Resolvendo o puzzle Unruly utilizando backtracking

Pedro H. L. Torres[1], **Pedro O. N. de O. Pinheiro**[1] **Wladimir A. Tavares**[1]

[1]Universidade Federal do Ceará - Campus de Quixadá

torres@alu.ufc.br, pedro.olimpio@hotmail.com, wladimir@lia.ufc.br

Abstract. *This paper presents a resolution of the puzzle Unruly using the backtracking technique. In order to improve performance, we present another technique, where we define a fixed set of rules to be used together with backtracking. Lastly, we establish a comparison between the two techniques.*

Resumo. *Este artigo apresenta a resolução do puzzle Unruly utilizando a técnica de backtracking. A fim de melhorar o desempenho, apresentamos outra técnica, onde definimos um conjunto fixo de regras para serem utilizadas juntas com backtracking. Por fim, estabelecemos um comparativo entre as duas técnicas.*

1. Introdução

O Unruly é um jogo que teve como inspiração o puzzle *Tohu wa Vohu* criado por Adolfo Zanellati[1]. É possível ter acesso ao Unruly por meio da coleção de puzzles de Simon Tatham's[2]. O objetivo do Unruly é preencher um tabuleiro usando as cores branca e preta, iremos definir os detalhes desse jogo na Seção 2.

O puzzle Unruly pode ser modelado como um problema de satisfação de restrição [Russell et al. 1995]. Formalmente, estes problemas são definidos por um conjunto de variáveis X_i com valores de um domínio D_i sujeitos a um conjunto de restrições. Uma das técnicas mais utilizadas para este tipo de problema são os algoritmo de *backtracking* [Russell et al. 1995].

No presente trabalho, usamos duas abordagens de resolução, uma usando o *backtracking*, a outra usando *backtracking* junto a aplicação de um conjunto de regras fixas definidas nesse trabalho. Realizamos uma comparação usando essas duas técnicas para diferentes instâncias, avaliando o tempo para resolver o problema, assim, como o número de nós criados na árvore de recursão.

O restante do artigo é dividido da seguinte forma: a Seção 2 apresenta a definição formal do problema e a introdução das nossas regras fixas. A Seção 3 define a técnica de *backtracking* para o nosso problema, na Seção 4 apresentamos a resolução do problema utilizando o conjunto de regras enquanto aplicamos o *backtraking*. Por fim, na Seção 5, apresentamos os resultados obtidos nesse trabalho e a conclusão.

2. Unruly

Dado uma grade de quadrados dse tamanho $N \times M$, onde $N, M \geq 6$ são inteiros pares, devemos pintar cada quadrado com a cor preto ou branco seguindo duas regras. A regra 1

[1]www.janko.at/Raetsel/Tohu-Wa-Vohu/

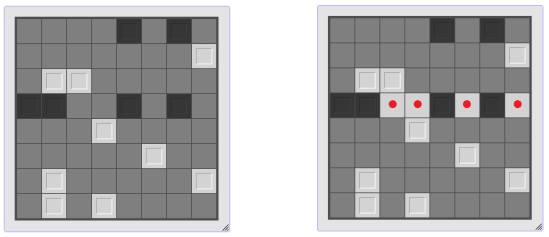
[2]www.chiark.greenend.org.uk/~sgtatham/puzzles/

diz que cada linha e coluna deve conter o número de quadrados pretos igual ao número de quadrados brancos. A regra 2 diz que em uma linha ou coluna não pode existir três quadrados consecutivos de mesma cor. O problema já inicia com alguns quadrados pintados. Todo quadrado que não foi pintado diremos que a cor dele é cinza. No restante do texto, vamos considerar que o oposto da cor branca será a cor preta e vice-versa.

Existem dois níveis de dificuldade, na dificuldade fácil, todas as instâncias do problema podem ser resolvidas apenas aplicando as quatro regras que iremos descrever a seguir. Na dificuldade difícil essas regras não são suficientes, para essas instâncias iremos propor uma solução usando *backtracking*.

2.1. Regra 1 - Linha

A aplicação dessa regra consiste em verificar se dada uma cor C, se existe uma linha na grade que tem o número de quadrados com a cor C igual a $N/2$ (máximo permitido para uma cor). Se a resposta for positiva então pintamos todos quadrados com a cor cinza com a cor oposta a C (chamaremos de $\neg C$) . Aplicamos isto para cada linha que obedece a condição. Um exemplo da aplicação dessa regra pode ser visto na mudança da configuração da Figura 1(a) para a Figura 1(b).

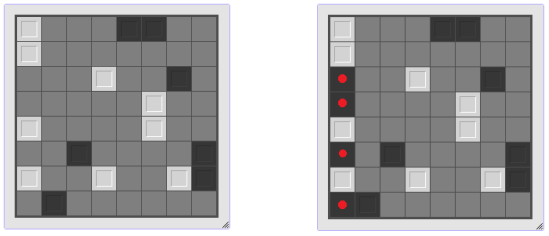


(a) (b)

Figura 1. Aplicação da Regra 1 - Linha

2.1.1. Regra 1 - Coluna

A aplicação dessa regra é semelhante a da regra anterior, com exceção de que no lugar de aplicarmos as linhas, aplicamos as colunas. Um exemplo da aplicação dessa regra pode ser visto na mudança da configuração da Figura 2(a) para a Figura 2(b).

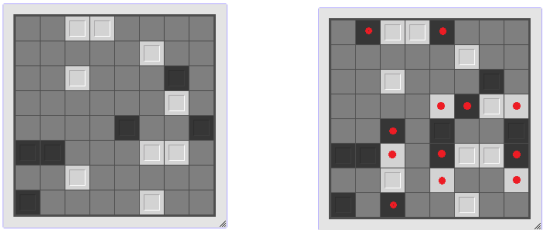


(a) (b)

Figura 2. Aplicação da Regra 1 - Coluna

2.2. Regra 2 - Extremos

A aplicação dessa regra consiste em achar duas células adjacentes (na vertical ou horizontal) que possuem a mesma cor, se encontramos, verificamos se as células que se encontram adjacentes a elas (no mesmo sentido) possuem a cor cinza, se sim, atribuímos a cor $\neg C$ a essa(s) célula(s). Um exemplo da aplicação dessa regra pode ser visto na mudança da configuração da Figura 3(a) para a Figura 3(b).

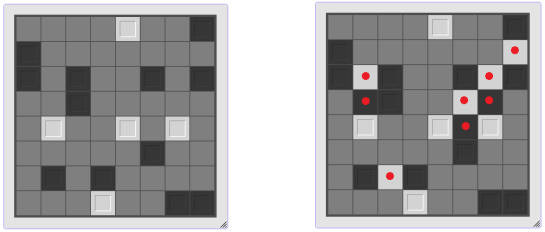


(a) (b)

Figura 3. Aplicação da Regra 2 - Extremos

2.3. Regra 2 - Meio

Essa regra consiste em achar uma célula k cinza que se encontra entre duas células de cor C, se encontramos atribuímos $\neg C$ a k. Um exemplo da aplicação dessa regra pode ser visto na mudança da configuração da Figura 4(a) para a Figura 4(b).



(a) (b)

Figura 4. Aplicação da Regra 2 - Meio

As regras devem ser aplicadas repetidas vezes, sempre que alguma regra retornar verdadeiro todas as regras devem ser novamente aplicadas para ambas as cores, branca e preta. Pois a coloração de uma célula pode possibilitar a aplicação de alguma regra que colore outra célula.

3. Backtracking

A técnica de aplicar as regras descritas finaliza sua execução rapidamente, porém, foi visto que ela é capaz de resolver somente as instâncias fáceis. Quando estamos trabalhando com as instâncias difíceis é preciso utilizar a técnica de *backtracking*. No Algoritmo 1, para a primeira célula cinza que encontramos, colocamos a cor branca e tentamos resolver o subproblema resultante recursivamente, se não for possível, repetimos o processo colocando a cor preta. Repetimos esse processo até que não exista nenhuma célula cinza na grade.

Algoritmo 1: BACKTRACKING

Entrada: $Grid, n$

```
1  Unruly(Grid, n) início
2      se n > N × M então
3          retorna verdade
4      i ← n/M + 1
5      j ← n mod M + 1
6      se cor de Grid_{i,j} é cinza então
7          retorna Unruly(Grid, n + 1
8      se Grid_{i,j} pode receber branco então
9          pinte Grid_{i,j} de branco
10         se Unruly(Grid, n + 1 então
11             retorna verdade
12         senão
13             pinte Grid_{i,j} de cinza
14     se Grid_{i,j} pode receber preto então
15         pinte Grid_{i,j} de preto
16         se Unruly(Grid, n + 1 então
17             retorna verdade
18         senão
19             pinte Grid_{i,j} de cinza
20 retorna falso
```

4. Backtracking com aplicação de regras

A fim de melhorar os resultados, propomos uma solução que utiliza a aplicação do conjunto de regras definidas na Seção 2 durante o *backtracking*. No Algoritmo 2, para cada chamada recursiva do subproblema, o conjunto de regras é aplicado para a configuração atual da $Grid$. Durante o *backtracking*, se uma cor é atribuída e não existe uma solução para a configuração atual, nós restauramos o estado anterior do problema.

5. Experimentos e conclusão

Os experimentos foram realizados em uma máquina com o processador Intel Core i5-2410M com 4 núcleos de 2.30GHz, e com memória de 1.7Gb. Os dois algoritmos foram implementados na linguagem de programação C++. Concluímos que a aplicação das regras em cada chamada recursiva diminui consideravelmente o número de nós criados na árvore de recursão. E mesmo realizando um maior número de operações para cada chamada, o tempo é reduzido drasticamente quando comparado ao tempo da técnica de *backtracking* sem a aplicação das regras, como pode ser verificado na Tabela 1. Quando o tempo limite de 3 horas é ultrapassado, tal ocorrência é assinalado como * na Tabela 1. Caso ocorra um estouro de memória, tal ocorrência será sinalizado como ** na Tabela 1.

Referências

[Russell et al. 1995] Russell, S., Norvig, P., and Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27.

Algoritmo 2: BACKTRACKING COM APLICAÇÃO DE REGRAS

Entrada: $Grid, n$

```
1  Unruly_com_Regras(Grid, n) início
2      se n > N × M então
3          retorna verdade
4      mem ← Grid
5      i ← n/M + 1
6      j ← n mod M + 1
7      se cor de Grid_{i,j} é cinza então
8          retorna Unruly(Grid, n + 1)
9      repita
10         Aplicar todas as regras em Grid
11     até nenhuma regra alterar Grid;
12     se Grid_{i,j} pode receber branco então
13         pinte Grid_{i,j} de branco
14         se Unruly_com_Regras(Grid, n + 1) então
15             retorna verdade
16         senão
17             pinte Grid_{i,j} de cinza
18     se Grid_{i,j} pode receber preto então
19         pinte Grid_{i,j} de preto
20         se Unruly_com_Regras(Grid, n + 1) então
21             retorna verdade
22         senão
23             pinte Grid_{i,j} de cinza
24 retorna falso
```

Tabela 1. Experimentos realizados usando o Algoritmo A (Backtracking) e Algoritmo B (Backtracking com aplicação de regras)

	Algortimo A		Algoritmo B	
Instância	Tempo	Número de nós	Tempo	Número de nós
U01_8x8	0.000027	82	0.000390	65
U02_8x8	0.00129	628	0.000440	65
U03_12x12	0.333156	6138453	0.001426	149
U04_14x14	0.678165	12930102	0.006890	443
U05_16x16	27.665460	761393175	0.007542	665
U06_16x16	0.042108	1066523	0.006157	726
U07_32x32	*	*	0.035206	1025
U08_64x64	*	*	**	**