ALGORITMO BUSCA TABU PARA A SOLUÇÃO DO NONOGRAMA

Marco César Goldbarg
Universidade Federal do Rio Grande do Norte
Campus Universitário Lagoa Nova, Natal, RN
marcocgold@gmail.com

Camila Nascimento de Oliveira Taumaturgo
Universidade Federal do Rio Grande do Norte
Campus Universitário Lagoa Nova, Natal, RN
milinhano@gmail.com

Elizabeth Ferreira Gouvêa Goldbarg
Universidade Federal do Rio Grande do Norte
Campus Universitário Lagoa Nova, Natal, RN
beth@dimap.ufrn.br

RESUMO

O Nonograma é um jogo lógico cujo problema de decisão associado é NP-completo. Ele possui aplicação em problemas de identificação de padrões e de compactação de dados, dentre outros. O jogo consiste em determinar uma alocação de cores em pixels distribuídos em uma matriz $n \times m$ atendendo restrições em linhas e colunas. Um Nonograma é codificado através de vetores cujos elementos especificam o número de pixels existentes em cada coluna e linha de uma figura, sem especificar suas coordenadas. Este artigo apresenta um algoritmo Busca Tabu para resolver o nonograma. O algoritmo é testado em 16 casos teste e seus resultados são comparados aos de um algoritmo evolucionário do estado da arte do problema.

PALAVARAS CHAVE. Nonograma, Busca Tabu, Metaheurística.

Metaheurísticas

ABSTRACT

Nonogram is a logical puzzle whose associated decision problem in NP-complete. It has applications in pattern recognition problems and data compression, among others. The puzzle consists in determining an assignment of colors to pixels distributed in an $n \times m$ matrix satisfying line and column constraints. A Nonogram is encoded by a vector whose elements specify the number of pixels in each row and column of a figure, without specifying their coordinates. This paper presents a Tabu Search algorithm to solve nonograms. The algorithm is applied to 16 instances and its results are compared to the ones produced by a state-of-art evolutionary algorithm.

KEYWORDS. Nonogram, Tabu Search, Metaheuristics.

Metaheuristics

1. Introdução

Os jogos de raciocino lógico que possuem natureza combinatória têm recebido uma crescente atenção na Ciência da Computação em virtude do avanço do estado da arte dos algoritmos e do fato de que diversos deles possuem importantes aplicações práticas. A possibilidade de desenvolvimento de algoritmos computacionais capazes de, primeiramente, enfrentar o raciocínio humano com razoável chance de vitória permeou a pesquisa nessa área nos anos 90. É emblemático o confronto entre Garry Kasparov e o supercomputador Deep Blue da IBM no xadrez. No primeiro confronto, em 1996, Deep Blue surpreendeu o mundo ao vencer uma partida contra Kasparov, mas o ex-campeão venceu três jogos e empatou dois, vencendo a disputa de seis rodadas. No ano seguinte, em uma revanche, o desafio foi concluído com uma vitória de Kasparov, três empates e duas vitórias de Deep Blue no dia 11 de maio de 1997. Entre os dias 11 e 18 de Novembro de 2003 na cidade de New York novo desafio se deu entre Kasparov e o computador X3D Fritz, um equipamento muito mais simples constituído por 4 processadores Intel Pentium Xeon com 2.8GHz. Nesse caso Kasparov ganhou uma, o computador X3D Fritz outra, e duas partidas terminaram empatadas. Tais partidas deixaram clara a capacidade competitiva dos algoritmos computacionais na solução desse tipo de situação. Com o auxílio da interface computacional e dos recentes meios de mídia novos jogos combinatórios também se popularizaram ao longo da década de 90. Vários desses jogos, em suas diversas versões, já tiveram sua complexidade de solução analisada como o Xadrez (Fraenkel e Lichtenstein, 1981), Damas (Frankel et al., 1978), Go (Lichtenstein e Sipser, 1980), Sokoban (Dor e Zwick, 1999), Soduko (Yato e Seta, 2003), Minesweeper (Kaye, 2000), Tantrix (Holzer e Holzer, 2004), Ciclomania (Biedl et al., 2002), Lemmings (Cormode, 2004), Spiral Galaxies (Friedman, 1992), Tertrix (Hoogeboom e Kosters, 2004), KPlumber (Kral et al. 2004), Nurikabe (McPhail, 2003), Light Up (McPhail, 2005), Mastermind (Stuckman e Zhang, 2006), Slither Link (Yato, 2000), Kakuro (Takahiro, 2001), Reflection Puzzles (Kempe, 2003), dentre muitos outros.

O encaminhamento de solução de jogos de tabuleiros através de técnicas algorítmicas tem sido proposto há anos (Laird, 2001; Herik et. al., 2000; Khoo e Zubek, 2002). Por outro lado, o interesse na aplicação de abordagens heurísticas para resolver quebra-cabeças e jogos tem sido crescente (Vaccaro e Guest, 2005; Jefferson et. al., 2006; Smith, 2007; Lucas e Kendal, 2006).

O presente trabalho aborda a solução de um jogo de raciocino lógico associado também ao problema real da otimização de transmissão de imagens, denominado Nonomino. A complexidade de solução dos Nonominos foi examinada por Ueda e Nagao(1996), demonstrando-se que o problema de decisão associado é NP-completo. O presente trabalho propõe um algoritmo Busca Tabu para a solução do problema e compara os resultados obtidos aos resultados de um algoritmo evolucionário híbrido apresentado por Ortiz-Gárcia et. al. (2009). O jogo é definido na Seção 2. O estado da arte dos algoritmos propostos para a solução do Nonograma é apresentado na Seção 3. A Seção 4 apresenta o algoritmo baseado em Busca Tabu proposto para a solução do problema. Na Seção 5 relata-se um experimento computacional de validação desse algoritmo. Finalmente, na Seção 6 são apresentadas conclusões.

2. O Problema de Reconstrução de Nonogramas

O jogo denominado Nonograma ou Nonominos ou *Paint-by-Numbers* consiste em determinar uma alocação de cores em pixels distribuídos em uma matriz $n \times m$ que atenda especificações em linhas e colunas. Existem dois tipos de jogos: preto-e-branco e coloridos. A figura 1(a) apresenta uma grade inicial 5×5 para o jogo preto-e-branco. Neste jogo um bloco é definido como uma sequência contígua de células pretas, delimitado por células brancas ou pelas bordas do tabuleiro. À esquerda das linhas e acima das colunas existem números que identificam o comprimento dos blocos que devem estar separados por pelo menos 1 pixel em branco. Por exemplo, na linha 1 da grade mostrada na figura 1(a) existem duas ocorrências do número 1. Isto significa que naquela linha devem existir dois blocos de comprimento 1 cada, que serão coloridos de preto e devem ter, ao menos, 1 pixel branco entre eles. Se for possível satisfazer todas as

restrições em linha e coluna, o Nonograma tem solução. A figura 1(b) mostra uma solução para o Nonograma da figura 1(a).

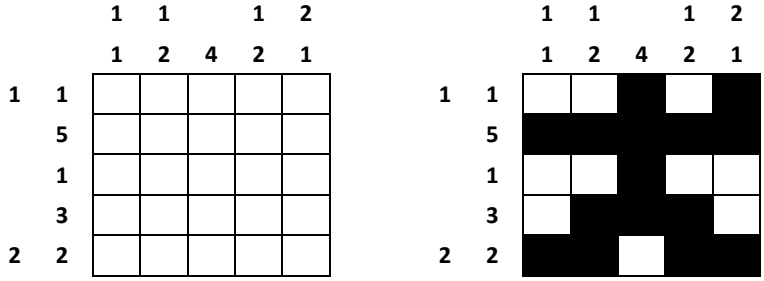


Figura 1: Nonograma preto-e-branco

Nonogramas coloridos seguem regras semelhantes ao jogo preto-e-branco, mas levando em conta que os blocos de mesma cor devem ser separados por pelo menos um bloco em branco e blocos de cor diferente podem ser separados ou não por blocos em branco (Dorant, 2011). A especificação dos blocos de mesma cor é feita em áreas específicas à esquerda das linhas e acima das colunas. A figura 2(a) exemplifica um caso em que devem ser alocados blocos cinza, pretos e brancos. Neste exemplo a alocação dos blocos é feita em uma grade 7×7. Na linha 3 existem 2 ocorrências do número 1 na área cinza e ocorrência dos números 1, 3 e 1 na área branco. Isto significa que 2 blocos cinza, cada um com comprimento 1, serão alocados na linha 3. A alocação dos blocos pretos na mesma linha são especificados pela sequência 1, 3, 1 que estabelece que existirão três blocos pretos na linha, um de tamanho 1, um de tamanho 3 e um de tamanho 1, nesta ordem. Uma solução é mostrada na figura 2(b).

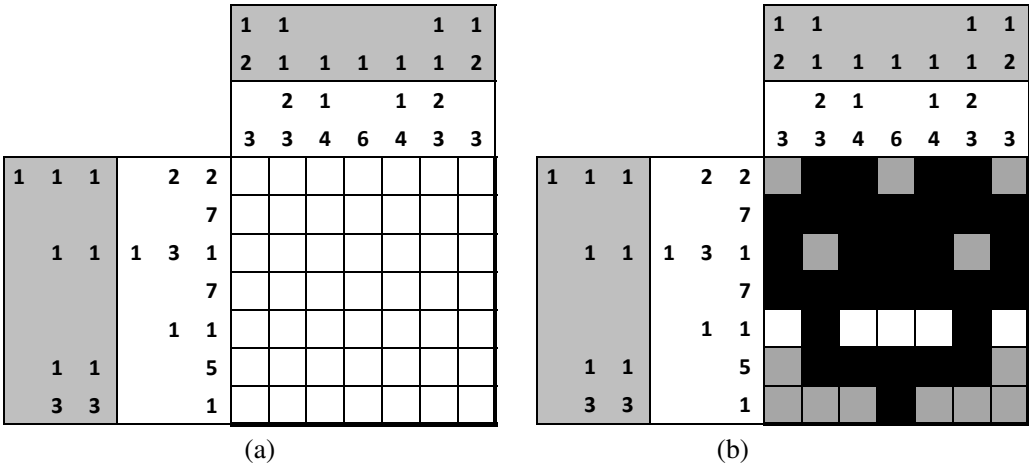


Figura 2: Nonograma colorido

Trata-se, portanto, de um problema de reconstrução de uma figura, dadas informações sobre os pixels dessa figura. Este problema foi demonstrado ser NP-completo por Ueda e Nagao (1996).

3. Estado da Arte na Solução do Nonograma

O problema da formação de imagens, a partir de informações sobre seus pixels, está recebendo intensa atenção, especialmente em anos recentes. Além de constituir em um desafio

lógico, pode implicar na redução da informação necessária à transmissão dos dados de uma figura. Um dos primeiros trabalhos publicados para a solução desse problema deve-se a Bosch (2000) que apresenta uma abordagem por Programação Linear.

Wiggers (2004) propõe um algoritmo genético e compara seu desempenho ao de um algoritmo em *depth first search* (DFS - busca em profundidade). O algoritmo genético proposto por Wiggers (2004) emprega uma população de 100 cromossomos, operando com uma taxa de mutação igual a 5% e uma taxa de *crossover* igual a 60%. Utiliza o método da roleta para realizar a seleção para a reprodução. O operador de mutação do algoritmo é simples e apenas altera o valor de um gene (um posição aleatória) em um cromossomo. Neste artigo, o desempenho dos algoritmos é definido a partir do número de avaliações necessárias para resolver o problema. Os algoritmos são comparados usando cinco tamanhos sucessivos do jogo: grades 5×5 a 10×10. Para tamanhos pequenos do jogo (5×5 e 6×6) o algoritmo o DFS encontra o resultado em um menor número de avaliações. Para tamanhos maiores o algoritmo genético encontra o resultado com menos avaliações do que o DFS.

Batenburg e Kosters (2004) desenvolvem um algoritmo memético. O algoritmo é clássico e executa após cada operação de recombinação ou mutação uma busca *hill-climbing* até que a solução atinja um ótimo local. O tamanho da população é definido igual a 1000, o número de filhos que são criados a cada geração é fixo e igual a 500. Os autores relatam que algoritmo resolve um jogo 25×25 em 80 minutos e um 30×30 em 12 horas em um Pentium IV com 2.4 GHz.

Salcedo-Sanz et al. (2007a) implementaram duas abordagem heurísticas *ad-hoc* para o Nonograma, uma combinatória e outra lógica. A primeira, *Combinatoral Ad-hoc heuristic*, baseia-se em tentar possíveis combinações de soluções em cada linha e coluna do jogo. A segunda, *Logic ad-hoc heuristic*, baseia-se em aspectos lógicos do jogo. A segunda heurística é composta por cinco sub-procedimentos lógicos que visam completar o jogo. Os autores utilizam como casos testes os exemplos dos Nonogramas B&W do *site Conceptis Puzzle*. Eles compararam as duas heurísticas propostas com o algoritmo apresentado por Salcedo-Sanz et.al. (2007b) e com o algoritmo de solução do *site Conceptis Puzzle*. O trabalho conclui que a abordagem *Logic ad-hoc heuristic* mostra-se superior às demais.

Salcedo-Sanz et al. (2007b) aplicam na solução do jogo algoritmos evolucionários com aprendizado. O trabalho emprega uma função objetivo para avaliar a configuração corrente do jogo que compara o número de 1`s e 0`s em cada linha e coluna de uma solução do jogo com o número desejado de 1`s e 0`s. Os autores descrevem três diferentes abordagens de solução. Na primeira um algoritmo genético canônico é apresentado, com a probabilidade de *crossover* Pc = 0,6 e a probabilidade de mutação Pm = 0,01. Segundo o relato do trabalho, esta abordagem não obtém um desempenho aceitável. Na segunda abordagem, operadores especiais de recombinação são sugeridos. A solução inicial é criada a partir da localização dos blocos dentro dos limites possíveis do quadro – o mais à esquerda e mais à direita possível, respeitando a viabilidade das colunas. O operador de recombinação troca alelos (que são colunas do tabuleiro de jogo) entre os pais. Segundo o relato do experimento, essa abordagem consegue viabilizar quase todas as alocações de blocos, todavia não consegue fazê-lo para todo o tabuleiro. A última abordagem é a segunda abordagem acrescida de uma busca local (*Hill Climbing*). Essa melhoria no algoritmo consegue fazer com que a solução do jogo seja encontrada. Os autores definiram que o número máximo de avaliações executadas pelos algoritmos evolucionários fosse foi fixado em 50.000 – a população em 100 indivíduos e a execução terminada após 500 gerações.

Ortiz-Gárcia et al. (2008) implementaram um algoritmo híbrido evolucionário-lógico para resolver qualquer tipo de Nonograma de duas cores, incluído aqueles classificados como muito difíceis. Eles propõem o cálculo de uma probabilidade *a priori* para um pixel ser colorido (pintado com a cor preta) ou não. Essas probabilidades são utilizadas para inicialização da população de cromossomos e em um operador de mutação especial. A seleção dos indivíduos é feita através do método da roleta. O operador de recombinação adotado é o de dois pontos. Há um operador de mutação *swap* que desliza os blocos pintados e um segundo operador de mutação denominado *diversity guided mutation*. O algoritmo é aparelhado também com uma busca local

que aplica as regras lógicas definidas em Salcedo-Sanz et. al. (2007a), buscando viabilizar a alocação dos blocos no tabuleiro.

Ortiz-Gárcia et. al. (2009) apresentam uma heurística baseada no mesmo trabalho (Ortiz-Gárcia et al. 2008) e aplicam essa heurística na solução tanto do Nonograma quanto do *Light-up puzzle*. Os autores aplicam sua abordagem a 16 casos teste, tabuleiros 10×10, classificados como difíceis. A comparação é feita com o algoritmo de Salcedo-Sanz (2007b) e eles mostram que a abordagem proposta é mais eficiente que a de comparação.

Batenburg e Kosters (2009) primeiramente consideram relaxações do problema que podem ser solucionadas em tempo polinomial. A partir dessas relaxações, definem pixels e obtém informações sobre relações entre pares de pixels. Essas informações são combinadas em um problema 2-SAT. Esse processo ocorre iterativamente até que a solução seja encontrada.

Jing et al. (2009) propõem um método para a resolução do jogo em duas fases. Na primeira fase 11 regras lógicas são deduzidas e utilizadas para fixar alguns quadrados como pretos ou brancos. Na segunda fase o algoritmo DFS é aplicado para resolver os quadrados desconhecidos remanescentes. O DFS é implementado conjuntamente com uma estratégia *branch and bound* para melhorar o tempo de processamento.

Ochoa et al. (2009) implementam uma abordagem multiobjetivo através de um algoritmo cultural para resolver o Nonograma. O algoritmo cultural implementado emprega uma busca local *hill-climbing*. Os autores apresentam uma solução para um caso teste 20×20 que levou 80 minutos para ser obtida em um Pentium IV com 2.4GHz.

4. Algoritmo Busca Tabu para o Nonograma

A Busca Tabu é referida na literatura como supostamente proposta por Fred Glover na década de 80 (Glover, 1986; Glover, 1989). Ainda que pouco conhecido, Hansen (1986) também apresenta uma proposta semelhante e independentemente desenvolvida denominada *steepest ascent mildest descent*.

A Busca Tabu, via de regra, emprega uma estrutura de memória simples, normalmente uma fila, que controla a diversidade de uma configuração corrente de um processo de busca. Esse controle da configuração é realizado de forma indireta através do controle das alterações que essa configuração sofre ao longo da busca. Controlando as modificações na configuração e não a própria configuração o esforço de memória é aliviado, contudo a revisita de uma mesma configuração torna-se possível. Dessa forma, a técnica propõe mecanismos adicionais de aceitação ou rejeição de uma dada configuração foco de busca.

Finalmente, a Busca Tabu também emprega mecanismos que permitem equilibrar a diversificação e a intensificação da busca através do uso de memória de médio e longo prazo, oscilação, elitização, etc (Glover e Laguna, 1997).

A busca tabu tem sido fracamente explorada na literatura para o desenvolvimento de algoritmos metaheurísticos eficientes para a solução de jogos lógicos. Uma recente exceção é o trabalho de Wang e Chiang (2010) que aplicam a busca tabu para a solução do *Eternity-II puzzles*.

Do conhecimento dos autores, ainda não foi relatado um algoritmo baseado em Busca Tabu para a solução do problema objeto do presente trabalho.

4.1 Algoritmo Tabugrama

O algoritmo tabu proposto baseia-se na ideia de solucionar o desafio de colorir corretamente os pixels da figura considerando a viabilidade garantida em linhas ou em colunas e através da análise de possíveis movimentos para um bloco escolhido.

A solução inicial do algoritmo é construída aleatoriamente dentro dos limites lógicos das alocações de blocos através do procedimento proposto por Ortiz-Gárcia et. al. (2009). Este procedimento busca uma alocação de blocos que atenda a viabilidade em linhas ou em colunas, o que pode ser feito trivialmente. A posição inicial de cada bloco é sorteada aleatoriamente respeitando os possíveis limites de alocação de cada bloco. A figura 3 exemplifica uma solução

inicial respeitando as restrições das linhas. Na figura 3 as linhas 1 e 3 têm 2 blocos e a linha 2 tem 1 bloco. A linha 1 possui um bloco de tamanho 1 e um bloco de tamanho 2. Portanto, na linha 1 os limites para o primeiro bloco correspondem ao intervalo entre a primeira e a segunda células da matriz, intervalo [1,2], uma vez que os blocos não podem ser alocados encostados. Já para o segundo bloco o início do pode estar no intervalo [3,4]. Supondo que o procedimento sorteie a posição inicial para o primeiro bloco na célula 1, como na figura 3, a posição inicial do segundo bloco poderá ser sorteada no intervalo [3,4]. No caso da figura a posição sorteada foi a 3. Caso o primeiro bloco da primeira linha fosse alocado na célula 2, o segundo bloco somente poderia ser alocado a partir da célula 4. A posição inicial do bloco da linha 2 pode ser localizada dentro do intervalo [1, 3]. No caso o bloco da linha 2 foi alocado entre as células 2 e 4. Procede-se de forma semelhante para a terceira linha.

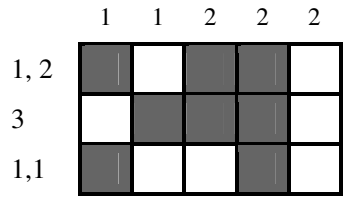


Figura 3: Exemplo de alocação inicial dos blocos

A avaliação de uma solução é feita com a equação 1 quando as restrições em linha são respeitadas. Uma fórmula análoga é utilizada para soluções onde se respeita restrições em coluna. Na equação 1 M é o número de colunas, N é o número de linhas, c_{pk} é o tamanho que deve ter o p-ésimo bloco alocado à coluna k, b_{pk} é o tamanho do p-ésimo bloco que está associado à coluna k na solução, x_{pk} é uma variável binária com valor 1 se o pixel na linha p e coluna k é preto ou com valor 0 caso o pixel seja branco, cb_k é o número de blocos que devem estar associados à coluna k, bb_k é o número de blocos associados à coluna k na solução.

$$f(X) = \sum_{k=1}^{M} \left| \sum_{p=1}^{cb_k} c_{pk} - \sum_{p=1}^{N} x_{pk} \right| + 100 \sum_{k=1}^{M} \sum_{p=1}^{\max\{cb_k, bb_k\}} \left| c_{pk} - b_{pk} \right| \quad (1)$$

O valor de $f(X)$ é zero se a solução for viável. O cálculo da função objetivo é ilustrado com a solução não viável exibida na figura 4.

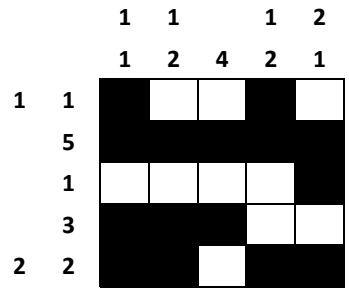


Figura 4: Solução não viável do Nonograma da figura 1

Neste Nonograma tem-se que $M = 5$ e $N = 5$. Os valores de cb_k, para $k = 1,...,5$, são, respectivamente, 2, 2, 1, 2 e 2. O primeiro termo de 1 é calculado por:

$$\sum_{k=1}^{M}\left|\sum_{p=1}^{cb_k} c_{pk} - \sum_{p=1}^{N} x_{pk}\right| = \left|(1+1)-(1+1+0+1+1)\right| + \left|(1+2)-(0+1+0+1=1)\right| +$$
$$\left|4-(0+1+0+1+0)\right| + \left|(1+2)-(1+1+0+0+1)\right| +$$
$$\left|(2+1)-(0+1+1+0+1)\right| = 4$$

No cálculo da segunda parcela, verifica-se, coluna a coluna, o comprimento que cada bloco tem em relação ao que deveria ter. Esse valor recebe uma ponderação 100 vezes maior que a da primeira parcela. O valor de $bb_k = 2$, $k = 1,...,5$, ou seja, existem 2 blocos em cada coluna. Na primeira coluna existem dois blocos com comprimento 2, entretanto, o problema requer que existam dois blocos ambos com comprimento 1. Cada conjunto é verificado separadamente. O primeiro bloco alocado à primeira linha conta a violação de 1 pixel. O mesmo se dá com o segundo bloco. Na coluna 3, deveria existir apenas um bloco de comprimento 4, mas existem dois blocos de comprimento 1 cada. O primeiro bloco da coluna é comparado ao único bloco que deveria ter na coluna. Como deveria existir 4 pixels no primeiro bloco e existe apenas 1, então é contada uma violação de 3 blocos. O segundo bloco da coluna 3 não tem correspondente. Um correspondente fictício de tamanho 0 é assumido neste caso. A violação então é igual a 1, o tamanho do segundo bloco existente na coluna 3. O cálculo da segunda parcela é mostrado a seguir.

$$100\sum_{k=1}^{M}\sum_{p=1}^{\max\{cb_k,bb_k\}}\left|c_{pk}-b_{pk}\right| = 100\left[\left(\left|1-2\right|+\left|1-2\right|\right)+\left(\left|1-1\right|+\left|2-2\right|\right)+\left(\left|4-1\right|+\left|0-1\right|\right)+\right.$$
$$\left.\left(\left|1-2\right|+\left|2-1\right|\right)+\left(\left|2-2\right|+\left|1-1\right|\right)\right]=800$$

Deste modo, temos que o valor da função objetivo da solução mostrada na figura 4 é 804.

Algoritmo TabuGrama

```
sol, solCor, mSol
iterG = 600; Lista_Tabu = { }
Enquanto (iterG > 0) faça
        s = rand(0,1)
        solCor = Solucao_inicial(s)
        cont = 300;
        enquanto (cont > 0) faça
                para iter de 1 até 1000 faca
                        sol = buscalocal(solCor,s, Lista_Tabu);
                        se (f(sol) < f(mSol))
                                mSol = sol;
                        fim se
                fim para
                sol = mudarLine(s);  cont = cont - 1
        fim enquanto
        iterG = iterG - 1
fim enquanto
```

Quadro 1: Pseudo-código do algoritmo Tabugrama

A memória de curto prazo do algoritmo tabu, a lista tabu, controla a escolha do bloco que deverá ser movimentado no quadro do jogo, não propriamente o movimento do bloco. De fato o movimento que será escolhido será o melhor possível, segundo a avaliação da função objetivo. O pseudocódigo do algoritmo tabugrama está descrito no quadro 1. Inicialmente cria-se uma solução no procedimento *Solucao_inicial*( ). A solução criada será viável em linhas ou em colunas de acordo com o parâmetro *s* passado como entrada do procedimento. Esse parâmetro é obtido no procedimento *rand*( ) que retorna 0 ou 1 com equiprobabilidade. Caso s = 0, a alocação será realizada considerando a viabilidade das linhas, e se s = 1será considerada a viabilidade das colunas.

A estratégia de busca tabu é desenvolvida dentro do procedimento *buscalocal*( ) do quadro 1. Este procedimento está detalhado no quadro 2.

Procedimento buscalocal (SolCor,s,Lista_Tabu)

```
Bloco ← Rand_Bloco (SolCor,s,Lista_Tabu)  // Bloco∉ Lista_Tabu //
f(mSol)  ←∞
Para toda a posição disponível de Inserção i
   Sol  ←  Insert (Bloco, SolCor, s, i)
   se (f(Sol) < f(mSol))
          mSol ←Sol
          Tabu ← Bloco;
fim para
Insere_Tabu (Lista_Tabu, Tabu)   // Lista_Tabu possui comprimento 5 //
SolCor  ← mSol
```

Quadro 2: Procedimento de busca tabu

O procedimento buscalocal( ) é desenvolvido da seguinte forma. No procedimento Rand_Bloco( ), escolhe-se, aleatoriamente, um bloco de uma linha (s=0) ou coluna (s=1), também escolhida aleatoriamente. Dentro dos limites possíveis de alocação do bloco escolhido na linha ou coluna, todas as posições possíveis são testadas para alocar o bloco no procedimento Insert( ) dentro do laço. A solução com a melhor alocação do bloco é guardada na variável *mSol*. No final do laço, a posição do bloco é guardada na Lista_Tabu no procedimento Insere_Tabu( ). O procedimento buscalocal( ) retorna a solução em *mSol*. Cada bloco selecionado se torna tabu durante 5 iterações.

A vizinhança explorada na busca tabu do quadro 2 é composta por todas as configurações possíveis de organizar através da movimentação de um bloco – em linha ou em coluna, conforme o tipo de solução corrente inicialmente gerado pelo procedimento do quadro 1. O procedimento tabu não necessita de uma estratégia de aspiração, uma vez que o controle se faz sobre o bloco a ser utilizado para gerar a vizinhança de busca.

Os passos do algoritmo descrito no quadro 1 que antecedem a chamada do procedimento de busca tabu têm por objetivo fornecer soluções iniciais diversificadas para serem exploradas por uma etapa de busca tabu.

O procedimento mudarLine( ) modifica uma linha (s=0) ou coluna (s=1) da solução corrente, escolhida aleatoriamente, utilizando a mesma estratégia para a construção de uma alocação viável de blocos que a empregada para criar uma solução inicial. Sorteia-se uma linha (ou coluna) aleatoriamente e a posição inicial de cada bloco é sorteada respeitando os limites do bloco como exemplificado anteriormente. O objetivo deste procedimento é diversificar as soluções exploradas pelo algoritmo.

5. Experimento Computacional

Todos os experimentos foram realizados em uma máquina com processador AMD Turion II Dual-Core M500 2.20 GHz e 4Gb de RAM. A codificação foi feita utilizando a linguagem C++ e sistema operacional Linux 2.6.26-1-amd64.

Foram utilizadas 16 instâncias classificadas como muito difíceis retiradas do *Benchmark Puzzles* (http://homepages.cwi.nl/~aeb/games/jpuzzlegraded/index.html). Neste site há cerca de 200 instâncias classificadas com estrelas de 1 a 9, as com 9 estrelas indicam os jogos mais difíceis. Neste trabalho utilizamos apenas os nonogramas com 8 e 9 estrelas. Todas as instâncias deste site possuem tamanho 10 x 10. Estas instâncias foram escolhidas por serem as mesmas utilizadas no trabalho de Ortiz-Garcia et al. (2009).

Os resultados são mostrados nas Tabelas 1 e 2. A Tabela 1 mostra a identificação das instâncias e o percentual de vezes que cada algoritmo solucionou a instância correspondente. Os resultados se referem a 50 execuções independentes de cada algoritmo para cada instância.

Tabela 1. Desempenho dos algoritmos TabuGrama e de Ortiz-Gárcia et. al. (2009) em % de soluções encontradas

Instância	**TabuGrama**	Ortiz-Gárcia et. al. (2009)	Instância	**TabuGrama**	Ortiz-Gárcia et. al. (2009)
222	100	100	230	100	100
223	100	100	231	100	100
224	100	100	232	100	98
225	100	100	233	100	94
226	100	100	234	100	96
227	100	100	235	100	100
228	100	100	236	100	96
229	100	100	237	100	100

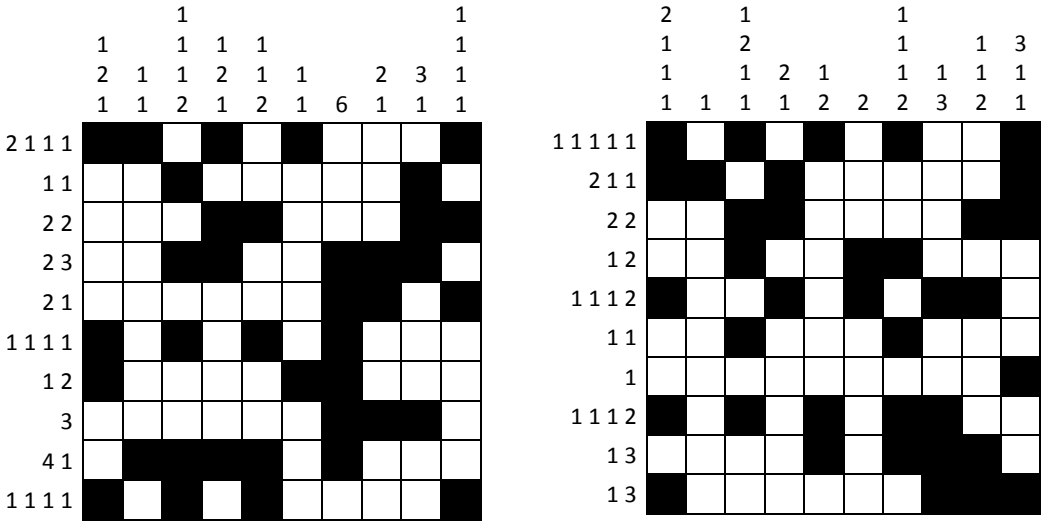
Como pode ser observado na Tabela 1, o algoritmo Tabugrama possui um desempenho qualitativo superior em quatro instâncias do conjunto proposto no trabalho de Ortiz-Gárcia et. al. (2009). O algoritmo de comparação não apresenta desempenho superior ao algoritmo proposto neste trabalho em nenhum caso teste, tendo este último um aproveitamento de 100% no conjunto de instâncias testadas.

Tabela 2. Resultados - Tempo em segundos do algoritmo TabuGrama

Instância	Média	Mediana	Instância	Média	Mediana
222	43,4	24,5	230	102,26	77,5
223	35,12	25	231	39,76	24
224	89,12	72,5	232	229,9	195,5
225	64	54,5	233	270,66	270
226	26,7	15,5	234	226,74	169,5
227	12,66	11	235	26,08	20
228	62,12	46	236	510,88	280
229	511,46	434,5	237	347,6	265,5

A tabela 2 resume os tempos de processamento do algoritmo TabuGrama em segundos. Os tempos de processamento do algoritmo de Ortiz-Gárcia et. al. (2009) não constam do trabalho publicado. Os resultados apresentados na tabela 2 permitem a verificação de que os tempos de execução do algoritmo Tabugrama não são proibitivos, pelo contrário.

As figuras 5(a) e 5(b) exibem as soluções produzidas pelo algortimo Tabugrama para os casos teste 236 e 237, ambos com 9 estrelas.



(a) (b)

Figura 5: Solução dos Nonogramas 9 estrelas (a) 236 e (b) 237

6. Conclusões e Trabalhos Futuros

Os testes preliminares descritos presentemente sugerem que o algoritmo busca tabu proposto no trabalho para a solução do problema de recuperação de nonogramas, mostra um potencial promissor para a solução desse tipo de problema.

Destaca-se a simplicidade do procedimento e seu bom desempenho qualitativo e quantitativo.

Como uma busca local é basicamente uma estratégia que pode ser facilmente composta com diversas outras metaheurísticas, o algoritmo proposto mostra potencial para aparelhar outros métodos mais complexos em busca da solução de problemas de maior porte.

Como trabalhos futuros sugere-se a implementação do algoritmo de Ortiz-Gárcia et. al. (2009) para que a dimensão da demanda em tempo computacional seja comparável em experimentos futuros e novos e maiores conjuntos de instâncias possam compor o teste de validação.

Referências

Batenburg, K. J. e Kosters, W. A. (2004). A discrete tomography approach to Japanese puzzles, *Proceedings of the Belgian-Dutch Conference on Artificial Intelligence*, 243-50.

Batenburg, K. J. e Kosters, W. A. (2009). Solving Nonograms by combining relaxations, *Pattern Recognition*, 42, 1672-1683.

Biedl, T., Demaine, E., Demaine, M., Fleischer, R., Jacobsen, L. e Munro, J. (2002). The Complexity of Clickomania. *More Games of No Chance* (ed. R. Nowakowski), 389-404, Cambridge University Press.

Bosch, R. A. (2000). Painting by numbers, Mathematical Programming Society Newsletter, OPTIMA 65, 16-17.

Cormode, G. (2004). The hardness of the lemmings game, or Oh no, more NP-completeness proofs, *Proceedings of the International Conference on Fun with Algorithms*, 65-76.

Dor, D. e Zwick, U. (1999). SOKOBAN and other motion planning problems, *Computational Geometry: Theory and Applications*, 13(4), 215–228.

Dorant, M. (2011) *Conceptis Puzzle*. Disponível em: <http://conceptispuzzles.com>. Acesso em 08 março 2011.
Fraenkel, A. S., Garey, M. R., Johnson, D. S., Schaefer, T. e Yesha, Y. (1978). The complexity of checkers on an N ×N board - preliminary report, *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, 55–64.
Fraenkel, A. S. e Lichtenstein, D. (1981). Computing a perfect strategy for n×n chess requires time exponential in n, *Journal of Combinatorial Theory*, Series A, 31, 199-214.
Friedman, E. (1992). *Spiral Galaxies Puzzles are NP-complete*, Disponível em: http://www.stetson.edu/˜efriedma/articles/spiral.pdf – acesso em 10 fev 2010.
Glover, F. (1986). Future paths for integer programming and links to artificial intelligence, Computers & Operations Research, 13, 533-549.
Glover, F. (1989). Tabu search – Part I, *ORSA Journal on Computing*, 1, 190-206.
Glover, F, e Laguna, M. (1997). *Tabu Search*, Kluwer Academic Publishers.
Hansen, P. (1986) The steepest ascent mildest descent heuristic for combinatorial programming, *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.
Herik, H. J., Van Den, H. e Iida, H. (2000). *Games in AI Research*. University Maastricht.
Holzer, M. e Holzer, W. (2004). Tantrix rotation puzzles are intractable, *Discrete Applied Mathematics*, 144, 345-358.
Hoogeboom, H. J. e Kosters, W. A. (2004). Tetris and Decidability, *Information Processing Letters*, 89, 267–272.
Jefferson, C., Miguel, A., Miguel, I. e Tarim. A. (2006). Modelling and solving english peg solitaire, *Computers & Operations Research*, 33(10), 2935-2959.
Jing, M-Q., Yu, C-H., Lee, H-L. e Chen, L-H. (2009). Solving japanese puzzles with logical rules and depth first search algorithm, *Proceedings of the Eighth International Conference on Machine Learning and Cybernetics*, Baoding, 2962-2967.
Kaye, R. (2000). Minesweeper is NP-complete, *Mathematical Intelligencer*, 22 (2), 9–15.
Kempe, D. (2003). *On the complexity of the reflections game*. Unpublished manuscript, January 2003. http://www-rcf.usc.edu/_dkempe/publications/reflections.pdf.
Khoo, A. e Zubek, R. (2002). Applying inexpensive AI techniques to computer games AI. *IEEE Intelligent System*, 17(4), 48-53.
Kral, D., Majerech, V., Sgall, J., Tichy, T., eWoeginger, G. (2004). It is tough to be a plumber, *TCS: Theoretical Computer Science*, 313(3), 473–484.
Laird, J. E. (2001). Using a computer game to develop advanced AI. *IEEE Computer*, 34(7), 70-75.
Lichtenstein, D. e Sipser, M. (1980). GO is polynomial-space hard, *Journal of the Association for Computing Machinery*, 27(2), 393–401.
Lucas, S. S. e Kendall, G. (2006). Evolutionary computation and games, *IEEE Computational Intelligence Magazine*, 1(1), 10-18.
McPhail, B. (2003). *The Complexity of Puzzles: NP-Completeness Results for Nurikabe and Minesweeper.* Thesis for BA, Division of Mathematics and Natural Sciences, Reed College.
McPhail, B. (2005). *Light Up is NP-Complete*. Disponível em: http://www.cs.umass.edu/~mcphailb/papers/2005lightup.pdf, acesso em 10 fev 2010.
Ochoa, A. (2009) Resolution of a combinatorial problem using cultural algorithms, *Journal of Computers*, 4(8), 738-741.
Ortiz-García, E. G., Salcedo-Sanz, S., Pérez-Bellido, A. M., Portilla-Figueras, A. e Yao, X. (2008). Solving very difficult japanese puzzles with a hybrid evolutionary-logic algorithm, *Lecture Notes in Computer Science*, 5361, 360-369.
Ortiz-García, E. G., Salcedo-Sanz, S., Pérez-Bellido, Á. M., Carro-Calvo, L., Portilla-Figueras, A. e Yao, X. (2009). Improving the performance of evolutionary algorithms in grid-based puzzles resolution, *Evolutionary Intelligence*, 2(4), 169-181.
Salcedo-Sanz, S., Ortíz-García, E. G., Pérez-Bellido, A. M., Portilla-Figueras, A. e Yao, X. (2007a). Solving japanese puzzles with heuristics, *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games*, 224-231.

Salcedo-Sanz, S., Portilla-Figueras, J.A., Ortiz-Garcia, E.G., Perez-Bellido, A. M. e Yao, X. (2007b). Teaching advanced features of evolutionary algorithms using japanese puzzles, *IEEE Transactions on Education* , 50(2), 151-156.

Smith, K. D. (2007). Dynamic programming and board games: A survey, *European Journal of Operational Research*, 176, 1299–1318.

Stuckman, J. e Zhang, G.-Q. (2006). Mastermind is NP-complete, *INFOCOMP Journal of Computer Science*, 5, 25-28.

Takahiro, S. (2001). *The complexities of puzzles, cross sum and their another solution problems (ASP).* Thesis for BSc, Department of Information Science, University of Tokyo.

Ueda, N. e Nagao, T. (1996). *NP-completeness Results for Nonogram via Parsimonius* Reductions Tecnical Report TR96008 – Departament of Computer Science – Tokyo Institute of Technology. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.5277 acesso em 10 fev 2010.

Vaccaro, M. e Guest, C. C. (2005). Planning an endgame move set for the game RISK: a comparison of search algorithms, *IEEE Transactions on Evolutionary Computation*, (9)6, 641-652.

Yato, T. (2000). On the NP-completeness of the slither link puzzle, *IPSJ SIGNotes Algorithms*, 74, 25-32.

Yato, T. e Seta, T. (2003). Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronic*s, Communications and Computer Sciences, 86(5), 1052-1060.

Wang, W-S. e Chiang, T-C. (2010). Solving Eternity-II puzzles with a tabu search algorithm, *Proceedings of International Conference on Methaheuristics and Nature Inspired Computing*, META 2010.

Wiggers, W. (2004). A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms, *Proceedings of the first Twente student conference on IT*, 1-6.