# Algoritmos heurísticos e exatos para o Problema de Inundação livre em grafos

**André Renato Villela da Silva**
Universidade Federal Fluminense - Instituto de Ciência e Tecnologia
R. Recife s.n. - Jardim Bela Vista - Rio das Ostras/RJ
arvsilva@id.uff.br

**Luiz Satoru Ochi, Uéverton Souza**
Universidade Federal Fluminense - Instituto de Computação
Av. Gal. Milton Tavares de Souza s.n. - São Domingos - Niterói/RJ
satoru@ic.uff.br,usouza@ic.uff.br

**Bruno José da Silva Barros, Rian Gabriel S. Pinheiro**
Universidade Federal Rural de Pernambuco - Unidade Acadêmica de Garanhuns
Av. Bom Pastor, s.n. Boa Vista - Garanhuns/PE
brrsbruno@gmail.com,rian.gabriel@ufrpe.br

**RESUMO**

Este artigo trata da versão livre do problema de inundação em grafos. Suponha que o grafo $G$ seja colorido em vértices. Uma inundação consiste em mudar um vértice $i$ para uma cor $c$, unindo-o a todos os vértices adjacentes de $i$ que já tenham a mesma cor $c$. O objetivo do problema é definir uma sequência mínima de vértices e cores, representando cada inundação, que torne o grafo $G$ totalmente monocromático. Esta versão do problema tem aplicações em algumas áreas como propagação da doenças, por exemplo, mas há apenas estudos teóricos na literatura. Duas meta-heurísticas, uma primeira formulação matemática e um algoritmo backtracking com podas são propostos. Os resultados obtidos mostram que as meta-heurísticas e o algoritmo backtracking tiveram excelente desempenho.

**PALAVRAS CHAVE. Problema de Inundação em Grafos, Meta-heurísticas, Formulação Matemática, Backtracking.**

**Tópicos (Otimização Combinatória, Meta-heurísticas)**

**ABSTRACT**

This paper deals with the free version of the Flooding Problem on graphs. Suppose graph $G$ colored on vertices. A flooding consists in changing a vertex $i$ to a color $c$, merging it to all adjacent vertices of $i$ that already have the same color $c$. The problem objective is to define a minimal sequence of vertices and colors, representing each flooding, that makes the graph $G$ totally monochromatic. This problem version has applications in some areas as disease propagation, for example, but there are only theoretical studies in the literature. Two metaheuristics, a first mathematical formulation and a prunning-backtracking algorithm are proposed. The obtained results show that the metaheuristics and the backtracking algorithm had excellent performance.

**KEYWORDS. Flooding Problem on Graphs, Metaheuristics, Mathematical Formulation, Backtracking.**

**Paper topics (Combinatorial Optimization, Metaheuristics)**

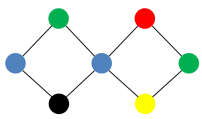
## 1. Introdução

O Problema de Inundação em Grafos é derivado de um popular videojogo chamado Flood-It [LabPixies, 2015]. O jogo é composto por uma matriz (ou tabuleiro) de tamanho qualquer $N \times M$. Cada elemento da matriz é representado por uma cor inicial proveniente de um conjunto de cores $C$. Elementos adjacentes (apenas na vertical e/ou horizontal) que forem da mesma cor formam regiões contíguas, chamadas de regiões monocromáticas. Essas regiões se comportam como se fossem um único elemento. A inundação é um processo de mudança de cor de uma região para uma cor $c$. Quando a inundação ocorre, a região em questão se une a todas as demais regiões vizinhas que já possuíam a cor $c$. O objetivo do jogo é, com o menor número possível de etapas (inundações), tornar toda a matriz monocromática através da mudança de cor de uma região inteira de cada vez.

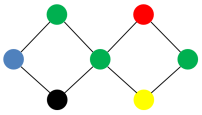
Existem duas versões do jogo Flood-It. Na versão chamada fixa, apenas uma das regiões monocromáticas pode mudar de cor. Essa região é chamada de pivô. As demais só poderão mudar quando forem inundadas pelo pivô. Na versão chamada livre, qualquer uma das regiões pode ser recolorida a qualquer momento. O objetivo, porém, permanece o mesmo: tornar a matriz monocromática com o menor número possível de etapas (inundações). O problema em ambas as versões é NP-difícil como mostrado em Clifford et al. [2012]. Portanto, não são conhecidos algoritmos eficientes para encontrar a solução ótima em tempo aceitável.

Embora as apresentações mais frequentes do problema utilizem uma estrutura matricial, computacionalmente é mais eficiente trabalhar com grafos. Assim, cada vértice do grafo corresponde a uma região monocromática da matriz. A cor inicial do vértice é a mesma cor inicial da região matricial. As arestas representam a relação de adjacência entres essas regiões. Pode-se, portanto, definir o problema utilizando esta interpretação por meio de grafos. Seja $G = (V, A)$, um grafo não-direcionado contendo um conjunto $V$ de vértices coloridos e um conjunto $A$ de arestas entre estes vértices, deseja-se tornar o grafo inteiramente monocromático com o menor número possível de inundações. Uma inundação pode ser descrita como o procedimento de mudança de cor de um vértice $v$ para uma cor arbitrária $c$.

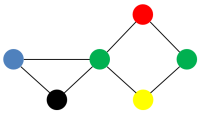
Também existem duas formas de tratar as inundações. Na forma mais simples, os vértices que mudaram de cor permanecem no grafo, apenas têm sua cor atualizada. Na forma mais complexa, o vértice que mudou para uma cor $c$ é contraído com todos os vizinhos de cor $c$, formando uma único vértice novo. O conjunto de vizinhos desse novo vértice é a união da vizinhança de todos os vértices que foram contraídos, exceto os próprios vértices. A Figura 1 mostra um exemplo das duas formas de atualização das cores de um vértice após uma inundação.



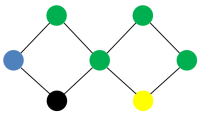
(a) Grafo original



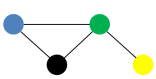
(b) Inundação do vértice central com cor verde



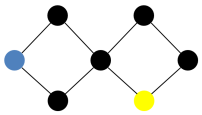
(c) Forma contraída do grafo (b)



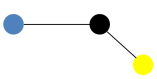
(d) Inundação do vértice superior direito com cor verde, após etapa (b)



(e) Forma contraída do grafo (d)



(f) Inundação do vértice central com cor preta, após etapa (d)



(g) Forma contraída do grafo (f)

Figura 1: Exemplo de uma sequência de inundações, com e sem contrações dos vértices inundados

Suponha o grafo original mostrado na Figura 1(a). Caso a inundação desejada mude a cor do vértice central para verde, o grafo ficaria igual à Figura 1(b). Neste ponto, existe a possibilidade de manter a topologia original do grafo ou contraí-lo para que os vértices verdes adjacentes fiquem como na Figura 1(c). Uma segunda inundação no vértice superior direito com a cor verde novamente deixaria o grafo original como o da Figura 1(d). Se a contração fosse aplicada, o grafo seria semelhante ao da Figura 1(e). A terceira inundação com a cor preta no vértice central produziria o grafo da Figura 1(f) sem a contração ou o grafo da Figura 1(g) com a contração. Em resumo, se a contração não for utilizada, o grafo final mantém a mesma topologia do grafo original com todos os vértices da mesma cor. Se a contração for adotada, o grafo final é apenas um único vértice.

O problema em questão pode ser utilizado como modelo para algumas aplicações reais, como propagação de doenças descritas por Adriaen et al. [2004]. Em Souza et al. [2013], é apresentada uma analogia do problema de inundação em árvores com um subcaso do problema da supersequência comum mais curta (SCS). Fellows et al. [2003] mostram que a dinâmica de inundações permitem uma melhor observação de alguns problemas e propriedades de *strings*. Também existem implicações na área de bioinformática, como: produção de *microarrays* [Rahmann, 2003], montagens de sequências de DNA [Barone et al., 2001], além de um relacionamento próximo ao problema de alinhamento de múltiplas sequências [Sim e Park, 2003].

Este artigo visa tratar a versão livre do problema, uma vez que são conhecidas apenas abordagens teóricas para ela. A metodologia proposta é por meio de algoritmos heurísticos e exatos. O restante deste trabalho é descrito a seguir. Na Seção 2, é feita uma revisão da literatura sobre o Problema de Inundação. Os métodos propostos para a versão livre do problema são apresentados na Seção 3. A Seção 4 apresenta os resultados computacionais obtidos. Por fim, na Seção 5 estão as conclusões e as propostas de trabalhos futuros.

## 2. Revisão da Literatura

Em Arthur et al. [2010], os autores mostraram que as duas versões (fixa e livre) para matrizes $N \times N$ coloridas com mais de três cores são NP-difíceis. Meeks e Scott [2013] mostraram que a versão livre pode ser resolvida em tempo polinomial para matrizes $1 \times N$. As duas versões são NP-difíceis para matrizes $3 \times N$ coloridas com pelo menos quatro cores. Entretanto, nestas mesmas matrizes o problema permanece em aberto se forem utilizadas apenas três cores.

Para matrizes $2 \times N$, na versão fixa do problema, foi apresentado um algoritmo de tempo polinomial em Clifford et al. [2012]. Para estas matrizes, a versão livre permanece NP-difícil, podendo ser resolvida em tempo polinomial em grafos 2-coloridos, de acordo com Meeks e Scott [2011]. Lagoutte [2010] mostrou 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 Souza et al. [2014]: grades circulares $2 \times N$, a segunda potência de um ciclo com $n$ vértices, $d$-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.

Em Silva [2016], foi proposto um Algoritmo Evolutivo para a versão fixa do problema. O algoritmo obteve bons resultados até para instâncias de maior porte e é considerado, até o nosso conhecimento, o melhor método heurístico para o problema. Uma formulação matemática (F2) na forma de um Programa Inteiro Misto, também para a versão fixa, foi proposta em Silva e Ochi [2016]. Esta formulação apresenta mecanismos para evitar simetria, quando comparada à formulação (F1) apresentada em Barros et al. [2015]. A formulação (F2) conseguiu ser executada em tempo menor e encontrar o ótimo em diversas instâncias testadas.

## 3. Métodos Propostos

Levando em consideração que o Problema de Inundação é NP-difícil, são propostas duas versões de meta-heurísticas, sendo um Algoritmo Evolutivo (EA) e um Iterated Local Search (ILS). O leitor pode encontrar mais embasamento teórico para estas meta-heurísticas respectivamente em

Goldberg [1989]; Gonçalves et al. [2016] e Lourenço et al. [2003]; Penna et al. [2013]. Também foram propostos dois métodos exatos: uma formulação matemática na forma de um Programa Inteiro Misto e um algoritmo de *backtracking*.

### 3.1. Modelagem Matemática

Antes de descrever a formulação matemática, é importante definir alguns conceitos. O problema de inundação aqui é tratado como um problema de escalonamento, onde cada etapa consiste em uma eventual inundação, ao escolher-se uma única cor e vértices que devem mudar para esta cor escolhida. Um *grupo* de vértices é um conjunto de vértices adjacente entre si que possuem a mesma cor. A cada etapa, mais de um vértice pode ser escolhido para mudar de cor, desde que todos esses vértices sejam do mesmo grupo. Inicialmente, cada vértice possui a sua cor fornecida como dado de entrada e, portanto, não há grupos existentes. Quando os vértices mudarem de cor, vértices adjacentes que terminarem uma etapa com a mesma cor formarão um grupo. Esse grupo poderá se expandir em etapas posteriores, ao incorporar outros vértices. Diz-se que as arestas do grafo *conectam* seus vértices extremos, se estes possuírem a mesma cor ao final de uma etapa.

A função-objetivo (1) visa minimizar o número de etapas para inundar totalmente o grafo. As restrições (2) indicam que uma nova etapa deve ser considerada sempre que uma cor for escolhida em uma etapa $t$ e que no máximo uma única cor pode ser escolhida a cada etapa (restrições 3). As restrições (4) garantem que se qualquer vértice for escolhido para mudar em uma etapa $t$, uma cor também deve ser escolhida nesta etapa. Cada vértice só pode encerrar uma etapa com uma única cor (5). A cor do vértice deve ser a mesma da etapa anterior, exceto se o vértice for escolhido para mudar de cor (6). Neste caso, a cor final do vértice será a cor escolhida nesta etapa (7).

A aresta $(i, j)$ conecta seus vértices extremos (restrições 10) se, para alguma cor $c$, ambos os vértices extremos terminam a etapa $t$ com a cor $c$ (restrições 8 e 9). Dois vértices $i$ e $j$ só podem ser escolhidos para mudar de cor em uma etapa $t$, se eles já participarem do mesmo grupo na etapa anterior (restrições 11). Vértices $i$ e $j$ adjacentes fazem parte do mesmo grupo, se a aresta incidente a eles estiver conectando-os (12). Se os vértices $i$ e $j$ não forem adjacentes, só farão parte do mesmo grupo se houver vértices $k$ e $q$ tal que $(i, k) \in E$, $(j, q) \in E$ e os vértices $k$ e $q$ façam parte do mesmo grupo recursivamente (restrições 13, 14, 15 e 16).

Cada uma das aresta não conecta os vértices extremos no início do problema (17), mas devem conectar até o final do horizonte de planejamento (restrições 18). As restrições (19) asseguram que a cor inicial de cada vértice $i$ é um dado de entrada do problema ($c(i)$). Inicialmente, os únicos grupos existentes são os próprios vértices isoladamente (restrições 20 e 21). Cada vértice é considerado conectado consigo mesmo desde o início do horizonte de planejamento, de acordo com as restrições (22).

Tabela 1: Significado das variáveis binárias da formulação matemática

| Var. | Descrição |
|---|---|
| $x^{c,t}$ | Se a cor $c$ é escolhida na etapa $t$ |
| $s_i^t$ | Se o vértice $i$ mudará de cor da etapa $t$ |
| $b_i^{c,t}$ | Se o vértice $i$ terminará a etapa $t$ com a cor $c$ |
| $w_{i,j}^{c,t}$ | Se a aresta $(i, j)$ conecta os vértices extremos com cor $c$ na etapa $t$ |
| $a_{i,j}^t$ | Se a aresta $(i, j)$ conecta os vértices extremos com qualquer cor |
| $g_{i,j}^t$ | Se os vértices $i$ e $j$ fazem parte do mesmo grupo na etapa $t$ |
| $d_{i,j,k,q}^t$ | Se os vértices não adjacentes $i$ e $j$ estão diretamente conectados aos vértices $k$ e $q$, respectivamente, na etapa $t$ |

$$min \quad z \tag{1}$$

$$s.a. \quad z \geq t * x^{c,t} \qquad \forall_{t=1,..,T} \tag{2}$$

$$\sum_{c \in C} x^{c,t} \leq 1 \qquad \forall_{c \in C} \, \forall_{t=1,..,T} \tag{3}$$

$$\sum_{c \in C} x^{c,t} \geq s_i^t \qquad \forall_{i \in V} \quad \forall_{t=1,..,T} \qquad (4)$$

$$\sum_{c \in C} b_i^{c,t} = 1 \qquad \forall_{i \in V} \quad \forall_{t=0,..,T} \qquad (5)$$

$$b_i^{c,t} - b_i^{c,t-1} + s_i^t \geq 0 \qquad \forall_{i \in V} \quad \forall_{c \in C} \quad \forall_{t=1,..,T} \qquad (6)$$

$$x^{c,t} + s_i^t - b_i^{c,t} \leq 1 \qquad \forall_{i \in V} \quad \forall_{c \in C} \quad \forall_{t=1,..,T} \qquad (7)$$

$$b_i^{c,t} \geq w_{i,j}^{c,t} \qquad \forall_{(i,j) \in E} \quad \forall_{c \in C} \quad \forall_{t=1,..,T} \qquad (8)$$

$$b_j^{c,t} \geq w_{i,j}^{c,t} \qquad \forall_{(i,j) \in E} \quad \forall_{c \in C} \quad \forall_{t=1,..,T} \qquad (9)$$

$$a_{i,j}^t = \sum_{c \in C} w_{i,j}^{c,t} \qquad \forall_{(i,j) \in E} \quad \forall_{t=1,..,T} \qquad (10)$$

$$s_i^t + s_j^t - g_{i,j}^{t-1} \leq 1 \qquad \forall_{i,j \in V} \quad \forall_{t=1,..,T} \qquad (11)$$

$$g_{i,j}^t = a_{i,j}^t \qquad \forall_{(i,j) \in E} \quad \forall_{t=0,..,T} \qquad (12)$$

$$g_{i,j}^t = \sum_{(i,k) \in E} \sum_{(j,q) \in E} d_{i,j,k,q}^t \qquad \forall_{(i,j) \notin E} \quad \forall_{t=1,..,T} \qquad (13)$$

$$a_{i,k}^t \geq d_{i,j,k,q}^t \qquad \forall_{(i,j) \notin E} \quad \forall_{(i,k) \in E} \quad \forall_{(j,q) \in E} \quad \forall_{t=1,..,T} \qquad (14)$$

$$a_{j,q}^t \geq d_{i,j,k,q}^t \qquad \forall_{(i,j) \notin E} \quad \forall_{(i,k) \in E} \quad \forall_{(j,q) \in E} \quad \forall_{t=1,..,T} \qquad (15)$$

$$g_{k,q}^{t-1} \geq d_{i,j,k,q}^t \qquad \forall_{(i,j) \notin E} \quad \forall_{(i,k) \in E} \quad \forall_{(j,q) \in E} \quad \forall_{t=1,..,T} \qquad (16)$$

$$a_{i,j}^0 = 0 \qquad \forall_{(i,j) \in E} \qquad (17)$$

$$a_{i,j}^T = 1 \qquad \forall_{(i,j) \in E} \qquad (18)$$

$$b_i^{c(i),0} = 1 \qquad \forall_{i \in V} \qquad (19)$$

$$g_{i,j}^0 = 0 \qquad \forall_{i,j \in V, i \neq j} \qquad (20)$$

$$g_{i,i}^0 = 1 \qquad \forall_{i \in V} \qquad (21)$$

$$a_{i,i}^0 = 1 \qquad \forall_{i \in V} \qquad (22)$$

### 3.2. Algoritmo *backtracking*

*Backtracking* é uma técnica de projeto de algoritmos que consiste em tratar um problema dividindo-o recursivamente em subproblemas menores. Alguns desses subproblemas possuem solução trivial, encerrando assim a recursão. O *backtracking* clássico pode ser classificado como um método de enumeração explícita, pois analisa todas as possíveis combinações de soluções a fim de indicar a solução ótima do problema. Dado que o Problema de Inundação é NP-difícil, o completo percorrimento da árvore de subproblemas gerada pelo algoritmo de *backtracking* consome um tempo computacional muito elevado. A simples descoberta de soluções de boa qualidade também pode demorar demais, tornando o método consideravelmente ineficiente.

Uma medida costumeira par tentar melhorar a performance do *backtracking* é a utilização de podas. A poda é uma interrupção precoce de um ramo da árvore de subproblemas gerada ao longo da execução do *backtracking*. Geralmente, as podas eliminam ramos que não são considerados promissores, ou seja, que mesmo completamente explorados não apresentarão qualquer solução de qualidade melhor do que uma outra já encontrada anteriormente.

As podas, portanto, devem espelhar algum conhecimento específico do problema. No Problema de Inundação, o objetivo é tornar o grafo completamente monocromático com a menor quantidade possível de inundações. Suponha uma solução conhecida $S$, com valor $|S|$. Se um algoritmo qualquer é capaz de realizar $F$ inundações iniciais que levem a um estado $e$, uma poda poder aplicada se ainda forem necessárias pelo menos $|S| - F$ inundações para alcançar o objetivo. Caso contrário, as inundações feitas a partir do estado $e$ resultarão em soluções $S'$ com valor $|S'| \geq |S|$, o que não melhora em nada a melhor solução já conhecida.

Neste trabalho, são estudados dois critérios para podas. O primeiro critério é baseado no número de cores que existem em um grafo $G$. Se o número de cores restantes $c_{rest}(G) = k$, então ainda serão necessárias no mínimo $k - 1$ inundações para tornar o grafo $G$ monocromático.

O segundo critério é baseado na topologia do grafo. Suponha um grafo $G$ propriamente colorido e dois vértices $u$ e $v$. Se o vértice $v$ for inundado a partir de escolhas feitas sobre o vértice $u$, serão necessárias, no mínimo, $d(u, v)$ inundações, onde $d(u, v)$ representa a distância entre os vértices $u$ e $v$, já que o grafo $g$ não tem pesos nas arestas. Essa quantidade total de inundações pode ser eventualmente reduzida se for utilizado um terceiro vértice $w$ tal que $d(w, u) < d(u, v)$ e $d(w, v) < d(w, v)$. No melhor caso, o número mínimo de inundações seria $max\{d(w, u), d(w, v)\}$. Estendendo esse raciocínio para contemplar todos os vértices do grafo $G$, a melhor hipótese seria considerar um vértice "central" $v'$ menos distante de todos os outros vértices do grafo, a partir do qual poderiam ser realizadas as menores quantidades de inundações até se atingir os vértices mais extremos do grafo. Vale destacar que essa é uma análise hipotética para o melhor caso, funcionando como um limite inferior para o número de inundações necessárias. Na prática, outros fatores precisam ser considerados como a distribuição das cores pelos vértices do grafo, por exemplo.

Para implementar esse limite inferior, é necessário calcular o vértice $i$ de menor excentricidade $e(i)$ e, consequentemente, o raio $r(G)$ do grafo $G$. Esse valor indicará que serão necessárias pelo menos $q \geq r(g)$. Também é importante destacar que, sempre que ocorrer uma inundação, o grafo $G$ será contraído, ou seja, todos os vértices de mesma cor que sejam adjacentes entre si são transformado em um único vértice cuja vizinhança é a união da vizinhanças dos vértices originais, exceto os próprios vértices contraídos. Essa contração é vantajosa para o algoritmo porque evita que sejam criados mais de um ramo para vértices adjacentes da mesma cor (problema de simetria). O pseudo-código do algoritmo de *backtracking* pode ser visto no Algoritmo 1.

**Algoritmo 1** *Backtracking*(inteiro inundacoes, grafo colorido G, solucao best)

```
1: if inundacoes + max{r(G), c_rest(G) − 1} ≥ Valor(best) then
2:     return ∞
3: end if
4: if inundado(G) then
5:     return 0
6: end if
7: melhor ← ∞
8: L ← OrdenaVertices(G)
9: for (i = 1 to |L|) do
10:    C ← OrdenaCor(L_i)
11:    for (k = 1 to |C|) do
12:        G' ← Inunda(G, L_i, C_k)
13:        x ← 1 + Backtracking(inundacoes + 1, G', best)
14:        if x < Valor(best) then
15:            Atualiza(best, L_i, C_k, x)
16:        end if
17:        if x < melhor then
18:            melhor ← x
19:        end if
20:    end for
21: end for
22: return melhor
```

O retorno principal do algoritmo *Backtracking* é o número de inundações que podem ser realizadas para alcançar o objetivo do problema. Os parâmetros de entrada representam o número de inundações já realizadas, o grafo atual e a solução incumbente do algoritmo. A primeira chamada ao algoritmo é feita com os seguintes valores (0,$G$,{}). As podas são realizadas nas linhas 1-3, seja pelo raio do grafo, seja pelas cores restantes. Caso alguns desses fatores não possibilite uma solução melhor do que a incumbente, ocorre a poda, retornando um valor muito elevado.

Se o grafo $G$ já estiver totalmente inundado (linhas 4-6), não é necessária mais nenhuma inundação subsequente. A variável $melhor$ (linha 7) indica quantas inundações ainda serão necessárias. Os vértices são então ordenados pela sua excentricidade, formando uma lista $L$, em ordem crescente (linha 8). Para cada vértice $L_i$ da lista, as possíveis cores também são ordenadas formando uma outra lista $C$. Esta, por sua vez, é ordenada de acordo com a quantidade cores dos vértices adjacentes a $L_i$, priorizando as cores mais frequentes nesta vizinhança (linha 10).

Para cada possível par $(L_i, C_k)$, o grafo sofre uma inundação e é contraído, formando um novo grafo $G'$ (linha 12). Ocorre então a recursão sobre este novo grafo $G'$ (linha 13). A variável $x$ indica quantas etapas foram necessárias para inundar totalmente o grafo, seguindo pelo ramo correspondente às escolhas feitas. No caso de uma poda, a variável ficará com valor muito elevado. No entanto, caso este valor seja melhor do que a solução incumbente, esta é atualizada (linhas 14-16). Por fim, o algoritmo computa a menor quantidade de inundações a partir desta chamada (linhas 17-19) e a retorna como resultado final (linha 22). A sequência de inundações pode ser recuperada pelo parâmetro $best$ que foi atualizado ao longo do processo.

É interessantes destacar que o algoritmo *Backtracking* pode ser realizado sem as ordenações propostas nas linhas 8 e 10. Contudo, estas ordenações melhoram muito a eficiência do algoritmo pois os vértices mais promissores (menor excentricidade) e as cores que permitem uma inundação maior a cada etapa são explorados primeiro. Consequentemente, a solução incumbente é aprimorada o quanto antes e mais podas são realizadas inicialmente.

### 3.3. Algoritmo Evolutivo (EA)

Em Silva [2016], foi apresentado um algoritmo evolutivo, chamado EA, para o Problema de Inundação na versão fixa. O algoritmo apresentou bons resultados. No entanto, são necessárias algumas adaptações para torná-lo utilizável na versão livre do problema.

A primeira adaptação ocorre no operador de avaliação da solução. Cada indivíduo é representado por uma lista de $n$ números reais, correspondendo à prioridade de escolha de cada um dos $n$ vértices do grafo original $G$. O vértice com maior prioridade é escolhido e, em seguida, é também escolhida a cor $c$. Esta cor é computada deterministicamente como sendo a cor predominante na vizinhança do grupo ao qual o vértice pertence. Todos os vértices deste grupo mudam para a cor $c$ e se incorporam a outros vértices, aumentando assim o tamanho desse grupo. Diferentemente do algoritmo *Backtracking*, o grafo não é contraído ao final de cada inundação.

A segunda adaptação é feita ao se gerar os indivíduos da população original. Na versão fixa do problema, o algoritmo EA utilizava uma fórmula para calcular a prioridade de cada vértice $i$ onde eram ponderados um valor probabilístico e a quantidade total vértices do grafo com a mesma cor de $i$. Na versão livre, foi utilizada também uma ponderação, mas agora considera-se a excentricidade do vértice $i$. A fórmula adaptada é $p_i = (w * (diam(G) - e(i)) + u)/d$, onde $p_i$ significa a prioridade calculada para o vértice $i$, $diam(G)$ é o diâmetro do grafo, $e(i)$ é a excentricidade do vértice $i$, $w$ é um peso arbitrariamente definido, $u$ é uma variável aleatória uniforme e $d$ é um parâmetro de escala para que o resultado da fórmula fique limitado a uma faixa de valores. Nos experimentos computacionais, os valores foram definidos como $w = 100$, $u = [0, 100]$ e $d = 1000$.

A última adaptação consiste na inclusão de um mecanismo de intensificação. A cada $\gamma = 100$ gerações, 50 indivíduos (com possibilidade de repetição) da população corrente passam por 10 etapas de intensificação. Em cada etapa, um vértice com excentricidade mínima é escolhido aleatoriamente e a esse vértice são associadas de 1 a 6 inundações de cores aleatórias. Essas inundações são posicionadas também de forma aleatória ao longo da sequência de inundações já existe para este indivíduo. Após esse ampliação da sequência de inundações ocorre uma busca local, chamada de $BL$. Durante a busca, toda a sequencia de inundações é revisitada. Se uma inundação $f$ puder ser retirada e o restante das inundações ainda permitir alcançar o objetivo do problema, a inundação $f$ é excluída em definitivo. Caso contrário, a inundação $f$ retorna para a sua posição original e a análise da lista prossegue com a inundação seguinte. Todos os outros operadores evolutivos e o critério de parada foram os mesmos utilizados em Silva [2016].

### 3.4. Iterated Local Search (ILS)

O Algoritmo 2 descreve a meta-heurística ILS proposta para o Problema de Inundação na versão livre. O ILS proposto contém componentes que podem ser usadas no Problema de Inundação na versão fixa. O algoritmo segue o padrão descrito por Lourenço et al. [2010], na qual iterativamente aplicando busca local e movimentos de perturbação obtém-se novas soluções de acordo com alguns critérios de aceitação. A meta-heurística utiliza uma função de perturbação que consiste em adicionar uma quantidade de elementos aleatórios à solução já pronta. Essa perturbação é semelhante ao mecanismo de intensificação do EA. Inundações aleatórias são inseridas em posições aleatórias da sequência de inundações. O número de inserções também varia de 1 a 6, mas qualquer vértice pode ser utilizado neste caso. A busca local BL do algoritmo EA também é empregada.

Para o cálculo da solução inicial, como mostra o Algoritmo 2, deve-se que escolher um pivô inicial. Para isso, é escolhido um vértice com a menor excentricidade possível (linha 1). Uma vez escolhido o pivô, o algoritmo *FLOODING-II* [Barros et al., 2015] (linha 2) é utilizado para a obtenção da solução inicial. Essa solução é criada sem variar o pivô de cada rodada de inundação, ou seja, cada rodada contém o mesmo pivô. Note que, quando o algoritmo entra no *loop* principal (linhas 4-10), é a perturbação (linha 5) que se encarrega de variar os pivôs de cada rodada. Podendo assim, melhorar a solução que logo em seguida passa pela busca local (linha 6).

**Algoritmo 2** ILS(grafo colorido G)

1: $pivo \leftarrow EscolheMenorExcentricidade(G)$
2: $solucao \leftarrow Flooding\_II(G, pivo)$
3: $solucao \leftarrow BuscaLocal(solucao)$
4: **while** (Condição de parada não for satisfeita) **do**
5: $\quad novaSolucao \leftarrow Perturbacao(solucao)$
6: $\quad novaSolucao \leftarrow BL(novaSolucao)$
7: $\quad$ **if** $Valor(novaSolucao) \leq Valor(solucao)$ **then**
8: $\quad\quad solucao \leftarrow novaSolucao$
9: $\quad$ **end if**
10: **end while**
11: **return** $solucao$

## 4. Resultados Computacionais

As instâncias utilizadas nos experimentos seguintes foram extraídas de Silva et al. [2016] por serem consideradas de menor porte em relação às apresentadas em Silva [2016]. Na verdade, o Problema de Inundação na versão livre é muito mais difícil, sob o ponto de vista prático, do que na versão fixa, uma vez que não existe vértice pivô na versão livre. Assim, a cada etapa, precisam ser definidas duas variáveis: qual vértice iniciará a inundação e qual a cor a ser utilizada.

As instâncias foram manipuladas na forma de grafos, extraídos de matrizes $N \times N$, contendo até 6 cores em cada matriz, indicado pelas colunas $C$. Para cada tamanho de matriz, foram definidas 5 instâncias diferenciadas pelo identificador # presente nas tabelas a seguir. Na construção das instâncias, as cores foram escolhidas com probabilidade uniforme.

Os algoritmos heurísticos, o *Backtracking* e a formulação matemática foram executados em um computador com processador Intel I7-3630QM, 8GB de memória RAM, sistema operacional Windows 8.0. Os códigos foram escritos em linguagem C++ e compilados com o GNU GCC 4.7.1. A formulação matemática foi executada pelo *solver* CPLEX 12.6.1.

| #1 | | #2 | | #3 | | #4 | | #5 | |
|---|---|---|---|---|---|---|---|---|---|
| L.Inf. | Tempo | L.Inf. | Tempo | L.Inf. | Tempo | L.Inf. | Tempo | L.Inf. | Tempo |
| 6,0 | 874,7 | 6,0 | 4427,6 | 6,0 | 2499,9 | 5,0 | 7200,0 | 6,0 | 3309,7 |

Tabela 2: Resultados da formulação matemática para instâncias com $N = 4$ e $|C| = 6$. A solução incumbente em todos os casos tem valor igual a 6,0. Tempo em segundos.

O primeiro experimento consistiu na execução da formulação matemática proposta. Na Tabela 2, foram testadas instâncias provenientes de matrizes $4 \times 4$. A formulação teve desempenho ruim, pois consumiu um tempo computacional considerável. Na instância $\#4$ inclusive, não foi possível encerrar a execução antes do tempo limite de 7200 segundos. As soluções incumbentes tiveram valor igual a 6,0. O mesmo teste para instâncias de maior porte confirmou o fraco desempenho da formulação matemática, pois em nenhum caso houve sequer uma solução factível encontrada dentro do tempo limite. Também é importante destacar que os tempos presentes na Tabela 2 só foram obtidos adicionando à modelagem proposta uma restrição que obriga o número de etapas a ser, pelo menos, igual ao raio do grafo $G$.

| Instância | | | | Melhor | | Quant. | | Média | |
|---|---|---|---|---|---|---|---|---|---|
| N | C | # | Tempo(s) | ILS | EA | ILS | EA | ILS | EA |
| | | 1 | 1,1 | **6** | **6** | 30 | 30 | 6,0 | 6,0 |
| | | 2 | 1,3 | **6** | **6** | 30 | 30 | 6,0 | 6,0 |
| 4 | 6 | 3 | 1,2 | **6** | **6** | 30 | 30 | 6,0 | 6,0 |
| | | 4 | 1,5 | **6** | **6** | 30 | 21 | 6,0 | 6,3 |
| | | 5 | 1,4 | **6** | **6** | 30 | 24 | 6,0 | 6,2 |
| | | 1 | 1,6 | **6** | **6** | 30 | 30 | 6,0 | 6,0 |
| | | 2 | 1,8 | **6** | **6** | 30 | 30 | 6,0 | 6,0 |
| 5 | 6 | 3 | 2,0 | **7** | **7** | 30 | 19 | 7,0 | 7,4 |
| | | 4 | 1,6 | **6** | **6** | 30 | 30 | 6,0 | 6,0 |
| | | 5 | 1,8 | **7** | **7** | 30 | 30 | 7,0 | 7,0 |
| | | 1 | 2,6 | **8** | **8** | 30 | 25 | 8,0 | 8,2 |
| | | 2 | 2,7 | **8** | **8** | 30 | 21 | 8,0 | 8,3 |
| 6 | 6 | 3 | 2,7 | **8** | **8** | 30 | 30 | 8,0 | 8,0 |
| | | 4 | 3,1 | **8** | **8** | 15 | 1 | 8,6 | 9,0 |
| | | 5 | 2,9 | **9** | **9** | 30 | 30 | 9,0 | 9,0 |
| | | 1 | 3,4 | **8** | **8** | 30 | 30 | 8,0 | 8,0 |
| | | 2 | 3,4 | **8** | **8** | 29 | 8 | 8,0 | 8,7 |
| 7 | 6 | 3 | 3,3 | **9** | **9** | 30 | 26 | 9,0 | 9,1 |
| | | 4 | 2,9 | **7** | **7** | 29 | 29 | 7,0 | 7,0 |
| | | 5 | 3,5 | **8** | **8** | 24 | 6 | 8,2 | 8,8 |
| | | 1 | 4,6 | **10** | **10** | 28 | 24 | 10,1 | 10,2 |
| | | 2 | 4,4 | **9** | **9** | 13 | 19 | 9,6 | 9,4 |
| 8 | 6 | 3 | 4,8 | **10** | **10** | 30 | 30 | 10,0 | 10,0 |
| | | 4 | 4,8 | **9** | **9** | 11 | 1 | 9,6 | 10,3 |
| | | 5 | 4,4 | **10** | **10** | 30 | 30 | 10,0 | 10,0 |
| | | 1 | 5,3 | **10** | **10** | 18 | 2 | 10,4 | 10,9 |
| | | 2 | 4,9 | **10** | **10** | 19 | 17 | 10,4 | 10,4 |
| 9 | 6 | 3 | 6,2 | **10** | **10** | 13 | 18 | 10,6 | 10,4 |
| | | 4 | 6,6 | **12** | **12** | 19 | 30 | 12,4 | 12,0 |
| | | 5 | 5,5 | **9** | **9** | 28 | 30 | 9,1 | 9,0 |
| | | 1 | 6,5 | **11** | **11** | 21 | 26 | 11,3 | 11,1 |
| | | 2 | 7,7 | **12** | **12** | 25 | 14 | 12,2 | 12,5 |
| 10 | 6 | 3 | 6,3 | **11** | **11** | 17 | 2 | 11,4 | 12,2 |
| | | 4 | 7,1 | **12** | **12** | 27 | 28 | 12,1 | 12,1 |
| | | 5 | 7,0 | **12** | **12** | 26 | 10 | 12,1 | 12,7 |

Tabela 3: Principais resultados das meta-heurísticas. Limite de tempo (em segundos) usado como critério de parada.

O segundo experimento tem por objetivo comparar a performance das meta-heurísticas propostas. Cada instância foi executada 30 vezes. Na Tabela 3, a coluna *Melhor* indica o melhor resultado dentre as 30 instâncias. A coluna *Quant.* indica a quantidade de vezes em que o melhor resultado foi encontrado e a coluna *Média* indica a média aritmética simples das soluções finais das 30 execuções. O critério de parada adotado foi o tempo limite equivalente a $|V|/10.0$ segundos, onde $|V|$ indica a ordem do grafo em questão.

Em relação às melhores soluções dentre as 30 execuções, tanto o EA quanto o ILS alcançaram os mesmos resultados. Em termos de robustez, todos os dois algoritmos foram relativamente bem. O algoritmo ILS foi o mai robusto, considerando as colunas *Quant.* e *Média*. Sob todos os aspectos analisados, o algoritmo ILS pode ser considerado o melhor método heurístico para o Problema de Inundação na versão livre.

| Instância | | | EA/ILS | Bt1 | | Bt2 | | Bt3 | |
|---|---|---|---|---|---|---|---|---|---|
| N | C | # | Sol. | Sol. | Tempo | Sol. | Tempo | Sol. | Tempo |
| | | 1 | 6 | 6 | 3,8 | 6 | 4,2 | 6 | 0,0 |
| | | 2 | 6 | 6 | 17,7 | 6 | 9,1 | 6 | 0,0 |
| 4 | 6 | 3 | 6 | 6 | 2,0 | 6 | 2,0 | 6 | 0,0 |
| | | 4 | 6 | 6 | 22,6 | 6 | 18,0 | 6 | 0,0 |
| | | 5 | 6 | 6 | 5,4 | 6 | 4,7 | 6 | 0,0 |
| | | 1 | 6 | 6 | 60,1 | 6 | 46,7 | 6 | 0,0 |
| | | 2 | 6 | 6 | 77,2 | 6 | 65,0 | 6 | 0,1 |
| 5 | 6 | 3 | 7 | 7 | 631,8 | 7 | 462,4 | 7 | 0,0 |
| | | 4 | 6 | 6 | 46,3 | 6 | 7,3 | 6 | 0,0 |
| | | 5 | 7 | 7 | 342,4 | 7 | 367,6 | 7 | 0,8 |
| | | 1 | 8 | 10 | 7200,0 | 9 | 7200,0 | 9 | 4,9 |
| | | 2 | 8 | 11 | 7200,0 | 10 | 7200,0 | 10 | 4,2 |
| 6 | 6 | 3 | 8 | 9 | 7200,0 | 9 | 7200,0 | 9 | 5,1 |
| | | 4 | 8 | 9 | 7200,0 | 9 | 7200,0 | 9 | 4,8 |
| | | 5 | 9 | 11 | 7200,0 | 9 | 7200,0 | 9 | 549,7 |
| | | 1 | 8 | 9 | 7200,0 | 9 | 7200,0 | 9 | 11,3 |
| | | 2 | 8 | 13 | 7200,0 | 10 | 7200,0 | 10 | 10,1 |
| 7 | 6 | 3 | 9 | 11 | 7200,0 | 10 | 7200,0 | 10 | 787,0 |
| | | 4 | 7 | 8 | 7200,0 | 8 | 7200,0 | 8 | 0,1 |
| | | 5 | 8 | 10 | 7200,0 | 9 | 7200,0 | 9 | 10,4 |
| | | 1 | 10 | 14 | 7200,0 | 10 | 7200,0 | 10 | 7200,0 |
| | | 2 | 9 | 12 | 7200,0 | 9 | 7200,0 | 9 | 5898,4 |
| 8 | 6 | 3 | 10 | 19 | 7200,0 | 11 | 7200,0 | 11 | 7200,0 |
| | | 4 | 9 | 19 | 7200,0 | 13 | 7200,0 | 13 | 7200,0 |
| | | 5 | 10 | 16 | 7200,0 | 15 | 7200,0 | 15 | 7200,0 |
| | | 1 | 10 | 19 | 7200,0 | 15 | 7200,0 | 15 | 7200,0 |
| | | 2 | 10 | 16 | 7200,0 | 12 | 7200,0 | 12 | 7200,0 |
| 9 | 6 | 3 | 10 | 21 | 7200,0 | 13 | 7200,0 | 13 | 7200,0 |
| | | 4 | 12 | 21 | 7200,0 | 15 | 7200,0 | 15 | 7200,0 |
| | | 5 | 9 | 18 | 7200,0 | 11 | 7200,0 | 11 | 7200,0 |
| | | 1 | 11 | 21 | 7200,0 | 14 | 7200,0 | 14 | 7200,0 |
| | | 2 | 12 | 32 | 7200,0 | 13 | 7200,0 | 13 | 7200,0 |
| 10 | 6 | 3 | 11 | 24 | 7200,0 | 16 | 7200,0 | 16 | 7200,0 |
| | | 4 | 12 | 16 | 7200,0 | 13 | 7200,0 | 13 | 7200,0 |
| | | 5 | 12 | 21 | 7200,0 | 15 | 7200,0 | 15 | 7200,0 |

Tabela 4: Soluções obtidas pelo Algoritmo de Bactracking. Tempo em segundos.

No último experimento, foi utilizado o algoritmo *Backtracking* com 3 configurações diferentes. Na primeira (Bt1), só é utilizada a ordenação pela excentricidade dos vértices (linha 8, Algoritmo 1) e nenhuma solução inicial é passada. Na segunda configuração (Bt2), é utilizada também a ordenação pelas cores mais frequentes (linha 10, Algoritmo 1). Na terceira configuração (Bt3), além das duas ordenações, a melhor solução produzida pelo EA/ILS também é passada inicialmente. Desta forma, o *Backtracking* realiza um número muito maior de podas, uma vez que ele só explora ramos que possam ser melhores que a solução corrente.

A Tabela 4 mostra, para cada instância, a melhor solução do EA/ILS, a melhor solução encontrada por cada configuração do *Backtracking* e o tempo que o algoritmo consumiu, considerando uma limite máximo de 7200 segundos. Nas instâncias pequenas, provenientes de matrizes com tamanho $4 \times 4$ e $5 \times 5$, a configuração mais simples conseguiu encontrar a solução ótima em um tempo muito menor, se comparado à formulação matemática. Entretanto, para instâncias maiores, ambas

as configurações Bt1 e Bt2 começaram a ter um desempenho mais fraco em relação aos resultados do EA. Na antepenúltima instância, por exemplo, a configuração Bt1 conseguiu uma solução com o valor 24 e Bt2 conseguiu com valor 16, enquanto os algoritmos EA e ILS encontraram uma solução com valor 11. Comparativamente, a configuração Bt2 é melhor do que a Bt1, mostrando que a ordenação por cores mais frequentes tem um impacto muito positivo na performance do algoritmo.

Adicionalmente, também é possível observar que a configuração Bt3 consegue ser ainda melhor. Por meio dela, não foi possível encontrar solução nenhuma melhor do que a fornecida pelo EA/ILS em instância alguma. Para instâncias de matrizes até $7 \times 7$, todas as soluções do EA/ILS foram provada como sendo ótimas além de uma instância de tamanho $8 \times 8$. Nas outras instâncias, a configuração Bt3 não conseguiu terminar a execução antes do tempo limite de 7200 segundos, mas pelo fato de também não ter encontrado solução melhor, existe bons indícios de que a solução do EA/ILS tem excelente qualidade, inclusive podendo ser possivelmente a solução ótima dessas instâncias. Nestes casos, o algoritmo *Backtracking* funciona também como certificador da qualidade da solução heurística fornecida. Em diversos casos, essa certificação consumiu menos de 0,1 segundos (valores 0,0 na Tabela 4).

## 5. Conclusões e Trabalhos Futuros

Este artigo tratou do Problema de Inundação em grafos na versão livre. Nesta versão, cada inundação é indicada por um vértice inicial e uma cor, sendo, portanto, uma generalização da versão fixa, onde o vértice inicial é sempre o mesmo. Na literatura, há apenas trabalhos teóricos sobre o problema que é considerado NP-difícil.

Foram propostas duas meta-heurísticas: um Algoritmo Evolutivo (EA) e um Iterated Local Search (ILS). Também foram propostos um algoritmo *backtracking* e uma modelagem matemática. A modelagem apresentou desempenho muito ruim mesmo para instâncias de pequeno porte. Por outro lado, as meta-heurísticas, especialmente o EA e o ILS, conseguiram resultados de excelente qualidade em instâncias de diversos tamanhos.

A qualidade das soluções obtidas pôde ser certificada pelo algoritmo de *backtracking* que possui bons mecanismos de poda e exploração priorizada de ramos mais promissores. Ao se passar uma solução inicial de boa qualidade, proveniente do EA/ILS, o *backtracking* conseguiu certificar que a solução era ótima em várias instâncias. Em outras de maior porte, após 2 horas de processamento, o algoritmo não conseguiu encontrar solução de melhor qualidade. Isto certifica, em parte, que a solução fornecida tem boa qualidade.

Como principal trabalho futuro está o desenvolvimento de uma formulação matemática mais eficiente, que seja capaz de encontrar a solução ótima para instâncias de porte médio. Também seria interessante o desenvolvimento de uma versão paralela do algoritmo *Backtracking*, para que a certificação da qualidade das soluções heurísticas possa ser feita com melhor desempenho.

## Referências

Adriaen, M., De Causmaecker, P., Demeester, P., e Vanden Berghe, G. (2004). 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*.

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

Barone, P., Bonizzoni, P., Vedova, G. D., e Mauri, G. (2001). An approximation algorithm for the shortest common supersequence problem: An experimental analysis. In *ACM Symposium on Applied Computing*, p. 56–60.

Barros, B., Pinheiro, R., e Souza, U. (2015). Métodos heurísticos e exatos para o problema de inundação em grafos. In *Anais do XLVII Simpósio Brasileiro de Pesquisa Operacional (SBPO2015)*.

Clifford, R., Jalsenius, M., Montanaro, A., e Sach, B. (2012). The complexity of flood filling game. *Theory Comput Syst*, 50:72–92.

Fellows, M. R., Hallett, M. T., e Stege, U. (2003). Analogs & duals of the mast problem for sequences & trees. *Journal of Algorithms*, 49:192–216.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc.

Gonçalves, J. F., Resende, M. G. C., e Costa, M. D. (2016). A biased random-key genetic algorithm for the minimization of open stacks problem. *International Transactions in Operational Research*, 23:25–46.

LabPixies (2015). Labpixies - the coolest games! `http://www.labpixies.com`. Acessado em 27/12/2015.

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

Lourenço, H. R., Martin, O. C., e Stutzle, T. (2003). Iterated local search. In Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics*, p. 321–353. Kluwer Academic Publishers.

Lourenço, H. R., Martin, O. C., e Stutzle, T. (2010). Iterated local search: Framework and applications. In Glover, F. e Kochenberger, G. A., editors, *Handbook of Metaheuristics, 2nd Edition*, volume 146, p. 363–397. Kluwer Academic Publishers.

Meeks, K. e Scott, A. (2011). The complexity of flood-filling games on graphs. *Discrete Applied Mathematics*, 160:959–969.

Meeks, K. e Scott, A. (2013). The complexity of free-flood-it on 2n boards. *Theoretical Computer Science*, 500:25–43.

Penna, P. H. V., Subramanian, A., e Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19:201–232.

Rahmann, S. (2003). The shortest common supersequence problem in a microarray production setting. *Bioinformatics*, 19:156–161.

Silva, A. R. V. (2016). Uma meta-heurística eficiente para o problema de inundação em grafos. In *Anais do XLVIII Simpósio Brasileiro de Pesquisa Operacional (SBPO2016)*.

Silva, A. R. V., Machado, R. V., e da Silva, M. D. (2016). Heurísticas construtivas para um problema de inundação em matrizes. In *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics (CNMAC2016)*.

Silva, A. R. V. e Ochi, L. S. (2016). Uma nova formulação matemática eficiente para o problema de inundação em grafos. In *Anais do XVIII Latin-Iberoamerican Conference on Operations Research (CLAIO2016)*.

Sim, J. e Park, K. (2003). The consensus string problem for a metric is np-complete. *Journal of Discrete Algorithms*, 1:111–117.

Souza, U., Protti, F., e da Silva, M. D. (2013). Parameterized complexity of flood-filling games on trees. *Lecture Notes in Computer Science*, 7936:531–542.

Souza, U., Protti, F., e da Silva, M. D. (2014). An algorithmic analysis of flood-it and free-flood-it on graph powers. *Discrete Mathematics and Theoretical Computer Science*, 16:279–290.