Periódicos Brasil. Pesquisa Científica ISSN 2674-9432

UTILIZANDO METAHEURÍSTICAS PARA RESOLVER O QUEBRA-CABEÇA SUDOKU

Simone Silva Frutuoso de Souza

RESUMO

O quebra-cabeça Sudoku é considerado um problema NP-completo, e por este fato desperta a atenção de pesquisadores. É um jogo muito popular, jogado por milhões de pessoas, apesar de ser difícil de resolver dependendo das condições inicias do quebra-cabeça. Neste artigo apresentam-se duas metodologias para resolver o quebra-cabeça Sudoku, isto é, as Metaheurísticas Simulated Annealing e Busca Tabu. Os algoritmos foram desenvolvidos na linguagem C++. Para evidenciar a eficácia destas metodologias foram realizados vários testes e foi possível comprovar que com as metodologias podem ser encontradas as soluções ótimas de diferentes níveis de dificuldade e dimensão do quebra-cabeça.

Palavras-chave: Metaheurística, Sudoku, Otimização, Simulated Annealing, Busca Tabu.

USING METAHEURISTICS TO SOLVE THE SUDOKU PUZZLE

ABSTRACT

The Sudoku puzzle is considered an NP-hard problem, and this fact caught the attention of researchers. It is a very popular game, played by millions of peoples, although it is difficult to solve depending on the initial conditions of the puzzle. This paper presents two methodologies to solve the Sudoku puzzle, i.e., metaheuristics Simulated Annealing and Tabu Search. The algorithms were developed in C++. To demonstrate the effectiveness of these methodologies were performed several tests and it was possible to prove that with the methodologies can found the optimal solution of different levels of difficulty and size of the puzzle.

Keywords: Metaheuristic, Sudoku, Optimization, Simulated Annealing, Tabu Search.

Instituição afiliada – Universidade do Estado de Mato Grosso (UNEMAT), campus Tangará da Serra
Dados da publicação: Artigo publicado em Agosto de 2025
DOI: https://doi.org/10.36557/pbpc.v4i2.384

Autor correspondente*: Simone Silva Frutuoso de Souza*

1 INTRODUÇÃO

O enigma Sudoku foi projetado por Howard Garns, um arquiteto aposentado de 74 anos, embora provavelmente inspirado na invenção de 1780 do matemático suíço Leonhard Euler, que propôs o quadrado latino que consistia na ideia de arranjar números de tal maneira que qualquer número ou símbolo iria ocorrer apenas uma vez em cada linha ou coluna, sendo seu objetivo utilizar os quadrados latinos para análise estatística. Howard Garns adicionou a restrição de cada sub-grade poder ter os números ocorrendo apenas uma vez, isto é, sem repetições nas linhas, colunas e nas sub-grades. Com a adição desta regra temos o quebra-cabeça Sudoku como o conhecemos hoje (Sudoku Made Easy, 2012).

O Sudoku foi publicado pela primeira vez no final da década de 70, mais precisamente no ano de 1979 na América do norte em Nova York na revista Math puzzles and Logic Problems, pelo editor da companhia Dell Magazines. Dell era uma companhia especialista em revistas de quebra-cabeças de lógica e capacidade. A editora publicou o jogo com o nome de “Number Place”, ou “alocação de números” traduzido em português, nome o qual é utilizado até hoje nos Estados Unidos (Sudoku Essentials, 2012).

O Sudoku é definitivamente uma invenção americana, porém o nome é de origem japonesa. Em 1984 quando Nikoli, a maior empresa de quebra-cabeças do Japão descobriu o jogo decidiu leva-lo, e então o mesmo foi introduzido no Japão sob o nome de “Suuji wa dokushin ni kagiru” que significa “Números que devem permanecer únicos”, está frase deu origem ao nome que se tornou a marca registrada da Nikoli. Em 1986 o Sudoku tornou-se um dos jogos mais vendidos do Japão (Sudoku Made Easy, 2012).

Apesar de toda a popularidade no Japão, o Sudoku só conquistou o mundo quando Wayne Gould, um juiz aposentado de Hong Kong, que era fã de quebra-cabeças e programador de computador, começou a criar seus próprios quebra-cabeças com um programa gerador de jogos com vários níveis de dificuldade que ele mesmo desenvolveu. Gould apresentou aos editores do jornal The Times (Londres, Inglaterra) seu programa gerador e não estava cobrando nada por ele. O The Times decidiu arriscar e no dia 12 de novembro de 2004 publicou seu primeiro Sudoku. A partir disto o Sudoku se tornou conhecido no mundo todo (Wikipédia, 2012).

O Sudoku é um quebra-cabeça que chama bastante a atenção dos pesquisadores e matemáticos, pelo fato do mesmo ser considerado um problema NP-completo, e ajudar no desenvolvimento de alunos e despertar o interesse em matemática. (Sudoku Science, 2006). Estima-se que existem 6.670.903.752.021.072.936.960 possibilidades para o quebra-cabeça Sudoku de ordem 3 (9 x 9 células). (Delahaye Jean-Paul, 2006). Este fato desperta o interesse em desenvolvimento de metodologias para resolver o quebra-cabeça. Neste contexto na literatura especializada encontram-se diversos trabalhos relacionados ao desenvolvimento de métodos e solver para resolver o quebra-cabeça Sudoku. Entre eles destacam-se os seguintes trabalhos. Em (Majumder et al., 2010) apresenta-se um algoritmo backtrack para resolver o Sudoku, já em (Davis, 2010) apresenta-se uma proposta para resolver o Sudoku com o uso de lógica por eliminação, ou como mais conhecido, técnica de força bruta. Em (Babu et al., 2010) o problema Sudoku é formalizado e resolvido como um sistema de equações lineares esparsos. Em (Moon et al., 2009) apresenta-se uma metodologia baseada no teorema de Sinkhorn, uma técnica de cálculo numérico para trabalho com matrizes. O trabalho de (Bartlett et al., 2008) destaca-se por apresentar a resolução do problema Sudoku através de métodos de cobertura e programação restrita.

No entanto entre estas metodologias o trabalho de (Lewis, 2007) resolve o quebra-cabeça Sudoku como um problema de otimização utilizando a Metaheurística Simulated Annealing.

Neste artigo apresentam-se duas metodologias baseadas nas Metaheurísticas Simulated Annealing e Busca Tabu para resolver o problema do quebra-cabeça Sudoku. Para comprovar a eficácia das metodologias e fazer uma análise do desempenho dos métodos foram realizados vários testes, e apresentam-se três casos de prova nos resultados, sendo dois deles quebra-cabeças de dimensão 3 (9 x 9 células) e um quebra-cabeça de dimensão 4 (16 x 16 células).

Este artigo está organizado como a seguir. Na seção 2 descrevem-se as características do quebra-cabeça Sudoku. Na seção 3 apresenta-se o algoritmo Simulated Annealing e na seção 4 o algoritmo da busca tabu. A metodologia proposta encontra-se na seção 5. Os resultados estão apresentados na seção 6 e na seção 7 está à conclusão para este trabalho.

2 SUDOKU

O Sudoku é um quebra-cabeça simples, onde se tem uma grade de dimensão $n^2 x n^2$ que é dividida em $n x n$ sub-grades distintas, sendo n a dimensão do problema. A dimensão mais popular é n = 3, ou seja, uma grade de 9 x 9 (81 células), dividida em sub-grades de 3 x 3 células. A Figura 1 ilustra um quebra-cabeça Sudoku.

O objetivo do quebra-cabeça é que o jogador preencha todas as células da grade com as pistas dadas inicialmente. As pistas são números dados em posições aleatórias da grade para auxiliar o jogador na resolução do problema. Para resolver o problema três critérios devem ser satisfeitos:

- Cada linha de células deve conter os inteiros de 1 até n^2 exatamente uma vez;
- Cada coluna de células deve conter os inteiros de 1 até n^2 exatamente uma vez;
- Cada n x n sub-grades devem conter os inteiros de 1 até n^2 exatamente uma vez;

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

Figura 1. Quebra-cabeça Sudoku.

Quando os três critérios acima são satisfeitos, o jogo estará solucionado, e o jogador terá completado a grade inicial proposta. Normalmente o quebra-cabeça Sudoku se classifica em níveis de dificuldade, de acordo com a relevância, posição e quantidade dos números iniciais. Efetivamente isto pode influenciar totalmente na solução do problema, pois um quebra-cabeça com o máximo de números iniciais pode

ser fácil de resolver e um com o mínimo de números pode ser extremamente complicado de resolver. Os níveis mais populares são classificados em fácil, médio, difícil e super-sudoku. O que difere um nível do outro é a quantidade de números iniciais preenchidos.

3 METAHEURÍSTICA SIMULATED ANNEALING

A Metaheurística Simulated Annealing proposta por Kirkpatrick (Kirkpatrick et al, 1983) é um método estocástico de otimização que consiste em uma busca local probabilística, e se fundamenta em uma analogia com a termodinâmica. Isto é, no processo térmico dito Annealing ou recozimento, utilizado em metalúrgicas para obtenção de estados de baixa energia em metais. Durante o recozimento o material passa por vários estados possíveis, que correspondem a soluções do espaço de busca.

De forma análoga, o algoritmo Simulated Annealing busca realizar uma melhoria na solução, substituindo a solução atual com uma próxima solução vizinha, em sua vizinhança, sendo a nova solução escolhida de acordo com a função objetivo e o parâmetro T conhecida como a temperatura corrente. Então quanto maior for T, maior a probabilidade de que uma solução vizinha de pior qualidade seja escolhida, e à medida que o algoritmo realiza iterações, o valor de T é diminuído, fazendo com que o algoritmo geralmente convirja para uma solução ótima global.

O algoritmo da Metaheurística Simulated Annealing é apresentado na Figura 2. (Gallego et al., 2006).

```
1   S* ← S;                    {Melhor solução obtida até então}
2   IterT ← 0;                 {Número de iterações na temperatura T}
3   T ← T0;                    {Temperatura corrente}
4   enquanto (T ▷ 0) faça
5       enquanto (IterT ◁ SA max) faça
6           IterT ← IterT + 1;
7           gere um vizinho qualquer S' ∈ N(s);
8           Δ = f(s') − f(s);
9           se (Δ ◁ 0) então
10              S ← S';
11              se (f(s') ◁ f(s*);) então
12                  S* ← S';
13              senão
14                  gere x ∈ [0,1];
15                  se (x ◁ e^(−Δ/T)) então
16                      S ← S';
17                  fim se
18              fim se
19          fim se
20      fim enquanto
21      T ← αT;
22      IterT ← 0;
23  fim enquanto
24  S ← S*;
25  Retorne S;
```

Figura 2. Algoritmo Simulated Annealing.

O algoritmo Simulated Annealing a cada iteração escolhe aleatoriamente uma solução vizinha, a partir da solução corrente, então se essa solução vizinha for de melhor qualidade que a solução corrente, essa solução vizinha passa a ser a nova solução corrente. Por outro lado, se a função objetivo da solução vizinha for de pior qualidade que da solução corrente em Δ unidades, então a probabilidade de se passar para essa solução vizinha é dada pela equação (1).

$$e^{-\Delta/T} \tag{1}$$

Onde Δ é a variação da função objetivo ou equivalente (diferença entre as funções objetivo da solução corrente e a solução vizinha é sempre positivo) e T é o parâmetro de controle, conhecido como temperatura. Em altas temperaturas, cada solução vizinha escolhida aleatoriamente tem probabilidade elevada de se transformar

na solução corrente, porém à medida que a temperaturas diminui então apenas as soluções vizinhas ligeiramente inferiores da solução corrente tem maior probabilidade de se tornar na solução corrente. Quando o algoritmo Simulated Annealing aceita uma solução pela probabilidade indica que uma solução de pior qualidade se transforma na solução corrente e essa estratégia permite fugir de um ótimo local.

4 METAHEURÍSTICA BUSCA TABU

A Metaheurística Busca Tabu proposta por Fred Glover (Glover, 1986) é um método de busca local que consiste em explorar o espaço de soluções movendo-se de uma solução para outra que seja seu melhor vizinho. A principal característica é a estrutura de memória para armazenar as soluções geradas (ou características das soluções geradas). Estas características possibilitam que o algoritmo escape de ótimos locais.

O algoritmo da Metaheurística Busca Tabu é apresentado na Figura 3. (Gallego et al., 2006).

O melhor movimento admissível é aquele com melhor avaliação na vizinhança da solução corrente em termos de valor da função objetivo e restrições tabu. A função de avaliação escolhe o movimento que produz a maior melhoria, ou a menor piora na função objetivo. A lista tabu é introduzida no sentido de guardar as características dos movimentos realizados, para que futuros movimentos que apresentem estas mesmas características possam ser classificados como tabu (isto é, proibidos de serem executados). A aceitação de movimentos que piorem o resultado final abre a possibilidade de retorno a soluções já visitadas, portanto, ciclos podem ocorrer e a função da lista tabu é evitar que tais ciclos ocorram. (Glover e Laguna, 1993).

```
1   S* ← S;                    {Melhor solução obtida até então}
2   Iter ← 0;                  {Contador do Número de Iterações}
3   MelhorIter ← 0;            {Iteração mais recente que forneceu S*}
4   Seja BTmax o número máximo de iterações sem melhora em S*;
5   T ← 0;                     {Lista Tabu}
6   Inicialize a função de aspiração A;
7   enquanto (Iter – MelhorIter ≤ BTmax) faça
8   Iter ← Iter + 1;
9   gere um vizinho qualquer s' ∈ N(s);
10  Seja s' ← s ⊕ m o melhor elemento de V ⊆ N(s) tal que o movimento m não seja tabu (m ∉ T)
        ou s' atenda a condição de aspiração ( f(s') < A(f(s)));
11  Atualize a Lista Tabu T;
12  s ← s';
13  se f(s) < f(s*) então
14      s* ← s;
15      MelhorIter ← Iter ;
16  fim se;
17  Atualize a função de aspiração A;
18  fim enquanto
19  S ← S*;
20  Retorne S;
```

Figura 3. Algoritmo Busca Tabu.

Para cada possível valor *V* da função objetivo existe um nível de aspiração *A(V)*: uma solução *S'* em *V* pode ser gerada se *f(S') < A(f(S'))*, mesmo que o movimento *m* esteja na lista tabu. A função de aspiração *A* é tal que, para cada valor *V* da função objetivo, retorna outro valor *A(V)*, que representa o valor que o algoritmo aspira ao chegar de *V*. Este critério se fundamenta no fato de que soluções melhores que a solução S^* corrente, ainda que geradas por movimentos tabu, não foram visitadas anteriormente, evidenciando que a lista de movimentos tabu pode impedir não somente o retorno a uma solução já gerada anteriormente, mas também outras soluções ainda não geradas. (Glover e Laguna, 1993).

Os parâmetros principais de controle do método de busca tabu são:

- A cardinalidade de *T* (lista tabu);
- A função de aspiração *A*;
- A cardinalidade do conjunto *V* de soluções vizinhas testadas em cada iteração;
- BTmax, o número máximo de iterações sem melhora no valor da melhor solução;

5 MÉTODO PROPOSTO

Para resolver o quebra-cabeça Sudoku como um problema de otimização propõe-se o quebra-cabeça em forma de um problema de otimização. Onde inicialmente constrói-se uma solução gerada aleatoriamente respeitando uma condição. A condição de que em cada sub-grade de n x n células devem conter os números inteiros de 1 até n^2 exatamente uma única vez, respeitando os números fixos inicialmente, ou seja, uma solução é construída de forma aleatória preenchendo as células vazias do jogo inicial satisfazendo somente a terceira regra do jogo. (Lewis, 2007).

Através da solução gerada inicialmente transforma-se o quebra-cabeça Sudoku em um problema de otimização em forma de custos, onde o objetivo é minimização. Os custos são obtidos através da somatória das repetições de números nas linhas e colunas que foram gerados na solução inicial aleatória. Isto é, cada número que se repete em uma linha ou coluna é equivalente a um erro, ou custo. Na figura 4 (a) pode-se observar que os números em negrito são as células fixas do jogo inicial, e as demais foram geradas aleatoriamente, sendo que ao final de cada linha e cada coluna encontram-se o número de repetições de números, e estas repetições totalizam 40 repetições para a solução aleatória inicial gerada para o quebra-cabeça da Figura 1.

A partir da formulação do quebra-cabeça Sudoku como um problema de otimização aplicam-se a Metaheurística para resolução do problema de otimização, visando encontrar a solução do problema, isto é a função objetivo igual à zero. Quando o custo da função objetivo é igual à zero significa que na grade do jogo não contém nenhum número repetido nas linhas ou colunas, e todas as regras para resolver o quebra-cabeça estão satisfeitas. A Figura 4 (b) ilustra a solução ótima do problema, ou seja, custo igual à zero.

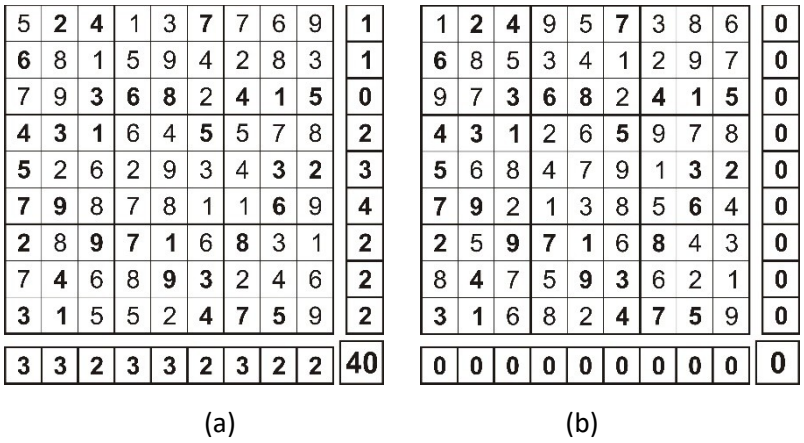

Figura 4. Solução inicial gerada aleatoriamente.

Para utilizar as Metaheurísticas *Simulated Annealing* e busca tabu o operador de vizinhança é a principal rotina, sendo responsável por buscar novas soluções de melhor qualidade. O tópico a seguir descreve o operador de vizinhança utilizado nas Metaheurísticas *Simulated Annealing* e Busca Tabu.

5.1. Operador de vizinhança

Como se observa nas Figuras 2 e 3, os algoritmos do *Simulated Annealing* e a Busca Tabu tem em comum a utilização de um operador de vizinhança para realizar a busca de novas soluções. Então o operador de vizinhança age iterativamente nos algoritmos escolhendo novas soluções, sendo que neste problema uma nova solução é caracterizada por um movimento de troca, onde células não fixas da grade são trocadas uma pela outra visando provocar uma mudança no custo da função objetivo, e assim encontrar uma nova solução. Se esta troca melhora a função objetivo do problema então a mesma é confirmada, em caso contrário o movimento de troca é desfeito. As células são trocadas da seguinte maneira. (Lewis, 2007).

1. Escolha aleatoriamente uma das n^2 sub-grades, isto é, uma das n x n células;
2. Dentro desta sub-grade realizar os seguintes passos:

a. Escolher aleatoriamente duas células da sub-grade em análise e que não tem seus valores fixados.

b. Encontrar a solução vizinha da solução corrente trocando os valores das células identificadas no passo anterior. Assim, a restrição dentro da sub-grade é preservada.

Com este operador de vizinhança os métodos foram implementados e testados com diversos problemas testes. No caso de *simulated annealing* os vizinhos são escolhidos aleatoriamente, mas no caso de tabu search, todos os vizinhos devem ser identificados e avaliados. Na sequência apresentam-se os resultados obtidos pelos métodos.

6 TESTES E RESULTADOS

Os algoritmos propostos foram desenvolvidos em C++ e todos os resultados foram obtidos utilizando um PC Intel Core 2 Duo 1.9 GHz, 2 GB de memória RAM, e sistema operacional Windows 7 Ultimate 32 bits.

As metodologias propostas se comprovaram com testes realizados com problemas de diferentes níveis de dificuldade. A Figura 5 (casos 1 e 2) ilustram os dois casos de prova para um quebra-cabeça de dimensão 3 (9 x 9 células) e a Figura 5 (caso 3) ilustra o caso de prova para um quebra-cabeça de dimensão 4 (16 x 16 células).

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

(Caso 1)

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

(Caso 2)

			1		12	3	7			2	14	15		6	11
10	12			8		4	13	6	0		11				
		2					1				10		4	12	
6	13			15		2			12		3	0	1	10	8
		7				13	2	0			5		11	15	
			14	9	7	10		12		6	1	5	13		
4	11	12				6	3	15	9				10	0	2
13	6			0	5				7	4		9			12
8			6		15	11				12	9			13	14
5	4	10				9	6	14	13				2	11	1
		3	11	7					1	10	0	8			
	14	13		5			8	3	6				15		
14	0	15	4	1		7			2		12			3	6
	8	9		13				1					5		
				2		5	14	8	3		7			9	15
11	3		12	6	4			9	10	14		1			

(Caso3)

Figura 5. Casos de prova.

Os resultados obtidos com os casos de prova são descritos a seguir.

6.1. Resultados para algoritmo *Simulated Annealing*

Para resolver os casos de prova apresentados na Figura 5 foram utilizados os parâmetros apresentados na Tabela 1 a seguir.

Tabela 1. Parâmetros utilizados.

Parâmetros	**Caso 1 e 2**	**Caso 3**
Alfa	0,75	0,9
SAmax	8	10
T_0	120	150

Onde Alfa é a taxa de decremento da temperatura ao longo das iterações do algoritmo, SAmax é o número máximo de iterações a cada valor de temperatura e T_0 é a temperatura Inicial.

Com a configuração apresentada na tabela 1 o algoritmo *Simulated Annealing* encontrou a solução ótima dos dois casos de prova para o quebra-cabeça de ordem 3, e para o caso de prova de ordem 4. Os resultados são apresentados na tabela 2.

Tabela 2. Resultados obtidos.

Resultados	**Caso 1**	**Caso 2**	**Caso 3**
Iterações	760	948	10220
Tempo	0,723 segundos	1,021 segundos	12,432 segundos

A Figura 6 ilustra o gráfico de convergência da função objetivo obtido pelo algoritmo *Simulated Annealing* para os três casos testados.

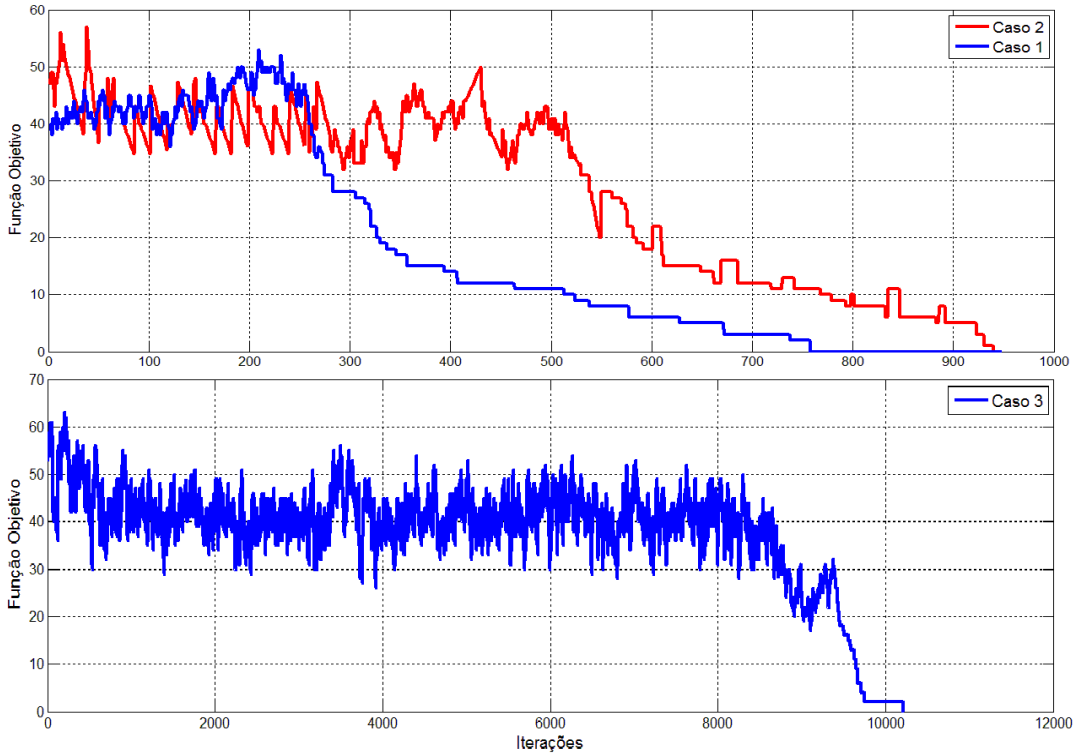

Figura 6. Convergência da função objetivo para todos os três casos.

6.2. Resultados para algoritmo Busca Tabu

Para resolver os casos de prova apresentados na Figura 5 foram utilizados os parâmetros apresentados na Tabela 3 a seguir.

Tabela 3. Parâmetros utilizados.

Parâmetros	Caso 1 e 2	Caso 3
Tamanho da Lista Tabu	25	35
BTmax	20	20

Onde o tamanho da lista tabu é a quantidade de elementos que podem ser armazenados na lista, para evitar que esta solução possa ocorrer novamente, e BTmax é o número máximo de iterações que possa ocorrer sem melhora na solução.

Com a configuração apresentada na tabela 3 o algoritmo da Busca tabu encontrou a solução ótima dos dois casos de prova para o quebra-cabeça de ordem 3, e para o caso de prova de ordem 4. Os resultados são apresentados na tabela 4.

Tabela 4. Resultados obtidos.

Resultados	**Caso 1**	**Caso 2**	**Caso 3**
Iterações	616	703	8756
Tempo	0,592 segundos	0,712 segundos	10,174 segundos

A Figura 7 ilustra o gráfico de convergência da função objetivo obtido pelo algoritmo de Busca tabu para os três casos testados.

6.3. Análise dos Resultados

Os resultados obtidos na resolução dos três casos apresentados neste artigo são satisfatórios, tendo em vista que as Metaheurísticas geralmente encontraram as soluções ótimas de cada um dos testes. Neste caso a solução encontrada pelos dois métodos sempre será a ótima, e o que se difere entre um método e outro é o desempenho computacional.

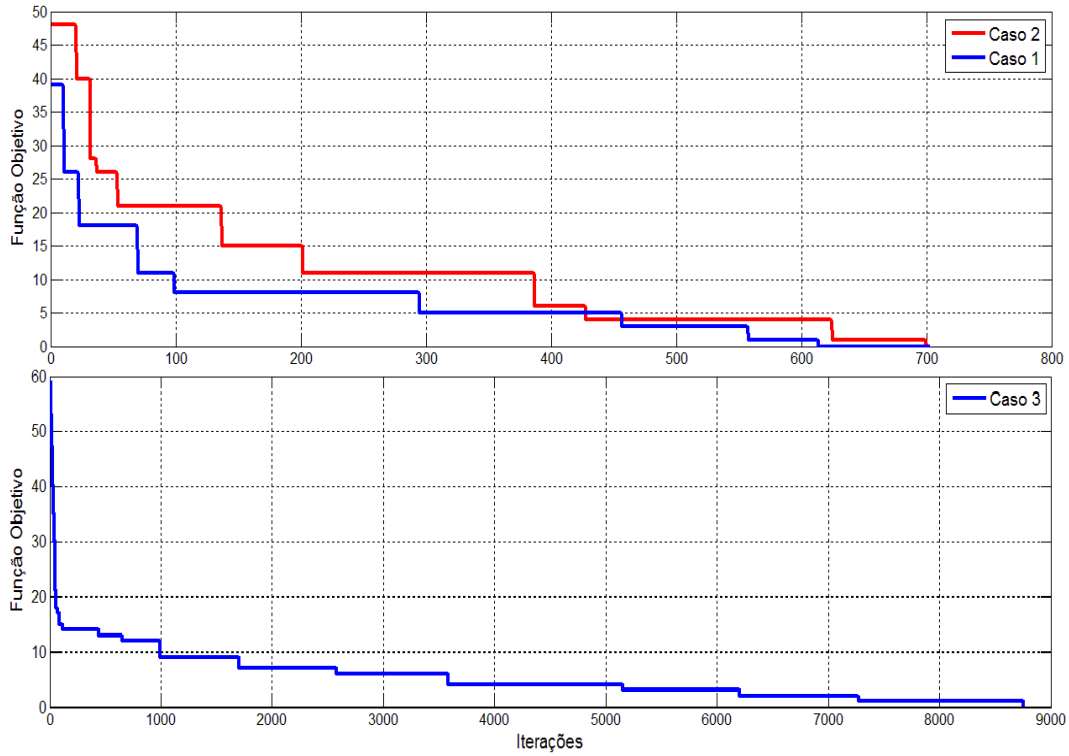

Figura 7. Convergência da função objetivo para todos os três casos.

Através dos resultados observa-se que o algoritmo da busca tabu obteve melhor desempenho computacional que o algoritmo *Simulated Annealing*. A justificativa para isto é o funcionamento dos métodos, o algoritmo do *Simulated Annealing* depende da queda da temperatura para convergir em soluções de vizinhança que sejam de melhoras, no entanto em níveis altos de temperatura o algoritmo aceita soluções de piora para escapar de ótimos locais, isto faz com que o algoritmo consuma mais tempo computacional. O algoritmo da busca tabu tem uma vantagem em relação ao *Simulated Annealing*, pois o mesmo tem uma lista onde são armazenadas características dos movimentos realizados onde encontrou-se uma solução de piora, e esta lista permite que os próximos movimentos com estas características sejam classificados como tabu, ou seja, proibidos de serem executados, esta função no algoritmo da busca tabu proporciona melhor desempenho computacional durante a execução do algoritmo.

Outro aspecto que deve ser ressaltado são os parâmetros utilizados nos métodos, sendo que os mesmos foram encontrados através de testes realizados com vários quebra-cabeças (de diferentes níveis de dificuldade e dimensão), e os

parâmetros apresentados no artigo são os que encontraram o melhor desempenho computacional na resolução de todos os casos durante a fase de testes. Vale a pena enfatizar que qualquer parâmetro utilizado nos métodos encontrará a solução, porém o tempo computacional pode não ser viável. Assim recomenda-se utilizar os parâmetros apresentados neste artigo, pois eles tiverem os melhores desempenhos computacionais em todos os testes.

7 CONCLUSÃO

Neste artigo foram apresentados dois métodos para resolução do quebra-cabeça Sudoku através de Metaheurísticas. Foram descritas as principais etapas e características dos algoritmos Simulated Annealing e a busca tabu e a aplicação dos mesmos no problema proposto. Para validação dos algoritmos foram realizados vários testes com quebra-cabeças de diferentes níveis de dificuldade e dimensão. Os algoritmos propostos apresentaram excelentes resultados encontrando a solução dos quebra-cabeças em um curto tempo computacional, tendo em média uma execução inferior a 1 segundo para quebra-cabeças de dimensão 3 (9x9 células), e em média uma execução inferior a 13 segundos para quebra-cabeças de dimensão 4 (16x16 células).

Os parâmetros utilizados nos métodos de resolução do problema influenciam diretamente no desempenho computacional, portanto para que os algoritmos tenham bons desempenhos computacionais os parâmetros devem ser ajustados precisamente. Neste trabalho os parâmetros foram ajustados através de vários testes realizados com diversos quebra-cabeças (os apresentados no artigo e outros), durante estes testes observou-se o tempo computacional e o número de iterações dos métodos na resolução do problema, e assim escolheram-se os parâmetros que tiveram um melhor desempenho computacional em todos os quebra-cabeças testados.

Sendo assim, conclui-se que o uso de Metaheurísticas para resolver o quebra-cabeça Sudoku tem um desempenho satisfatório nos testes realizados, e a metodologia proposta é bastante confiável para encontrar a solução dos jogos. Na sequência desta pesquisa pretende-se realizar uma modificação no operador de vizinhança apresentado, visando tornar a aplicação mais competitiva (eficiência, confiabilidade e tempo computacional).

8 REFERÊNCIAS

Babu, P. Pelckmans, K. and Stoica, P. (2010). Linear Systems, sparse solutions, and Sudoku. IEEE Signal Processing Letters, Vol. 17. Pg. 40-43.

Bartlett , A. and Langville, A. (2008). An integer programming model for the Sudoku problem. Online Math. Applicat., vol. 8.

Davis, T. (2010). The mathematics of Sudoku. Disponível em: http://www.geometer.org/mathcircles/sudoku.pdf.

Delahaye Jean-Paul. “The science behind Sudoku”. Scientific American. 2006. Pg. 80-87.

Gallego, R. R. A.; Escobar, Z. A. e Romero, R. "Técnicas de Optimización Combinatorial", Primera edición. grupo de Investigación en Planeamiento de Sistemas Eléctricos. Universidad Tecnológica de Pereira. Pereira abril de 2006.

Glover, F. (1986) “Future Paths for integer Programming and Links to artificial inteligence” Computers and Operations Research, Vol. 13, Nº 5, pg. 533-549.

Glover, F. and Laguna, M. (1993) “Tabu Search, Modern Heuristic Techniques for Combinatorial Problems”. Blackwell, Oxford, pg. 70–150.

Kirkpatrick, S.; Gelett, C. e Vechht, M. (1983). Optimization by simulated annealing. Science 4598, pg. 621-630.

Lewis, R. (2007). Metaheuristics can solve Sudoku puzzles, SpringerLink, Springer Science+Business Media, LLC 2007.

Majumder, A.; Kumar, A.; Das, N. and Chakraborty, N. (2010). The game of Sudoku-Advanced Backtrack approach. IJCNS – International Journal of Computer Science and Network Security, Vol. 10. Pg. 255-258.

Moon, T. K., Gunther, J. H. and Kupin, J. J. (2009) Sinkhorn Solves Sudoku, IEEE Transactions on Information Theory, Vol. 55, Nº 4, Pg. 1741-1746

Souza, M. J. F. (2010). Inteligência computacional para otimização. Notas de Aulas, Departamento de computação Universidade Federal de Ouro Preto. Acesso em 03/2012, Disponível em: http://www.iceb.ufop.br/prof/marcone.

Sudoku Essentials. (2012). Acessado em 04/2012 em: http://www.sudokuessentials.com/history-of-sudoku.html

Sudoku Made Easy. (2012). Acessado em 04/2012 em: http://www.sudokumadeeasy.com/history-of-sudoku/

Sudoku Science. (2006). A popular puzzle helps researchers dig into deep math. IEEE Spectrum, pg. 16-17.

Wikipédia. (2012). Enciclopédia Online. Acessado em 04/2012 em: http://en.wikipedia.org/wiki/Sudoku_algorithms.