O jogo de lógica Sudoku: modelagem teórica, NP-completude, algoritmos e heurísticas*

Kevin Takano[1], Rosiane de Freitas[1], Vinícius G. Pereira 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)
21941-590 – Rio de Janeiro-RJ, Brasil

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

Resumo. *Abordamos os aspectos matemáticos e computacionais do jogo de lógica Sudoku. O Sudoku é um problema NP-completo estreitamente relacionado a problemas como satisfatibilidade única, cobertura exata, pré-coloração estendida de vértices e quadrado latino. Estudamos e implementamos as principais técnicas e algoritmos exatos da literatura: enumeração implícita (backtracking com podas), manipulação de bits, dancing links, programação por restrições e programação inteira. Nossa principal contribuição está em dois novos métodos: um exato, baseado em propagação de restrições e com melhor desempenho do que os similares encontrados na literatura, e um meta-heurístico GRASP para o Sudoku genérico $n \times n$. São também propostos algoritmos polinomiais para o caso particular do grid inicialmente vazio, com aplicação na geração de instâncias do Sudoku em níveis variados de dificuldade.*

Abstract. *We tackle the mathematical and computational aspects of the logic puzzle Sudoku. The Sudoku is an NP-complete problem which bears a strong connection with problems such as unique satisfiability, exact cover, vertex precoloring extension, and Latin square. We have studied and implemented the main techniques and exact algorithms from the literature, to wit: implicit enumeration (backtracking with pruning), bit manipulation, dancing links, constraint programming, and integer programming. Our main contribution comes in the form of two new methods: an exact method, based on constraint propagation, with better performance than similar ones from the literature; and a GRASP metaheuristic method for the generic $n \times n$ Sudoku. We also propose polynomial-time algorithms for the particular case of the initially empty grid, which suit well the generation of Sudoku instances in varying levels of difficulty.*

1. Introdução

O Sudoku é um passatempo lógico bastante conhecido, construído em um tabuleiro quadrado $n \times n$, com $n = k^2$, para algum inteiro positivo k. Isto é, tratam-se de n^2 **células** dispostas em n **linhas**, n **colunas** e n **blocos** $k \times k$. O objetivo do jogo é preencher todo o tabuleiro — que já vem com algumas casas previamente preenchidas — com inteiros de 1 a n. A solução deve atender as seguintes condições: cada casa deve conter um único

*Apoiado pelo Programa Institucional de Bolsas de Iniciação Científica - PIBIC (UFAM e CAPES).

número, e cada linha, coluna e bloco do tabuleiro deve conter, sem repetição, todos os inteiros de 1 a n. A Figura 1 apresenta uma instância do Sudoku no tamanho clássico 9×9 e uma possível solução.

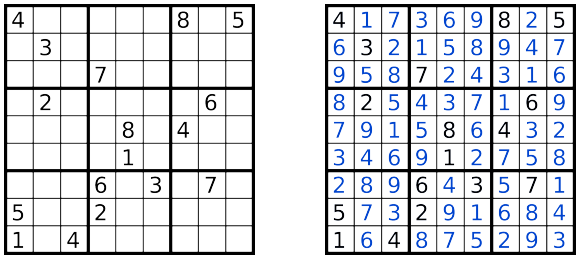


Figura 1. Exemplo de jogo clássico de Sudoku. (a) Instância com 17 casas inicialmente preenchidas. (b) Solução.

Além dos conceitos de célula, linha, coluna e bloco, utilizamos neste trabalho a seguinte terminologia. Uma **grid** é um tabuleiro $n \times n$ do jogo. Um **valor** é um inteiro entre 1 e n. Um **candidato** é todo valor que, em dado momento, pode ser inserido em uma determinada célula (não violando as condições do problema). Uma **unidade** é uma referência geral para linha, coluna ou bloco. Uma **grid incompleta** é uma solução parcial do Sudoku, com algumas células preenchidas, mas não todas. Uma **grid completa** é uma solução final do Sudoku, com todas as células devidamente preenchidas.

Este artigo está organizado como se segue. Na Seção 2, são apresentados os aspectos teóricos, modelo em grafos e em programação matemática são apresentados. A Seção 3 aborda a NP-completude do Sudoku. Na Seção 4, são apresentadas as principais técnicas e algoritmos da literatura para o Sudoku, incluindo os novos métodos (um exato e um meta-heurístico GRASP) propostos. Na Seção 5, tratamos um caso polinomial do Sudoku, que é aquele em que inicialmente nenhuma célula se encontra preenchida. Nossa técnica permite gerar automaticamente instâncias do jogo. Por fim, na Seção 6, são feitas as considerações finais sobre o trabalho e os resultados obtidos.

2. Aspectos e modelos teóricos

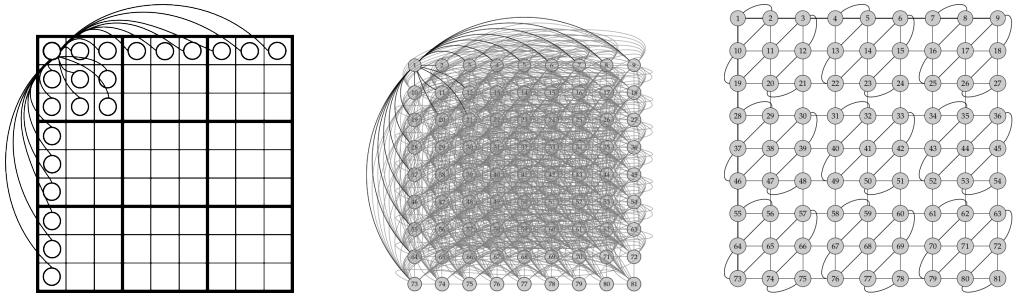
No jogo Sudoku, deseja-se determinar uma configuração final válida para um dado tabuleiro, isto é, deseja-se estender a grid inicial de forma a se obter uma grid com todas as células preenchidas e sem valores repetidos em uma unidade. O Sudoku é um problema NP-difícil [Yato and Seta 2002], e os algoritmos conhecidos possuem tempo de execução exponencial em n. Existe uma correlação forte entre o Sudoku e o Quadrado Latino proposto por Euler no século XVII. De fato, o Sudoku é um caso particular do Quadrado Latino, com a restrição adicional de que cada bloco não pode conter valores 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]. Entretanto, eliminando-se 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 enorme de soluções [Sloane 2004b]. Para o Sudoku clássico 9×9, é estimado um número de soluções válidas de $6.670.903.752.021.072.936.960$

[McNair 2005]. Com a eliminação de soluções simétricas, o número final diminui para $5.472.730.538$ [Sloane 2005]. Assim como no Sudoku, completar um Quadrado Latino parcialmente preenchido é problema NP-Completo [Colbourn 1984]. Também nos dois casos, se a grid inicial estiver vazia, o problema é polinomial, como veremos adiante. O número máximo de dígitos preenchidos para que um tabuleiro inicial do Sudoku não tenha solução única é 77, segundo a literatura. Por outro lado, o número mínimo de valores previamente preenchidos que garante solução única é 17. Uma **conjectura** famosa é de que bastam 16 dígitos preenchidos para garantir solução única, mas não se conhece instância do Sudoku que satisfaça tal condição.

Nas seções seguintes, serão apresentados modelos do Sudoku utilizando grafos simples e hipergrafos, um modelo em programação linear inteira, e um modelo de programação por restrições.

2.1. Modelagens via Grafos Simples e Hipergrafos

O Sudoku pode ser modelado como um grafo considerando cada célula do tabuleiro como um vértice e adicionando uma aresta entre cada par de células que estiverem em uma mesma unidade. Sendo assim, cada unidade do tabuleiro formará uma clique. Para o caso tradicional 9×9, temos 81 vértices, cada qual conectado a 20 outros, totalizando 810 arestas. As Figuras 2a e 2b ilustram o modelo. O Sudoku pode ser modelado também como um hipergrafo. Um hipergrafo $H = (V, E)$ consiste de um conjunto V de elementos unitários (os vértices) e um conjunto E de subconjuntos não-vazios de V (as hiperarestas) indicando elementos vizinhos. No modelo de hipergrafo para o Sudoku 9×9, temos apenas uma hiperaresta para cada unidade do tabuleiro, em um total de 27 hiperarestas. A Figura 2c exemplifica tal modelagem, que é utilizada pelos métodos propostos para 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 via Programação Linear Inteira

Embora o Sudoku não seja um problema de otimização, ele pode ser modelado em programação linear inteira (PLI) $0 - 1$ (binária) se a função objetivo for considerada como um vetor de coeficientes nulos e as regras do Sudoku como restrições[Barllet et al. 2008]. A variável de decisão tri-indexada x_{ijk} será 1 se a célula (i,j) do tabuleiro contiver o valor k e 0 caso contrário. Assim, tem-se o seguinte modelo de PLI:

Minimizar $0^T x$.

Sujeito a:

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

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

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

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

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

$$x_{ijk} \in \{0, 1\}. \tag{6}$$

Em (1), (2) e (3) são definidas as regras do jogo, respectivamente: deve haver exatamente um valor k em cada linha, em cada coluna e em cada bloco, para todo k de 1 a n. A restrição (4) força que todas as células do Sudoku sejam preenchidas. A restrição (5) faz com que valores já preenchidos atribuam valor 1 às respectivas variáveis tri-indexadas (o conjunto T representa as células já preenchidas). 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 de testes e obtenção de resultados computacionais.

2.3. Modelagem via Programação por Restrições

O Sudoku também pode ser modelado como um problema de programação por restrições (PR) 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 de PR do Sudoku:

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

Cada variável bi-indexada x_{ij} tem seu respectivo domínio $D_{ij} = \{1, \ldots, n\}$. O conjunto de restrições $C = \{C_1, C_2, C_3, C_4\}$ será formado basicamente por restrições *all different* (alldif), 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:

$$C_1 = alldif(x_{i1}, x_{i2}, \ldots, x_{in}) \qquad \forall i \in \{1, \ldots, n\}, \tag{1}$$

$$C_2 = alldif(x_{1j}, x_{2j}, \ldots, x_{nj}) \qquad \forall j \in \{1, \ldots, n\}, \tag{2}$$

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

$$C_4 = (x_{ij} = k), \forall (i, j, k) \in T. \tag{4}$$

Novamente, o conjunto T é o conjunto de triplas (i, j, k) indicando células (i, j) já preenchidas com valores k. As restrições (1), (2) e (3) definem, respectivamente, que os valores em cada linha, coluna e bloco devem ser diferentes. A restrição (4) força que todas as variáveis tenham seu valor fixado naqueles que já foram previamente preenchidos.

Este modelo também foi implementado utilizando o módulo CP-Optimizer para PR do CPLEX.

3. NP-completude

Nesta seção, será apresentada uma prova da NP-completude do Sudoku. Para isto, consideramos a versão de decisão do problema, que significa verificar se existe ou não uma solução válida para uma dada instância do Sudoku.

Não é difícil ver que o problema está em NP. Como exercício, elaboramos um algoritmo de reconhecimento de uma solução do Sudoku em tempo $O(n^2)$ — isto é, linear no número de células do tabuleiro — usando estrutura de manipulação de bits. Essa estrutura garante consulta aos dígitos em tempo constante $O(1)$. Omitimos o algoritmo em razão do espaço limitado.

Para a prova da NP-dificuldade, foi realizada uma redução polinomial do Quadrado Latino (NP-completo, vide [Colbourn 1984]) para o Sudoku. Outros problemas NP-completos também podem ser reduzidos polinomialmente ao Sudoku. Dentre eles, citamos Unique SAT, Pré-Coloração Estendida e Cobertura Exata, também omitidos neste texto por limitação de espaço.

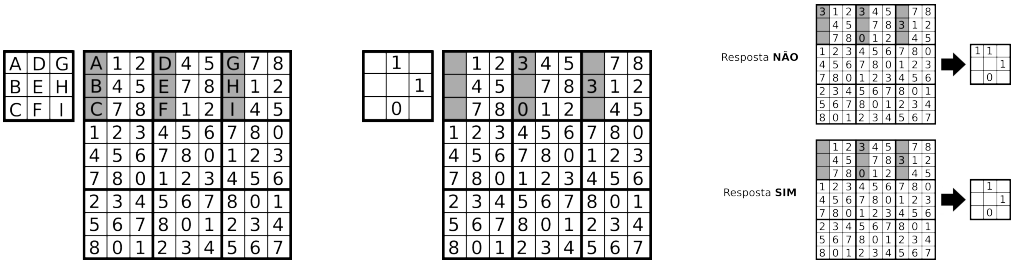
Dado um Quadrado Latino $k \times k$ parcialmente preenchido, obtemos uma instância do Sudoku $n \times n$, para $n = k^2$, da seguinte maneira. Seja $s(i, j)$ o valor atribuído à célula do Sudoku da linha i e coluna j, e seja $\ell(p, q)$ o valor contido na célula do Quadrado Latino da linha p e coluna q.

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

onde

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

O conjunto B contém todas as células do Sudoku que serão correlacionadas com o Quadrado Latino. O símbolo $\varnothing$ representa uma célula não preenchida. A Figura 3 representa o resultado de uma transformação e a relação de validade entre as instâncias do Sudoku e do Quadrado Latino: se o Sudoku tiver solução válida, então o Quadrado Latino também terá, e vice-versa.



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

Figura 3. (a) Células correspondentes no Quadrado Latino e no Sudoku. (b) Resultado da transformação de uma instância do Quadrado Latino para uma do Sudoku. (c) Correspondência entre instâncias SIM e NÃO dos dois problemas.

4. Estratégias algorítmicas

Nesta sessão serão apresentadas as técnicas e os algoritmos da literatura para o Sudoku que foram estudadas e implementadas por nós. Resumidamente:

- ***Minimum Remaining Values*** é uma heurística 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 a cada vez que for necessário atribuir valor a uma nova célula ainda não preenchida.
- ***Forward Checking*** é uma técnica que propõe o término da busca pela solução (ou poda da sub-árvore de busca) caso alguma variável ainda não testada não tenha mais valores possíveis em seu domínio. Para o caso do Sudoku, toda vez que o algoritmo de backtracking encontrar uma célula não-preenchida em que não seja mais possível inserir qualquer valor, a busca atual é finalizada [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. 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.
- **Propagação de Restrições** (*Constraint Propagation*) é uma técnica que remove valores do domínio das variáveis a que sabidamente não poderão mais ser atribuídos. Enquanto o algoritmo for executado através da busca em profundidade (backtracking padrão), a técnica irá remover candidatos das células do tabuleiro à medida em que se infira que eles ali não podem mais ser inseridos [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 por Skiena utilizando a técnica de Forward Checking e a heurística de Minimum Remaining Values; algoritmo de backtracking implementado com técnica de Manipulação de Bits; algoritmo proposto por Norvig baseado em propagação de restrições (considerado aquele de melhor desempenho até então para o Sudoku); implementação do modelo em PLI usando o método branch-and-cut da ferramenta CPLEX; e implementação do modelo de PR com o módulo CP-Optimizer do CPLEX. Além destes, propomos uma versão mais eficiente do algoritmo de propagação de restrições de Norvig, em que conseguimos melhorias estruturais com o emprego de Manipulação de Bits e *heaps* e consideramos dois novos casos de propagação.

O Sudoku também pode ser resolvido através de técnicas meta-heurísticas 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, 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 da vizinhança é feita escolhendo-se cada valor não-fixo de cada bloco e trocando-se dois elementos de uma mesma posição. Cada troca é um novo vizinho, e 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 de restrições, uso de *heaps* e manipulação de bits; e, uma meta-heurística GRASP para resolver o Sudoku com técnica de Manipulação de Bits.

Quanto ao **algoritmo exato**, nossa proposta é uma otimização da **propagação de restrições** de um dos melhores algoritmos conhecidos [Norvig 2006]. Nesse caso, os candidatos são representados através de vetores de bits, utilizando Manipulação de Bits para inserção, remoção e verificação de candidatos de forma muito rápida. Além disso, também lapidamos a heurística Minimum Remaining Values já implementada, para que a busca da célula menos restritiva seja em $O(1)$ e não $O(n^2)$. Obtivemos desempenho superior ao do algoritmo original nos testes computacionais (omitidos aqui por restrição de espaço).

Quanto ao **método meta-heurístico** GRASP, a representação e custo de cálculo da função objetivo foi otimizada novamente através de técnicas de Manipulação de Bits, em que a verificação de um dígito será feito em $O(1)$. 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 que contém todos os candidatos de inserção ordenados pelo custo. Seleciona-se assim os $\alpha\%$ elementos da lista C (parte gulosa) para inserir na LRC, para um certo α pré-definido. 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 meta-heurística prossegue, guardando sempre a melhor solução encontrada, até que alguma condição de parada seja verdadeira. A condição de parada pode ser definida como um total de iterações ou quando se encontra a solução é ótima (custo zero). Quanto maior o coeficiente α, mais aleatório será; quanto menor, mais guloso.

5. Geração de Instâncias

Para se gerar uma instância válida (grid incompleto) do Sudoku é necessário, primeiramente, gerar uma solução válida (grid completo) e, assim, remover alguns valores adequadamente. O desafio é gerar instâncias que apresentem solução única. Um processo iterativo para modificação do início da construção determinística da solução foi aplicado de tal forma a garantir a geração de grandes conjuntos de soluções. Com muito poucas células preenchidas ou muito poucas não-preenchidas, a instância é considerada fácil.

Dois algoritmos polinomiais para a criação de soluções do Sudoku, de complexidade $O(n^3)$, são apresentados a seguir.

5.1. Caso polinomial do Sudoku: grid inicial vazia

O Sudoku pode ser resolvido em tempo polinomial quando toda a grid está vazia, ou seja, quando a instância não contiver nenhuma célula já preenchida. Os algoritmos esboçados 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 a 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 neste projeto de pesquisa são:

- 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, 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, Unique-Sat, Pré-Coloração Estendida de Vértices e Cobertura Exata) para o Sudoku;
- elaboração e implementação do algoritmo de reconhecimento de uma solução (prova de pertinência a 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: algoritmo exato de propagação de restrições, com uso de heaps e Manipulação de Bits; meta-heurística GRASP com Manipulação de Bits;
- estudo sobre casos polinomiais do Sudoku, onde foram propostos dois algoritmos determinísticos de tempo $O(n^2)$ para grids inicialmentevazias;
- 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 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 Sudoku.

Acreditamos ter sido de grande valia toda a teoria estudada e os algoritmos desenvolvidos e implementados durante este trabalho. Pretendemos aplicar este conhecimento para novas descobertas, tanto no Sudoku quanto com relação a outros jogos lógicos, especialmente em grids, seguindo linhas de pesquisa das áreas de Otimização Combinatória, Complexidade de Algoritmos 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)*.