbing*. O algoritmo Hill Climbing irá selecionar a vizinhança de melhor custo e irá retorná-la. O custo de uma solução é calculado pela soma da quantidade de dígitos que faltam ser inseridos em todo o tabuleiro. E assim a metaheurística guardando sempre a melhor solução até alguma condição de parada for verdadeira. A condição de parada pode ser definida como um total de iterações ou quando se encontra a solução é ótima (custo da melhor solução é zero). Quanto maior o coeficiente alfa mais aleatório será, e quanto menor mais guloso.

## 5. Geração de Instâncias

Para se gerar uma instância válida (grid incompleto) do Sudoku é necessário, primeiramente, se gerar uma solução válida (grid completo) e, assim, se remover os dígitos adequadamente, sendo o desafio, se gerar instâncias de forma que só haja uma única solução. Neste trabalho, um caso polinomial do Sudoku foi considerado para a construção de uma solução viável para o Sudoku e que, resultando em dois algoritmos determinísticos de tempo polinimial, A partir daí, um processo iteratuvo para modificação do início da constução determinística da solução foi aplicado de tal forma a garantir a geração de grandes conjuntos de soluções. Se com muito poucas células preenchidas ou muito poucos não preenchidas, a instância é considerada de nível fácil. Senão, existe a conjectura de que com $16$ células pré-preenchidas há solução única, mas, se consegue gerar com $17$, e isto foi considerado para a construção de instâncias de nível difícil (e dependendo da distribuição no grid e da quantidade de célulcas pré-preenchidas, gera-se as de nível intermediário). Este processo pode ser refinado e isto faz parte dos trabalhos em andamento. Os dois algoritmos polinomiais, de complexidade $O(n^3)$, para a criação de soluções do Sudoku são apresentados a seguir.

### 5.1. Caso Polinomial de Sudoku

O Sudoku pode ser resolvido em tempo polinomial quando todo seu grid está vazio, ou seja, quando a instância não contiver nenhuma célula pré-preenchida. Os algoritmos a seguir resolvem o problema para diferentes tipos de soluções e de diferentes tamanhos. Cada um deles insere os dígitos seguindo a configuração de uma Lista Circular que contém valores de $\{1, \ldots, n\}$ distintos.

- O primeiro algoritmo preenche bloco por bloco o tabuleiro, começando no primeiro bloco superior a esquerda, e descendo. Após terminar o preenchimento de um bloco, o preenchimento irá iniciar através do próximo elemento da lista circular e assim por diante.
- O segundo algoritmo tem o preenchimento feito em um mesmo padrão para cada $m$ linhas. E assim, para cada conjunto das $m$ linhas, o inicio preenchimento de cada linha deste conjunto se dará na $i$-ésima coluna, tal que $i$ cresce em num fator de $m$.

## 6. Contribuições do Trabalho

Os pontos a destacar como contribuições deste projeto de pesquisa, tanto aos envolvidos quanto à literatura sobre o jogo do Sudoku, foram:

- realização de uma ampla revisão de literatura sobre definições, história e aspectos teóricos do Sudoku, bem como sobre as técnicas e algoritmos aplicados;
- estudo aprofundado e aplicação de conceitos e técnicas de Otimização Combinatória, Complexidade Computacional (NP-completude, e análise e projeto de algoritmos) e Teoria dos Grafos;
- modelagem teórica do Sudoku, envolvendo a modelagem em grafos simples e em hipergrafos, como também em programação matemática (formulações em programação linear inteira e programação por restrições);
- estudo e detalhamento da prova da NP-Completude do Sudoku, envolvendo o algoritmo de reconhecimento de uma solução (prova NP) e a redução polinomial de problemas clássicos (Quadrado Latino (pré-preenchido), Unique-Sat, Pré-Coloração Estendida e Cobertura Exata) para o Sudoku;
- elaboração e implementação do algoritmo de reconhecimento de uma solução (prova NP) e do algoritmo de transformação de instâncias e solução entre Quadrado Latino e Sudoku;
- estudo e implementação (com algumas adaptações) dos principais algoritmos de resolução do Sudoku para instâncias retiradas da literatura [Cook 2006];
- proposta e implementação de dois novos métodos: lgoritmo exato de propagação por restrição, com uso de heaps e manipulação de bits; metaheurística GRASP com manipulação de bits;
- estudo sobre casos polinomiais do Sudoku, principalmente o caso do grid vazio, onde foram propostos dois algoritmos determinísticos de tempo $O(n^2)$;
- estudo sobre geração de instâncias do Sudoku e garantia de solução única, com a elaboração de um algoritmo para a geração automática de instâncias partindo-se da geração determinística de uma solução inicial (algoritmo polinomial para o caso do grid vazio), em um processo iterativo de alteração do padrão inicial;
- análise comparativa entre os algoritmos implementados da literatura, incluindo os métodos propostos, com testes massivos envolvendo instâncias da literatura e as propostas, de níveis fácil, intermerdiário e difícil.

## 7. Considerações Finais

Este trabalho é resultado de um projeto de pesquisa do Programa Institucional de Bolsas de Iniciação Científica, do período de $2014/2015$, abordando os aspectos matemático-computacionais do jogo do Sudoku.

Toda a teoria estudada e algoritmos desenvolvidos foi de grande valia e pretende-se aplicar este conhecimento e ferramental para novas descobertas tanto para o Sudoku quanto para outros jogos de lógica em grids, seguindo linhas de pesquisa das áreas de Otimização Combinatória, Complexidade e Teoria dos Grafos.

## Referências


Barllet, A. C., Chartier, T. P., and Langville, A. N. (2008). An integer programming model for the sudoku problem. *Journal of Online Mathematics and its Applications*.

Colbourn, C. (1984). The complexity of completing partial latin square. *Elsevier Science Plubishers*.

Cook, P. (2006). Solving every sudoku puzzles with python. `http://www2.warwick.ac.uk/fac/sci/moac/people/students/peter_cock/python/sudoku`. Acessado em 26 de abril de 2015.

Hoeve, W.-J. (2001). The all different constraint: A survey. *In 6th Annual Workshop of the ERCIM Working Group on Constraints*.

Lewis, R. (2007). Metaheuristcs can solve sudoku puzzles. *Jornal of Heuristics*.

McNair, R. (2005). Number of (completed) sudokus (or sudokus) of size $n^2 \times n^2$. `http://oeis.org/A107739`. Acessado em 26 de abril de 2015.

Norvig, P. (2006). Solving every sudoku puzzle. `http://norvig.com/sudoku.html`. Acessado em 26 de abril de 2015.

Skiena, S. S. (2008). *Algorithm Design Manual*. Springer Publishing.

Sloane, N. J. A. (2004a). Number of latin squares of order n; or labeled quasigroups. `http://oeis.org/A002860`. Acessado em 26 de abril de 2015.

Sloane, N. J. A. (2004b). Number of reduced latin squares of order n; also number of labeled loops (quasigroups with an identity element) with a fixed identity element. `http://oeis.org/A000315`. Acessado em 26 de abril de 2015.

Sloane, N. J. A. (2005). Number of inequivalent (completed) $n^2 \times n^2$ sudokus (or sudokus). `http://oeis.org/A109741`. Acessado em 26 de abril de 2015.

Yato and Seta (2002). Complexity and completeness of finding another solution and its application to puzzles. *In Procceedings of The National Meeting of the Information Society of Japan (IPSJ)*.