Proceeding Series of the Brazilian Society of Applied and Computational Mathematics, Vol. 5, N. 1, 2017.

Trabalho apresentado no CNMAC, Gramado - RS, 2016.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Heurísticas construtivas para um problema de inundação em matrizes

André Renato Villela da Silva[1]
Ricardo Vilela Machado[2]
Maise Dantas da Silva[3]
Departamento de Computação, ICT, UFF, Rio das Ostras, RJ

Resumo. O problema de inundação é derivado de um popular jogo de computador denominado Flood-It. O objetivo é tornar monocromática uma matriz colorida através do menor número de etapas de inundação. Por inundação entende-se o processo de modificar a cor de uma região monocromática da matriz, fazendo assim com que regiões vizinhas se aglomerem. Pelo nosso conhecimento, a literatura para o problema (que é NP-difícil no caso geral) trata apenas de provas teóricas para diversas possíveis situações de instâncias. Este trabalho apresenta um estudo empírico de diversas heurísticas construtivas para o problema. Diversas instâncias de portes distintos também são propostas para servirem de *benchmark* para testes futuros.

Palavras-chave. Problemas de inundação, heurísticas construtivas, otimização combinatória

1 Introdução

O popular jogo Flood-It é [4] composto por uma matriz $N \times M$, onde cada posição possui uma cor. Posições vertical e/ou horizontalmente adjacentes que possuem a mesma cor são consideradas contíguas, formando uma só região monocromática. O objetivo do jogo é tornar toda a matriz monocromática através da mudança de cor de uma região de cada vez, com o menor número possível de etapas. Sempre que uma região muda para uma cor c, ela agrega a si todas as regiões monocromáticas adjacentes que tenham a mesma cor c. A Figura 1 mostra a sequência de inundações que tornam monocromática a matriz em questão.

Existem duas versões principais do jogo Flood-It. Na versão fixa, apenas uma das regiões monocromáticas pode mudar de cor inicialmente. As demais só poderão mudar quando forem inundadas pela região inicial. Na versão livre, qualquer uma das regiões pode ser recolorida a qualquer momento, inundando assim as regiões adjacentes que sejam da mesma cor. O objetivo, porém, permanece o mesmo: tornar a matriz monocromática

[1]avillela@ic.uff.br
[2]ricardovilela@id.uff.br
[3]maisedantas@id.uff.br

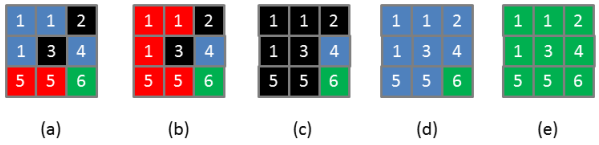

Figura 1: Sequência de colorações que tornam a matriz monocromática: (a) inicialmente a matriz apresenta 6 regiões monocromáticas, sendo a região 1 fixa; (b) escolhida a cor vermelha, o que inunda a região 5; (c) escolhida a cor preta, inundando também as regiões 2 e 3; (d) escolhida a cor azul; (e) escolhida a cor verde, tornando toda a matriz monocromática em 4 etapas.

com o menor número possível de etapas (inundações). Algumas implementações do jogo definem um número máximo Q de etapas para se cumprir o objetivo, criando uma espécie de problema de decisão, já que pode ser formulado da seguinte maneira: "é possível tornar a matriz monocromática em no máximo Q inundações?".

O problema em sua forma de minimização é NP-difícil, justificando o emprego de técnicas heurísticas para sua resolução. Este trabalho apresenta algumas propostas de critérios heurísticos a serem empregados, bem como duas formulações matemáticas para o problema. Também são propostas instâncias para serem utilizadas como *benchmark* para algoritmos a serem produzidos posteriormente pelo comunidade científica.

O jogo em questão pode ser utilizado como modelo para algumas aplicações reais, como propagação de doenças descritas em [1], que podem ocorrer de forma similar ao jogo.

O restante deste trabalho se divide da seguinte forma: na Seção 2 são apresentados alguns resultados de trabalhos anteriores da literatura. Na Seção 3, os métodos heurísticos propostos neste trabalho são descritos. Os resultados computacionais para os experimentos realizados são mostrados na Seção 4. As Seções 5 e 6 apresentam as conclusões e as propostas de trabalhos futuros, respectivamente.

2 Revisão da literatura

Em [2], os autores mostraram que as versões fixa e livre são NP-difíceis em matrizes $N \times N$ coloridas com mais de três cores. Em [6], foi mostrado que a versão livre pode ser resolvida em tempo polinomial para matrizes $1 \times N$. As duas versões permanecem NP-difíceis para matrizes $3 \times N$ coloridas com pelo menos quatro cores. A complexidade do problema em matrizes $3 \times N$ com três cores permanece em aberto.

Em [3], foi apresentado um algoritmo de tempo polinomial para a versão fixa do problema para matrizes $2 \times N$. Para estas matrizes, a versão livre permanece NP-difícil, podendo ser resolvida em tempo polinomial em grafos 2-coloridos, de acordo com [7]. Em [5], mostrou-se que a versão livre é polinomialmente solucionável em ciclos e que as duas versões são NP-difíceis em árvores.

Também existem algoritmos de tempo polinomial para os seguintes casos, como mostrado em [8]: grades circulares $2 \times N$, a segunda potência de um ciclo com n vértices,

DOI: 10.5540/03.2017.005.01.0458

d-tabuleiros $2 \times N$ (grades $2 \times N$ onde a d-ésima coluna é monocromática). Nestes dois últimos casos, a versão livre é NP-difícil.

Apesar de já existirem alguns trabalhos tratando da complexidade teórica do problema em diversos cenários, como os citados anteriormente, não é do nosso conhecimento a existência de técnicas heurísticas ou exatas para tratar do caso geral (matrizes $N \times M$ com N,M quaisquer e número de cores $K \geq 3$).

3 Métodos e Técnicas

São analisadas quatro heurísticas construtivas que levam em consideração diversos aspectos do problema, a fim de resolver instâncias de grande porte. Antes de explicitar essas técnicas, é interessante destacar que as abordagens utilizadas neste trabalho não atuam diretamente sobre a matriz de cores, mas sim sobre um grafo colorido em vértices derivado dessa matriz.

O principal objetivo em se trabalhar com um grafo ao invés de uma matriz consiste em reduzir a quantidade de elementos a serem manipulados pelas técnicas computacionais. Duas posições adjacentes na matriz que possuam a mesma cor formam uma única região monocromática, que será totalmente inundada em determinado momento. Assim, a redução dessas duas posições a um único vértice permite tratar toda uma região monocromática como sendo um único elemento.

A criação do grafo é bastante trivial, pois cada região monocromática é representada por um vértice de mesma cor. As regiões vizinhas, encontradas recursivamente, indicam a vizinhança do vértice em questão. Após a criação do grafo, o vértice 1 corresponderá à região inicial fixa da matriz, sendo automaticamente considerado já inundado. É a partir dele que todos os demais vértices também serão inundados, de acordo com as escolhas de cores a serem feitas. Todos os vértices adjacentes ao vértice 1 formam a vizinhança inicial de inundação. A Figura 2 mostra uma matriz 4×4 contendo 11 regiões e seu grafo equivalente.

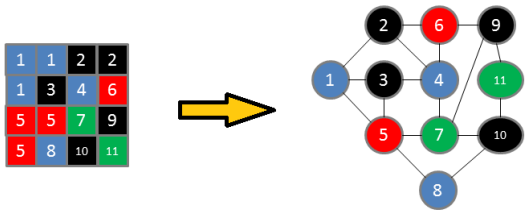

Figura 2: Matriz de tamanho 4×4, contendo 11 regiões monocromáticas e a conversão para o grafo correspondente. A região 1 é considerada inicialmente inundada.

3.1 Heurísticas

Como mencionado, o caso geral do problema de otimização é NP-difícil. Dessa forma, justifica-se o emprego de algoritmos heurísticos para tentar obter uma solução em tempo

DOI: 10.5540/03.2017.005.01.0458

computacional aceitável. A seguir, são apresentados cada um dos quatro critérios testados neste trabalho. Alguns dos critérios se utilizam da noção de vizinhança dos vértices inundados. Um vértice v não-inundado está na vizinhança de inundação se for adjacente a outro vértice u já inundado.

Critério 1: escolher a cor do vértice da vizinhança de inundação cujo grau seja o maior. Nesse critério, o objetivo é ampliar o máximo possível o tamanho da região de inundação para a escolha seguinte.

Critério 2: escolher a cor predominante da vizinhança de inundação. O objetivo é fazer com que um maior número de vértices seja inundado desde o começo do algoritmo.

Critério 3: encontrar o vértice i não inundado cujo caminho mínimo P_i a outro vértice já inundado seja o maior possível. As cores desse caminho são usadas em ordem inversa (do vértice inundado para o não inundado). Portanto, é criada uma sequência de cores c_j ($j = 1..|P_i|$) que serão escolhidas obrigatoriamente nesta ordem. O processo se repete até que todos os vértices tenham sido inundados.

Critério 4: descobrir o caminho mínimo, como no critério 3. A ordem de escolha das cores, porém, não é fixa. Se uma determinada cor c_j só puder inundar x vértices, mas neste momento houver uma outra cor c' capaz de inundar pelo menos kx vértices ($k \in \mathbb{R}$), a cor c' é escolhida primeiro e posteriormente a cor c_j, se ainda houver algum vértice não inundado. O processo também se repete até que todos o grafo esteja inundado.

4 Resultados Computacionais

Infelizmente, a literatura acerca do problema tratado neste trabalho não apresenta instâncias que sirvam de referência para os testes computacionais realizados. Como existem diversas implementações do jogo, disponíveis em dezenas de sites da internet, também não é possível definir com precisão o método de criação das matrizes nessas implementações. Dessa forma, foi necessário criar um programa gerador de instâncias para o problema, disponível em `www.professores.uff.br/asilva/flooding_problems.html`. Foram implementadas três formas de se criar a matriz de cores.

Na primeira forma, chamada Normal, cada posição da matriz tem uma cor escolhida aleatoriamente, com probabilidade uniforme. Na segunda forma, chamada Prioritária, existe um conjunto de cores prioritárias que têm somadas 70% de chance de serem escolhidas. Em todas as instâncias testadas, esse conjunto prioritário é composto por três cores, enquanto as outras cores são consideradas não-prioritárias.

Por fim, na terceira forma, chamada Sequencial, a matriz é colorida em blocos. A cada nova posição da matriz, uma das cores é escolhida aleatoriamente com probabilidade uniforme. A mesma cor escolhida é utilizada também nas posições contíguas da matriz (vertical ou horizontalmente), criando assim um bloco de posições de mesma cor, cujo tamanho, nas instâncias testadas, variou de 2 a 4 posições. Blocos muito grandes tornam a instância mais simples, pois a tendência é que haja um menor número de vértices no grafo gerado. Todos esses parâmetros podem ser definidos através do programa gerador de instâncias.

Para cada uma das três formas de se gerar a matriz de cores, foram criadas diversas

DOI: 10.5540/03.2017.005.01.0458

instâncias de matrizes $N \times N$, com N valendo 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 25 ou 30. Todas as matrizes possuem no máximo 6 cores, exceto para as instâncias de grandes porte (N =20, 25 ou 30), que possuem também matrizes com 7 e 8 cores. No total, foram geradas 10 instâncias para cada valor de N até 10. Para N =12, 15, 20, 25 ou 30, foram criadas 5 instâncias. Todas podem ser obtidas em `www.professores.uff.br/asilva/flooding_problems.html`. Os experimentos computacionais foram realizados utilizando um *notebook* com as seguintes configurações: processador Intel I7-3630QM, 8GB de memória RAM, sistema operacional Windows 8.0, compilador GCC (32 bits) 4.7.1.

Um primeiro experimento computacional foi realizado com o objetivo de identificar os melhores valores para o parâmetro k do critério heurístico 4. Quanto maior for esse valor, mais exigente é a heurística em aceitar trocar uma cor definida pela sequência de cores do maior caminho mínimo por outra cor que permitirá inundar mais vértices. Foram definidas três faixas de valores para k: [1,2], (2,5] e (5,10]. Em cada faixa, os valores de k foram testados em intervalos de 0,5. A Tabela 1 mostra a porcentagem de melhores soluções obtidas para os valores de k em cada faixa.

Tabela 1: Porcentagem de melhores resultados obtidos por valores de k do critério 4, em cada uma das três faixas. Empates também são contabilizados como melhores resultados.

Instâncias	[1,2]	(2,5]	(5,10]
Normal	31,4	39,6	36,6
Prioritária	36,5	42,1	42,1
Sequencial	39,2	59,8	72,0

Para instâncias pequenas (matrizes até 5×5), a variação dos valores de k tem pouca influência no resultado final da heurística. As diferenças começam a aparecer conforme aumenta o tamanho da instância e, consequentemente, o número de vértices que formam a vizinhança de inundação. Neste cenário, a troca de uma cor indicada pela sequência do maior caminho mínimo por outra cor c' pode ser muito vantajosa, pois c' pode inundar um número significativo de vértices. Os tempos computacionais não tiveram variação significativa em relação ao valor de k. Nas maiores instâncias com oito cores, a heurística consumiu em média 0,2 segundos.

A Tabela 2 mostra o número de melhores resultados obtidos (ou empates) por cada critério nas instâncias divididas em grupos de mesmo tamanho e número de cores. No caso do critério 4, foi utilizado o melhor valor de k dentre aqueles testados anteriormente.

Na Tabela 2, o número de cores das instâncias é indicado pela coluna $|C|$. Cada agrupamento (linha) contém uma quantidade de instâncias correspondente à coluna "Qtd". As demais colunas indicam em quantas das instâncias o referido critério heurístico obteve o melhor resultado em comparação com os demais, considerando também os eventuais empates. Por exemplo, para o agrupamento de instâncias com matrizes 25×25 de 6 cores, o critério 1 obteve um resultado melhor, enquanto o critério 4 obteve cinco resultados melhores. Obviamente, um destes resultados tem o mesmo valor do critério 1, configurando assim um empate para uma das cinco instâncias desse agrupamento. A linha "Total" fornece o somatório das linhas anteriores. Através destes somatórios, percebe-se que o critério 4 consegue fornecer um número expressivo de melhores resultados dentre os

DOI: 10.5540/03.2017.005.01.0458

Tabela 2: Comparações entre as heurísticas construtivas para instâncias agrupadas por tamanho e número de cores ($|C|$).

			Normal				Prioritária				Sequencial			
$N \times N$	$\|C\|$	Qtd	C1	C2	C3	C4	C1	C2	C3	C4	C1	C2	C3	C4
4×4	6	10	8	3	0	8	7	4	0	8	8	6	0	9
5×5	6	10	7	3	0	8	7	0	1	9	9	3	0	9
6×6	6	10	6	0	0	9	6	4	0	9	5	1	0	9
7×7	6	10	6	0	0	7	5	0	0	10	9	0	0	7
8×8	6	10	5	0	0	9	6	0	0	6	3	0	0	8
9×9	6	10	0	0	0	10	4	0	0	8	5	0	0	7
10×10	6	10	4	0	0	9	3	0	0	10	3	0	0	9
12×12	6	5	0	0	0	5	1	0	0	5	2	0	0	5
15×15	6	5	0	0	0	5	1	0	0	4	1	0	0	5
20×20	6	5	2	0	0	5	0	0	0	5	0	0	0	5
25×25	6	5	1	0	0	5	0	0	0	5	0	0	0	5
30×30	6	5	0	0	0	5	1	0	0	4	0	0	0	5
20×20	7	5	0	0	0	5	1	0	0	5	0	0	0	5
25×25	7	5	0	0	0	5	0	0	0	5	1	0	0	5
30×30	7	5	1	0	0	4	1	0	0	4	0	0	0	5
20×20	8	5	0	0	0	5	1	0	0	4	1	0	0	4
25×25	8	5	0	0	0	5	1	0	0	5	0	0	0	5
30×30	8	5	0	0	0	5	0	0	0	5	0	0	0	5
Total			40	6	0	114	45	8	1	111	47	10	0	112

critérios heurísticos testados, alcançando o melhor valor de solução em cerca de 90% das instâncias testadas. O critério 1 encontra o melhor resultado em 35% das instâncias. Os demais critérios obtiveram resultados praticamente insignificantes.

Os tempos computacionais não foram apresentados, pois foram necessários no máximo 0,2 segundos para executar as maiores instâncias. A diferença entre os critérios também é pouco significativa. No caso do uso do critério 4, não é possível saber *a priori* qual o melhor valor de k para uma instância qualquer. Embora os testes preliminares tenham mostrado que valores acima de 2,0 são bons, qualquer valor $k \in \mathbb{R}$ pode ser escolhido. Desta forma, o tempo computacional consumido pelo critério 4 pode ser maior ou menor, dependendo da amostragem de valores de k utilizada. Como cada execução consome um tempo muito pequeno, aplicações práticas poderão utilizar uma amostragem razoavelmente grande sem maiores problemas, uma vez que o critério 4 apresenta grande potencial.

5 Conclusões

No presente trabalho abordamos o problema de inundação, que modela o jogo Flood-It. Na versão em grafos, cada vértice possui uma cor. Um vértice inicial é considerado inundado e todos os seus vizinhos são candidatos a sofrerem inundação. O objetivo é escolher uma sequência de cores de forma a inundar todos os vértices do grafo com o menor número possível de etapas. A cada escolha, todos os vértices vizinhos aos já inundados e que possuem a mesma cor escolhida passam também a ser considerados inundados.

DOI: 10.5540/03.2017.005.01.0458

Foram propostos quatro critérios heurísticos para algoritmos construtivos, já que em instâncias de médio e grande porte a formulação matemática não consegue encontrar sequer uma solução viável dentro de um tempo computacional aceitável. Os experimentos computacionais indicaram que os critérios 1 e 4 obtiveram os melhores desempenhos, sendo o critério 4 aquele que alcançou o melhor valor de solução em cerca de 90% das instâncias testadas.

Agradecimentos

Agradecemos à FAPERJ pelo apoio na realização desta pesquisa.

Referências

[1] M. Adriaen, P. De Causmaecker, P. Demeester, and G. Vanden Berghe. Spatial simulation model for infectious viral disease with focus on sars and the common flu. In *37th Annual Hawaii International Conference on System Sciences, IEEE Computer Society, ISBN: 0-7695-2056-1*, 2004.

[2] D. Arthur, R. Clifford, M. Jalsenius, and A. Montanaro ans B. Sach. The complexity of flood filling games. In *FUN, volume 6099 of Lecture Notes in Computer Science, Springer, ISBN 978-3-642-13121-9*, pages 307–318. 2010.

[3] R. Clifford, M. Jalsenius, A. Montanaro, and B. Sach. The complexity of flood filling game. *Theory Comput Syst*, 50:72–92, 2012. DOI: 10.1007/s00224-011-9339-2.

[4] LabPixies. Labpixies - the coolest games! Disponível em: `http://www.labpixies.com`, 2015. Acessado em 27/04/2015.

[5] A. Lagoutte. Jeux d'inondation dans les graphes. Technical report, ENS Lyon, HAL: hal-00509488, 2010.

[6] K. Meeks and A. Scott. The complexity of free-flood-it on 2n boards. *Theoretical Computer Science*, 500:25–43, 2013. DOI: 10.1016/j.tcs.2013.06.010.

[7] K. Meeks and A. Scott. The complexity of flood-filling games on graphs. *Discrete Applied Mathematics*, 160:959–969, 2011. DOI:10.1016/j.dam.2011.09.001.

[8] U.S. Souza, F. Protti, and M. Dantas da Silva. An algorithmic analysis of flood-it and free-flood-it on graph powers. *Discrete Mathematics and Theoretical Computer Science*, 16:279–290, 2014.

