Proteção de Software por Marcas d'água Baseadas em Grafos

Lucila Bento[1,3], Davidson Boccardo[3], Raphael Machado[3], Vinícius Pereira de Sá[1], Jayme Szwarcfiter[1,2,3]

Instituto de Matemática – Universidade Federal do Rio de Janeiro[1]
COPPE – Universidade Federal do Rio de Janeiro[2]
INMETRO – Instituto Nacional de Metrologia, Qualidade e Tecnologia[3]

Roteiro

1. Motivação
2. Conceitos preliminares
3. Marca d'água de Chroni e Nikolopoulos
4. Recuperação de ataques
5. Resultados Computacionais

Marca d'água

- É uma marca distintiva impressa, usada para:

Marca d'água

- É uma marca distintiva impressa, usada para:
  - Identificação de propriedade

Marca d'água

- É uma marca distintiva impressa, usada para:
  - Identificação de propriedade
  - Identificação de autenticidade

Marca d'água

- É uma marca distintiva impressa, usada para:
  - Identificação de propriedade
  - Identificação de autenticidade

- Também é utilizada em objetos digitais

Marca d'água

- É uma marca distintiva impressa, usada para:
  - Identificação de propriedade
  - Identificação de autenticidade

- Também é utilizada em objetos digitais

Pode ser usada em software? Como?

Marca d'água em software

Marca d'água em software

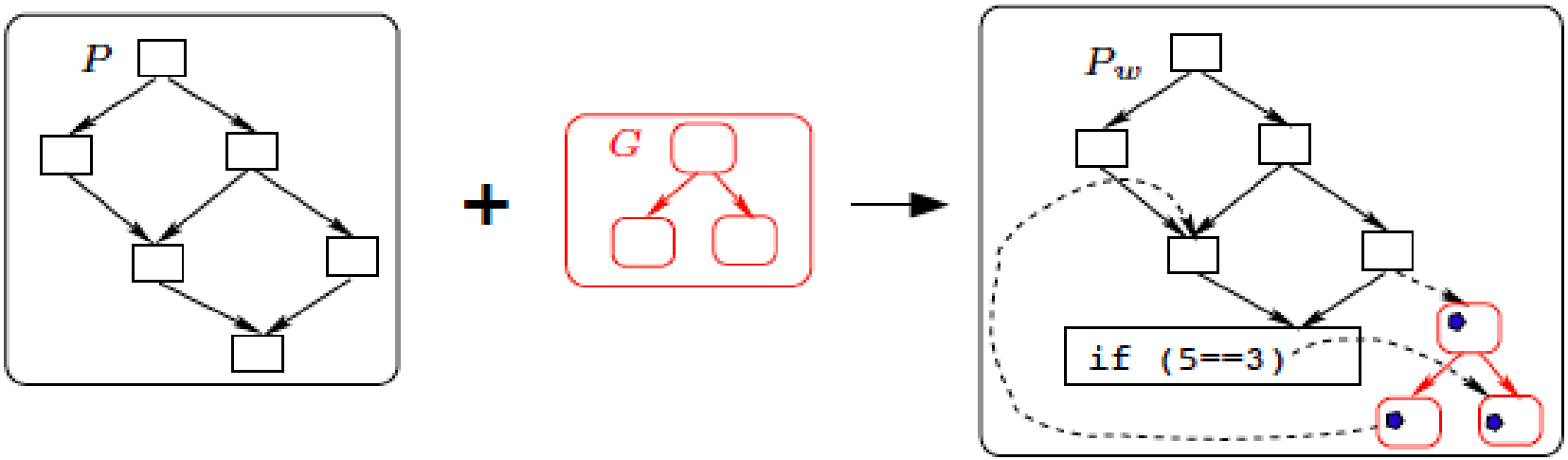
- Todo programa está associado a um grafo direcionado (grafo de fluxo de controle)

Marca d'água em software

- Todo programa está associado a um grafo direcionado (grafo de fluxo de controle)
- A ideia é inserir a marca d'água no grafo de fluxo

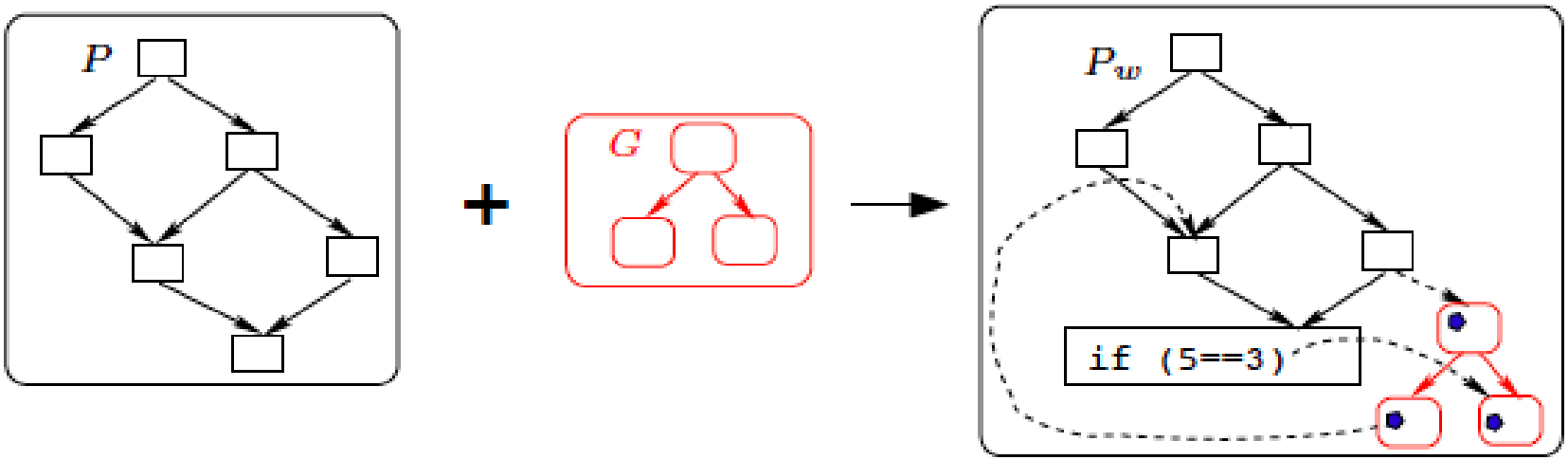
Marca d'água em software

- Todo programa está associado a um grafo direcionado (grafo de fluxo de controle)
- A ideia é inserir a marca d'água no grafo de fluxo



Marca d'água em software

- Todo programa está associado a um grafo direcionado (grafo de fluxo de controle)
- A ideia é inserir a marca d'água no grafo de fluxo



Propriedades da marca d'água aplicada a software:

- Difícil de identificar
- Não alterar a funcionalidade do software

Marcas d'água baseadas em grafos

O esquema é composto por:

Marcas d'água baseadas em grafos

O esquema é composto por:

- um algoritmo de codificação;
- um algoritmo de decodificação;
- uma função *embarcadora*;
- uma função *extratora*.

Ataques

Marcas d'água para software estão sujeitas a:

Ataques

Marcas d'água para software estão sujeitas a:

- ataques de adição;
- ataques de subtração;
- ataques de distorção;

Ataques

Marcas d'água para software estão sujeitas a:

- ataques de adição;
- ataques de subtração;
- ataques de distorção;

Criptografia e ofuscação podem ajudar a contornar os ataques de adição e subtração

Ataques

Marcas d'água para software estão sujeitas a:

- ataques de adição;
- ataques de subtração;
- ataques de distorção;

Criptografia e ofuscação podem ajudar a contornar os ataques de adição e subtração

Estamos interessados em ataques de distorção

Geração da marca d'água

Chave $\omega = 26$

Geração da marca d'água

Chave $\omega = 26$

$$B = 11010$$

Geração da marca d'água

Chave $\omega = 26$

$$\overline{B} = 00101$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010100$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010\,10$$

$$Z_0 = (6, 7, 9, 11)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010\,10$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010100$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

$$P_s = (6, 7, 9, 11, 10, 1, 2, 8, 3, 5, 4)$$

Geração da marca d'água

Chave $\omega = 26$

$$B^* = 111110010100$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

$$P_s = (6, 7, 9, 11, 10, 1, 2, 8, 3, 5, 4)$$

Construindo o Grafo G:

Geração da marca d'água

Chave $\omega = 26$

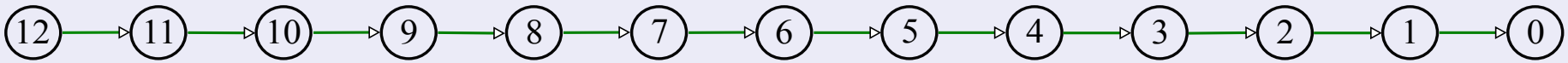
$$B^* = 111110010100$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

$$P_s = (6, 7, 9, 11, 10, 1, 2, 8, 3, 5, 4)$$

Construindo o Grafo G:



Geração da marca d'água

Chave $\omega = 26$

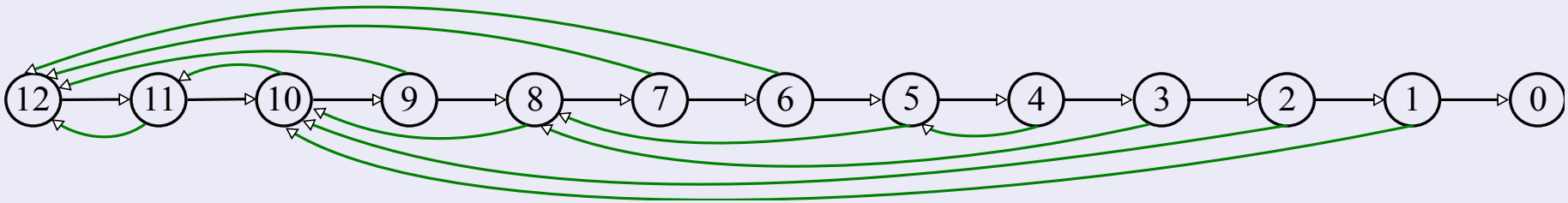
$$B^* = 111111001010$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

$$P_s = (6, 7, 9, 11, 10, 1, 2, 8, 3, 5, 4)$$

Construindo o Grafo G:



Geração da marca d'água

Chave $\omega = 26$

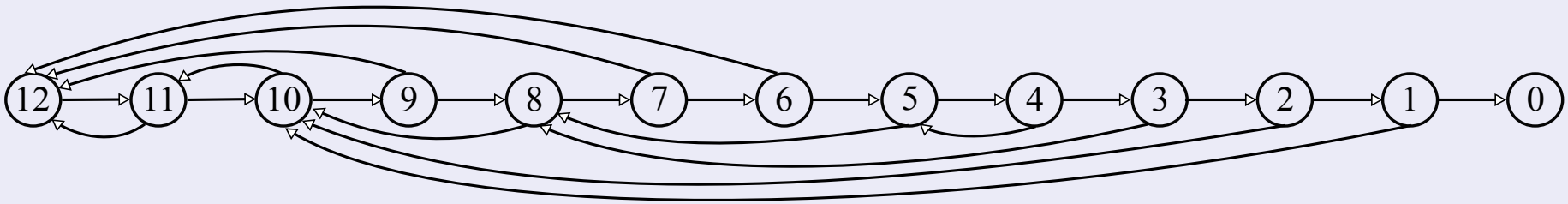
$$B^* = 111110010110$$

$$Z_0 = (6, 7, 9, 11) \text{ e } Z_1 = (1, 2, 3, 4, 5, 8, 10)$$

$$P_b = (6, 7, 9, 11, 10, 8, 5, 4, 3, 2, 1)$$

$$P_s = (6, 7, 9, 11, 10, 1, 2, 8, 3, 5, 4)$$

Construindo o Grafo G:



Caracterização

Caracterização da família de grafos gerados pelo codec de Chroni e Nikolopoulos

Caracterização

Caracterização da família de grafos gerados pelo codec de Chroni e Nikolopoulos

- Grafos de permutação redutíveis auto-rotuláveis
- Árvores representativas
  - Tipo 1
  - Tipo 2
- Percurso em pré-ordem sem raiz
- Permutação auto-inversível canônica
- Grafos de permutação redutíveis canônicos

Algoritmos

Algoritmos

- A caracterização permite o reconhecimento de um grafo da classe em tempo linear

Algoritmos

- A caracterização permite o reconhecimento de um grafo da classe em tempo linear
- Possibilita algoritmo mais rápido de decodificação

Algoritmos

- A caracterização permite o reconhecimento de um grafo da classe em tempo linear
- Possibilita algoritmo mais rápido de decodificação
- Algoritmo linear para recuperação de marca d'água que sofreu a remoção de k arestas, $k \leq 2$

Algoritmos

- A caracterização permite o reconhecimento de um grafo da classe em tempo linear
- Possibilita algoritmo mais rápido de decodificação
- Algoritmo linear para recuperação de marca d'água que sofreu a remoção de k arestas, $k \leq 2$
- Algoritmos polinomiais para recuperação de marca d'água que sofreu a remoção de k arestas

Recuperação de k arestas

Algoritmo 1

Entrada: estrutura da marca d'água danificada G'

Saída: lista das marcas d'água candidatas

Passo 1 cria uma lista L vazia

Passo 2 determina o número k de arestas removidas

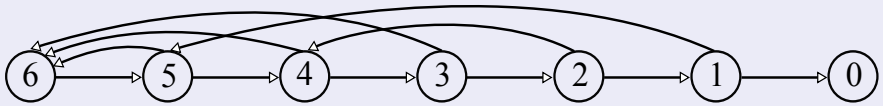
Passo 3 para cada conjunto K de k não-arestas de G'

Passo 3.1 adiciona K a G'

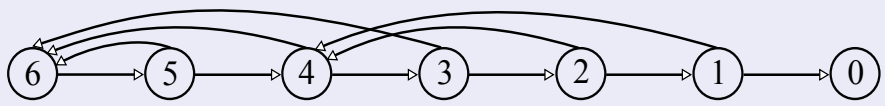
Passo 3.2 verifica se a marca d'água obtida é válida

Passo 3.3 se a marca d'água for válida, adiciona a L

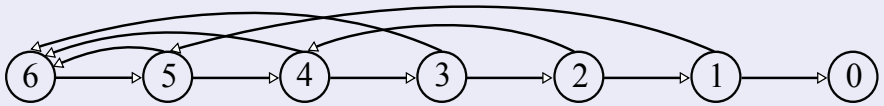
$\omega = 2$ e $B = 10$

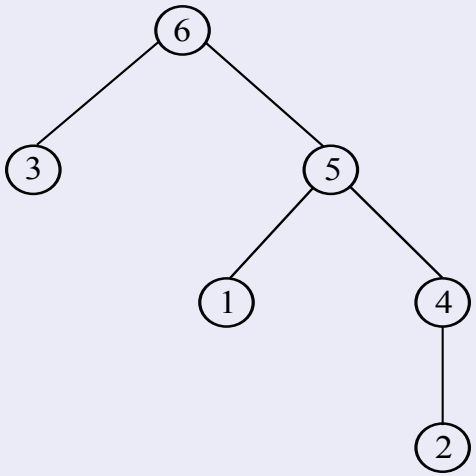


$\omega = 3$ e $B = 11$

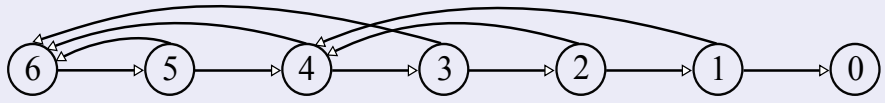


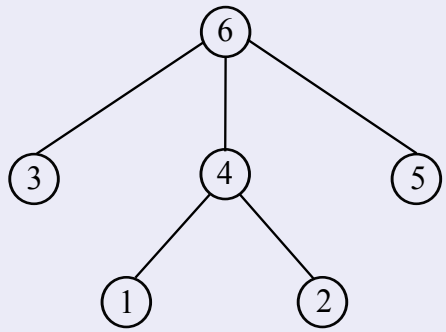
$\omega = 2$ e $B = 10$



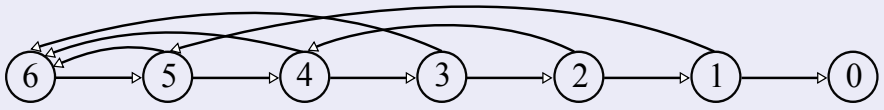


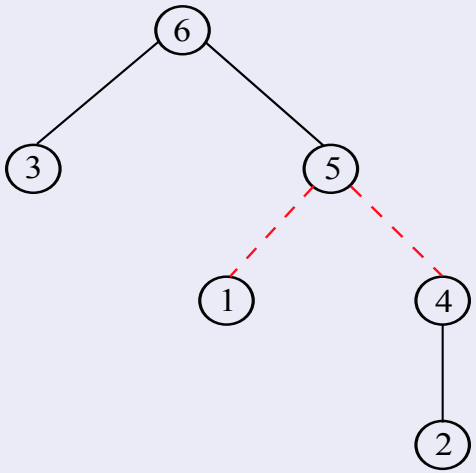
$\omega = 3$ e $B = 11$



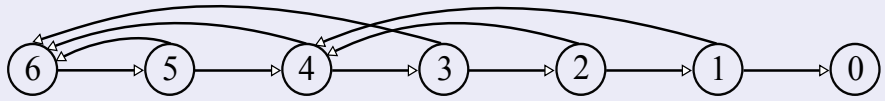


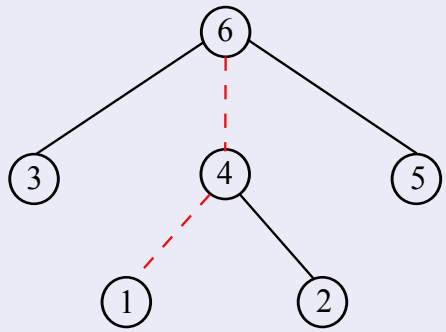
$\omega = 2$ e $B = 10$



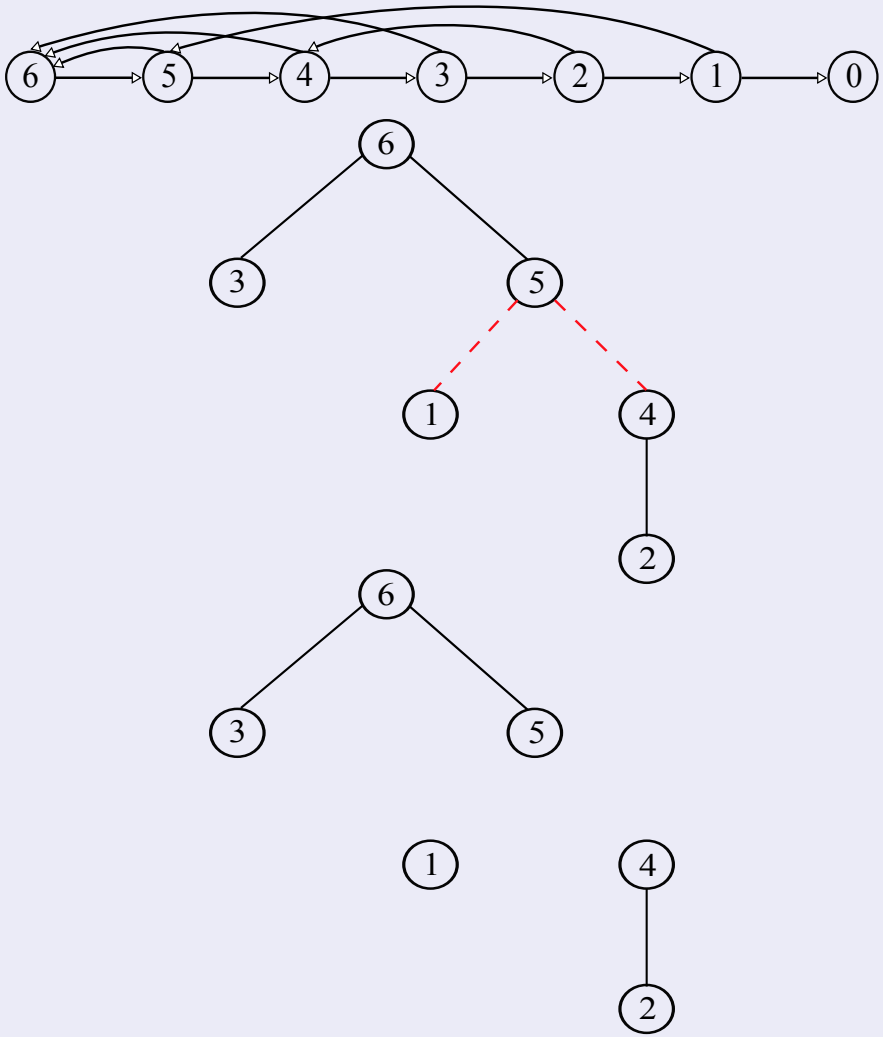


$\omega = 3$ e $B = 11$

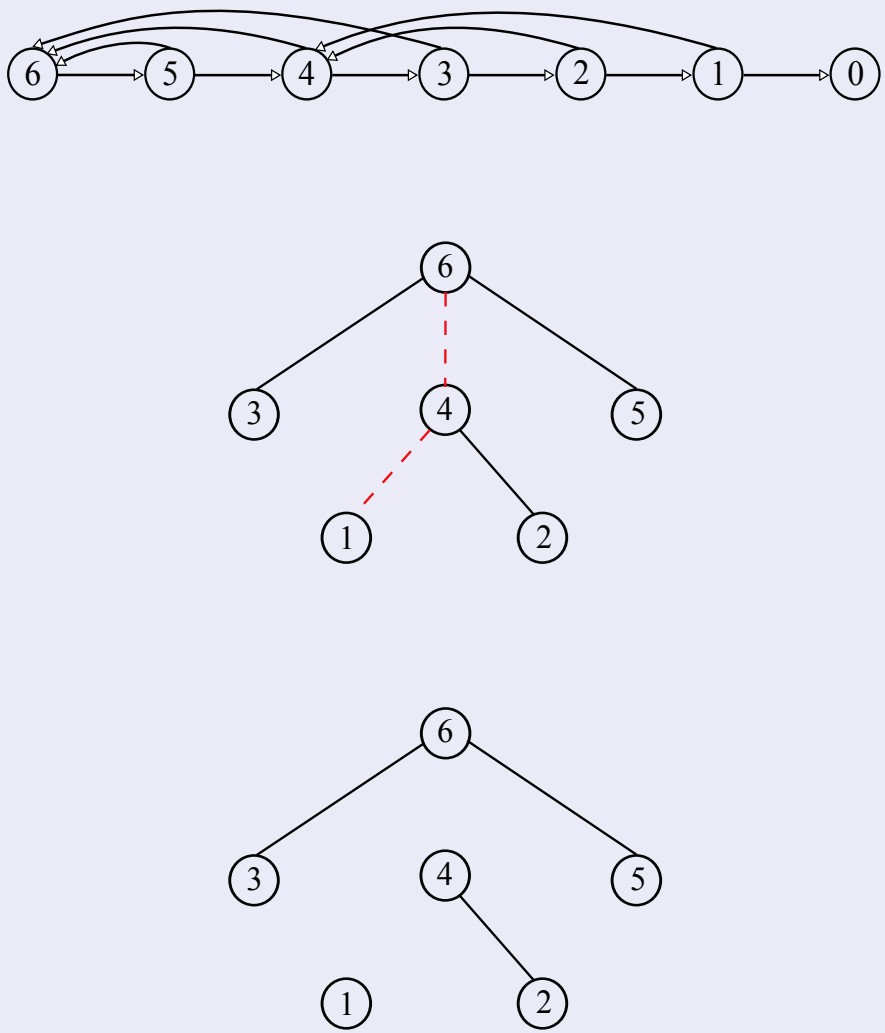




$\omega = 2$ e $B = 10$



$\omega = 3$ e $B = 11$



Recuperação de k arestas

Algoritmo 1

Entrada: estrutura da marca d'água danificada G'

Saída: lista das marcas d'água candidatas

Passo 1 cria uma lista L vazia

Passo 2 determina o número k de arestas removidas

Passo 3 para cada conjunto K de k não-arestas de G'

Passo 3.1 adiciona K a G'

Passo 3.2 verifica se a marca d'água obtida é válida

Passo 3.3 se a marca d'água for válida, adiciona a L

Complexidade: Há $\binom{(2n+3)(2n+2)/2-(4n+3-k)}{k} = O(n^{2k})$ subconjuntos de não arestas de G'. Logo, a complexidade é $O(n^{2k}) \cdot O(n) = O(n^{2k+1})$.

Recuperação de k arestas

Algoritmo 1 — aprimorado

Entrada: estrutura da marca d'água danificada G'

Saída: lista das marcas d'água candidatas

Passo 1 cria um lista L vazia
Passo 2 determina o número k de arestas removidas de G
Passo 3 para cada conjunto de k origens possíveis
Passo 3.1 escolhemos k origens
Passo 3.2 escolhemos o destino de cada uma das k arestas
Passo 3.3 verificamos se a marca d'água obtida é válida
Passo 3.4 adicionamos a marca d'água a L caso seja válida

Recuperação de k arestas

Algoritmo 1 — aprimorado

Entrada: estrutura da marca d'água danificada G'

Saída: lista das marcas d'água candidatas

Passo 1 cria um lista L vazia
Passo 2 determina o número k de arestas removidas de G
Passo 3 para cada conjunto de k origens possíveis
Passo 3.1 escolhemos k origens
Passo 3.2 escolhemos o destino de cada uma das k arestas
Passo 3.3 verificamos se a marca d'água obtida é válida
Passo 3.4 adicionamos a marca d'água a L caso seja válida

Complexidade: $|M^*| = 2.(2n+3) - (4n+3-k) = k+3.$

Recuperação de k arestas

Algoritmo 1 — aprimorado

Entrada: estrutura da marca d'água danificada G'

Saída: lista das marcas d'água candidatas

Passo 1 cria um lista L vazia
Passo 2 determina o número k de arestas removidas de G
Passo 3 para cada conjunto de k origens possíveis
Passo 3.1 escolhemos k origens
Passo 3.2 escolhemos o destino de cada uma das k arestas
Passo 3.3 verificamos se a marca d'água obtida é válida
Passo 3.4 adicionamos a marca d'água a L caso seja válida

Complexidade: $|M^*| = 2.(2n+3) - (4n+3-k) = k+3$. Cada origem é escolhida de $\binom{|M^*|}{k} = O(k^3)$ maneiras e tem $O(n)$ destinos possíveis.

Recuperação de k arestas

Algoritmo 1 — aprimorado

Entrada: estrutura da marca d'água danificada G'

Saída: lista das marcas d'água candidatas

Passo 1 cria um lista L vazia
Passo 2 determina o número k de arestas removidas de G
Passo 3 para cada conjunto de k origens possíveis
Passo 3.1 escolhemos k origens
Passo 3.2 escolhemos o destino de cada uma das k arestas
Passo 3.3 verificamos se a marca d'água obtida é válida
Passo 3.4 adicionamos a marca d'água a L caso seja válida

Complexidade: $|M^*| = 2.(2n+3) - (4n+3-k) = k+3$. Cada origem é escolhida de $\binom{|M^*|}{k} = O(k^3)$ maneiras e tem $O(n)$ destinos possíveis. Logo, a complexidade é $O(k^3) \cdot O(n^k) \cdot O(n) = O(k^3 \cdot n^{k+1})$.

Recuperação de k arestas

Algoritmo 2

Entrada: $G' = (V, E')$ e os rótulos de $v \in V$

Saída: lista das marcas d'água candidatas

Passo 1 cria um lista L vazia

Passo 2 cria uma lista M com a raiz de cada componente conexa diferente de $2n + 2$

Passo 3 para cada $R = \{e_1, ..., e_k\}$, com $e_i = (v_i, w) \notin E'$, $v_i \in M$, $v_i < w \in V (i = 1, ..., k)$

Passo 3.1 verifica se $G(V, E' \cup R)$ é uma marca d'água e adiciona a L

Recuperação de k arestas

Algoritmo 2

Entrada: $G' = (V, E')$ e os rótulos de $v \in V$

Saída: lista das marcas d'água candidatas

Passo 1 cria um lista L vazia

Passo 2 cria uma lista M com a raiz de cada componente conexa diferente de $2n + 2$

Passo 3 para cada $R = \{e_1, ..., e_k\}$, com $e_i = (v_i, w) \notin E'$, $v_i \in M$, $v_i < w \in V (i = 1, ..., k)$

Passo 3.1 verifica se $G(V, E' \cup R)$ é uma marca d'água e adiciona a L

Complexidade: Cada uma das k origens tem $O(n)$ destinos possíveis.

Recuperação de k arestas

Algoritmo 2

Entrada: $G' = (V, E')$ e os rótulos de $v \in V$

Saída: lista das marcas d'água candidatas

Passo 1 cria um lista L vazia

Passo 2 cria uma lista M com a raiz de cada componente conexa diferente de $2n + 2$

Passo 3 para cada $R = \{e_1, ..., e_k\}$, com $e_i = (v_i, w) \notin E'$, $v_i \in M$, $v_i < w \in V (i = 1, ..., k)$

Passo 3.1 verifica se $G(V, E' \cup R)$ é uma marca d'água e adiciona a L

Complexidade: Cada uma das k origens tem $O(n)$ destinos possíveis. Logo, temos $O(n^k) \cdot O(n) = O(n^{k+1})$.

Recuperação do caminho hamiltoniano

- O algoritmo de menor complexidade utiliza os rótulos

Mas quão razoável é assumir que os rótulos estão disponíveis?

- O caminho hamiltoniano está intacto
- Os rótulos estão explicitados nos blocos do código associados aos vértices do grafo
- É possível recuperar o caminho hamiltoniano

Recuperação do caminho hamiltoniano

- O algoritmo de menor complexidade utiliza os rótulos

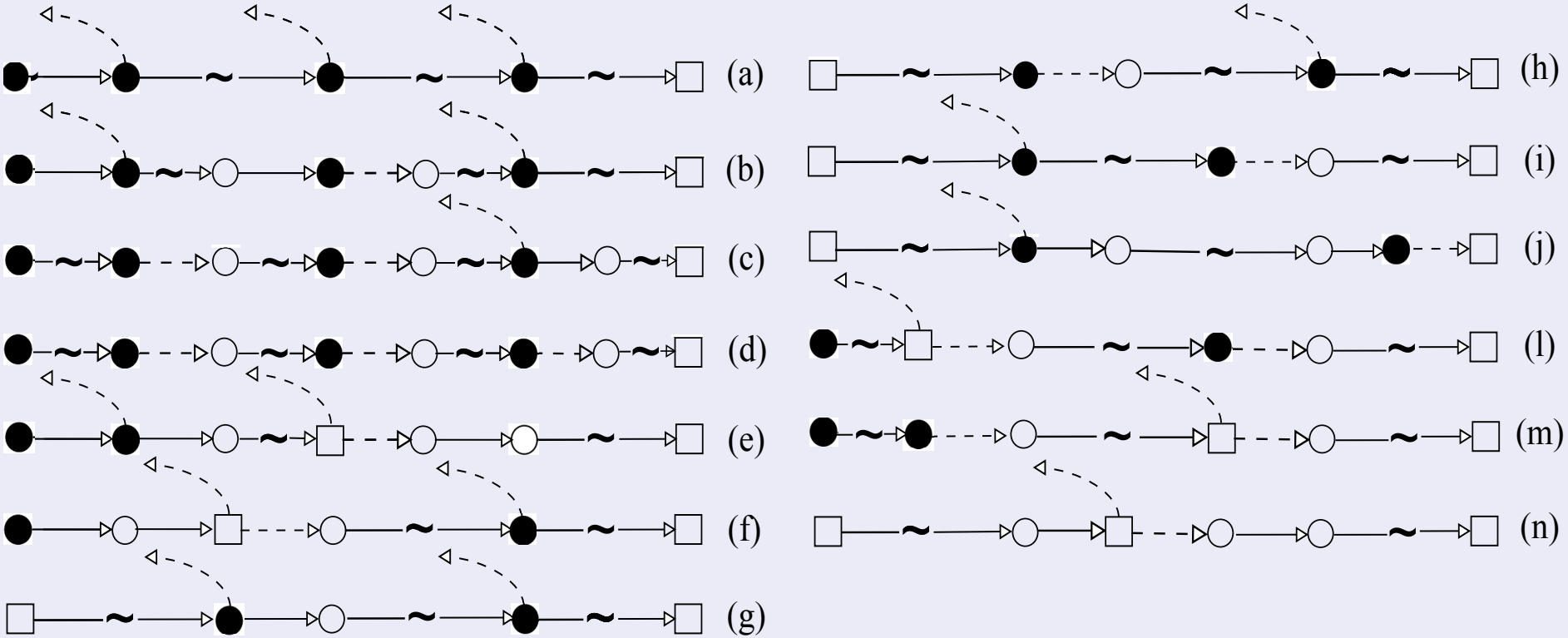
Mas quão razoável é assumir que os rótulos estão disponíveis?

- O caminho hamiltoniano está intacto
- Os rótulos estão explicitados nos blocos do código associados aos vértices do grafo
- É possível recuperar o caminho hamiltoniano

Temos um algoritmo linear que recupera o caminho hamiltoniano após a remoção de até duas arestas [1]

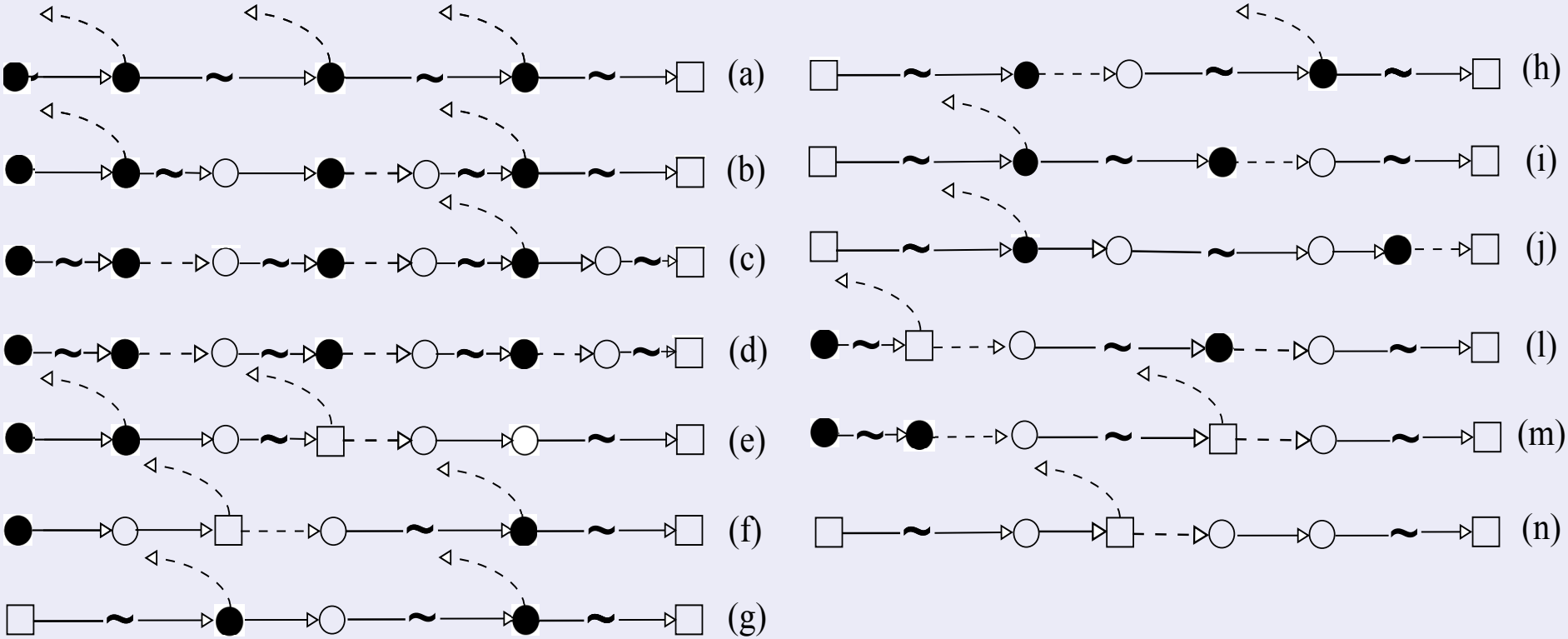
Recuperação do caminho hamiltoniano

Casos analisados – 3 arestas removidas



Recuperação do caminho hamiltoniano

Casos analisados – 3 arestas removidas



Complexidade: Cada subcaminho é computado em tempo $|H(v_i)|$.
Logo, o algoritmo roda em tempo $O(n)$.

Resultados computacionais

- As marcas d'água utilizadas nos testes compreendem:
  1. todas no intervalo $[2^4, 2^5 - 1]$ e $[2^9, 2^{10} - 1]$
  2. 10 mil escolhidas aleatória e uniformemente no intervalo $[2^{19}, 2^{20} - 1]$
  3. 10 mil escolhidas aleatória e uniformemente no intervalo $[2^{29}, 2^{30} - 1]$
  4. 10 mil escolhidas aleatória e uniformemente no intervalo $[2^{99}, 2^{100} - 1]$

Resultados computacionais

- As marcas d'água utilizadas nos testes compreendem:
  1. todas no intervalo $[2^4, 2^5 - 1]$ e $[2^9, 2^{10} - 1]$
  2. 10 mil escolhidas aleatória e uniformemente no intervalo $[2^{19}, 2^{20} - 1]$
  3. 10 mil escolhidas aleatória e uniformemente no intervalo $[2^{29}, 2^{30} - 1]$
  4. 10 mil escolhidas aleatória e uniformemente no intervalo $[2^{99}, 2^{100} - 1]$
- Primeiro teste: comparação de desempenho
- Segundo teste: compreenção da resiliência diante de ataques de distorção

Comparativo dos algoritmos de decodificação

Chroni et al. [2] x Bento et al. [1]

n	alg. Chroni et al. (sem remoções)	alg. Bento et al. (sem remoções)	alg. Bento et al. (-1 aresta)	alg. Bento et al. (-2 arestas)
5	82.2 (4.4) μs	56.5 (3.2) μs	63.9 (6.7) μs	78.0 (16.4) μs
10	132.3 (9.3) μs	95.7 (5.8) μs	104.2 (9.4) μs	122.8 (24.8) μs
20	240.9 (11.8) μs	177.5 (9.7) μs	190.7 (17.4) μs	219.9 (44.9) μs
30	357.7 (14.4) μs	268.9 (13.2) μs	281.3 (18.2) μs	328.1 (66.0) μs
100	1406.7 (45.7) μs	1135.4 (39.5) μs	1151.2 (89.8) μs	1248.5 (260.4) μs

Resiliência da marca d'água de Chroni e Nikolopoulos

n (bits)	$p(n,3)$	$p(n,4)$	$p(n,5)$
3	5.71429%	8.57143%	33.33333%
4	2.38095%	2.57937%	5.09607%
5	1.21212%	1.93182%	2.91899%
6	0.69930%	1.23876%	1.92696%
7	0.43956%	0.76190%	1.21360%
8	0.29412%	0.47731%	0.75934%
9	0.20640%	0.30999%	0.48427%
10	0.15038%	0.20897%	0.31842%
11	0.11293%	0.14571%	0.21637%
12	0.08696%	0.10463%	0.15169%
13	0.06838%	0.07707%	0.10939%
14	0.05473%	0.05803%	0.08086%
15	0.04449%	0.04453%	0.06108%

Probabilidade $p(n,k)$ de uma marca d'água se tornar irrecuperável após a remoção de k arestas.

Trabalhos futuros

- Recuperação do caminho hamiltoniano para $k \geq 3$
- Marcas d'água randomizadas
- Uso de código de detecção e correção de erro
  - Reed-Solomon

Obrigado!

Proteção de Software por Marcas d'água Baseadas em Grafos

Lucila Bento[1,3], Davidson Boccardo[3], Raphael Machado[3], Vinícius Pereira de Sá[1], Jayme Szwarcfiter[1,2,3]

Instituto de Matemática – Universidade Federal do Rio de Janeiro[1]
COPPE – Universidade Federal do Rio de Janeiro[2]
INMETRO – Instituto Nacional de Metrologia, Qualidade e Tecnologia[3]

Bibliografia

- L. Bento, D. Boccardo, R. Machado, V. Pereira de Sá, J. Szwarcfiter (2013). Towards a provably robust scheme for graph-based software watermarking. http://arxiv.org/abs/1302.7262.
- M. Chroni and S.D. Nikolopoulos, Encoding watermark numbers as cographs using selfinverting permutations, 12th Int'l Conference on Computer Systems and Technologies, CompSysTech'11, ACM ICPS 578 (2011), 142–148 (Best Paper Award).
- M. Chroni and S.D. Nikolopoulos, An efficient graph codec system for software watermarking, 36th IEEE Conference on Computers, Software and Applications (COMPSAC'12), IEEE Proceedings (2012), 595–600.
- C. Collberg, A. Huntwork, E. Carter, G. Townsend and M. Stepp, More on graph theoretic software watermarks: implementation,

analysis and attacks, Information and Software Technology 51 (2009), 56–67.


C. Collberg, S. Kobourov, E. Carter and C. Thomborson, Error-correcting graphs for software watermarking, 29th Workshop on Graph-Theoretic Concepts in Computer Science, WG'03, LNCS 2880 (2003), 156–167.


R. L. Davidson and N. Myhrvold, Method and system for generating and auditing a signature for a computer program, US Patent 5.559.884, Microsoft Corporation (1996).


J. Zhu, Y. Liu and K. Yin, A novel dynamic graph software watermark scheme, 1st Int'l Workshop on Education Technology and Computer Science 3 (2009), 775–780.


R. Venkatesan, V. Vazirani, S. Sinha, A graph theoretic approach to software watermarking, 4th International Information Hiding Workshop (2001), 157–168.