

Estudo Algorítmico de Problemas de Infecção em Grafos

Lucas Rodrigues Keiler - Engenharia de Computação - UFC



Banca:

- Prof. Dr. Rudini Menezes Sampaio (Orientador)
- Prof. Dr. Pablo Mayckon Silva Farias
- Prof. Dr. Carlos Estêvão Rolim Fernandes
- Prof. Dr. Jorge Herbert Soares de Lira

03 de dezembro de 2019

Conteúdo

1 Introdução

- Visão Simplificada do Problema
- Histórico do Problema

2 Objetivos e Metodologia

- Objetivos
- Subproblemas estudados
- Metodologia

3 Resultados

4 Conclusão

Conteúdo

1 Introdução

- Visão Simplificada do Problema
- Histórico do Problema

2 Objetivos e Metodologia

- Objetivos
- Subproblemas estudados
- Metodologia

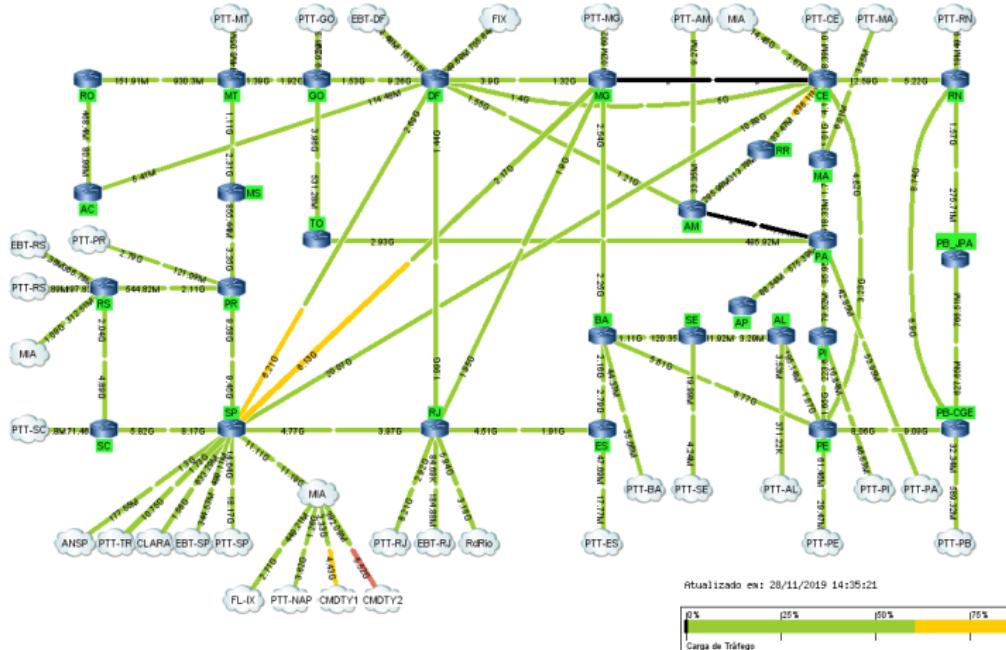
3 Resultados

4 Conclusão

Introdução - Visão Simplificada do Problema

- Seja G um grafo
- Cada vértice $v \in G$ possui um estado: *infetado* ou *não-infetado* (sadio).
- Define-se regras de transição entre estes estados (determinísticas ou não).
- Estuda-se métricas do processo de infecção.

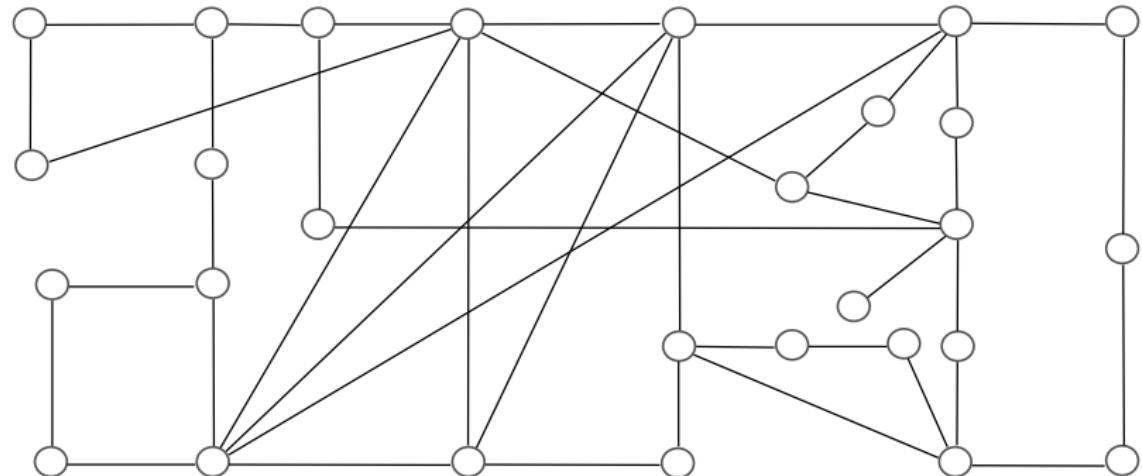
Introdução - Exemplo



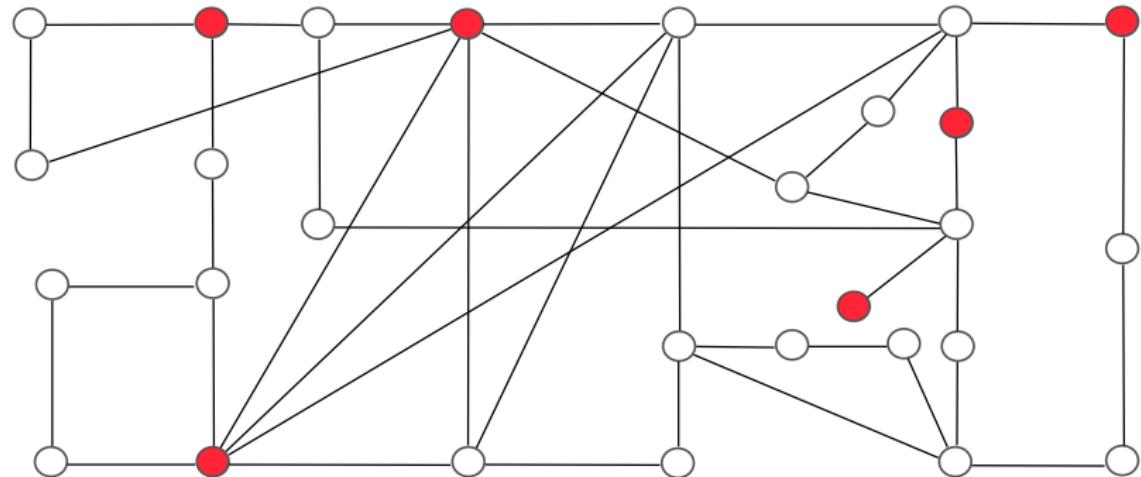
Topologia da Rede Nacional de Pesquisa - RNP

Disponível em: <https://www.rnp.br/sistema-rnp/ferramentas/panorama-de-trafego>

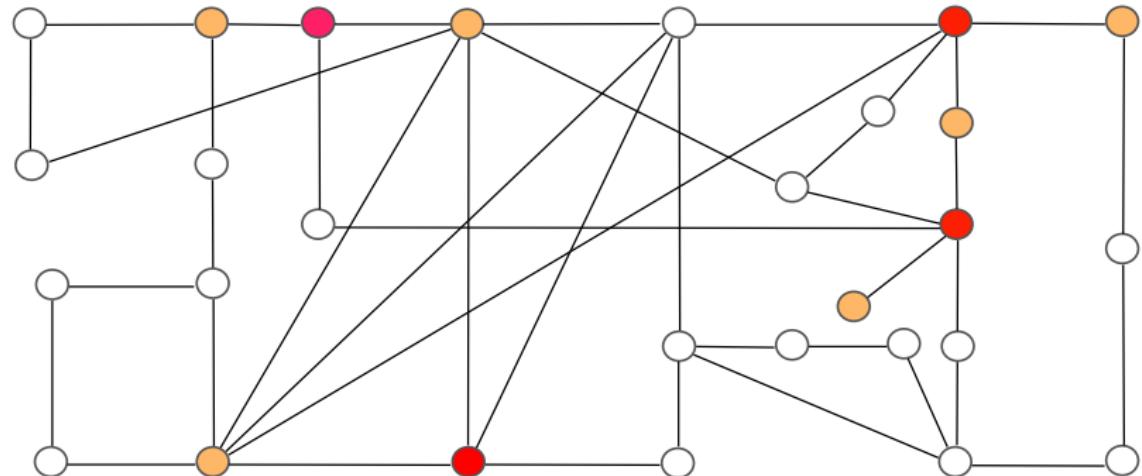
Introdução - Exemplo



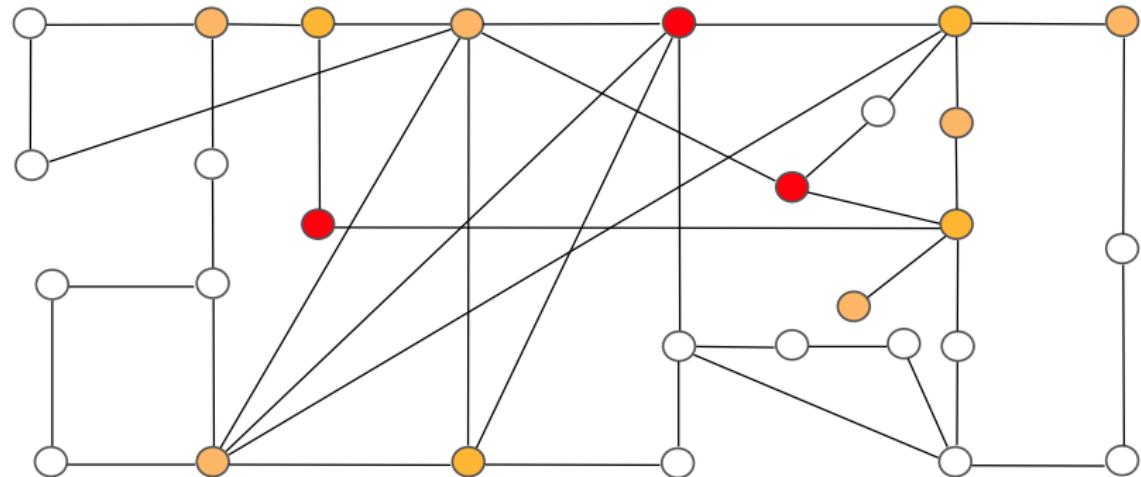
Introdução - Exemplo



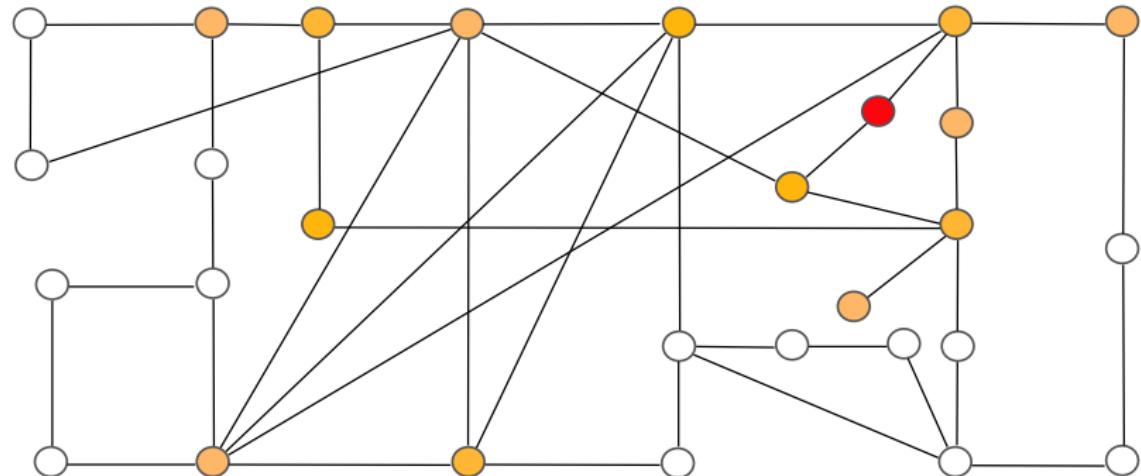
Introdução - Exemplo



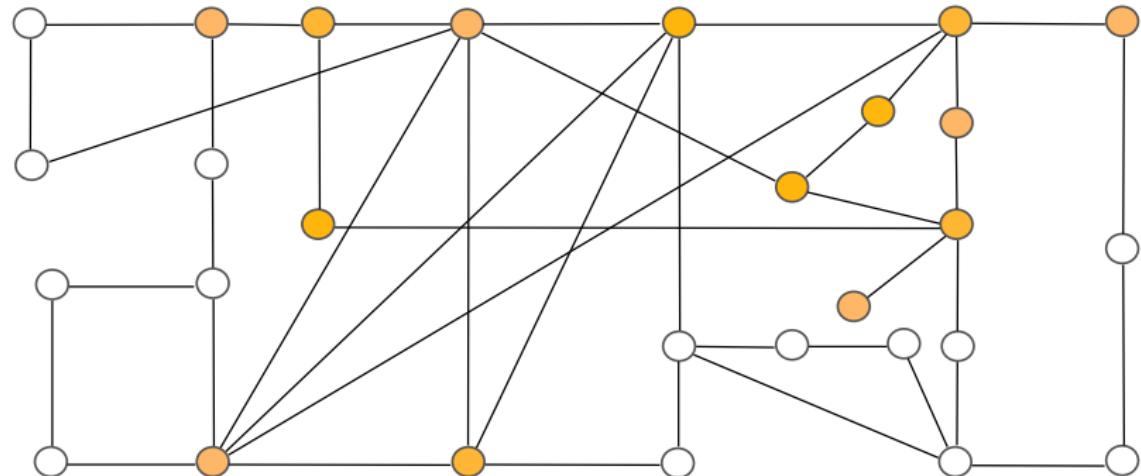
Introdução - Exemplo



Introdução - Exemplo

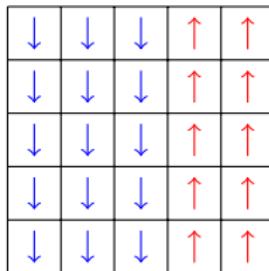


Introdução - Exemplo



Introdução - Histórico do Problema

- Modelo de *Ising*:
 - Elaboração do problema - Wilhelm Lenz - 1920
 - Solução para uma dimensão - Ernst Ising - 1924
- Modelo de Ferromagnetismo na Matéria
- A Estrutura do Modelo:
 - Átomos possuem *spins* igual a 1 ou -1 .
 - *Spins* são arranjados em *grids*. Assim, ocorre interação entre *spins vizinhos*.



Introdução - Histórico do Problema

- *Spins* vizinhos alinhados possuem estado de energia inferior.
- A estrutura como um todo tende a menor configuração energética.
- O Calor provoca distúrbios na estrutura.

Fonte: ByHeMath-Ownwork, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=37327967>

- Em dimensão 2d, permite a análise de mudanças de fase na matéria.

Introdução - Histórico do Problema

- Autômatos Celulares:

- John von Neumann - Autorreplicação de Máquinas - 1948.
- Sugestão de Stanislaw Ulam, colega de von Neumann em Los Alamos.

- Estrutura de Autômatos Celulares:

- Grids d-dimensionais de elementos discretos (células).
- Conjunto de $k \geq 2$ estados possíveis para as células.
- Função local de transição de estados: estado atual da célula e de sua vizinhança.

Introdução - Histórico do Problema

Fonte: ByKieff-Ownwork, CCBY-SA3.0, <https://commons.wikimedia.org/w/index.php?curid=101736>

Introdução - Histórico do Problema

- A estrutura de infecção em grafos continua aparecendo em diversas áreas da ciência:
 - Difusão de Fluídos - (BROADBENT; HAMMERSLEY, 1957)
 - Física de Estado Sólido (CHALUPA et al., 1979)
 - Infectologia - (GRASSBERGER, 1983)
- Em especial, possui algumas aplicações em computação:
 - Propagação de falhas em *clusters* de armazenamento (KICKPATRICK el al., 2002)
 - Algoritmos de Consenso - Recuperação de falhas (PELEG, 2002)
 - Propagação de *worms* [Segurança da Informação] (GRIFFIN, BROOKS, 2006)

Conteúdo

1 Introdução

- Visão Simplificada do Problema
- Histórico do Problema

2 Objetivos e Metodologia

- Objetivos
- Subproblemas estudados
- Metodologia

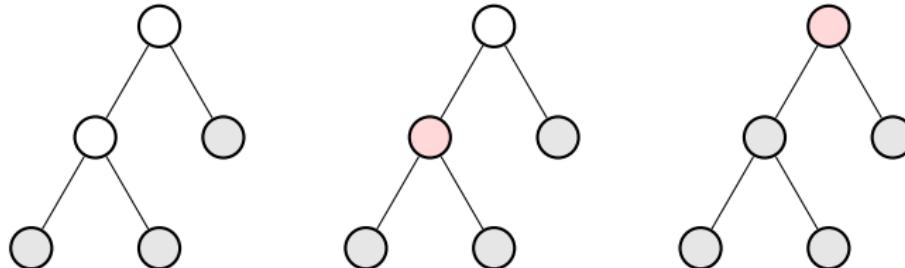
3 Resultados

4 Conclusão

- Motivação: Grande número de aplicações do problema em diferentes áreas.
- **Objetivo Principal**
 - Estudar subproblemas de infecção em grafos de um ponto de vista algorítmico, propondo implementações dos algoritmos estudados.
- **Objetivo Secundário**
 - Estratégias de Validação
- Dois subproblemas são estudados:
 - *2-Bootstrap Percolation*
 - *Target Set Selection*

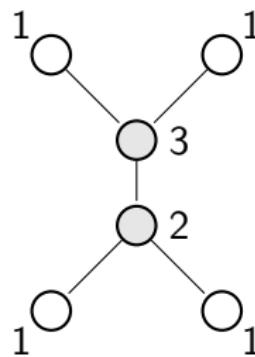
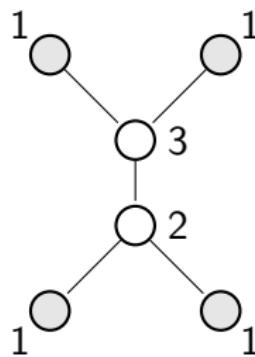
2-Bootstrap Percolation (Convexidade P3)

- Regra de Infecção: Vértice *sadio* é infectado no tempo atual, caso dois vizinhos tenham sido infectados em tempo inferior ao atual.
- Não existe transição *infectado* \rightarrow *não-infectado*.
- Neste trabalho, a métrica estudada é o tempo máximo de infecção $t(G)$.



Target Set Selection (TSS)

- Regra de infecção: $\forall v \in V(G)$ define-se um limite $I(v) \in [1, d(v)]$ de vizinhos que, todos infectados, provocam a infecção do vértice v .
- Não existe transição *infetado* \rightarrow *não-infetado*.
- Objetivo: Encontrar o menor conjunto (*Target Set*) inicialmente infectado que percola o grafo completamente.



Objetivos e Metodologia - Metodologia

- Implementações:
 - Análise de pseudocódigo (desenvolver um, caso não exista).
 - Verificação da Complexidade (analisar caso a publicação do algoritmo não o faça).

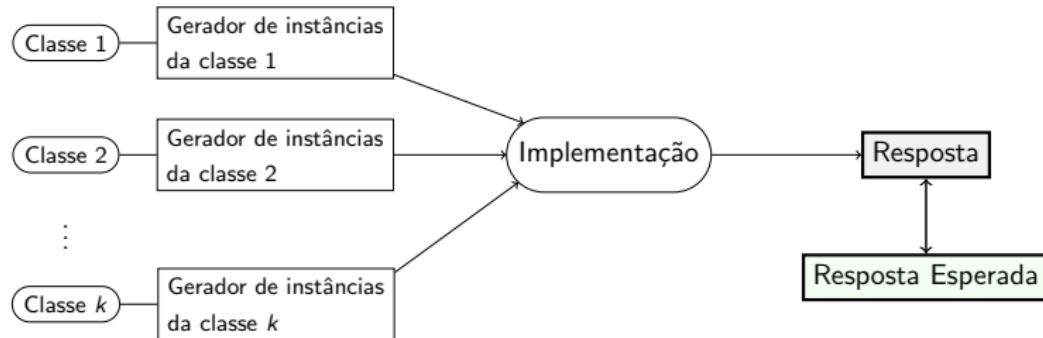
- Testes:
 - Validade da Implementação: está resolvendo o problema proposto?
 - Cumprimento da Complexidade: mesma complexidade da teoricamente proposta?

Objetivos e Metodologia - Verificações

Estratégias de teste da implementação:

- Casos de Teste Determinísticos:

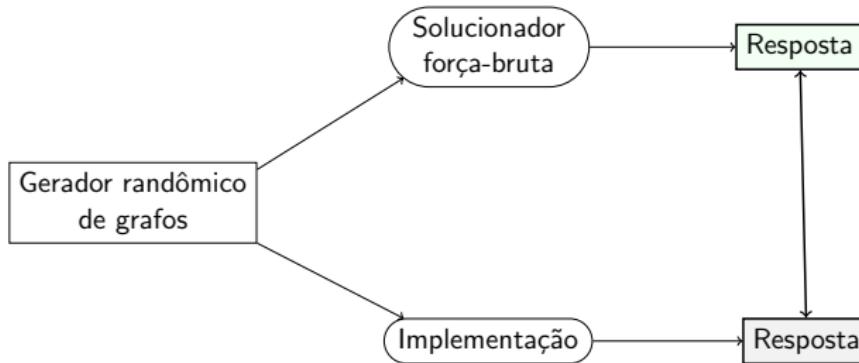
- Seleção de classes de grafos determinísticas que possuam resultados “equacionados”
- Gerar instâncias de várias cardinalidades para estas classes.
- Validar solução da implementação com a equacionada.



Objetivos e Metodologia - Verificações

- Casos de Teste Randômicos:

- Implementação de um gerador randômico de instâncias do problema específico.
- Execução das instâncias pelo algoritmo.
- Comparação com um verificador “força bruta”.



Testes de Complexidade da Implementação

- Geração de instâncias randômicas (ou determinística de pior caso, se possível).
- Execução do algoritmo com tomada de tempo.
- Plotagem de gráficos dos tempos de execução:
 - Curvas de tempo de execução da implementação desenvolvida.
 - Curvas “guia” de complexidade esperada.

```
        double r;
        for(int i=1;i<=n;i++){
            r = 273761253/1232;
            r = 86418113515/56448;
            r = 54723168516/215;
            r = 245724858115/515;
            r = 218564815188/475;
            r = 784318435181/541;
        }
```

Objetivos e Metodologia - Verificações

- Plotagem de tempo de execução já é utilizada como comparação de desempenho.

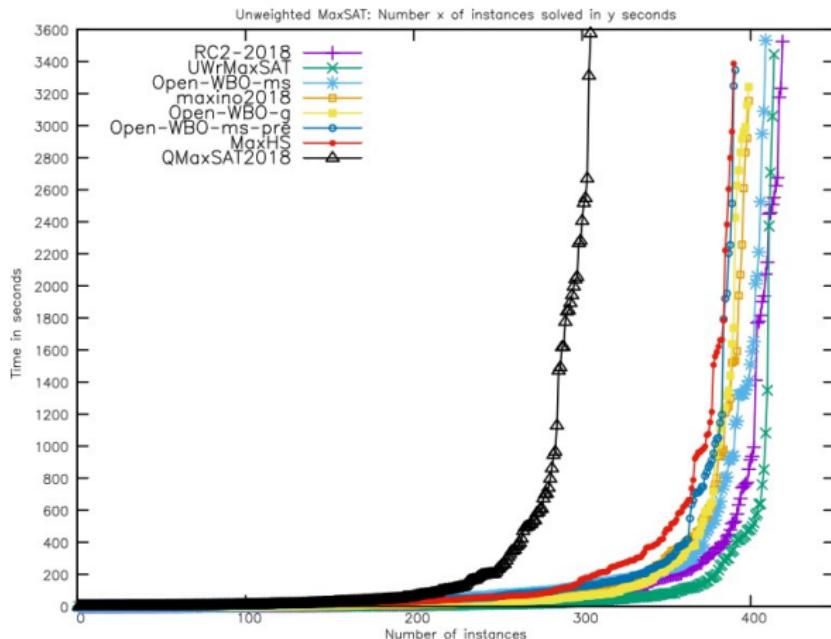


Gráfico de tempo de execução para resolvedores MAXSAT

Disponível em: <https://maxsat-evaluations.github.io/2019/results/complete/unweighted/summary.html>

Conteúdo

1 Introdução

- Visão Simplificada do Problema
- Histórico do Problema

2 Objetivos e Metodologia

- Objetivos
- Subproblemas estudados
- Metodologia

3 Resultados

4 Conclusão

Resultados - Visão Geral dos Algoritmos

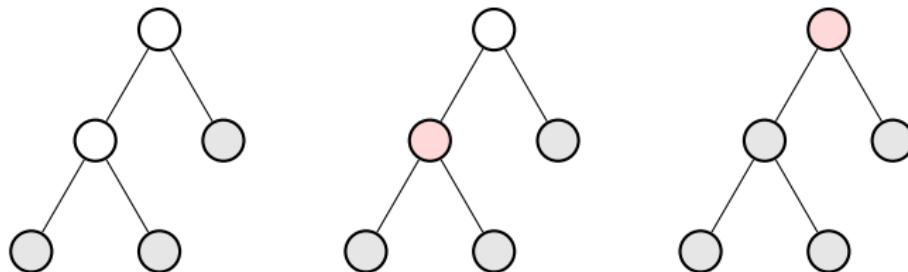
- Problema de Decisão $t(G) \geq k$ em grafos com $\Delta \leq 3$. m
(MARCILON; SAMPAIO, 2018a)
 - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$. m (MARCILON; SAMPAIO, 2018a)
 - Problema de Decisão $t(G) \geq n - k$ em grafos quaisquer. m
(MARCILON; SAMPAIO, 2018a)
 - Problema de Decisão $t(G) \geq 3$ em grafos bipartidos. m (MARCILON; SAMPAIO, 2018b)
 - Problema de Decisão $t(G) \geq 3$ em grafos quaisquer. m (MARCILON; SAMPAIO, 2018b)
- *Target Set Selection em Árvores.* (CHEN, 2009)
- Cálculo de $t(G)$ em Árvores.

Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

Regra de Infecção: 2-bootstrap percolation.

Entrada: Grafos quaiser com grau máximo 3 e um inteiro positivo k .

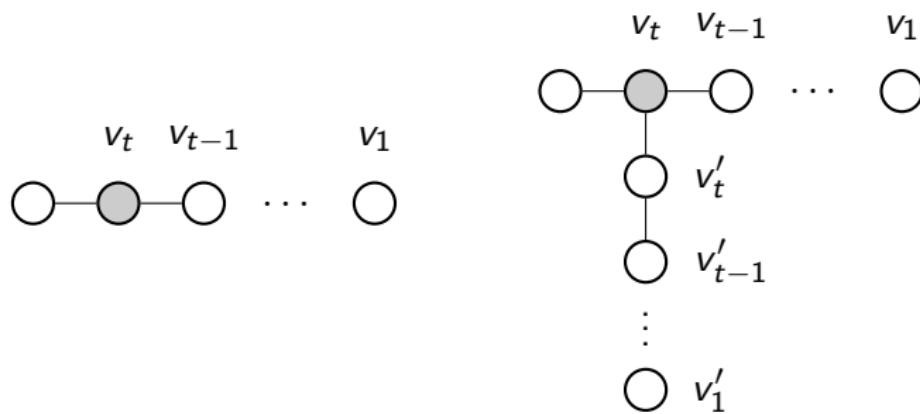
Saída: Veredicto de $t(G) \geq k$.



Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

Lema

Seja G um grafo conexo com $\Delta(G) \leq 3$ e k um inteiro não negativo. Então, $t(G) \geq k$ se, e somente se, G possui um caminho induzido P tal que, ou P tem $2k - 1$ vértices, todos com grau três; ou P tem k vértices, todos com grau três, à exceção de um extremo, que possui grau dois.



Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

Algorithm 1 $t(G) \geq k$ em grafos com $\Delta \leq 3$

```
1: resp ← falso
2: for all vértice  $v$  em  $G$  do
3:   if  $grau(v) = 2$  then
4:     profundidade ←  $k - 1$ 
5:     dfsMod( $v, 0$ )
6:   else if  $grau(v) = 3$  then
7:     profundidade ←  $2k - 2$ 
8:     dfsMod( $v, 0$ )
9:   end if
10: end for
11: Retorna resp
```

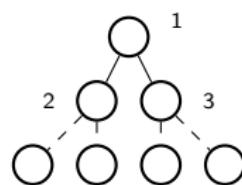
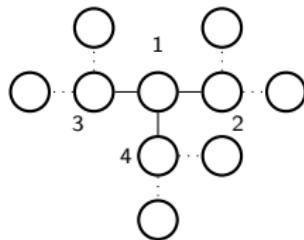
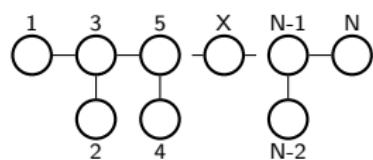
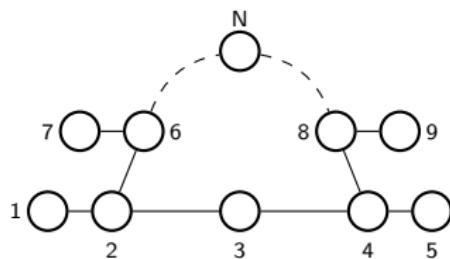
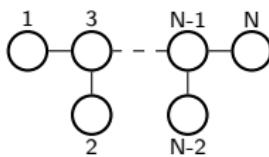
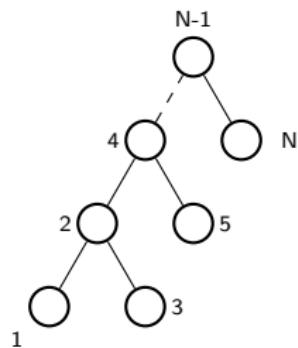
Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

Algorithm 2 dfsMod

```
1: marcar( $v$ )
2: if  $d = \text{profundidade}$  then
3:   resp  $\leftarrow$  verdadeiro
4:   Retorna
5: end if
6: for all vértice  $s$  vizinho de  $v$  do
7:   if  $s$  não marcado e induzido( $v,s$ ) then
8:     dfsMod( $s,d + 1$ )
9:   end if
10: end for
11: desmarcar( $v$ )
```

Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

- Verificação por classes de grafos definidas:



Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

- Para cada classe de grafo apresentada, é possível equacionar o tempo máximo de infecção:

Classe	Número de Vértices	$t(G)$
G1	$2K + 1, K \geq 1$	K
G2	$K, K \geq 3$	1
G3	$2K, K \geq 3$	K
G4	$2K, K \geq 3$	$\lceil K/2 \rceil$
G5	$2 * (a + b + 1) + 1$	$(\max(a, b) - 1)/2$
G6	$1 + 3(2^k - 1)$	k
G7	N	$\log(N)$

Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

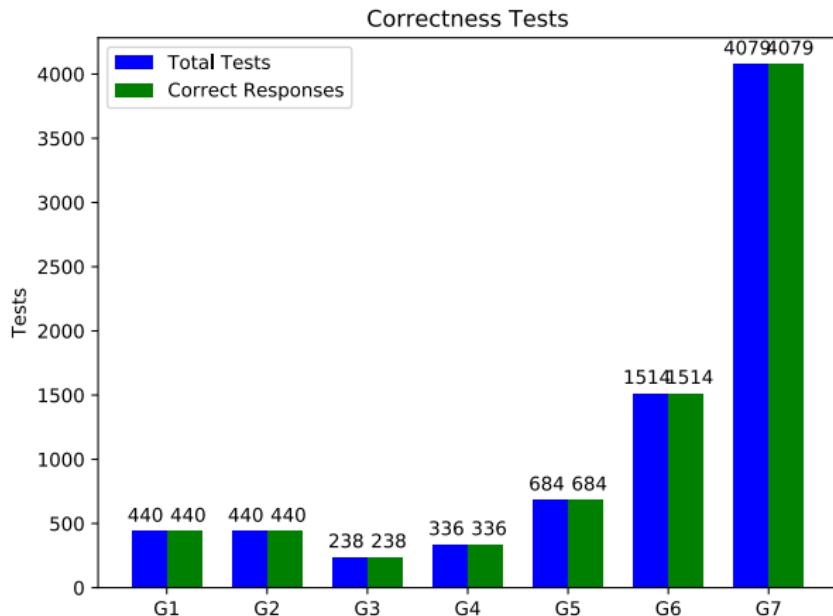


Figura: Teste de Implementação (Fonte: o autor)

Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

Complexidade:

- Supondo todo vértice com grau três, para cada vértice inicial, tem-se o número de caminhos:

$$3 \cdot 2 \cdot 2 \dots 2 = 3 \cdot 2^{l-1}$$

- Como, para vértices de grau 3, $l = 2k - 2$:

$$3 \cdot 2^{l-1} = 3 \cdot 2^{2k-3} = \frac{3}{8} \cdot 2^{2k}$$

- Por fim, considerando $k \leq c \cdot \log_2 n$:

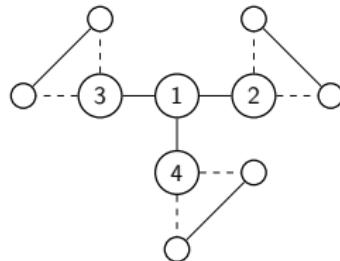
$$\frac{3}{8} \cdot 2^{2k} \leq \frac{3}{8} \cdot n^{2c}$$

- Como inicia-se a busca em cada vértice:

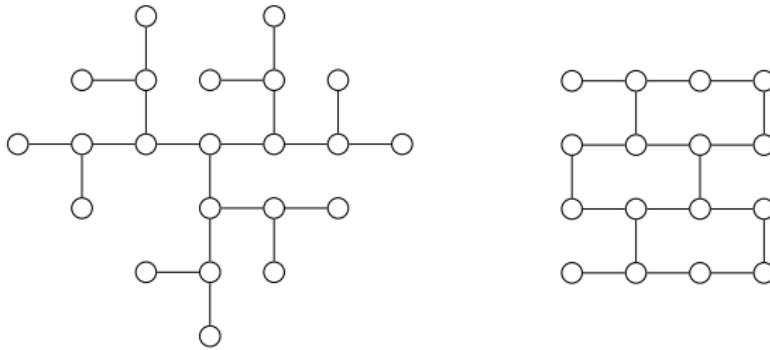
$$n \cdot \frac{3}{8} \cdot n^{2c} = O(n^{2c+1})$$

Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

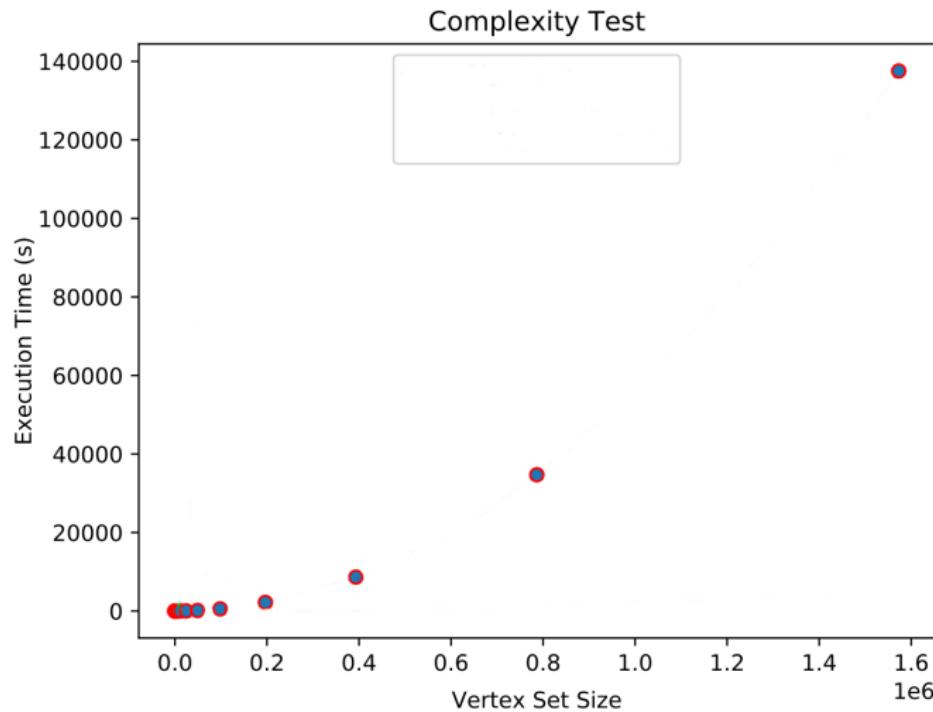
- Escolha da classe de teste:



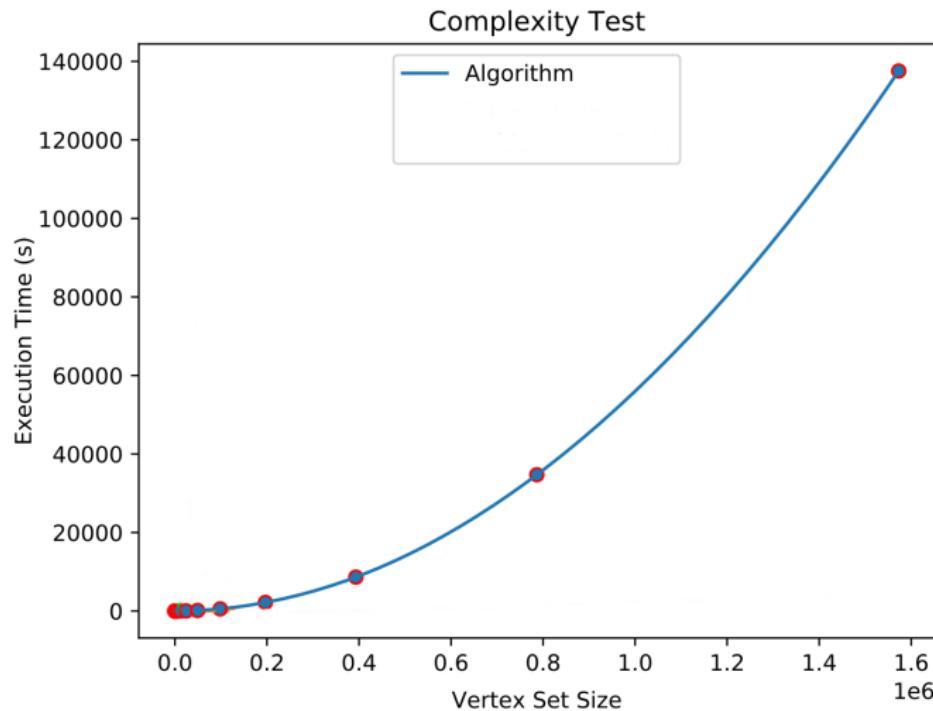
- Escolha não trivial



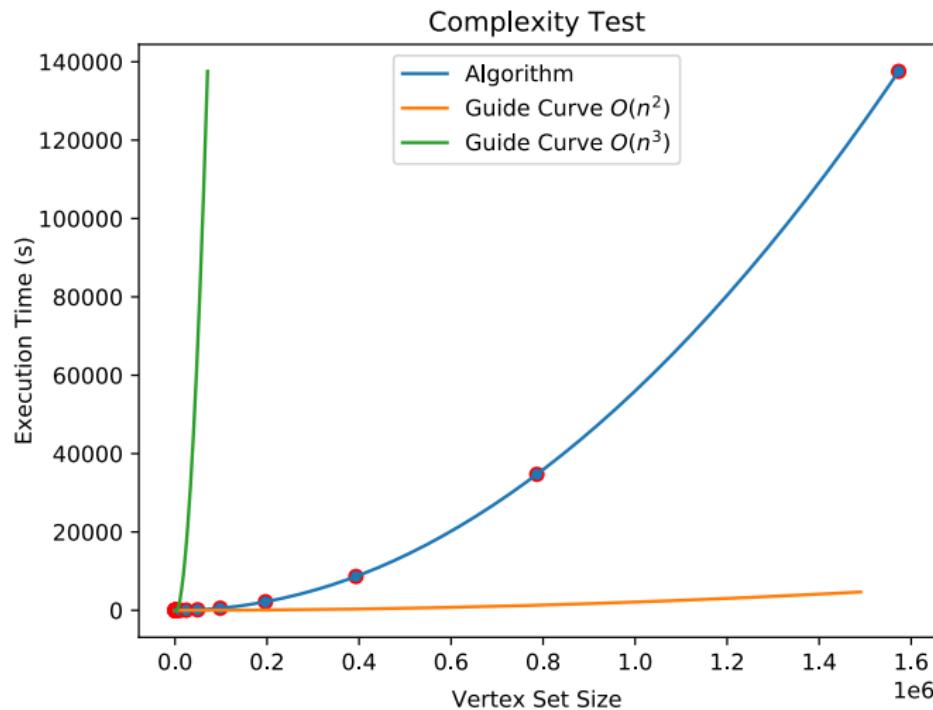
Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$



Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$



Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$



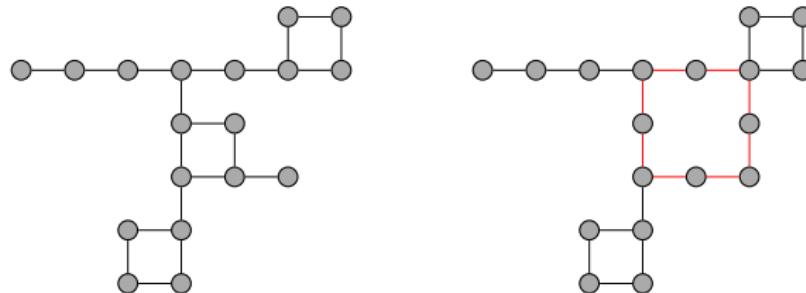
Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

Regra de Infecção: 2-bootstrap percolation.

Entrada: Grafos *solid grid* com grau máximo 3.

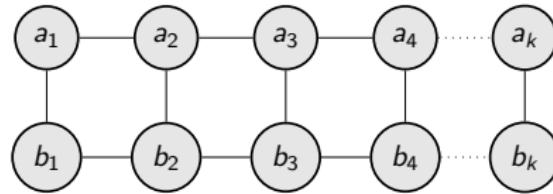
Saída: $t(G)$.

- *Solid Grids* são grafos *grid* com áreas delimitadas de valor unitário



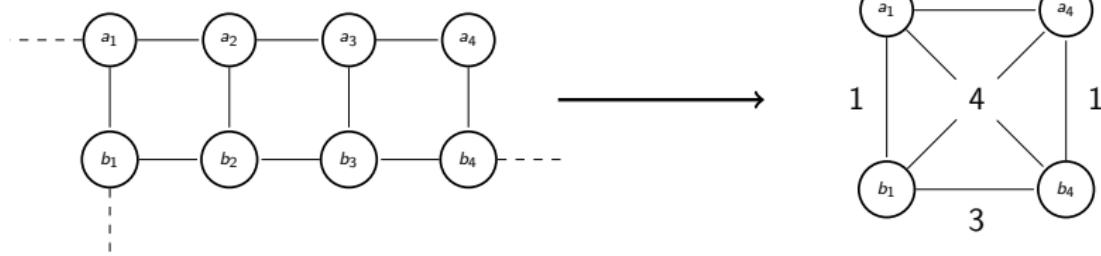
Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

- Utiliza-se do mesmo lema anteriormente apresentado: maior caminho induzido.
- Escadas em um grid sólido:



Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

- Redução no número de possibilidades para a busca de caminhos.
- Transformação de escada em K_4 ponderado (pesos são os maiores caminhos induzidos):



Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

- Formalmente:

$$w(u,v) = \begin{cases} 1, & \text{se } u, v \text{ vizinhos} \\ d_b(k), & \text{se ambas as outras extremidades têm grau 3} \\ d_b(k-1) + 1, & \text{se apenas uma extremidade possui grau 2} \\ d_b(k-2) + 2, & \text{se ambas as outras extremidades têm grau 2 e } k > 2 \\ -\infty, & \text{caso contrário.} \end{cases}$$

Onde,

$$d_b(k) = \begin{cases} (k-1) + 2 \cdot \lfloor (k+1)/4 \rfloor, & \text{se } b = 0 \\ k + 2 \cdot \lfloor (k-1)/4 \rfloor, & \text{se } b = 1 \end{cases}$$

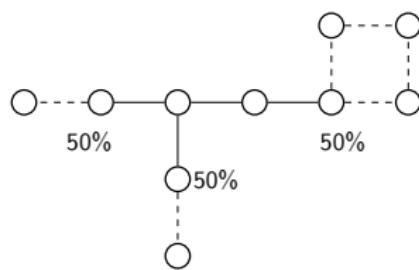
Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

Algorithm 3 Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

```
1:  $G' \leftarrow$  Transformar( $G$ )
2:  $t(G) \leftarrow 0$ 
3: for all vértice  $v$  em  $G'$  com  $d(v) \geq 2$  do
4:   if  $grau(v) = 2$  then
5:      $atual \leftarrow \text{dfsPond}(G, v) + 1$ 
6:   else
7:      $atual \leftarrow \lfloor (dfsPond(G, v) + 2)/2 \rfloor$ 
8:   end if
9:    $t(G) \leftarrow \max(t(G), atual)$ 
10: end for
11: Retorna  $t(G)$ 
```

Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

- Escolha da classe de teste:



- Validação “força-bruta” do tempo obtido.

Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

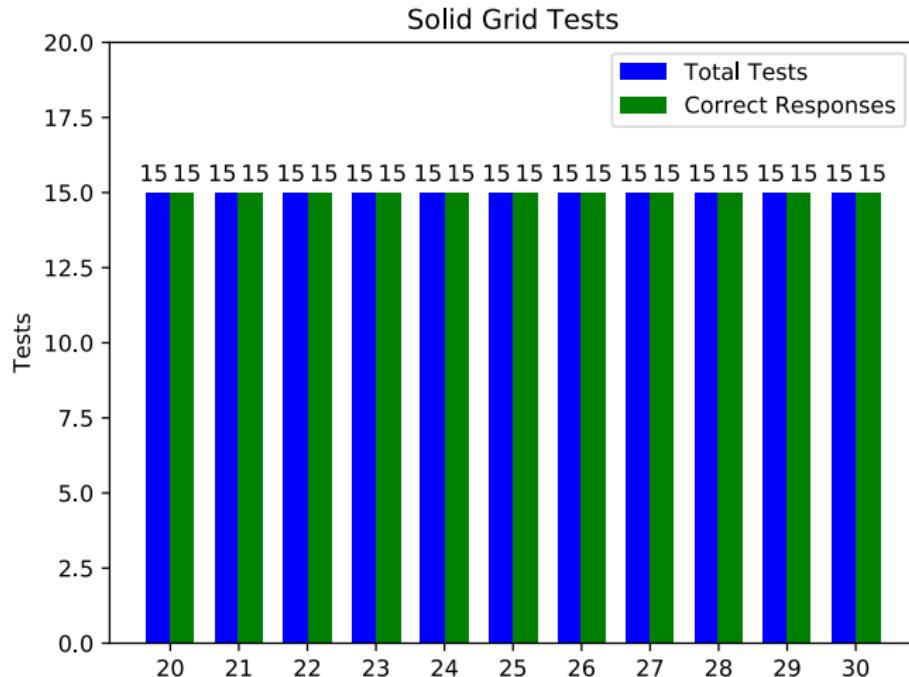


Figura: Teste de Implementação (Fonte: o autor)

Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

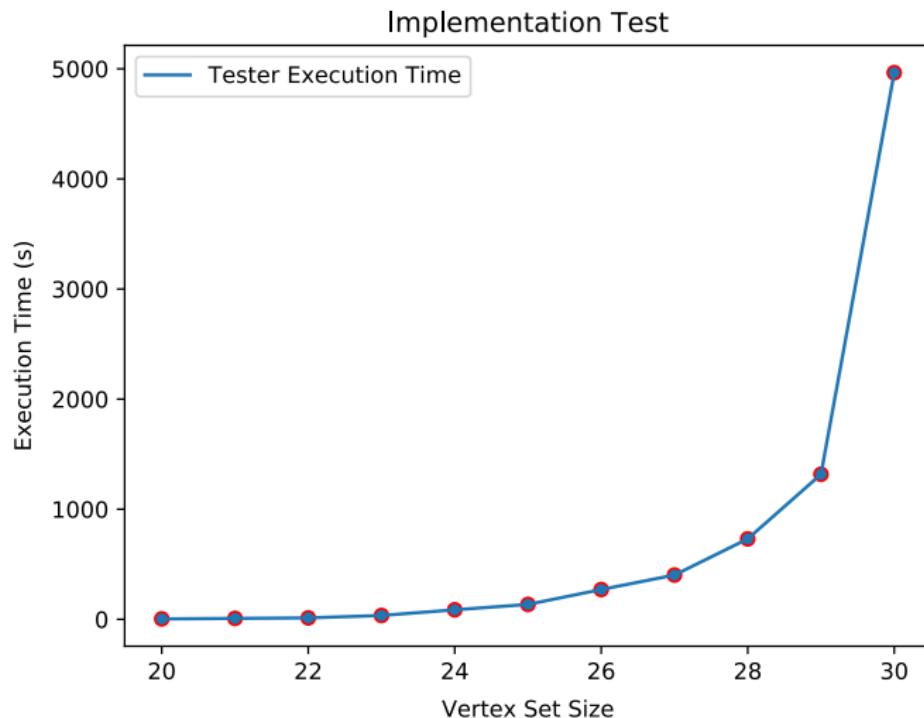


Figura: Tempos para Teste de Implementação (Fonte: o autor)

Complexidade:

- Após a transformação de escadas, existe apenas um caminho induzido entre cada par de vértice.
- A DFS modificada tem complexidade $O(m + n) = O(n)$.
- Para cada vértice, executa a DFS, portanto, complexidade final $O(n^2)$.

Resultados - Cálculo de $t(G)$ em Grids Sólidos com $\Delta = 3$

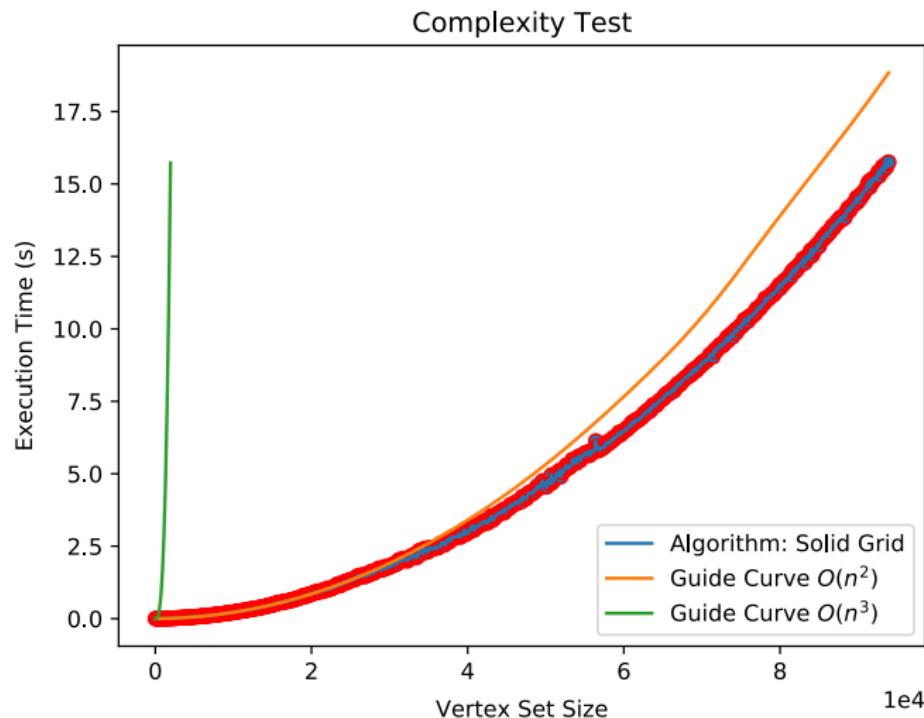


Figura: Teste de Complexidade (Fonte: o autor)

Resultados - Problema de Decisão $t(G) \geq n - k$ em grafos quaisquer

Regra de Infecção: 2-bootstrap percolation.

Entrada: Um grafo G qualquer e um inteiro $k \in [0, n]$.

Saída: Veredicto de $t(G) \geq n - k$.

Observação

Suponha que um grafo G possui $t(G) \geq n - k$. Em um processo de infecção com tempo $n - k$, há pelo menos um vértice infectado em tempo 1, um vértice infectado em tempo 2 e, assim por diante, um vértice infectado em tempo $n - k$. Deste modo, existem, no máximo, k vértices infectados no tempo 0.

- Ideia: Testar todos os possíveis conjuntos de tamanho até k vértices.

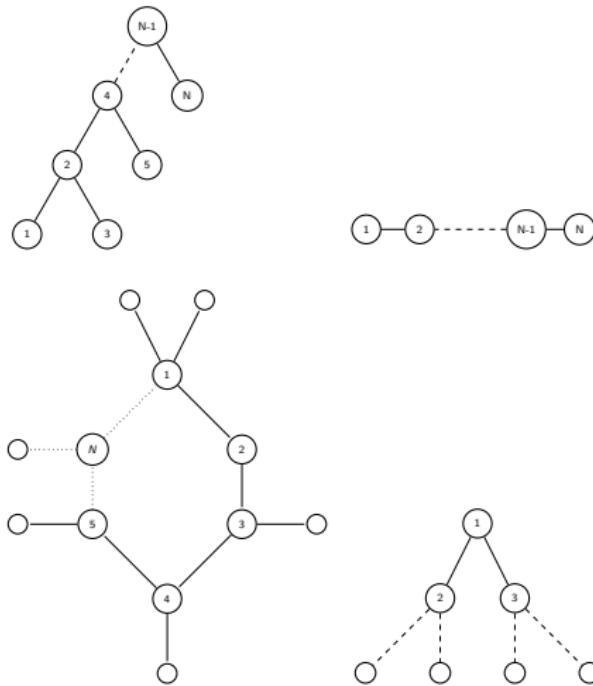
Resultados - Problema de Decisão $t(G) \geq n - k$ em grafos quaisquer

Algorithm 4 Decisão $t(G) \geq n - k$ em grafos quaisquer

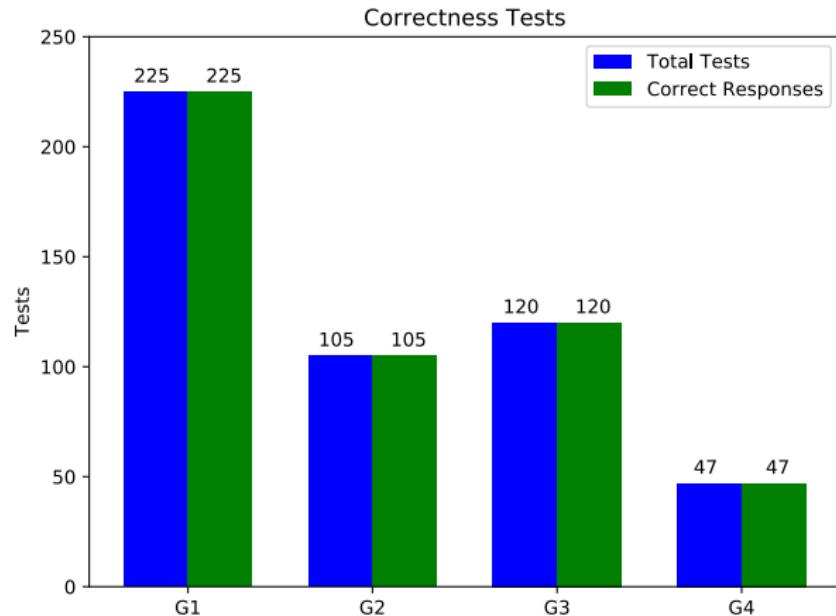
```
1: for all permutação P de vértices em G do
2:   tempo  $\leftarrow 0$ 
3:   while houver infecção em G do
4:     for all vértice v não infectado do
5:       if mais de um vizinho de v está infectado then
6:         infectar(v)
7:       end if
8:     end for
9:     tempo  $\leftarrow$  tempo + 1
10:   end while
11:   if tempo - 1  $\geq n - k$  then
12:     Retorna verdadeiro
13:   end if
14: end for
15: Retorna falso
```

Resultados - Problema de Decisão $t(G) \geq n - k$ em grafos quaisquer

- #### • Classes de Grafo de Teste.

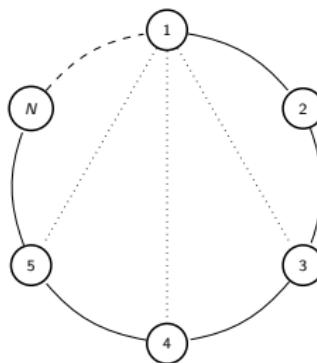


Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

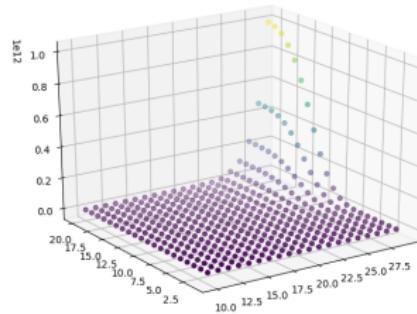


Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

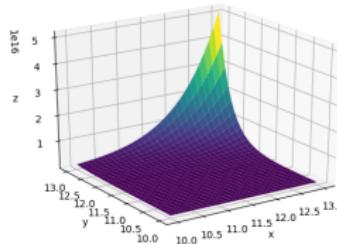
- Como existem $O(n^k)$ subconjuntos de vértices de tamanho k , e cada simulação de infecção leva um tempo $O(n \cdot m)$, a decisão $t(G) \geq n - k$ é feita em $O(m \cdot n^{k+1})$.
- Testes para o par (n, k) : Árvores
- Testes para o par (n, m) : Ciclo de cardinalidade de arestas variável



Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$

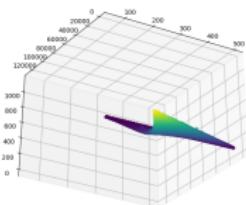


(a) Vista 1 do resultado experimental

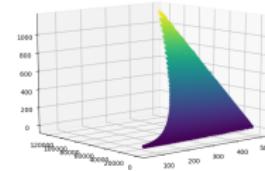


(b) Superfície de $f(x, y) = x^{(y+2)}$

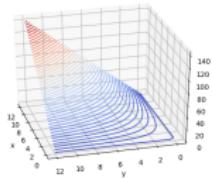
Resultados - $t(G) \geq k$ em grafos com $\Delta \leq 3$



(a) Vista 1 do resultado experimental



(b) Vista 2 do resultado experimental



(c) Superfície de $f(x, y) = x \cdot y$

Resultados - Problema de decisão $t(G) \geq 3$ em grafos bipartidos

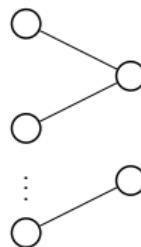
Processo de Infecção: 2-bootstrap percolation

Entrada: Grafo G bipartido.

Saída: Veredicto de $t(G) \geq 3$.

- Grafos bipartidos:

X Y



Problema de decisão $t(G) \geq 3$ em grafos bipartidos

Lema

Seja G um grafo bipartido e T_0 o conjunto de vértices v de G com $d(v) = 1$. Assim, $t(G) \geq 3$ se, e somente se, há três vértices $u, v \in N(u)$ e $s \in N_2(u)$ tal que $T_0 \cup N_{\geq 3}(u) \cup \{v, s\}$ infecta u em tempo 3.

- Ideia: Busca iterada de vértices u, v, s que satisfazem a condição de $t(u) \geq 3$.

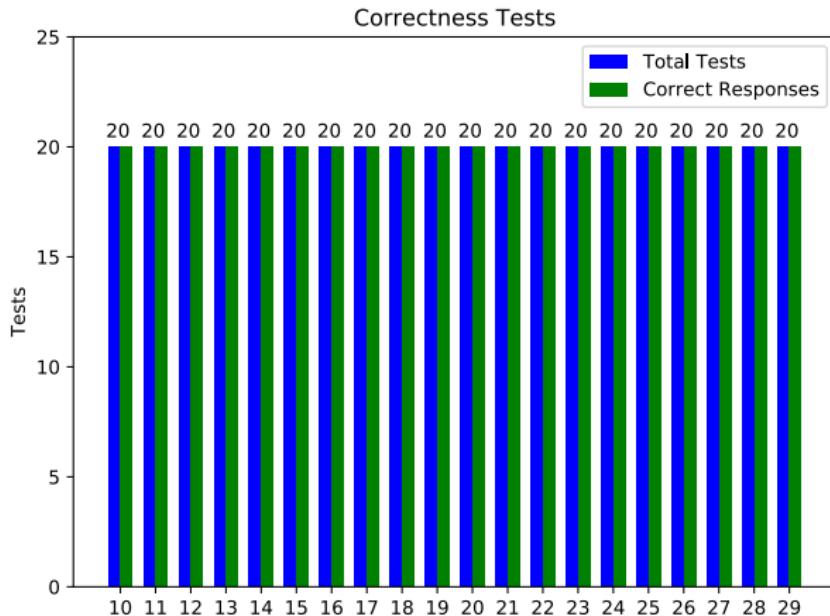
Problema de decisão $t(G) \geq 3$ em grafos bipartidos

Algorithm 5 Decisão $t(G) \geq 3$ em grafos bipartidos

```
1: for all  $u \in V(G)$  do
2:   for all  $v \in N(u)$  do
3:     for all  $s \in N_2(u)$  do
4:       if  $T_0 \cup N_{\geq 3}(u) \cup \{v, s\}$  infecta  $u$  no tempo 3 then
5:         Retorna verdadeiro
6:       end if
7:     end for
8:   end for
9: end for
10: Retorna falso
```

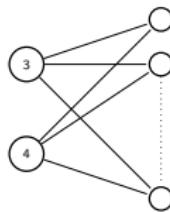
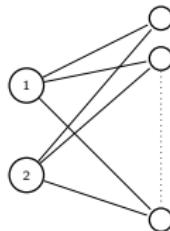
Problema de decisão $t(G) \geq 3$ em grafos bipartidos

- Validação por geração randômica de grafos bipartidos e comparação com teste “força-bruta”.

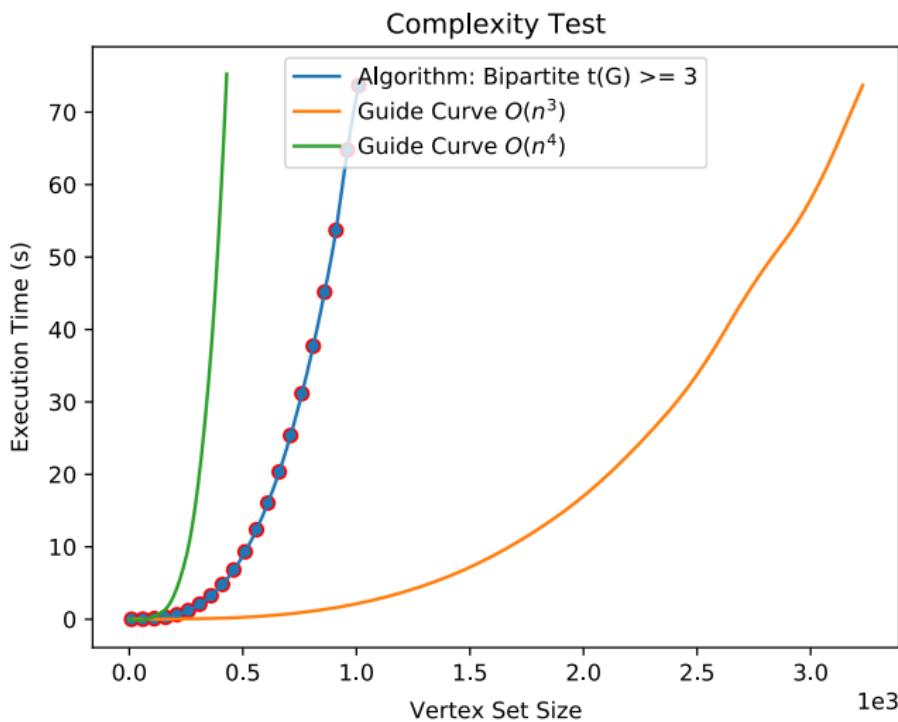


Problema de decisão $t(G) \geq 3$ em grafos bipartidos

- Complexidade esperada de $O(m \cdot n^3)$.
- Validação de complexidade pelo redimensionamento da classe de grafo abaixo.



Problema de decisão $t(G) \geq 3$ em grafos bipartidos



Resultados - Problema de decisão $t(G) \geq 3$ em grafos quaisquer

Processo de Infecção: 2-bootstrap percolation

Entrada: Grafo G qualquer.

Saída: Veredicto de $t(G) \geq 3$.

Definição

Seja \mathcal{T}_0^u uma família de subconjuntos de $V(G)$ tal que um conjunto $T_0 \in \mathcal{T}_0^u$ se, e somente se, para cada articulação v e cada componente conexo $H_{v,i}$ de $G - v$ tal que $u \notin V(H_{v,i})$ e $V(H_{v,i}) \subseteq N(v)$, T_0 contém exatamente um vértice de $H_{v,i}$ e todo vértice de T_0 possui essa propriedade.

Resultados - Problema de decisão $t(G) \geq 3$ em grafos quaisquer

Lema

Seja G um grafo conexo. $t(G) \geq 3$ se, e somente se, existe um vértice u , um conjunto $T_0 \in \mathcal{T}_0^u$ e um conjunto F com, no máximo, 4 vértices tal que $T_0 \cup N_{\geq 3}(u) \cup F$ infecta u no tempo 3.

Lema

Se existe um vértice u , um conjunto $T_0 \in \mathcal{T}_0^u$ e um conjunto de vértices F , com $|F| \leq 4$, tal que $T_0 \cup N_{\geq 3}(u) \cup F$ infecta u no tempo 3, então para qualquer conjunto $T'_0 \in \mathcal{T}_0^u$ existe um conjunto de vértices F' , com $|F'| \leq 4$, tal que $T_0 \cup N_{\geq 3}(u) \cup F$ infecta u no tempo 3.

- Ideia: Busca iterada de u e F , com pré-processamento de T_0 .

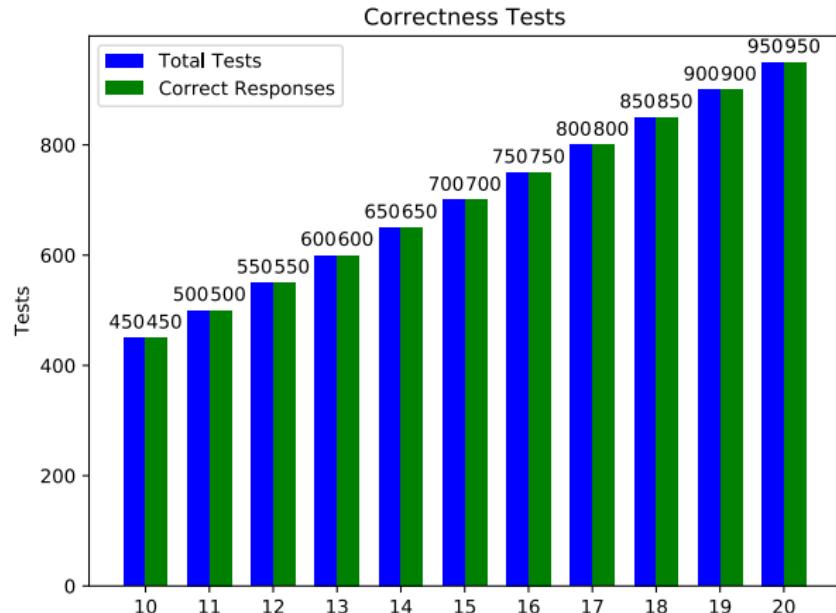
Resultados - Problema de decisão $t(G) \geq 3$ em grafos quaisquer

Algorithm 6 Decisão $t(G) \geq 3$ em grafos quaisquer

```
1: for all  $u \in V(G)$  do
2:   Encontrar um conjunto  $T_0 \in \mathcal{T}_0^u$ 
3:   for all  $F \subseteq V(G)$ , onde  $|F| \leq 4$  do
4:     if  $F \cup N_{\geq 3}(u) \cup T_0$  infecta  $u$  no tempo 3 then
5:       Retorna verdadeiro
6:     end if
7:   end for
8: end for
9: Retorna falso
```

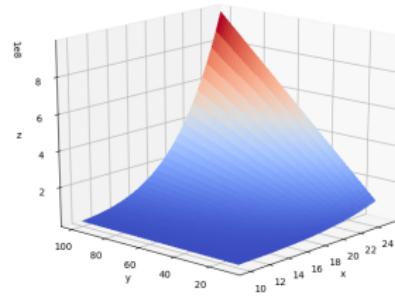
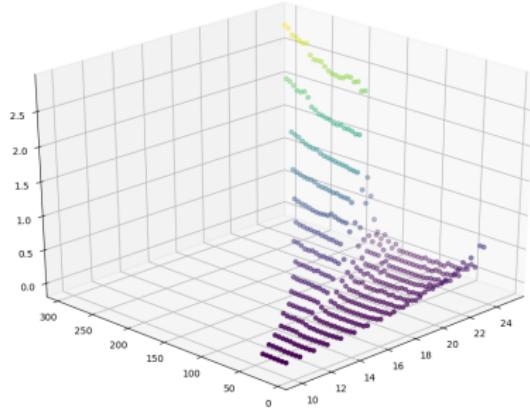
Resultados - Problema de decisão $t(G) \geq 3$ em grafos quaisquer

- Geração de grafos randômicos.
- Para cada grafo, varia-se $m \in [n, 2n]$.
- Validação “força-bruta”.



Resultados - Problema de decisão $t(G) \geq 3$ em grafos quaisquer

- Complexidade esperada de $O(m \cdot n^5)$.
- Geração de grafos randômicos com número de arestas variável.

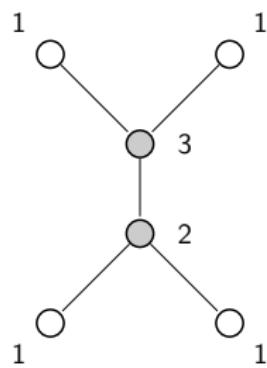


Resultados - Target Set Selection em Árvores

Processo de Infecção: Target Set Selection

Entrada: Grafo G árvore (grafo acíclico) e valores
 $I(v) \in [1, d(v)]$, $\forall v \in V(G)$.

Saída: Target Set de G .



Observação

Como $I(v) \in [1, d(v)]$, é certo que, para folhas, $I(v) = 1$. Assim, pode-se descartar as folhas na busca por um Target Set.

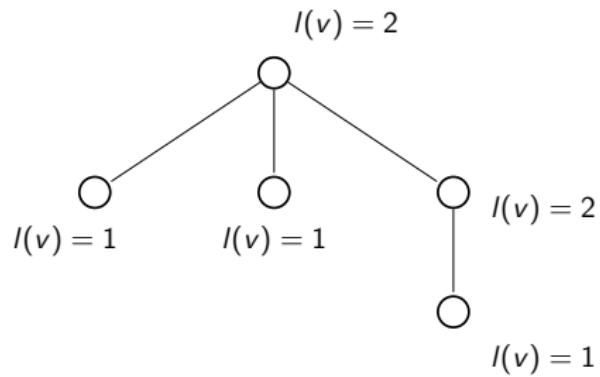
- Analisa-se a árvore enraizada.
- Descarta-se as folhas.
- Atualiza-se o limite de cada vértice, há medida que seus filhos forem escolhidos ou não.
- Analisa-se vértices cujos filhos já foram escolhidos ou descartados. Se $I(v) \geq 2$, insere-se o vértice no Target Set.

TSS em árvores - Algoritmo proposto

ALG-TREE

1. Let $t'(v) = t(v)$, for $v \in V$
2. Let $x(v) = 0$, for each leaf $v \in V$
3. While there is $x(v)$ not defined yet
4. for any vertex u where all $x(\cdot)$'s of its children have been defined
5. let w be u 's parent
6. if $t'(u) \geq 2$
7. let $x(u) = 1$
8. let $t'(w) \leftarrow t'(w) - 1$
9. else
10. let $x(u) = 0$
11. if $t'(u) \leq 0$
12. let $t'(w) \leftarrow t'(w) - 1$
13. Output the target set $\{v \in V \mid x(v) = 1\}$

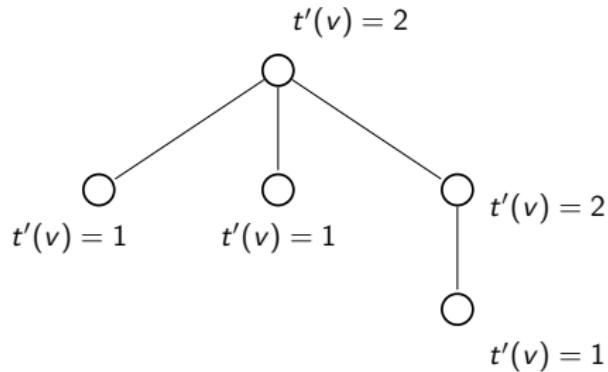
TSS em árvores - Inconsistência no algoritmo



TSS em árvores - Inconsistência no algoritmo

ALG-TREE

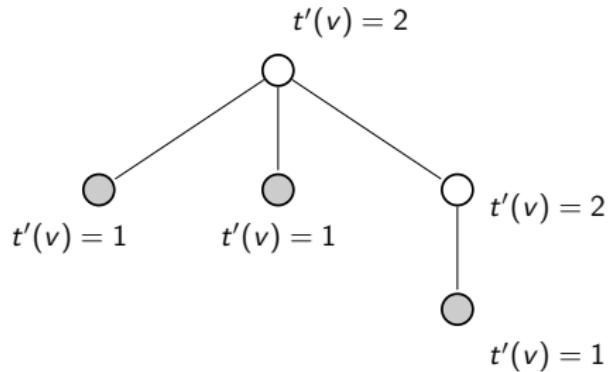
- 1. Let $t'(v) = t(v)$, for $v \in V$
- 2. Let $x(v) = 0$, for each leaf $v \in V$
- 3. While there is $x(v)$ not defined yet
 - 4. for any vertex u where all $x(\cdot)$'s of its children have been defined
 - 5. let w be u 's parent
 - 6. if $t'(u) \geq 2$
 - 7. let $x(u) = 1$
 - 8. let $t'(w) \leftarrow t'(w) - 1$
 - 9. else
 - 10. let $x(u) = 0$
 - 11. if $t'(u) \leq 0$
 - 12. let $t'(w) \leftarrow t'(w) - 1$
 - 13. Output the target set $\{v \in V \mid x(v) = 1\}$



TSS em árvores - Inconsistência no algoritmo

ALG-TREE

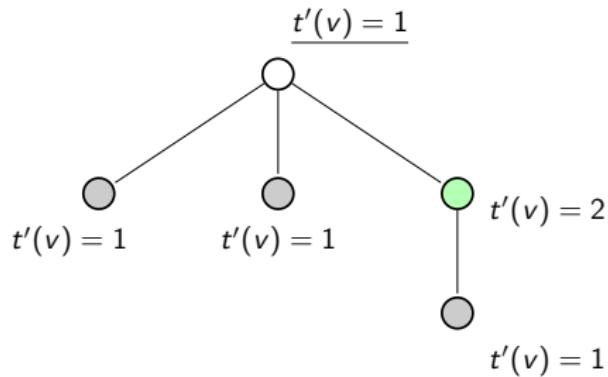
1. Let $t'(v) = t(v)$, for $v \in V$
2. Let $x(v) = 0$, for each leaf $v \in V$
3. While there is $x(v)$ not defined yet
4. for any vertex u where all $x(\cdot)$'s of its children have been defined
5. let w be u 's parent
6. if $t'(u) \geq 2$
7. let $x(u) = 1$
8. let $t'(w) \leftarrow t'(w) - 1$
9. else
10. let $x(u) = 0$
11. if $t'(u) \leq 0$
12. let $t'(w) \leftarrow t'(w) - 1$
13. Output the target set $\{v \in V \mid x(v) = 1\}$



TSS em árvores - Inconsistência no algoritmo

ALG-TREE

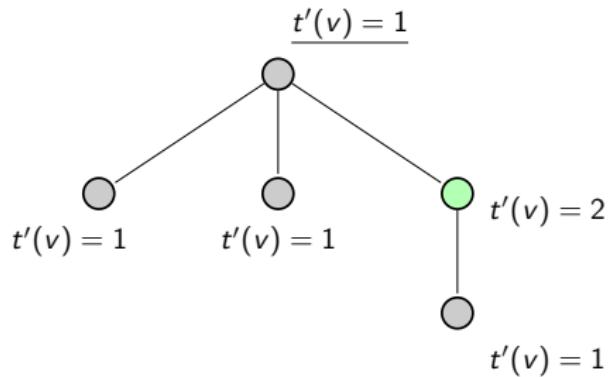
1. Let $t'(v) = t(v)$, for $v \in V$
2. Let $x(v) = 0$, for each leaf $v \in V$
3. While there is $x(v)$ not defined yet
4. for any vertex u where all $x(\cdot)$'s of its children have been defined
5. let w be u 's parent
6. if $t'(u) \geq 2$
7. let $x(u) = 1$
8. let $t'(w) \leftarrow t'(w) - 1$
9. else
10. let $x(u) = 0$
11. if $t'(u) \leq 0$
12. let $t'(w) \leftarrow t'(w) - 1$
13. Output the target set $\{v \in V \mid x(v) = 1\}$



TSS em árvores - Inconsistência no algoritmo

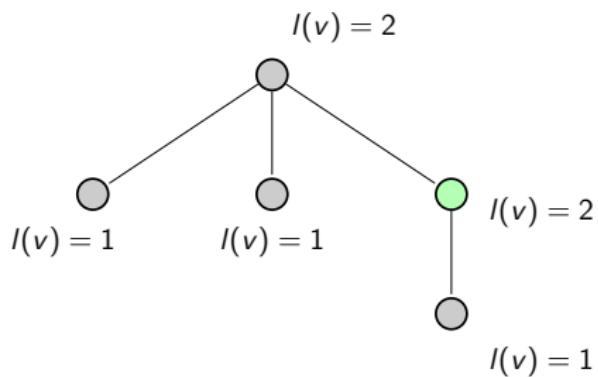
ALG-TREE

1. Let $t'(v) = t(v)$, for $v \in V$
2. Let $x(v) = 0$, for each leaf $v \in V$
3. While there is $x(v)$ not defined yet
4. for any vertex u where all $x(\cdot)$'s of its children have been defined
5. let w be u 's parent
6. if $t'(u) \geq 2$
7. let $x(u) = 1$
8. let $t'(w) \leftarrow t'(w) - 1$
9. else
10. let $x(u) = 0$
11. if $t'(u) \leq 0$
12. let $t'(w) \leftarrow t'(w) - 1$
13. Output the target set $\{v \in V \mid x(v) = 1\}$



TSS em árvores - Inconsistência no algoritmo

- Target Set retornado não percola o grafo

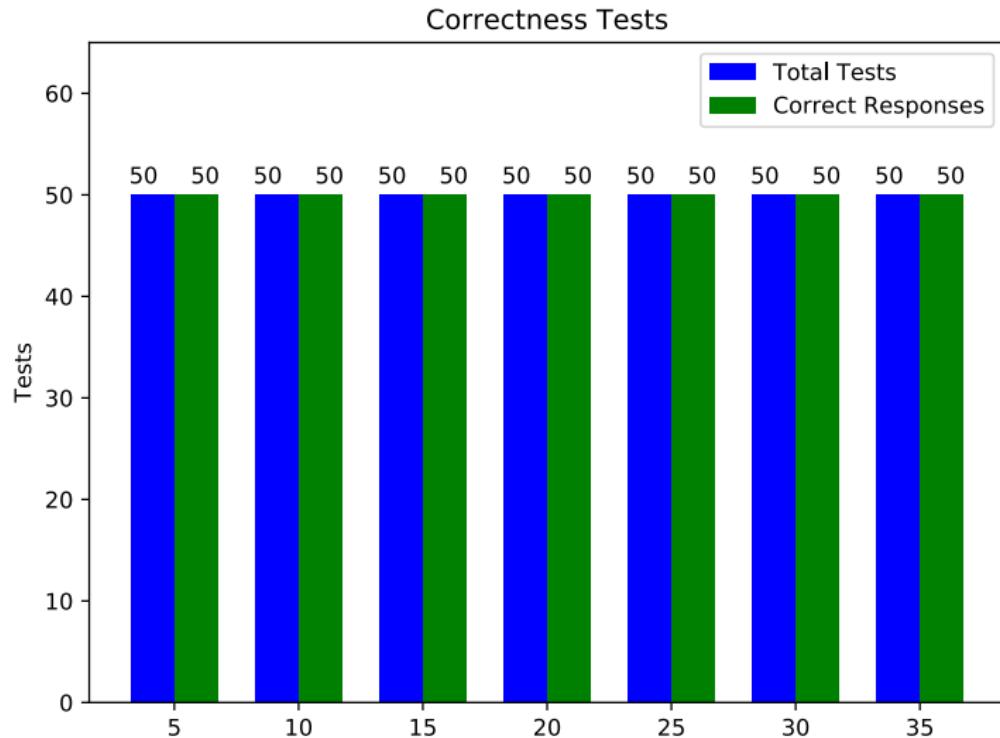


TSS em árvores - Algoritmo Corrigido

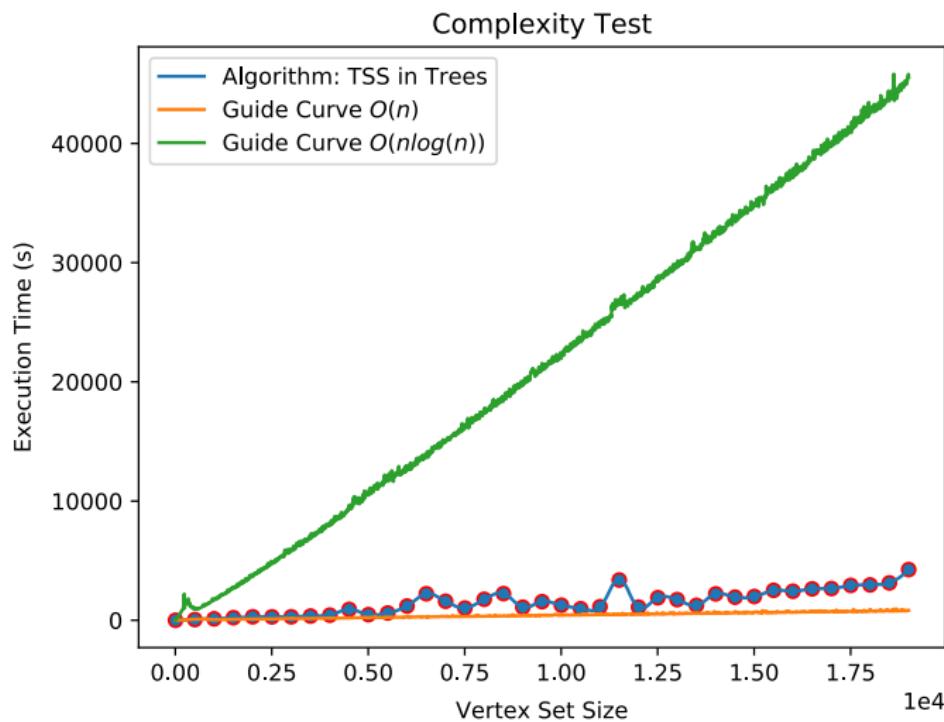
Algorithm 7 TSS em Árvores

```
1:  $t'(v) \leftarrow t(v)$ , para cada  $v \in V(G)$ 
2:  $tss(v) \leftarrow 0$ , para cada  $v$  folha de  $G$ 
3: while existe vértice não determinado do
4:   for all  $v$  cujos filhos foram determinados do
5:     if  $t'(v) \geq 2$  ou ( $v$  é raiz e  $t'(v) \geq 1$ ) then
6:        $tss(v) \leftarrow 1$ 
7:        $t'(p) \leftarrow t'(p) - 1$ , sendo  $p$  pai de  $v$ 
8:     else
9:        $tss(v) \leftarrow 0$ 
10:      if  $t'(v) \leq 0$  then
11:         $t'(p) \leftarrow t'(p) - 1$ , sendo  $p$  pai de  $v$ 
12:      end if
13:    end if
14:  end for
15: end while
16: Retorna  $tss$ 
```

TSS em árvore - Resultados dos Testes de Implementação



TSS em árvore - Resultados dos Testes de Complexidade



Tempo Máximo de Infecção em Árvores - Visão Geral

Processo de Infecção: TSS “expandido”

Entrada: Grafo G árvore (grafo acíclico) e valores
 $I(v) \geq 1, \forall v \in V(G)$.

Saída: $t(G)$.

Algoritmo de $t(G)$ em árvores - Ideia de Corretude

- Conjunto de vértices inicialmente infectados \rightarrow Hull Set: S .
- Conjunto final de vértices infectados \rightarrow Convex Hull: $H(S)$.

Lema

Sendo G uma árvore, $\forall v \in V(G), \exists S : H(S) = S$ e $H(S \cup \{v\}) = V(G)$.

Demonstração.

Por indução forte no número de vértices n de G .

- **Caso Base:** $n = 2$.



Sem perda de generalidade, faça $v = v_1$.

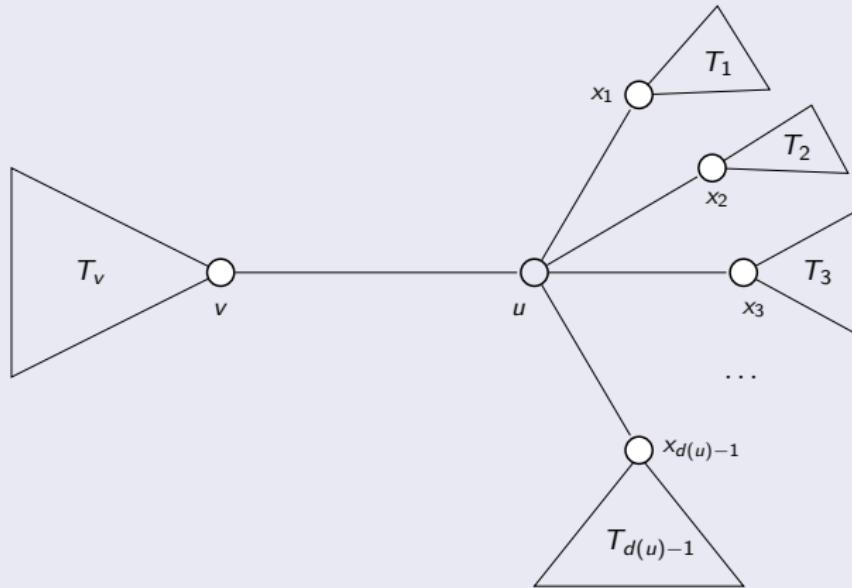
Fazendo $S = \emptyset$, tem-se:

- $H(S) = S$, por vacuidade.
- $H(S \cup \{v\}) = V(G)$, pois $I(v_2) \in [1, d(v_2)] \Rightarrow I(v_2) = 1$.

Algoritmo de $t(G)$ em árvores - Ideia de Corretude

Demonstração (Cont.)

- **Hipótese de indução:** O lema vale para $n \leq k - 1$.
- **Passo de indução:** $n = k$.



Algoritmo de $t(G)$ em árvores - Ideia de Corretude

- Estratégia recursiva de $t_{MAX}(u), \forall v \in V(u)$. Naturalmente, $t(G) = \max_{u \in V(G)} t(u)$.
- $s(u, v)$: maior tempo de infecção do vértice u na subárvore $G - \{v\}, v \in N(u)$.
- Tem-se que:

$$t(u) = \begin{cases} 0, & \text{se } d(u) < l(u) \\ 1 + \text{o maior valor } s(x, u) : x \in N(u), & \text{se } d(u) \geq l(u) \end{cases}$$

- De maneira similar:

$$s(u, v) = \begin{cases} 0, & \text{se } d(u) \leq l(u) \\ 1 + \text{o maior valor } s(x, u) : x \in N(u) \setminus \{v\}, & \text{se } d(u) > l(u) \end{cases}$$

Algoritmo de $t(G)$ em árvores - Pseudocódigo

Algorithm 8 Tempo Máximo de Infecção em Árvores

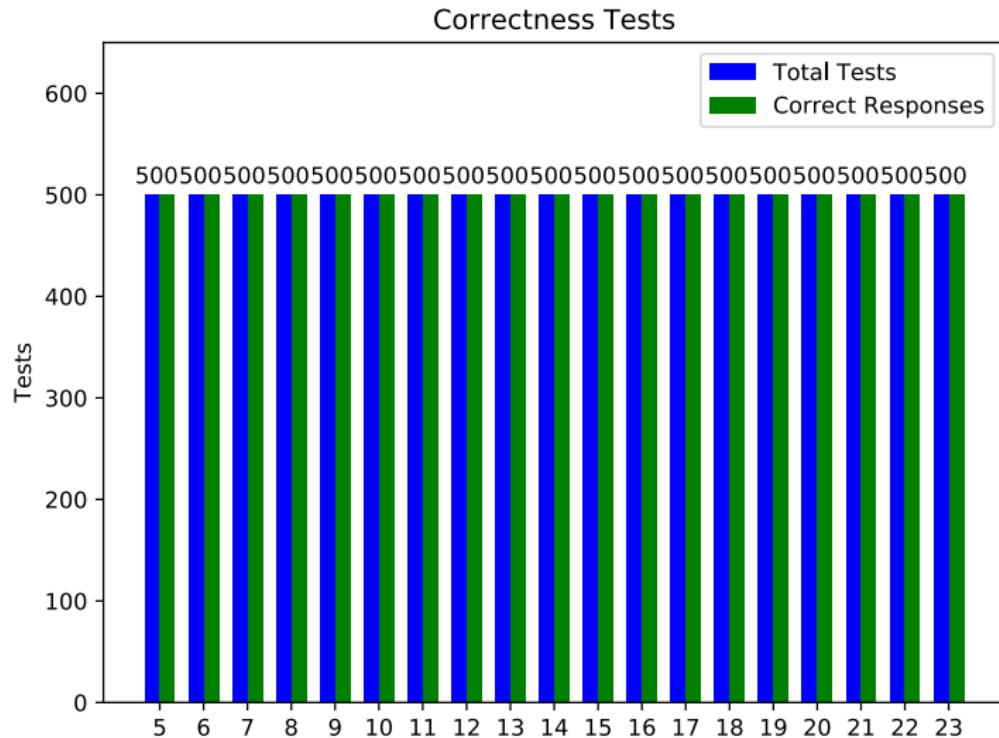
```
1:  $I' \leftarrow I$ 
2: Preprocessamento( $G, I'$ )
3: for all  $u$  em  $G$  do
4:   if  $t(u)$  já foi determinado then
5:      $s(u, v) \leftarrow t(u)$ , para todo  $v \in N(u)$ 
6:   else
7:     if  $d(u) \leq I(u)$  then
8:        $s(u, v) \leftarrow 0$ , para todo  $v \in N(u)$ 
9:     else
10:       $s(u, v) \leftarrow 1 + \max(s(x, u))$ , para todo  $x \in N(u) \setminus v$ 
11:    end if
12:  end if
13: end for
14:  $\forall u \in G$  com  $t(u)$  não det.,  $t(u) \leftarrow \max(s(x, u))$ ,  $\forall x \in N(u)$ 
15:  $t(G) \leftarrow \max(t(v))$ , para todo  $v$  em  $G$ 
16: Retorna  $t(G)$ 
```

Algoritmo de $t(G)$ em árvores - Pseudocódigo

Algorithm 9 Tempo Máximo de Infecção em Árvores

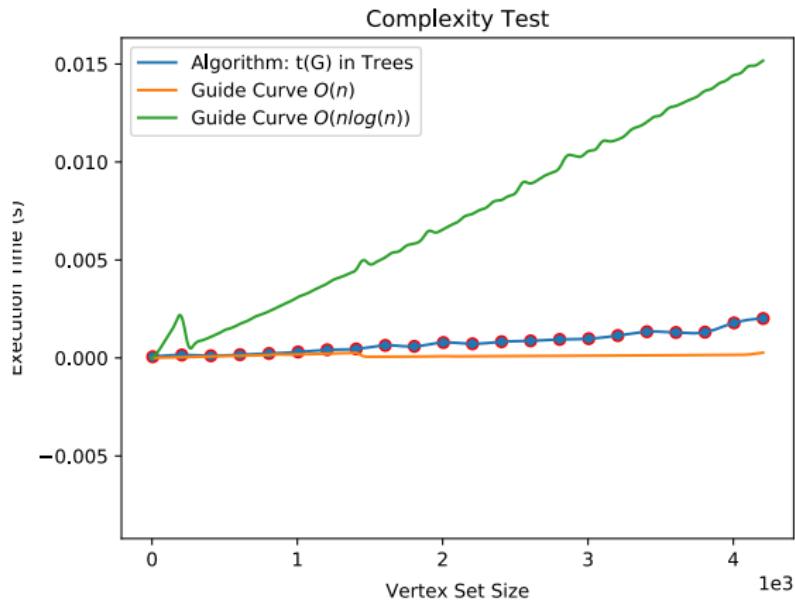
```
1:  $R \leftarrow$  fila vazia
2: for all  $v$  em  $G$  com  $I(v) > d(v)$  do
3:    $t(v) \leftarrow 0$ 
4:   for all  $u$  vizinho de  $v$  do
5:      $I'(u) \leftarrow I'(u) - 1$ 
6:     if  $I'(u) = 0$  then
7:       insere  $u$  em  $R$ 
8:        $t(u) \leftarrow \max(t(x)) + 1$ , onde  $x \in N(u)$ 
9:     end if
10:    end for
11:  end for
12:  while  $R$  não vazia do
13:     $x \leftarrow$  próximo de  $R$ 
14:    for all  $u$  vizinho de  $x$  do
15:       $I'(u) \leftarrow I'(u) - 1$ 
16:      if  $I'(u) = 0$  then
17:        insere  $u$  em  $R$ 
18:         $t(u) \leftarrow \max(t(y)) + 1$ , onde  $y \in N(u)$ 
19:      end if
20:    end for
21:  end while
```

Algoritmo de $t(G)$ em árvores - Testes da Implementação



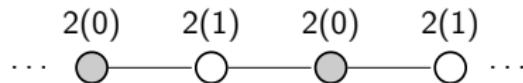
Algoritmo de $t(G)$ em árvores - Testes de Complexidade

- Tempo esperado: $O(n)$.

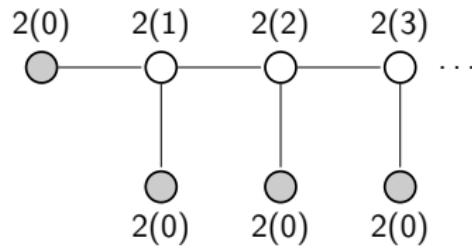


Algoritmo de $t(G)$ em árvores - Outra Abordagem

- **Vértice Crítico:** Vértice em que $I(v) = d(v)$.



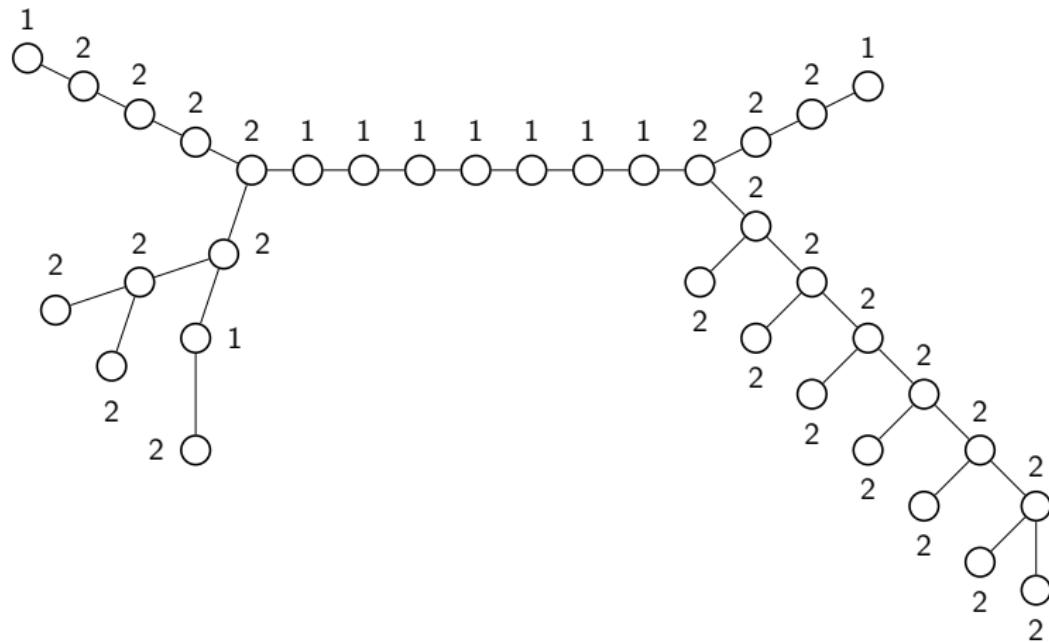
- A “propagação da infecção” ocorre através de vértices não-críticos.



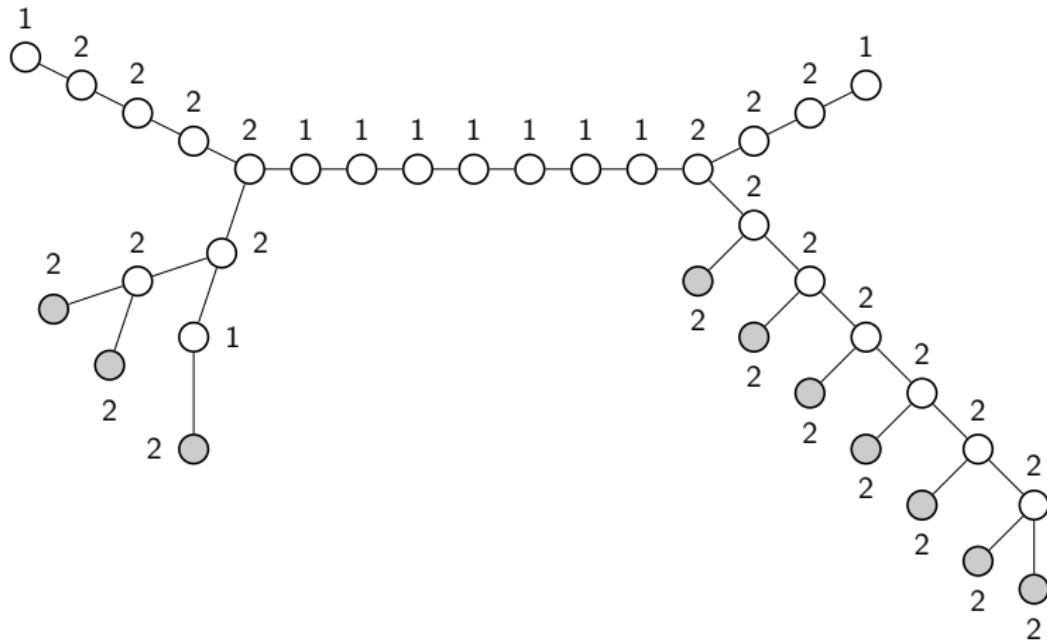
Algoritmo*

- Pré-processamento.
- Busca por caminho não-crítico máximo (Através de Busca em Largura Modificada).
- **Resultado:** maior caminho não-crítico + maior tempo de vizinho da extreminadade no pré-processamento +1 (vizinho não crítico).

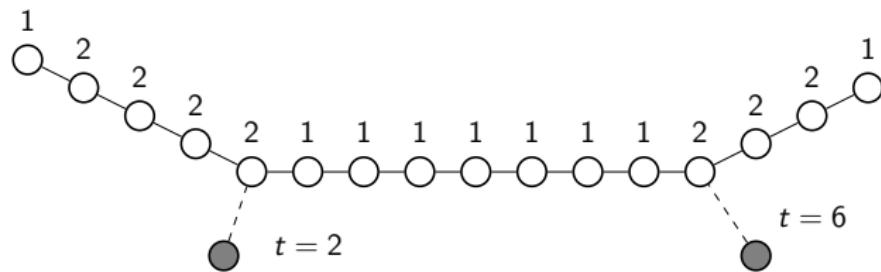
Algoritmo de $t(G)$ em árvores - Outra Abordagem (Exemplo)



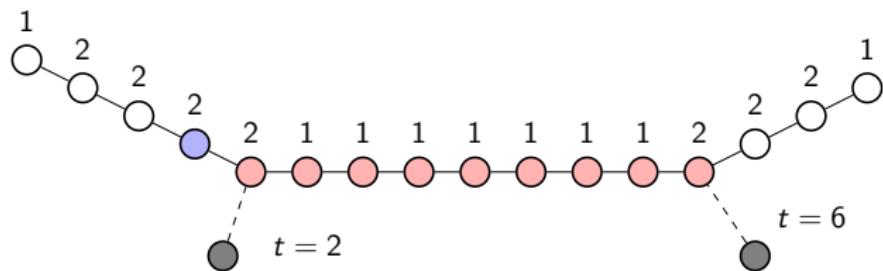
Algoritmo de $t(G)$ em árvores - Outra Abordagem (Exemplo)



Algoritmo de $t(G)$ em árvores - Outra Abordagem (Exemplo)



Algoritmo de $t(G)$ em árvores - Outra Abordagem (Exemplo)



$$t(G) = 9 + 6 + 1 = 16$$

Conteúdo

1 Introdução

- Visão Simplificada do Problema
- Histórico do Problema

2 Objetivos e Metodologia

- Objetivos
- Subproblemas estudados
- Metodologia

3 Resultados

4 Conclusão

Conclusão

- Todos os objetivos atingidos: Implementações e Verificações.
- Contribuições:
 - Correção do algoritmo de TSS em árvores.
 - Algoritmo de $t(G)$ em árvores.
- Possibilidades de Melhoria/Expansão:
 - Métricas de Complexidade.
 - Aumentar volume e padronizar testes.
 - Mais Algoritmos.

Nota sobre reproduzibilidade

- Todos os códigos e documentos deste trabalho estão disponíveis em repositório público, incluindo:
 - Implementações dos Algoritmos.
 - Scripts de geração de gráficos.
 - Rotinas de Teste.
 - Monografia
 - Esta apresentação.
- Link para repositório:
<https://github.com/lucaskeiler/AlgoritmosTCC>.
- Licença MIT.

Referências I



Thiago Marcilon and Rudini Sampaio.

The maximum infection time of the p3 convexity in graphs with bounded maximum degree.

Discrete Applied Mathematics, 251:245 – 257, 2018.



Thiago Marcilon and Rudini Sampaio.

The maximum time of 2-neighbor bootstrap percolation: Complexity results.

Theoretical Computer Science, 708:1 – 17, 2018.



Ning. Chen.

On the approximability of influence in social networks.

SIAM Journal on Discrete Mathematics, 23(3):1400–1415, 2009.

Referências II



Martin Niss.

History of the lenz-ising model 1920–1950: From ferromagnetic to cooperative phenomena.

Archive for History of Exact Sciences, 59(3):267–318, Mar 2005.



Joel Schiff.

Cellular automata : a discrete view of the world.

Wiley-Interscience, Hoboken, N.J, 2008.



P. Grassberger.

On the critical behavior of the general epidemic process and dynamical percolation.

Mathematical Biosciences, 63(2):157 – 172, 1983.



J Chalupa, P L Leath, and G R Reich.

Bootstrap percolation on a bethe lattice.

Journal of Physics C: Solid State Physics, 12(1):L31–L35, jan 1979.

Referências III

-  S. R. Broadbent and J. M. Hammersley.
Percolation processes: I. crystals and mazes.
Mathematical Proceedings of the Cambridge Philosophical Society,
53(3):629–641, 1957.
-  Scott Kirkpatrick, Winfried W Wilcke, Robert B Garner, and Harald Huels.
Percolation in dense storage arrays.
Physica A: Statistical Mechanics and its Applications, 314(1):220 – 229, 2002.
Horizons in Complex Systems.
-  David Peleg.
Local majorities, coalitions and monopolies in graphs: a review.
Theoretical Computer Science, 282(2):231 – 257, 2002.
FUN with Algorithms.

Referências IV



C. Griffin and R. Brooks.

A note on the spread of worms in scale-free networks.

IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 36(1):198–202, Feb 2006.

Estudo Algorítmico de Problemas de Infecção em Grafos

Lucas Rodrigues Keiler - Engenharia de Computação - UFC



Banca:

Prof. Dr. Rudini Menezes Sampaio (Orientador)
Prof. Dr. Pablo Mayckon Silva Farias
Prof. Dr. Carlos Estêvão Rolim Fernandes
Prof. Dr. Jorge Herbert Soares de Lira

03 de dezembro de 2019