Comparação de codificações para solução de puzzles Sudoku via algoritmo DPLL

Savio Lopes Rabelo, Helio Henrique Barbosa Rocha, e Thiago Alves Rocha

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE), Eixo Tecnológico de Computação, Campus Maracanaú – CE – Brazil
saviorabelo.ti@gmail.com, hique.rocha@gmail.com, thiago.alves@ifce.edu.br

Resumo Na presente investigação, exemplos de puzzles Sudoku serão codificados como problemas SAT. O propósito deste trabalho é avaliar o desempenho do algoritmo DPLL para resolver puzzles Sudoku considerando formas de codificação para sua solução: minimal e estendida. O algoritmo e as codificações foram implementados na linguagem de programação Python. Exemplos de entrada de jogos aleatórios foram considerados. O tempo para solução do jogo pelo algoritmo pode depender, além da codificação empregada, da configuração do mesmo.

Palavras-chave: lógica computacional, codificação SAT, algoritmo DPLL

1 Introdução

Sudoku é um puzzle combinatório baseado no posicionamento lógico de números. Ainda que as suas regras possam ser consideradas simples, a sua resolução pode se tornar um desafio intelectual. Na sua versão padrão, o jogo tem por objetivo a colocação dos números de 1 a 9 em cada uma das células vazias numa grade de dimensão 9x9 (que contem 81 células), constituída por sub-grades de dimensão 3x3 denominadas regiões [10]. Há, portanto, 9 regiões e cada região contém 9 células. Inicialmente o quebra-cabeça está parcialmente preenchido, contendo números inseridos em algumas células, de maneira a permitir uma indução ou dedução dos números em células que estejam vazias. Em cada coluna, linha e região, os números de 1 a 9 aparecem uma única vez.

Pela simplicidade de suas regras e também por ser membro de uma classe de problemas de satisfação de restrições, o Sudoku tem sido objeto de muitos estudos, em particular no que diz respeito às suas propriedades matemáticas e algorítmicas, como por exemplo enumeração de possíveis grades de jogo [6,7] e NP-completude da sua versão generalizada [18]. Além disso, vários métodos foram utilizados para resolver o Sudoku: Formulações em problemas de satisfação de restrições [17,15], métodos de busca [8], algoritmos genéticos [12] e o método *particle swarm optimization* [14].

Uma maneira de resolver tais puzzles combinatórios consiste na sua modelagem como um problema de satisfazibilidade (SAT) da lógica proposicional. O problema SAT envolve a busca por uma atribuição de valores verdade que

torna verdadeira uma fórmula da lógica proposicional [16]. O SAT foi o primeiro problema identificado como NP-Completo[1], sendo ainda hoje um dos mais estudados dessa classe. Destacam-se na literatura recente, trabalhos que tratam de problemas SAT com o uso de hipergrafos. Foram evidenciadas em [2] novas perspectivas sobre a representação de problemas SAT com base em hipergrafos direcionados, além de um novo algoritmo tipo DPLL. É notório o mérito no estudo destes problemas na medida em que resolvedores bem elaborados associados a codificações apropriadas favorecem a resolução de muitas instâncias úteis em diversas áreas do conhecimento. O algoritmo DPLL frequentemente serve como ponto de partida no desenvolvimento de métodos capazes de resolver de maneira eficiente instâncias do SAT. Será empregado o algoritmo DPLL[2] como descrito em [16] e com escolha de literais feita de forma aleatória.

O principal objetivo deste trabalho é estudar a codificação de puzzles Sudoku com fórmulas da lógica proposicional. É pretendido também comparar o desempenho do algoritmo DPLL com codificações diferentes e instâncias distintas do problema. Neste sentido, objetiva-se buscar a codificação mais eficiente para o caso geral do problema: dada uma situação inicial observar qual codificação apresenta melhor comportamento. Ainda, visa-se avaliar, considerando casos específicos, a importância da codificação para resolver de forma eficiente instâncias do problema. Assim sendo, a elucidação computacional de jogos Sudoku é desenvolvida considerando duas etapas: sua codificação em forma normal conjuntiva (FNC) com posterior resolução via algoritmo DPLL. São introduzidas duas codificações diretas para o Sudoku [11]: codificação minimal e codificação estendida. A codificação minimal é suficiente para caracterizar o jogo, enquanto a codificação estendida acrescenta cláusulas excedentes para a codificação minimal.

2 Codificações SAT para o problema Sudoku

A codificação minimal assegura que há pelo menos um número (entre 1 e 9) em cada célula, e que cada número aparece no máximo uma vez em cada linha, em cada coluna e em cada sub-grade 3x3. Considerando que os índices x,y e z da variável booleana p_{xyz} representam linha, coluna e número, respectivamente, formalmente temos [11]:

"Há, pelo menos, um número em cada célula":

$$\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigvee_{z=1}^{9} p_{xyz}, \tag{1}$$

[1] Por ser NP-completo, não se conhece algoritmo com tempo melhor (no pior caso) que o exponencial.

[2] O DPLL é reconhecidamente capaz de decidir corretamente se um conjunto de cláusulas é satisfazível ou não e adequado por aceitar novas heurísticas para escolha de literais quando implementado na resolução de problemas SAT

"Cada número aparece, no máximo, uma vez em cada linha":

$$\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{x=1}^{8}\bigwedge_{i=x+1}^{9}(\neg p_{xyz} \vee \neg p_{iyz}), \tag{2}$$

"Cada número aparece, no máximo, uma vez em cada coluna":

$$\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{y=1}^{8}\bigwedge_{i=y+1}^{9}(\neg p_{xyz} \vee \neg p_{xiz}), \tag{3}$$

"Cada número aparece, no máximo, uma vez em cada sub-grade (3x3)":

$$\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=y+1}^{3}(\neg p_{(3i+x)(3j+y)z} \vee \neg p_{(3i+x)(3j+k)z}), \tag{4}$$

$$\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=x+1}^{3}\bigwedge_{l=1}^{3}(\neg p_{(3i+x)(3j+y)z} \vee \neg p_{(3i+k)(3j+l)z}). \tag{5}$$

Na codificação minimal, a fórmula resultante terá 8829 cláusulas, sem contar as cláusulas unitárias representando as células pré-preenchidas. Destas cláusulas, 81 cláusulas têm tamanho nove e as 8748 restantes possuem tamanho dois.

A codificação estendida assegura que cada entrada na grade possui exatamente um número, e o mesmo para cada linha, cada coluna e cada sub-grade 3x3. A codificação estendida inclui todas as cláusulas da codificação minimal, bem como as seguintes restrições:

"Há, no máximo, um número em cada entrada":

$$\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{8}\bigwedge_{i=z+1}^{9}(\neg p_{xyz} \vee \neg p_{xyi}), \tag{6}$$

"Cada número aparece pelo menos uma vez em cada linha":

$$\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{x=1}^{9} p_{xyz}, \tag{7}$$

"Cada número aparece pelo menos uma vez em cada coluna":

$$\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{y=1}^{9} p_{xyz}, \tag{8}$$

"Cada número aparece pelo menos uma vez em cada sub-grade (3 x 3)":

$$\bigvee_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3} p_{(3i+x)(3j+y)z}. \tag{9}$$

Na codificação estendida, a fórmula resultante possuirá 11988 cláusulas, descontadas as cláusulas unitárias representando as entradas pré-alocadas (contendo números). Destas cláusulas, 324 têm tamanho nove e as 11664 restantes possuem tamanho dois.

2.1 O algoritmo DPLL

O algoritmo DPLL (ou método Davis–Putnam–Logemann–Loveland) [3,4] vem sendo implementado com os mais diversos métodos ou processos para soluções de problemas [16]. Basicamente, uma valoração para uma fórmula fornecida (na forma de um conjunto de cláusulas) deve ser construída. No início, todos os átomos da fórmula apresentam um valor desconhecido para sua valoração. A cada iteração do algoritmo (descrito no Algoritmo 1 [16]), um literal é escolhido, ficando determinada sua valoração como sendo verdadeira. Caso esse literal seja negativo, a valoração da atômica do literal é falsa. Uma vez valorada, a fórmula é simplificada (Algoritmo 2 [16]) pela eliminação das cláusulas que contêm o literal e pela eliminação da negação do literal das cláusulas restantes. Se essa valoração satisfizer todas as cláusulas, tem-se uma valoração que satisfaz a fórmula inicial. Se alguma cláusula for falsificada, altera-se a escolha da valoração para falso. Se nenhuma cláusula for falsificada, nem todas as cláusulas foram satisfeitas, procede-se à próxima escolha de literal. O processo para quando uma valoração for encontrada (fórmula satisfazível) ou quando não há mais átomos para serem testados (fórmula insatisfazível). Para uma instância SAT, um algoritmo completo é aquele que acha uma solução (ou prova que tal solução não existe). O método DPLL apresentado abaixo é chamado de SAT-completo, ou seja, ele sempre é capaz de decidir corretamente se um conjunto de cláusulas é ou não é satisfazível.

Algoritmo 1: Algoritmo DPLL(F).

Entrada: Uma fórmula F no formato CNF.
Saída: verdadeiro, se F é satisfazível ou falso, caso contrário.

```
1  Fazer v(p) = * para todo átomo p;
2  F'= Simplifica(F);
3  se F' = ∅ então
4  |  retorna verdadeiro;
5  senão se F' contém uma cláusula vazia (falsa) então
6  |  retorna falso;
7  fim
8  Escolha um literal L com v(L) = *;
9  se DPPL(F' ∪ L) = verdadeiro então
10 |  retorna verdadeiro;
11 senão se DPPL(F' ∪ ¬L) = verdadeiro então
12 |  retorna verdadeiro;
13 senão
14 |  retorna falso;
15 fim
```

Algoritmo 2: Algoritmo Simplifica(F).

Entrada: Uma fórmula F no formato CNF.
Saída: Uma fórmula na forma clausal equivalente a F porém mais simples.

```
1 enquanto F possui alguma cláusula unitária L faça
2 |   Apaga de F todas cláusulas que contém L;
3 |   Apaga ¬L das cláusulas restantes;
4 fim
5 retorna F
```

3 Metodologia

Inicialmente, uma amostra de jogo escolhida ao acaso serve como entrada no programa. Neste momento, apenas a codificação minimal é considerada. Uma vez que o jogo tenha sido resolvido[3], cada célula contendo um número é sequencialmente eliminada. A cada eliminação de célula (cumulativamente) o jogo é novamente resolvido.

Devemos observar um ponto[4] a partir do qual a solução torna-se dispendiosa (mais lenta). Neste ponto deve se guardar a quantidade de células preenchidas. Em seguida, outros exemplos de jogos contendo aproximadamente a mesma quantidade de células preenchidas são analisados em relação ao tempo de solução. Nesta etapa os resultados obtidos pelas codificações minimal e estendida serão contrastados quanto ao seu impacto na solução.

Ao longo deste trabalho empregamos o algoritmo DPLL[5] que trata fórmulas no formato clausal (chamada de Forma Normal Conjuntiva). Esta restrição visa facilitar a implementação de resolvedores sem perda de generalidade[6]. O problema SAT consiste em determinar se uma fórmula na FNC é satisfazível ou não.

Os arquivos com instâncias de jogos aleatórias foram obtidos on-line[7], sendo posteriormente convertidos no formato padrão CNF. Os algoritmos 1 e 2 supramencionados [16] foram implementados na linguagem de programação Python versão 2.7.11, bem como as codificações SAT e as entradas de problemas Sudoku. Para efeito de reprodutibilidade, os requisitos mínimos de hardware são: Processador Intel Core i3, 250 MB de espaço livre em disco e 4 GB de memoria RAM. Os requisitos mínimos de software incluem sistema operacional Windows 10.

[3] As 81 células preenchidas com números conforme as regras do Sudoku.
[4] Tendo em vista a quantidade de células que restam na grade.
[5] O literal é escolhido aleatoriamente.
[6] É demonstrável que qualquer fórmula da lógica proposicional clássica tem uma fórmula equivalente no formato clausal.
[7] `http://www.sudoku.name/`

4 Resultados e Discussões

Tendo em vista os resultado obtidos quando da solução de um exemplo e jogo, as células são eliminadas uma a uma em sequência. Após esta eliminação o jogo é novamente resolvido, e, desta vez, o tempo para a sua resolução é registrado.

O resultado deste processo, a partir do ponto em que a grade do jogo encontrava-se preenchida com 62 células pré-preenchidas, está ilustrado na Fig. 1. Em cada eliminação foram realizadas 5 computações de tempo. Aparentemente, considerando apenas a codificação minimal, uma grande quantidade de células pré-preenchidas implica em uma menor variabilidade e tempo de solução. A partir do momento em que a grade passa a conter 55 números, o tempo para solucionar o jogo começa a sofrer aumento considerável acompanhado de uma maior variabilidade. Para a instância considerada, com 52 restrições não se fez possível resolver o jogo em menos de 11 minutos.

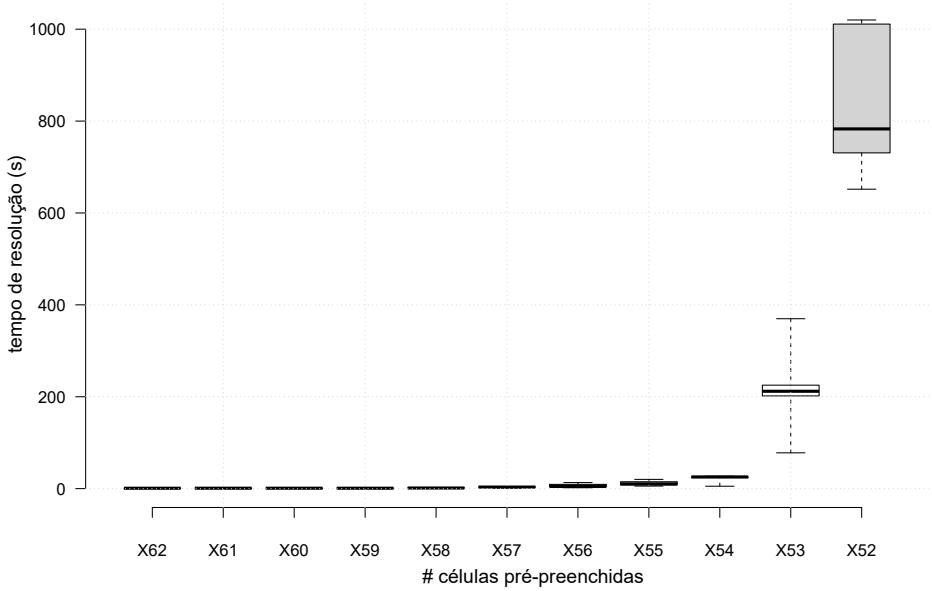

Figura 1: Tempo de resolução do jogo, usando codificação minimal (Eqs.1-5), em termos do número de células pré-preenchidas.

Foram tomadas aleatoriamente outras três instâncias de jogo diferentes quanto à quantidade de células pré-preenchidas. Estas amostras foram computadas considerando cinco situações, sendo a primeira apenas com codificação minimal e as posteriores contendo, além da codificação minimal, as codificações estendidas, acrescentadas acumulativamente uma por vez. Com isto, é pretendido analisar

o possível impacto da redundância no desempenho associado ao programa. Os resultados desta etapa encontram-se condensados na Tab. 1.

Tabela 1: Tempo médio (s) para resolução dos jogos.

	restrições				
# **células pré-preenchidas**	Eqs.(1)-(5)	Eqs.(1)-(6)	Eqs.(1)-(7)	Eqs.(1)-(8)	Eqs.(1)-(9)
54	4,062	6,422	6,360	6,547	6,563
53	3,703	6,297	6,359	6,359	6,516
45	3,688	6,247	6,203	6,344	6,344

Na Tab. 1, temos a quantidade de células pré-preenchidas em cada uma das três amostras analisadas e as restrições observadas, em que se considerou desde a codificação minimal (Eqs.1-5) até o acréscimo de todas as codificações estendidas (Eqs.1-9). Cada computação foi repetida sete vezes, sendo apresentado o valor médio resultante. Há duas observações sobre os resultados obtidos: o tempo de solução do jogo aumenta à medida que são incluídas as codificações estendidas, e a solução para um jogo com menos células pré-preenchidas é obtida num tempo menor. Além disto, para uma mesma amostra, o acréscimo das codificações estendidas não apresenta alterações significativas de tempo. Tal constatação aparentemente contrasta com as observações iniciais: quanto menos células pré-preenchidas, mais tempo se levaria para solucionar um problema. É sugerido que a quantidade de células pré-preenchidas, apesar de claramente importante, tem dominância possivelmente limitada sobre o resultado do experimento.

Continuando, insistiu-se na redução da quantidade de células pré-preenchidas alimentadas. Foi selecionado um jogo com apenas 22 células pré-preenchidas. Neste caso, a solução envolvendo unicamente a codificação minimal (Eqs.1-5) não se mostrou suficiente para obtenção de uma resposta no tempo estipulado de 20 minutos. No entanto, envolvendo a codificação estendida completa (Eqs.1-9), o mesmo jogo foi resolvido num tempo médio de 6.3 segundos.

5 Conclusão e Trabalhos Futuros

Neste trabalho estudamos o impacto da codificação em relação ao tempo de resposta de problemas SAT na forma de amostras de puzzles Sudoku com emprego do algoritmo DPLL para sua resolução. Embora não tenha sido verificado neste trabalho, é possível que, em geral, um jogo possua mais de uma solução. A unicidade da resolução de instâncias Sodoku é estudada em [13], sendo tal requisito implementado em [9], conforme descrito em [10]. A princípio, a quantidade parece influenciar no tempo de reposta obtido. Em conformidade com os experimentos realizados, jogos com menor quantidade de células preenchidas podem ser resolvidos em menor tempo que jogos com maior quantidade de células pré-preenchidas. Além disso, uma redução ainda maior da quantidade de células

pré-preenchidas supostamente evidencia a importância da redundância da codificação estendida no contexto da resolução do problema. Portanto, a configuração dos jogos parece exercer forte influência sobre o tempo em que os mesmos são elucidados. Isso foi analisado em outros trabalhos, como [5] e [19].

Para investigações posteriores, pretende-se fazer uma comparação com variações de heurísticas de escolha de literais no DPLL [16] e comparações com outras abordagens, a exemplo de algoritmos genéticos [12], *particle swarm optimization* [14] e *simulated annealing* [1].

Referências

1. Chi, E. C., Lange, K. Techniques for Solving Sudoku Puzzles. CoRR, abs/1203.2295, 2012.
2. Croitoru, C.; Croitoru, M. Combinatorial Results on Directed Hypergraphs for the SAT Problem. Graph Structures for Knowledge Representation and Reasoning: 4th International Workshop, GKR 2015, Buenos Aires, Argentina, July 25, 2015, Revised Selected Papers. Cham: Springer International Publishing, 2015. p. 72–88.
3. Davis, M.; Putnam, H. A Computing Procedure for Quantification Theory. J. ACM, v. 7, n. 3, p. 201–215, 1960.
4. Davis, M.; Logemann, G.; Loveland, D. W. A machine program for theorem-proving. Commun. ACM, v. 5, n. 7, p. 394–397, 1962.
5. Ercsey-Ravasz, M.; Toroczkai, Z. The Chaos Within Sudoku. CoRR, abs/1208.0370, 2012.
6. Felgenhauer, B.; Jarvis, F. Enumerating possible Sudoku grids. 2005. Disponível em: `http://www.afjarvis.staff.shef.ac.uk/sudoku/`.
7. Felgenhauer, B.; Jarvis, F. Mathematics of Sudoku I. 2006. Disponível em: `http://www.afjarvis.staff.shef.ac.uk/sudoku/`.
8. Geem, Z. W. Harmony Search Algorithm for Solving Sudoku. In: . KnowledgeBased Intelligent Information and Engineering Systems: 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007. Proceedings, Part I. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 371–378.
9. Jain, S., Shakher, C. Mathematical and C Programming Approach for Sudoku Game. Journal of Game Theory, v. 3, n. 1, p. 1–6, 2014.
10. Lee, W. Programming Sudoku. New York: Apress, 2006. (Technology in action).
11. Lynce, I.; Ouaknine, J. Sudoku as a SAT Problem. In: In Proc. of the Ninth International Symposium on Artificial Intelligence and Mathematics: Springer, 2006.
12. Mantere, T.; Koljonen, J. Solving, Rating and Generating Sudoku Puzzles with GA. In: IEEE Congress on Evolutionary Computation: IEEE, 2007. p.1382–1389.
13. McGuire, G., Tugemann, B. and Civario, G. There Is No 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration. Experimental Mathematics, v. 23, n. 2, p. 190–217, 2014.
14. Moraglio, A.; Chio, C. D.; Poli, R. Geometric Particle Swarm Optimisation. In: Genetic Programming: 10th European Conference, EuroGP 2007, Valencia, Spain, April 11-13, 2007. Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 125–136.
15. O'Sullivan, B.; Horan, J. Generating and Solving Logic Puzzles Through Constraint Satisfaction. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, 2007. p. 1974–1975.

16. Silva, F. da; Melo, A. de; Finger, M. Lógica para Computação. São Paulo: Thomson Pioneira, 2006.
17. Simonis, H. Sudoku as a Constraint Problem. In: In Proc. 4th Int. Workshop on Modelling and Reformulating Constraint Satisfaction Problems, 2005. p. 13–27.
18. Yato, T.; Seta, T. Complexity and completeness of finding another solution and its application to puzzles. IEICE Trans Fundam Electron Commun Comput Sci (Inst Electron Inf Commun Eng), E86-A, n. 5, p. 1052–1060, 2003.
19. Zhai, G., Zhang, J. Solving Sudoku Puzzles Based on Customized Information Entropy. International Journal of Hybrid Information Technology, v. 6, p. 77-92, 2013.