METAHEURÍSTICAS SIMULATED ANNEALING E BUSCA TABU APLICADAS NA RESOLUÇÃO DO QUEBRA-CABEÇA SUDOKU

SIMONE S. F. DE SOUZA[1], RUBEN ROMERO[1]

[1] *Departamento de Engenharia Elétrica, Faculdade de Engenharia de Ilha Solteira (FEIS), Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), Caixa postal 31, 15385-000, Ilha Solteira, SP, BRASIL*

E-mails: simonefrutuoso.mat@gmail.com, ruben@dee.feis.unesp.br

Abstract— The Sudoku puzzle is considered an NP-hard problem, and this fact calls the attention of researchers. It is a very popular game, played by millions of people, although it is difficult to solve depending on the initial conditions of the puzzle. This paper presents two methodologies to solve the Sudoku puzzle: the metaheuristics Simulated Annealing and Tabu Search. The algorithms were developed in C++ Builder 6.0, using the graphical interface. To demonstrate the effectiveness of these methodologies were performed several tests, it is possible to prove that the methodologies find the optimal solution for different levels of difficulty: easy, medium, hard (sudoku with dimension 3x3) and super sudoku (sudoku with dimension 4x4) for the sudoku puzzle.

Keywords— Sudoku Puzzle, Metaheuristics, Simulated Annealing, Tabu Search, NP-hard.

Resumo— O quebra-cabeça Sudoku é considerado um problema NP-difícil, e por este fato desperta a atenção de pesquisadores. É um jogo muito popular, jogado por milhões de pessoas, sendo de difícil resolução dependendo das condições inicias do quebra-cabeça. Neste artigo apresentam-se duas metodologias para resolver o quebra-cabeça Sudoku: as metaheurísticas *Simulated Annealing* e Busca Tabu. Os algoritmos foram desenvolvidos na linguagem C++ Builder 6.0, utilizando interface gráfica. Para evidenciar a eficácia destas metodologias foram realizados vários testes, sendo possível comprovar que as metodologias encontram a solução ótima para diferentes níveis de dificuldade: fácil, médio, difícil (sudoku com dimensão 3x3) e super-sudoku (sudoku com dimensão 4x4) para o quebra-cabeça sudoku.

Palavras-chave— Quebra-cabeça Sudoku, Metaheurísticas, Simulated Annealing, Busca Tabu, NP-difícil.

1 Introdução

O quebra-cabeça Sudoku foi desenvolvido por Howard Garns em 1979, inspirado no conceito matemático do quadrado latino, invenção do matemático suíço Leonhard Euler. O quadrado latino foi utilizado para análises estatísticas e consistia-se em preencher um conjunto de números de forma que não houvesse repetições de números em linhas ou colunas. Além destas regras, Howard Garns adicionou a regra de não poder existir números repetidos em cada sub-grade. Desta forma o quebra-cabeça sudoku foi formulado como o conhecemos atualmente. A partir da primeira publicação no jornal The Times (Londres, Inglaterra) em 2004, o Sudoku se tornou conhecido mundialmente (Sudoku Essentials, 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-difícil, e ajudar no desenvolvimento de alunos e despertar o interesse em matemática (Sudoku Science, 2006). Este fato desperta o interesse em desenvolvimento de metodologias para resolver o quebra-cabeça. Estima-se que existem 6.670.903.752.021.072.936.960 possibilidades para criar diferentes quebra-cabeças Sudoku válidos (Delahaye, 2006).

Na literatura pesquisada 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. No artigo de (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. O artigo de (Lewis, 2007) propõe que o quebra-cabeça Sudoku seja formulado como um problema de otimização e utiliza-se a metaheurística *Simulated Annealing* (SA) para resolver o problema.

Neste artigo apresentam-se duas metodologias baseadas nas metaheurísticas SA e Busca Tabu (BT) para resolver o problema do quebra-cabeça Sudoku. Uma das vantagens em se utilizar estas técnicas é a baixa complexidade computacional. A principal contribuição deste trabalho é o operador de vizinhança proposto com o intuito de reduzir o tempo

computacional e o número de iterações de cada método. De forma a realizar uma comparação entre os dois métodos. Para avaliar as metodologias foram solucionados 1000 instâncias do quebra-cabeça sudoku, sendo 90% quebra-cabeças de ordem 3 (9 x 9 células) nos níveis fácil, médio e difícil, e 10% quebra-cabeças de ordem 4 (16 x 16 células) se tratando do nível super-sudoku. Destaca-se que os algoritmos foram implementados com interface gráfica, visando facilitar a interação com o usuário.

Este artigo esta organizado como a seguir. Na seção 2 descrevem-se as características do quebra-cabeça Sudoku. Na seção 3 apresentam-se as metaheurísticas SA e BT. A metodologia proposta encontra-se na seção 4. Os resultados estão na seção 5 e na seção 6 apresenta-se a conclusão para este trabalho.

2 Sudoku

O Sudoku é um quebra-cabeça simples, composto por uma grade de dimensão n x n dividida em n^2 x n^2 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 (9 células). A Figura 1 ilustra um quebra-cabeça Sudoku.

	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 com dimensão 3x3
Fonte: (Lewis, 2007, p.2).

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

Quando os três critérios 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 de números preenchidos inicialmente. 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 populares se classificam em fácil, médio, difícil e super-sudoku. O que difere um nível do outro é a quantidade de números inicialmente preenchidos e a ordem do jogo (3x3, 4x4).

3 Metaheurísticas

As metaheurísticas são técnicas de solução que gerenciam uma interação entre as estratégias de busca local e as estratégias de nível superior para criar um processo de otimização com capacidade de sair de soluções ótimas locais e realizar uma busca robusta através do espaço de busca (Glover e Kochenberger, 2003). Neste trabalho optou-se por utilizar as metaheurísticas SA e BT devido as suas características especiais de busca em vizinhança.

Na sequência apresentam-se as metaheurísticas SA e BT.

3.1 Simulated Annealing (SA)

A metaheurística SA 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 as soluções do espaço de busca.

O algoritmo da metaheurística SA 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 ◁ SAmax) 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 SA

N(s) representa a vizinhança da solução corrente.

De forma análoga, o algoritmo SA busca realizar uma melhoria na solução, substituindo a solução corrente por uma solução de menor custo em sua vizinhança, sendo a nova solução escolhida de acordo com o custo e o parâmetro *T*, conhecido como a temperatura corrente.

O algoritmo SA a cada iteração gera aleatoriamente uma solução vizinha, a partir da solução corrente. Desta forma, se o custo é menor aceita o movimento passando-se para uma nova solução, caso contrário mantem-se a solução corrente em Δ unidades, e então a probabilidade de se passar para a nova solução é dada pela equação a seguir:

$$e^{-\Delta / T} \qquad (1)$$

onde Δ é a variação do custo ou equivalente (diferença entre o custo da solução corrente e da solução vizinha é sempre positiva). Em altas temperaturas, cada solução vizinha escolhida aleatoriamente tem probabilidade elevada de se transformar na solução corrente, porém à medida que a temperatura diminui então apenas as soluções vizinhas ligeiramente inferiores da solução corrente tem maior probabilidade de se tornar na solução corrente. Quando o algoritmo SA 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.

O algoritmo SA possui como parâmetros principais o Alfa que representa a taxa de decremento da temperatura, o *SAmax* que é o número máximo de iterações a cada valor de temperatura e T_0 que é a temperatura Inicial.

3.2 Busca Tabu (BT)

A metaheurística BT proposta por Fred Glover (Glover, 1986) foi baseada no conhecimento existente no campo da otimização matemática, 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 melhor movimento admissível é aquele com menor custo e restrições tabu na vizinhança da solução corrente. A função de avaliação escolhe o movimento que produz a maior melhoria, ou a menor piora no custo. 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).

Para cada possível valor do custo existe um nível de aspiração *A(f(S))*: uma solução *S'* pode ser gerada se *f(S')<A(f(S))*, mesmo que o movimento *m* esteja na lista tabu. 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).

O algoritmo da metaheurística BT é apresentado na Figura 3 (Gallego et al., 2006).

```
1   S* ← S;            {Melhor solução obtida até então}
2   Iter ← 0;          {Contador de iterações}
3   MelhorIter← 0;     {Iteração mais recente que forneceu S*}
4   seja BTmax o número máximo de iterações sem melhora de S*
5   T ←0;              {Lista tabu}
6   inicialize a função de aspiração A;
7   enquanto ( IterT – MelhorIter≤ BTmax ) faça
8       Iter ← Iter +1;
9       gere um vizinho qualquer S'∈ N(s);
10      seja S ← S'⊕m o melhor vizinho, 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      se ( f(s') ◁ f(s*); ) então
13              S* ← S';
14              MelhorIter← Iter;
15      fim se
16      Atualize a função de aspiração A;
17  fim enquanto
18  S ← S*;
19  Retorne S;
```

Figura 3. Algoritmo BT

Os parâmetros principais de controle do método de BT são:

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

4 Metodologia Proposta

Nesta seção propõe-se o quebra-cabeça em forma de um problema de otimização. Então, inicialmente uma proposta de solução é gerada aleatoriamente respeitando apenas uma condição, isto é, em cada sub-grade de *n* x *n* células deve conter os números inteiros de 1 até n^2 exatamente uma vez, respeitando as células fixas do quebra-cabeça. Assim a solução é construída de forma aleatória preenchendo as células vazias do jogo inicial satisfazendo somente a terceira regra do jogo (Lewis, 2007).

Ao realizar este processo o quebra-cabeça sudoku se transforma em um problema de otimização de minimização de custos. 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 custo. Na Figura 4 (a) as células preenchidas na cor cinza, com os números em negrito são as células fixas do jogo inicial, e as demais foram geradas aleatoriamente fazendo parte da proposta de solução inicial, assim ao final de cada linha e cada coluna encontra-se o número de repetições de números, e estas repetições totalizam 40 custos para a solução inicial gerada para o quebra-cabeça apresentado na Figura 1.

A partir da formulação do quebra-cabeça sudoku como um problema de otimização aplicam-se as metaheurística para solucionar o problema, isto é, obter um custo igual à zero. Quando o custo é igual à zero significa que na grade do jogo não contém nenhum número repetido nas linhas ou colunas, e todas as regras (restrições) para resolver o quebra-cabeça estão satisfeitas. A Figura 4 (b) ilustra a solução ótima do quebra-cabeça da Figura 1, onde se destaca o custo igual à zero.

5	**2**	**4**	1	3	**7**	7	6	9	**1**
6	8	1	5	9	4	2	8	3	**1**
7	9	**3**	**6**	**8**	2	**4**	**1**	**5**	**0**
4	**3**	**1**	6	4	**5**	5	7	8	**2**
5	2	6	2	9	3	4	**3**	**2**	**3**
7	**9**	8	7	8	1	1	**6**	9	**4**
2	8	**9**	**7**	**1**	6	**8**	3	1	**2**
7	**4**	6	8	**9**	**3**	2	4	6	**2**
3	**1**	5	5	2	**4**	**7**	**5**	9	**2**
3	**3**	**2**	**3**	**3**	**2**	**3**	**2**	**2**	**40**

(a)

1	**2**	**4**	9	5	**7**	3	8	6	**0**
6	8	5	3	4	1	2	9	7	**0**
9	7	**3**	**6**	**8**	2	**4**	**1**	**5**	**0**
4	**3**	**1**	2	6	**5**	9	7	8	**0**
5	6	8	4	7	9	1	**3**	**2**	**0**
7	**9**	2	1	3	8	5	**6**	4	**0**
2	5	**9**	**7**	**1**	6	**8**	4	3	**0**
8	**4**	7	5	**9**	**3**	6	2	1	**0**
3	**1**	6	8	2	**4**	**7**	**5**	9	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

(b)

Figura 4. Solução (a) inicial e (b) final.

Uma característica das metaheurísticas SA e BT é a utilização de um operador de vizinhança para caminhar de solução em solução no espaço de busca do problema. O operador de vizinhança é responsável por realizar este processo gerando novas soluções na vizinhança da solução corrente. O tópico a seguir descreve o operador de vizinhança proposto neste trabalho utilizado nas metaheurísticas SA e BT.

4.1 Operador de Vizinhança

O operador de vizinhança é o responsável por gerar novas soluções na vizinhança da solução corrente. Neste contexto visando proporcionar inteligência, rapidez, eficiência e robustez no processo de busca em vizinhança, propõe-se uma estratégia baseada no controle dos movimentos de troca que geram os novos vizinhos.

Uma nova solução vizinha é caracterizada por trocas de células a partir da solução corrente, isto é, movimentos que geram uma nova solução modificada. No quebra-cabeça sudoku só é possível realizar troca de células não fixas na grade, onde o objetivo é provocar uma mudança no custo.

Durante o processo iterativo das metaheurísticas SA e BT identifica a quantidade de células que possuem valores repetidos nas linhas e colunas da grade. Sendo possível incorporar uma rotina sem custo computacional adicional que armazena quais as células da grade do jogo que possuem valores repetidos. Para este processo utiliza-se uma grade auxiliar de dimensão n^2 *x* n^2 com n^4 células. Desta forma durante a rotina ao calcular o custo total, identifica e armazena na grade auxiliar o status das células não fixas da grade do jogo.

O status das células não fixas representa sua condição no jogo, isto é, se a célula possui valores que geram custo (célula problemática) ou se a célula possui valores que não geram custo (célula não problemática). Com esta informação armazenada é possível controlar os movimentos de troca no momento de gerar um novo vizinho a cada iteração das metaheurísticas.

Diferentemente do que foi realizado em (Lewis, 2007), o operador de vizinhança proposto neste artigo não escolhe células aleatoriamente para serem trocadas na grade do jogo. O operador de vizinhança utiliza a grade auxiliar montada a cada iteração como uma memória de conhecimento para "forçar" movimentos de troca que possibilitem reduzir o custo total. Estes movimentos são escolhidos de forma inteligente.

Ao criar movimentos de trocas controladas, utilizando as células problemáticas, o processo se torna mais eficiente, pois trocando estas células o custo é reduzido rapidamente.

A Figura 5 ilustra a grade auxiliar com as células que geram custos para a solução inicial aleatória da Figura 4 (a), as células brancas representam as células fixas do jogo, as células azuis representam as células que geram custo (células problemáticas) e as células verdes representam as células que não geram custo (células não problemáticas).

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
19	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**
28	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
37	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**
46	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**
55	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
64	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**
73	**74**	**75**	**76**	**77**	**78**	**79**	**80**	**81**

Figura 5. Grade auxiliar.

O operador de vizinhança assume os seguintes passos:

1. Escolha aleatoriamente uma das n^2 sub-grades, isto é, uma das *n* x *n* células;
2. Após escolher a sub-grade realize:

a. Se a sub-grade possui somente células problemáticas, então nesse contexto, a troca de duas células problemáticas gera uma solução vizinha;

b. Se a sub-grade possui células problemáticas e células não problemáticas, então a troca de uma célula que gera custo com uma célula que não gera custo, produz uma solução vizinha;

Uma vez escolhida a sub-grade, então no caso do SA os vizinhos são escolhidos aleatoriamente, mas no caso da BT, todos os vizinhos devem ser identificados e avaliados.

4.2 Interface do Solver

Na Figura 6 apresenta-se a interface do solver desenvolvido em C++ Builder 6.0, nesta Figura exemplifica-se a execução do solver para o quebra-cabeça da Figura 1.

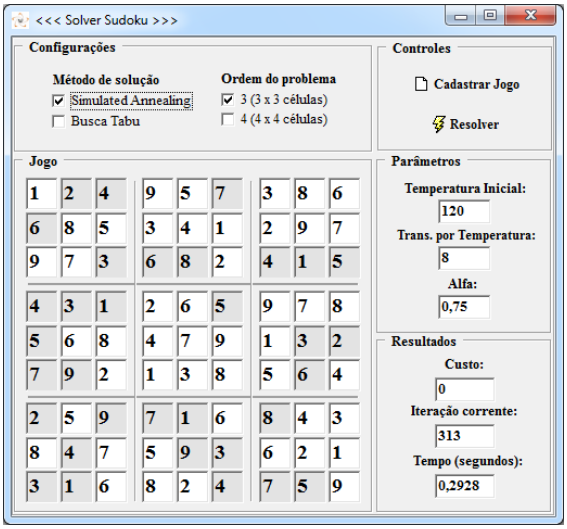

Figura 6. Interface do solver implementado.

No solver pode-se configurar o método para resolver o problema, a ordem e os parâmetros (que varia dependendo do método). Ao resolver o quebra-cabeça o solver apresenta iterativamente as trocas no visual, o número de iterações, o tempo e o custo. Ressalta-se que quando o custo é igual à zero, a solução do problema é encontrada.

Ao final do processo iterativo o solver exibe um gráfico da convergência do método. A Figura 7 ilustra este gráfico para o exemplo apresentado na interface do solver.

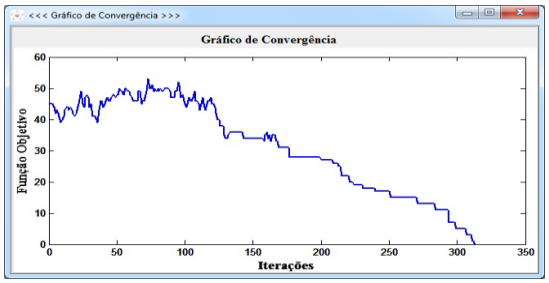

Figura 7. Gráfico de convergência apresentado pelo solver.

5 Resultados

Os algoritmos propostos foram desenvolvidos em C++ Builder versão 6.0 devido a sua facilidade de implementação e componentes para construção da interface gráfica. 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.

Para avaliar as metodologias foram utilizadas 1000 instâncias do quebra-cabeça sudoku retiradas das revistas Coquetel sudoku (Coquetel sudoku, 2013), sendo 900 quebra-cabeças de ordem 3, com níveis fácil (300), médio (300) e difícil (300), e 100 quebra-cabeças de ordem 4, se tratando do nível super-sudoku.

5.1 Resultados para o SA

Para resolver as 1000 instâncias do quebra-cabeça sudoku foram utilizados os parâmetros apresentados na tabela 1.

Tabela 1. Parâmetros utilizados.

Parâmetros	Ordem 3	Ordem 4
Alfa	0,75	0,9
SAmax	8	10
T_0	120	150

Os valores dos parâmetros foram escolhidos através de uma fase experimental de testes. Utiliza-se como critério de parada um número máximo de iterações ou quando o custo for igual a zero. Para os níveis fácil, médio, difícil e super-sudoku foram utilizados como número máximo de iterações 1000, 1100, 1200 e 10000 respectivamente.

Com a configuração apresentada na tabela 1 o algoritmo SA foi executado para as 1000 instâncias do quebra-cabeça sudoku e os resultados computacionais médios são apresentados na tabela 2.

Tabela 2. Resultados computacionais (médios) obtidos pelo SA.

Resultados computacionais	Fácil	Médio	Difícil	Super-sudoku
Iterações	322	388	413	2132
Tempo (segundos)	0,314	0,382	0,431	92,632
Instâncias testadas	300	300	300	100

Para todas as instâncias testadas foi encontrada a solução ótima, isto é, custo igual a zero. Na tabela 2 apresentam-se os resultados computacionais médios, ou seja, a média do número de iterações e o tempo gasto pelo método ao encontrar a solução ótima, para todas as instâncias.

O método apresentou melhor desempenho computacional quando comparado com a literatura (Lewis, 2007), (Majumder et al., 2010) e (Dittrich et al., 2010).

5.2 Resultados para a BT

Foram utilizados os parâmetros apresentados na tabela 3, para resolver as 1000 instâncias do quebra-cabeça sudoku.

Tabela 3. Parâmetros utilizados.

Parâmetros	Ordem 3	Ordem 4
Tamanho da Lista	25	35
BTmax	20	20

Através de testes realizados foram escolhidos os valores dos parâmetros. O critério de parada é realizado quando o custo for igual à zero ou por um número definido de iterações. Foram utilizados para os níveis fácil, médio, difícil e super-sudoku um número definido de iterações 900, 1000, 1100 e 8000 respectivamente.

O algoritmo BT foi executado para as 1000 instâncias do quebra-cabeça sudoku, com os parâmetros apresentados na tabela 3 e os resultados computacionais médios são apresentados na tabela 4.

Tabela 4. Resultados computacionais (médios) obtidos pela BT.

Resultados computacionais	Fácil	Médio	Difícil	Super-sudoku
Iterações	269	315	371	1353
Tempo (segundos)	0,217	0,253	0,302	61,445
Instâncias testadas	300	300	300	100

Em todas as instâncias testadas foi encontrado custo igual à zero. A média do número de iterações e o tempo gasto pelo método ao encontrar a solução ótima, para todas as instâncias, são mostrados na tabela 4.

Comparado com a literatura (Lewis, 2007), (Majumder et al., 2010) e (Dittrich et al., 2010), o método apresentou melhor desempenho computacional.

Ao observar os resultados apresentados percebe-se que o algoritmo da BT possui uma vantagem em relação ao SA, isto devido a uma estratégia do próprio método, a lista tabu. Esta lista permite que os movimentos de troca que levam a uma solução de má qualidade sejam proibidos de serem executados, permitindo encontrar somente soluções de melhor qualidade na vizinhança.

6 Conclusão

Neste artigo foram apresentados dois métodos para resolução do quebra-cabeça Sudoku através das metaheurísticas SA e a BT. Para validação dos algoritmos foram realizados vários testes com quebra-cabeças de diferentes níveis de dificuldade e ordem. Os algoritmos propostos apresentaram excelentes resultados, uma vez que foi encontrada a solução ótima para todas as instâncias testadas com pouco esforço computacional e um número reduzido de iterações. Destaca-se que a solução encontrada pelos dois métodos são sempre a mesma (com custo igual a zero), e o que difere um método do outro é o desempenho computacional (número de iterações e tempo de processamento).

O operador de vizinhança proposto neste trabalho tem por objetivo realizar movimentos inteligentes de troca que possibilitem a redução do custo da solução corrente, desta forma, esta estratégia, contribuiu para tornar o processo mais competitivo, de forma a reduzir o tempo computacional e o número de iterações, proporcionando eficiência, rapidez, confiabilidade e robustez na resolução das instâncias do quebra-cabeça sudoku.

O solver apresentado é de fácil utilização, uma vez que foi implementado com interface gráfica, o que facilita a interação com os usuários e possibilita um ambiente intuitivo.

Agradecimentos

Os autores agradecem a CNPq pelo apoio financeiro de pesquisa.

Referências Bibliográficas

Babu, P.; Pelckmans, K. and Stoica, P. (2010). Linear Systems, sparse solutions, and Sudoku. IEEE Signal Processing Letters, Vol. 17. Pg. 40-43. DOI: 10.1109/LSP.2009.2032489

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

Coquetel Sudoku. (2013). Disponível em: http://coquetel.uol.com.br/

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

Delahaye, J. P. (2006). The science behind Sudoku. Scientific American. Pg. 80-87. Disponível: http://www.lifl.fr/~delahaye/dnalor/SudokuScia m2006.pdf DOI: 10.1038/scientificamerican0606-80

Dittrich, M.; Preuber, T. B and Spellek, R. G. (2010) Solving Sudokus through an Incidence Matrix. Field Programmable Technology – FTP2010, p. 465-469.

Gallego, R. R. A.; Escobar, Z. A. and Romero, R. Técnicas de Optimización Combinatorial. Primera edición. Grupo de Investigación en Planeamiento de Sistemas Eléctricos. Universidad Tecnológica de Pereira - UTP. Pereira, 2006.

Glover, F. (1986) Future Paths for integer Programming and Links to artificial intelligence. Computers and Operations Research, Vol. 13, Nº 5, pg. 533-549. DOI: 10.1016/0305-0548(86)90048-1

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

Glover, F.; Kochenberger, G. Handbook of metaheuristics. Kluwer Academic Publishers. 570 p. 2003.

Kirkpatrick, S.; Gelett, C. and 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 DOI: 10.1109/TIT.2009.2013004

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

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

