PROJETO DE ALGORITMO PARA EXTENSÃO *K*-TOLERANTE A FALHAS DE GRAFOS CIRCULANTES

ALGORITHM PROJECT TO EXTENSION K-TOLERANT FAULT OF CIRCULANT GRAPHS

Leandro Gallinari
leandrogallinari@terra.com.br

Michael Damico
michael.damico@uol.com.br

Tales Pinheiro de Andrade
talespin@universiabrasil.net

Orientado por

Luiz Carlos da Silva Rozante
IMES – Universidade Municipal de São Caetano do Sul
lrozante@imes.edu.br

RESUMO

Sistemas multiprocessados são freqüentemente utilizados para resolução de problemas computacionais. Mas em alguns desses sistemas são necessários não somente a resposta mais rápida, mas uma resposta em um tempo inferior a um tempo máximo pré-determinado. Este trabalho descreve um método de projeto de sistemas multiprocessados k-tolerante a falhas em uma configuração de grafo circulante, adicionando k-processadores, de modo que se até k processadores falhem, o sistema continue fornecendo a resposta em um tempo menor que o limite.

Palavras-chave: sistemas multiprocessados, sistemas multiprocessados K-tolerante, projeto de algoritmo, grafos circulante, problemas computacionais.

ABSTRACT

Multiprocessed systems are used frequently for resolution of computational problems. But in some of these systems are necessary not only the fastest reply, but a response in a lesser time to the one best predetermined time. This work relates a method of project of k-tolerant multiprocessed systems relative to faults in a configuration of circulating graph, adding k-processors, so that if until k-processors fail, the system still be giving the reply in a short time than the limit.

Keywords: multiprocessed systems, k-tolerant multiprocessed systems, algorithm project, circulating graph, computational problems.

1. INTRODUÇÃO

Este trabalho se baseia no artigo de Farrag, [1996], que descreve uma forma muito eficiente de geração de extensões *k*-tolerante a falhas. Iremos descrever detalhadamente o funcionamento desse procedimento e, em seguida, descrever algumas formas de melhorar a complexidade do algoritmo, além de modificá-lo para melhorar sua performance.

O artigo está dividido em quatro seções. Além desta primeira seção, contendo a introdução, na seção 2 descrevemos o problema de forma geral, apresentando diversos conceitos necessários para a compreensão da solução proposta por Farrag, além de apresentar uma descrição simples e direta dessa solução. Na seção três, apresentamos a técnica de forma detalhada, demonstrando o pseudocódigo de forma completa e analisando sua complexidade, a partir da complexidade de cada parte interna do algoritmo. Por fim, na seção quatro, damos uma conclusão, além de sugestões para trabalhos futuros, baseados neste texto.

Diversas pesquisas são desenvolvidas na área de processamento paralelo. A computação paralela resolve, de forma mais rápida, diversos problemas, como mapeamento genético, simulações climáticas e controle de trafego aéreo, reduzindo muito o tempo requerido para a solução de um problema.

Podemos clarificar, de forma bem simplista, a computação paralela em duas categorias, quanto a sua necessidade: como primeiro caso, podemos citar a necessidade de encontrar a solução para um problema o mais rápido possível; o segundo caso seria resolver um problema em um tempo máximo predeterminado. Na primeira categoria estão problemas onde a solução na computação comum levaria, em alguns casos, até centenas de anos, como decifrar seqüencias genéticas, simulações climáticas e de reações químicas e decodificação de sinais de rádio de telescópios.

Nosso trabalho se aplica, principalmente, ao segundo caso. Como exemplo de aplicação, podemos citar computadores multiprocessados que efetuam controle de tráfego aéreo. Nesse tipo de sistema, não adianta dar a resposta em um tempo que tenda a zero, mas é crítico que ele responda em um tempo suficiente para evitar um desastre. Assim, é possível projetar um sistema que forneça a resposta antes de um tempo predeterminado. Se essa resposta vier antes, não irá gerar nenhum problema. Por isso, é necessário atribuir propriedades de tolerância a falhas nesses sistemas, com a inserção de novos processadores, por exemplo. O sistema se tornará mais rápido, porém será também mais seguro.

Além disso, muito do que está descrito neste trabalho é essencialmente escrito em termos da teoria dos grafos, podendo, portanto, ter utilizações além das inicialmente pensadas.

2. DESCRIÇÃO DO PROBLEMA

Diversas técnicas de extensões *k*-tolerante a falhas foram criadas. Dentre elas, podemos citar Farrag, [1995], Dutt [1990], Wu, 2000, Banâtre et al. [1989] e Lee and Wang [1995]. Uma técnica muito eficiente chamada particionamento em distâncias *m* foi definida em Farrag [1996].

Vamos descrever detalhadamente e implementar o algoritmo para particionamento em distancias *m* para o projeto de uma extensão *k*-tolerante a falhas de um grafo circulante. Para que possamos iniciar, vamos, primeiro, definir alguns conceitos que serão utilizados ao longo do texto.

Definição 1 *Grafo circulante*

Um grafo circulante é um grafo $G(\mathcal{V}, \mathcal{E})$ *de* n *vértices, onde existe pelo menos um ciclo no qual podem ser embutidos todos os vértices de* $\mathcal{V}$*, no qual o* i*-ésimo vértice é adjacente ao* $(i+a_j(mod\ n))$*-ésimo e ao* $(i-a_j(mod\ n))$*-ésimo vértice, para todo* a_j *em um conjunto* $\mathcal{A}$*, para* $1 \le a_j \le \left[\frac{n}{2}\right]$, $1 \le j \le \left[\frac{n}{2}\right]$, para $1 \le i \le \left[\frac{n}{2}\right]$ [Weisstein, 2000], [Farrag, 1996].

Os elementos do conjunto *A* serão denotados como *offsets*, [Farrag, 1996].

Definição 2 *Offset*

Estando os vértices do grafo circulante dispostos simetricamente e de forma eqüidistante ao longo de uma circunferência, um offset a é o número de arcos entre os vértices que estão no arco menor formado por uma corda entre dois vértices x *e* y *mais 1, se* $a \le \left[\frac{n}{2}\right]$ *, caso contrário,* $a = n - a$*. Assim, podemos dizer que no grafo circulante, dois vértices x e y são unidos por uma aresta, se e somente se, existe um offset* a_i *tal que se x e y são vértices numerados do grafo circulante, então* $a_i = (x - y) \bmod n$ [Farrag, 1996], [Dutt and Hayes, 1991].

Exemplo 1 *Considere um grafo de 15 vértices e o conjunto* $\mathcal{A} = \{1, 3, 5\}$*. A Tabela 1 mostra suas adjacências.*

Vértice	j = 1		j = 3		j = 5	
i	$i+j$	$i-j$	$i+j$	$i-j$	$i+j$	$i-j$
0	1	14	3	12	5	10
1	2	0	4	13	6	11
2	3	1	5	14	7	12
3	4	2	6	0	8	13
4	5	3	7	1	9	14
5	6	4	8	2	10	0
6	7	5	9	3	11	1
7	8	6	10	4	12	2
8	9	7	11	5	13	3
9	10	8	12	6	14	4
10	11	9	13	7	0	5
11	12	10	14	8	1	6
12	13	11	0	9	2	7
13	14	12	1	10	3	8
14	0	13	2	11	4	9

Tabela 1: Cálculo da adjacência dos vértices de um grafo circulante G [1, 3 ,5: 15]. Todos os valores estão em (*mod n*).

O grafo circulante G será denotado por $G[a_1, a_2,..., a_n: n]$, onde $0 < i < \lfloor \frac{n}{2} \rfloor$, $\mathcal{A} = \{a_1, a_2,..., a_i\}$, é o conjunto de *offsets* e n é o número de vértices no grafo G.

Devido à simetria do grafo circulante, neste artigo tomamos como o número máximo de *offsets* de um grafo o valor $\lfloor \frac{n}{2} \rfloor$, ou seja, $1 \leq |\mathcal{A}| \leq \lfloor \frac{n}{2} \rfloor$, e $a_i \leq \lfloor \frac{n}{2} \rfloor$, $\forall a_i \in \mathcal{A}$. Devemos notar também que se substituirmos um *offset* a_i pelo *offset* $n - a_i$ (ou por $-a_i$), o grafo terá exatamente as mesmas arestas. Por isso, se um *offset* a_i for calculado e não estiver nessa faixa de valores, este deverá ser convertido para o seu *offset* equivalente dentro da faixa $1 \leq a_i \leq \lfloor \frac{n}{2} \rfloor$, isto é, se a é um *offset* e $a > \lfloor \frac{n}{2} \rfloor$, então $a = n - a \bmod n$.

Definição 3 *Extensão* k *-ft*

Um grafo G' é dito ser uma extensão k*-tolerante a falhas (ou* k *– ft) de um grafo G se para cada subconjunto $\mathcal{F}$ de no máximo* k *vértices, o grafo $G' - \mathcal{F}$ que é obtido removendo-se os vértices em $\mathcal{F}$ de G' deve conter um subgrafo isomórfico à G.*

Agora que já temos alguns conceitos, vamos começar a explicar o algoritmo, lembrando que nosso problema consiste em, dado um grafo G e um inteiro k, determinar a extensão k-tolerante a falhas de G.

Seguindo a Definição 3, a idéia básica da geração de extensões k-tolerante a falhas é adicionar ao grafo G um conjunto $\mathcal{S}$ de vértices sobressalentes, sendo $|\mathcal{S}| = k$, isto é, um conjunto com k vértices, de modo a formar um novo grafo H. Assim, removendo-se um conjunto $\mathcal{F}$ de vértices de H, onde $\mathcal{F}$ representa o conjunto de vértices falhos, sendo $1 \leq |\mathcal{F}| \leq |\mathcal{S}|$, o grafo I resultante da remoção dos vértices falhos será isomórfico ao grafo G.

Para que o novo grafo H seja uma extensão k-tolerante a falhas do grafo G e para que a extensão k-tolerante a falhas seja eficiente, algumas propriedades [Dutt and Hayes, 1991] devem ser satisfeitas. São elas:

1. Baixa redundância, isto é, o número de vértices e arestas sobressalentes deve ser reduzido o máximo possível.
2. Reconfiguração rápida, incremental e distribuída. A reconfiguração incremental é a capacidade do sistema de se reconfigurar em caso de novas falhas, sem a necessidade de desfazer configurações para contornar falhas anteriores. Já a reconfiguração distribuída, é necessária para permitir que diferentes partes do sistema falhem e se reconfigurem concorrentemente, sem a necessidade de um controlador central para reconfiguração.
3. Projeto incremental, o que permite projetar um sistema com alta tolerância a falhas a partir de um sistema comum, sem grandes custos.
4. Amenabilidade para sobressalência local, isto é, a diminuição ao mínimo da necessidade de ligações e processadores sobressalentes.

Assim, foram criados alguns métodos para projetar extensões k-tolerante a falhas que atendam esses requisitos.

Um método básico consiste em adicionar os vértices usando o Teorema 1, [Farrag, 1996]:

Teorema 1 *Seja um grafo circulante $G[a_1, a_2, ..., a_n: n]$, nós podemos construir um grafo circulante H com $(n +$* k*$)$ vértices que é uma extensão* k*-tolerante a falhas de G, onde o conjunto A de offsets de H consiste da seguinte união:*

$$\{a_1, a_1 + 1, a_1 + 2, ..., a_1 + k\} \cup \{a_2, a_2 + 1, ..., a_2 + k\} \cup ... \cup \{a_i, a_i + 1, a_i + 2, ..., a_i + k\}.$$

Porém, uma extensão k-tolerante a falhas construída de acordo com o Teorema 1 só será eficiente em relação ao grau dos vértices (conseqüentemente ao número de arestas que deles saem) se os *offsets* de G forem consecutivos, isto é, tenham a forma $\{a, a + 1, a + 2, ...\}$. Podemos visualizar isso através dos seguintes exemplos.

Exemplo 2 *Considere o grafo circulante G* [1, 2: 13], *mostrado na Figura 1. Uma extensão 2-ft do grafo G, seria construída de acordo com o Teorema 1 da seguinte forma:*

$\{1, 1 + 1, 1 + 2\} \cup \{2, 2 + 1, 2 + 2\} = \{1, 2, 3, 4\}$

A extensão 2-ft de G [1, 2: 13] *seria o grafo H* [1, 2, 3, 4: 15] *exibido na Figura 2.*

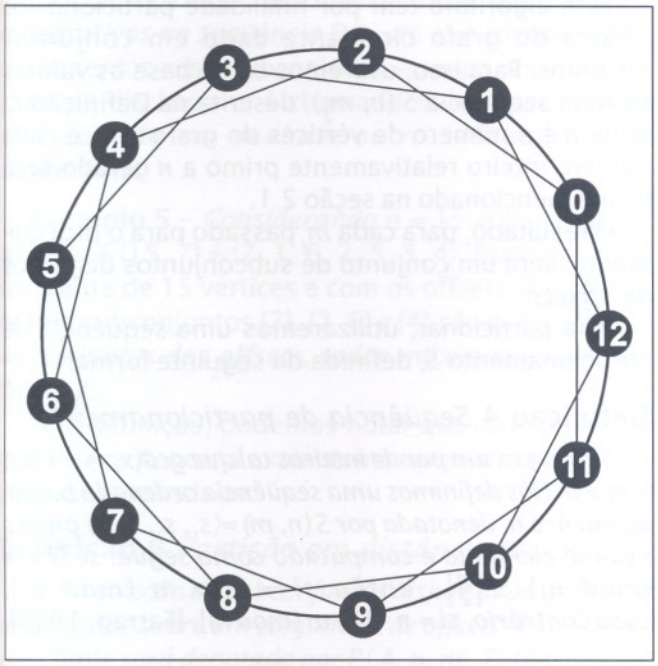

Figura 1: Um grafo circulante com $n = 13$ e $A = \{1, 2\}$.

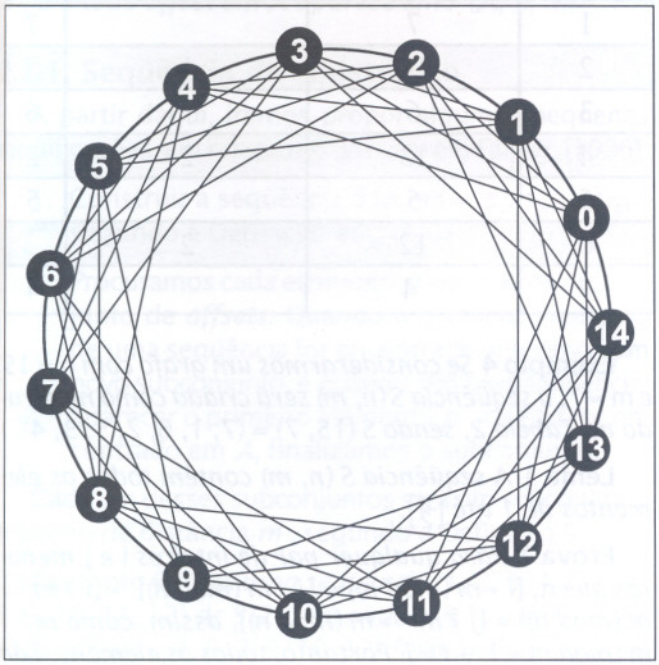

Figura 2: A solução 2-tolerante a falhas do grafo da Figura 1.

Exemplo 3 *Agora, se considerarmos o grafo G* [1, 3: 13], *sua extensão 2-ft será gerada da seguinte forma:*

$\{1, 1 + 1, 1 + 2\} \cup \{3, 3 + 1, 3 + 2\} = \{1, 2, 3, 4, 5\}$

sendo então o grafo H [1, 2, 3, 4, 5: 15].

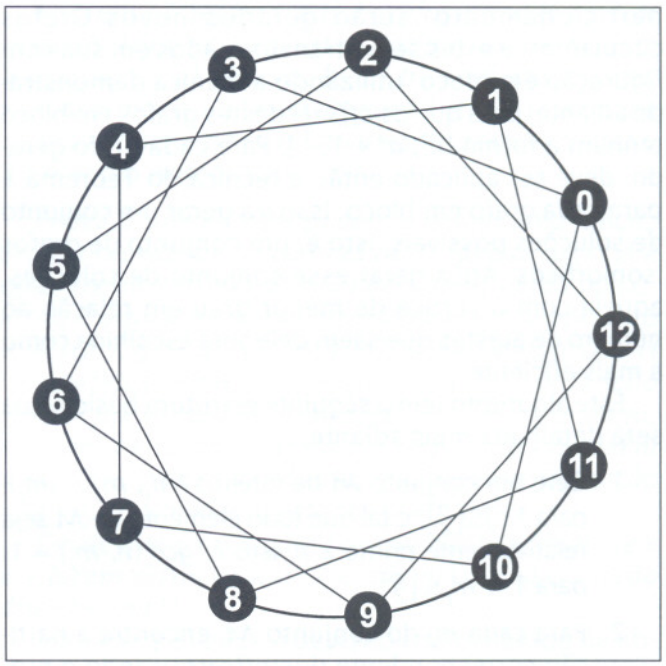

Figura 3: Um grafo circulante com $n = 13$ e $A = \{1, 3\}$.

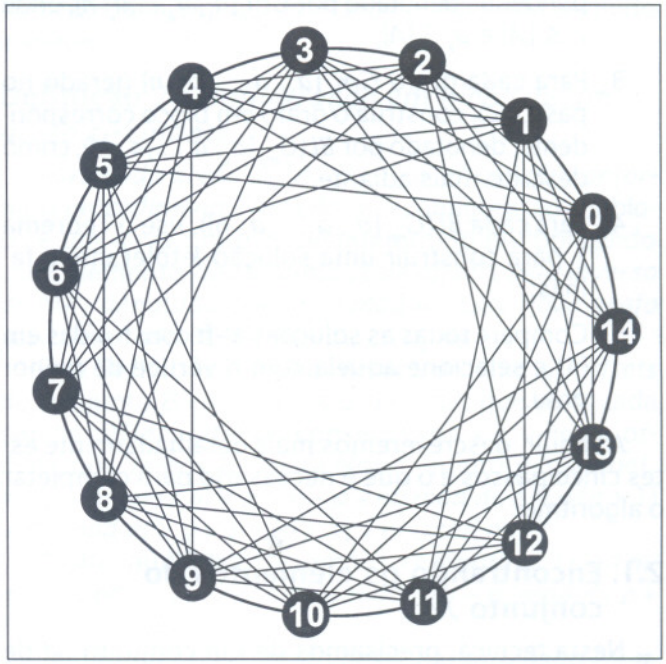

Figura 4: A solução 2-tolerante a falhas do grafo da Figura 3.

Podemos perceber que o método do Teorema 1 gera arestas desnecessárias na extensão *k*-tolerante a falhas para conjuntos de *offsets* não consecutivos.

Conforme foi descrito [Farrag, 1996], um método muito eficiente para encontrar soluções *k*-tolerante a falhas é através do particionamento em Distancias *m*. Sua idéia consiste em particionar o conjunto de *offsets* em conjuntos disjuntos de elementos, utilizando uma seqüência de particionamento que será gerada da forma descrita a seguir. Após o particionamento, serão gerados novos Grafos circulantes, e estes serão transformados em sua configuração em bloco, utilizando a técnica demonstrada adiante, para que os *offsets* destes grafos em bloco tenham a forma $\{a^1, a^1 + 1, ...\}$. Para cada grafo gerado, deve ser aplicado então a técnica do Teorema 1 para cada grafo em bloco. Isso irá gerar um conjunto de soluções possíveis, isto é, um conjunto de grafos isomórficos. Após gerar esse conjunto de soluções, aquela com o vértice de menor grau em relação ao número de arestas que saem dele será escolhida como a mais eficiente.

Este algoritmo tem a seguinte estrutura básica, que será detalhada mais adiante:

1. Gere um conjunto $\mathcal{M}$ de inteiros $\{m_1, m_2, ..., m_j\}$, para $1 \leq j \leq \lfloor \frac{n}{2} \rfloor$, e tal que todo elemento de $\mathcal{M}$ seja relativamente primo a *n*, isto é, $gcd(n, m_j) = 1$, para $1 \leq m_j \leq \lfloor \frac{n}{2} \rfloor$;
2. Para cada *mj* do conjunto $\mathcal{M}$, encontre a partição correspondente dos *offsets* usando o procedimento Partição($\mathcal{A}$, *n*, m_j) descrito adiante, e para cada partição construa o grafo correspondente, denotado por $Gm_j[a_1, a_2, ..., a_i; n]$, onde $i \leq |\mathcal{A}|$ e $a_i \in |A|$;
3. Para cada grafo $Gm_j[a_1, a_2, ..., a_i; n]$ gerado no passo (2), construa o grafo em bloco correspondente, denotado por $BL(G_{mj}[a_1, a_2, ..., a_i; n])$, como descrito mais adiante;
4. Para cada $BL(G_{mj}[a_1, a_2, ..., a_i; n])$, use o Teorema 1 para construir uma solução *k*-tolerante a falhas;
5. Compare todas as soluções *k*-ft construídas em (4) e selecione aquela com o vértice de menor grau.

A seguir, descreveremos mais detalhadamente estes cinco passos e o que é necessário para completar o algoritmo.

2.1. Encontrando os elementos do conjunto $\mathcal{M}$

Nesta técnica, precisamos de um conjunto $\mathcal{M}$ de números *m* relativamente primos a *n*.

Aqui será usado um procedimento *gcd*(*a*, *b*), que retorna o maior divisor comum entre dois números *a* e *b*. Sabendo que, se $gcd(a, b) = 1$, então os números *a* e *b* são relativamente primos, podemos gerar todos os inteiros $\{m_1, m_2, ..., m_j\}$, tal que para todo m_j tenhamos $gcd(n, m_j) = 1$, para $1 \leq m_j \leq \lfloor \frac{n}{2} \rfloor$.

2.2. O particionamento e a criação dos grafos $G_{mj}[a_1, a_2, ..., a_i; n]$

Esse algoritmo tem por finalidade particionar os *offsets* do grafo circulante dado em conjuntos disjuntos. Para isso, usaremos como base os valores de uma seqüência $S(n, m_j)$, descrita na Definição 4, onde *n* é o número de vértices do grafo e m_j é cada número inteiro relativamente primo a *n* gerado conforme mencionado na seção 2.1.

O resultado, para cada m_j passado para o procedimento, será um conjunto de subconjuntos disjuntos de *offsets*.

Para particionar, utilizaremos uma seqüência de particionamento *S*, definida da seguinte forma:

Definição 4 *Seqüência de particionamento*

Seja n *e* m *um par de inteiros tal que* $gcd(x, y) = 1$ *e* $n > m > 0$. *Nós definimos uma seqüência ordenada baseada em m e n, denotada por* $S(n, m) = \langle s_1, s_2, ..., s_i \rangle$ *onde o* i*-ésimo elemento é computado como segue: se* $[i * m \pmod n] \leq \lfloor \frac{n}{2} \rfloor$, *então* $si = [i * m \pmod n]$, *caso contrário,* $si = n - [i * m \pmod n]$, [Farrag, 1996].

Tabela 2: Seqüência *S* para *n* = 15 e *m* = 7.

Posição *i*	$[i * m \pmod n]$	$n - [i * m \pmod n]$	*si*
1	7		7
2	14	1	1
3	6		6
4	13	2	2
5	5		5
6	12	2	3
7	4		4

Exemplo 4 *Se considerarmos um grafo com* n = 15 *e* m = 7, *a seqüência* $S(n, m)$ *será criada como mostrado na Tabela 2, sendo* $S(15, 7) = \langle 7, 1, 6, 2, 5, 3, 4 \rangle$.

Lema 1 *A seqüência* $S(n, m)$ *contém todos os elementos de 1 até* $\lfloor \frac{n}{2} \rfloor$.

Prova 1 *Para qualquer par de inteiros* i *e* j *menores que* n, $[i * m \pmod n] = [j * m \pmod n] \rightarrow [i * m^{-1} * m \pmod n] = [j * m^{-1} * m \pmod n]$, *assim, como* $m^{-1} * m \pmod n = 1 \rightarrow i = j$. *Portanto, todos os elementos de* $S(n, m)$ *devem ser distintos. E como* $S(n, m)$ *contém*

exatamente $\left[\frac{n}{2}\right]$ *elementos inteiros positivos, cada um deles deve ser no máximo igual a* $\left[\frac{n}{2}\right]$, *além de conter todos os números de 1 até* $\left[\frac{n}{2}\right]$.

Definição 5 *Subconjunto máximo de distância* m

Um subconjunto de um conjunto de offsets de um grafo circulante de n *vértices é chamado subconjunto de distância* m, *onde* m *é um inteiro relativamente primo a* n, *se existe uma subseqüência de elementos consecutivos na seqüência* $S(n, m)$ *que contém todos* os elementos neste subconjunto. A seguir, um subconjunto de distância m é chamado *máximo se ele não está contido em nenhum outro subconjunto de distância* m.

Exemplo 5 - *Considerando* n = 15 *e* m = 7, *a seqüência* $S(15, 7) = \langle 7, 1, 6, 2, 5, 3, 4\rangle$. Para um grafo circulante de 15 vértices e com os offsets {2, 4, 6, 7}, os três subconjuntos {7}, {2, 6} *e* {4} *são subconjuntos do conjuntos dos offsets, todos máximos e com* distância 7.

Pela definição, podemos notar que cada *offset* do grafo vai pertencer a exatamente um subconjunto máximo de distância *m*.

Definição 6 *Partição em distância* m

Uma coleção de subconjuntos disjuntos de distância m definidos sobre um conjunto A de offsets de um grafo circulante será denotado por $P(\mathcal{A}, n, m)$. *Então,* $P(\mathcal{A}, n, m)$ *será chamado de Partição de A em distância* m *se cada subconjunto de distância* m *em* $P(\mathcal{A}, n, m)$ *é máximo e todo offset em* $\mathcal{A}$ *aparece em* $P(\mathcal{A}, n, m)$.

2.2.1. Seqüência de construção

A partir daqui, iremos propor algumas pequenas modificações, em relação ao descrito em Farrag, [1996].

1. Construir a seqüência $S(n, m) = \langle s_1, s_2, \ldots, s_{[\frac{n}{2}]}\rangle$, segundo a Definição 4.
2. Procuramos cada elemento *si* em $S(n, m)$ no conjunto de *offsets*. Quando o primeiro elemento de uma seqüência for encontrado, iniciamos um novo subconjunto, e quando, em uma seqüência, aparecer o primeiro elemento *si* que não for encontrado em $\mathcal{A}$, finalizamos o subconjunto.

Cada um desses subconjuntos será um subconjunto máximo de distância *m*, segundo a Definição 5.

Exemplo 6 *Para uma seqüência* $\langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\rangle$ *de* $S(25, m)$ *onde* $m = 1$, *e um conjunto de offset* {1, 4, 6, 7, 9, 10, 12} *formamos a seguinte partição:* $P(A, n, m) = \{\{1\}, \{4\}, \{6, 7\}, \{9, 10\}, \{12\}\}$.

2.3. Gerando os grafos $BL(G_m[a_1, a_2, \ldots, a_i; n])$

Baseado nos grafos formados de acordo com o procedimento na seção 2.2, temos então que construir para cada G_{mj} uma forma de grafo em bloco, denotado por $BL(G_{mj}[a_1, a_2, \ldots, a_i; n])$.

A idéia aqui é transformar os conjuntos particionados de distâncias *m* em seqüências de números inteiros consecutivos $\{j, j+1, j+2, \ldots\}$.

Primeiro, vamos definir o seguinte teorema:

Teorema 2 *Sejam G e H 2 grafos circulantes de* n *vértices com os seguintes conjuntos de offsets* $\mathcal{A} = \{a_1, a_2, \ldots, a_i\}$ *e* $\mathcal{B} = \{b_1, b_2, \ldots, b_i\}$, *respectivamente, e tal que* $bi = p * a_i (mod\ n)$, *para* $1 \leq i \leq j$, *onde p é qualquer inteiro que é relativamente primo a* n. Então G e H serão isomórficos.

Prova 2 *Sabendo que podemos definir uma função* φ *para reordenar os vértices de G, tal que se* $\{x, y\}$ *é uma aresta em não direcionada de G, então* $\{\varphi(x), \varphi(y)\}$ *é uma aresta não orientada em H (e portanto* $b_i = (\varphi(x) - \varphi(y))(mod\ n)$*), sendo então G e H isomórficos. Podemos então dizer que a função* $\varphi(x) = p * x\ (mod\ n)$. *Assim,* $\{x, y\} \equiv \{p * x\ (mod\ n), p * y\ (mod\ n)\}$ *pois* $(x-y)(mod\ n) \equiv (p*x - p*y)(mod\ n) = p*(x-y)(mod\ n)$ *[Knuth, 1973]. Pela Definição 1,* $a_i = (x-y)(mod\ n)$, *portanto* $bi = p * a_i$.

Desde que n e p sejam relativamente primos, se x e y forem distintos em G, então $\varphi(x)$ *e* $\varphi(y)$ *serão distintos em H.*

Em Farrag, [1996], é descrito como método de obter a forma em bloco de um grafo o processo de multiplicar cada elemento do conjunto de *offsets* de cada grafo $G_{mj}[a_1, a_2, \ldots, a_i; n]$ gerado pelo procedimento Partição $(\mathcal{A}, n, m_j)$ por $m_j^{-1}(mod\ n)$. Isto está definido a seguir.

Definição 7 *Configuração em bloco de um grafo circulante*

Para cada grafo $G_{mj}[a_1, a_2, \ldots, a_i; n]$ *gerado conforme o procedimento Partição, sua* configuração em bloco é um novo grafo circulante obtido multiplicando cada elemento *do conjunto* $\mathcal{A}$ *de offsets pelo inverso de* mj , *denotado por* $m_j^{-1}(mod\ n)$. *Esse novo grafo será denotado* $BL(G_{mj}[a_1, a_2, \ldots, a_i; n])$.

Pela Definição 5, nós podemos ordenar os *offsets* do conjunto $\mathcal{B}$ em alguma ordem tal, que para cada par de *offsets* inteiros consecutivos *x* e *y* nesta ordem, nós vamos ter, para algum inteiro *p* qualquer, $x = p * m_j (mod\ n)$ V $x = n - p * m_j (mod\ n)$, e $y = (p+1) * m_j (mod\ n)$ V $y = n - (p+1) * m_j (mod\ n)$.

Assim, multiplicando ambos *x* e *y* por $m_j^{-1}(mod\ n)$, iremos produzir o par ordenado $(p$ V $-p\ (mod\ n), p+1$ V $-p-1 (mod\ n))$.

Se o primeiro elemento neste par ordenado é $-p$, ou o segundo é $-(p+1)$, *offset* será convertido para a

faixa de *offset* de 1 até $\left[\frac{n}{2}\right]$, então o novo par correspondente à $\{x, y\}$ terá sempre a forma $(p, p+1)$. Isto é aplicável a qualquer par consecutivo no subconjunto ordenado $\mathcal{B}$, isto é, os *offsets* de $\mathcal{B}$ serão convertidos para a forma $\{j, j+1, j+2, ...\}$.

2.4. Construindo as soluções *k*-tolerante a falhas para cada $BL(G_{mj}[a_1, a_2, ..., a_i: n])$

Anteriormente, na seção 2.3, geramos os grafos em bloco que consistem de grafos com o conjunto de *offsets* consecutivos. é importante gerar esses grafos em bloco para termos um melhor resultado do algoritmo definido no Teorema 1, pois esse algoritmo é ótimo para construção de soluções *k*-tolerante a falhas onde o grafo possui *offsets* consecutivos.

Para gerar as extensões *k*-tolerante a falhas dos grafos $BL(G_m[a_1, a_2, ..., a_i: n])$, conforme descrito em Farrag, [1996] e em Dutt and Hayes, [1991], nós simplesmente adicionamos *k* vértices a cada grafo $BL(G_m[a_1, a_2, ..., a_i: n])$ utilizando o método do Teorema 1, já que os elementos do conjunto de *offsets* de cada grafo $BL(G_m[a_1, a_2, ..., a_i: n])$ são todos inteiros consecutivos.

2.5. Comparação das soluções *k*-tolerante a falhas de $BL(G_m[a_1, a_2, ..., a_i: n])$

É fácil perceber que este procedimento irá gerar diversas soluções. Farrag assumiu como a melhor aquela que gera o conjunto de *offsets* do grafo circulante *H*, que é a representação da extensão *k*-tolerante a falhas de *G*, com o menor número de elementos. Isso porque o grau de qualquer vértice *v* pertencente ao conjunto *V'* é calculado como sendo o número de elementos no conjunto de *offsets* de *H* vezes dois, isto é, $|H| * 2$. Este calculo é feito assim devido à alta simetria do grafo circulante.

Neste ponto, devemos então comparar os conjuntos de *offsets* de todos os grafos $BL(G_{mj}[a_1, a_2, ..., a_i: n])$ gerados na seção 2.4, eliminar os *offsets* repetidos de cada conjunto de *offsets*, e escolher o conjunto com menor número de elementos.

3. O ALGORITMO COMO UM PSEUDOCÓDIGO

Nesta seção, iremos descrever detalhadamente o algoritmo, fornecendo seu pseudocódigo e explicando como ele funciona.

3.1. Encontrando a melhor solução *k*-tolerante a falhas

Conforme apresentado em Farrag, [1996], podemos escrever o pseudocódigo do algoritmo, de uma forma mais geral, como apresentado no Algoritmo 1. Sua função é, após encontrar várias soluções *k*-tolerante a falhas possíveis, selecionar a melhor, conforme descrito na seção 2

Para todas as listas usadas, convencionamos usar uma estrutura de dados que contém um número inteiro que indica o tamanho da lista, e uma lista encadeada de números inteiros, na qual a inserção será sempre no começo da lista. Para simplificação do pseudocódigo, convencionamos acessá-la da mesma forma que um vetor estático. Mas no caso de inserção de novos elementos, estamos assumindo que esse vetor crescerá dinamicamente em uma posição. Assim, para o procedimento Tamanho (Lista), a ordem de complexidade é $O(1)$, pois ele apenas deve ler e retornar o número inteiro dessa estrutura que representa o tamanho da lista. Para inserção, mesmo sendo necessário alocar espaço para mais um elemento e incrementar o valor desse número inteiro em 1, como a inserção será sempre no começo da lista, sua complexidade será também $O(1)$, porém, a busca terá complexidade $O(n)$.

Para o Algoritmo 1 são passados dois parâmetros: o primeiro é o grafo circulante G, na forma de uma estrutura de dados contendo o número de vértices do grafo circulante original e uma sub-estrutura que contém uma lista dos *offsets* $\{a_1, a_2, ..., a\ n/2\}$. O segundo parâmetro é o número *k*, que indica o número de vértices falhos a serem tolerados.

```
1: KFT(G, k)
2: Lista_de_Primos ← ListaRPrimos(G.n)
3: TamanhoLista ← Tamanho(Lista de Primos)
4: L ← 1
5: para i ← 1 até TamanhoLista faça
6:     ListaParticoes ← Particao(G,Lista de Primos[i])
7:     para J ← 1 ate Tamanho(ListaParticoes) faça
8:         GrafoBloco ← GeraGrafoBloco(ListaParticoes[J])
9:         KFT ← GeraKFT(GrafoBloco[i],k)
10:        ListaKFT[L] ← KFT
11:        L ← L + 1
12:    fim para
13: fim para
14: OtimoKFT ← BuscaOtimoKFT(ListaKFT)
15: Retorne OtimoKFT
```

Algoritmo 1: Encontra a solução *k*-tolerante a falhas do grafo G.

Após analisarmos a complexidade de cada passo intermediário do Algoritmo 1, retornaremos à análise de sua complexidade.

3.2. Os passos intermediários

Aqui, iremos descrever cada passo intermediário do algoritmo e suas funcionalidades.

3.2.1. Encontrando a lista de números *m* relativamente primos à *n*

O Algoritmo 2 é usado para encontrar uma lista de números *mi* relativamente primos a um número *n*, que, no nosso caso, é o número de vértices do grafo G fornecido.

É fácil perceber que, no pior dos casos, tanto a atribuição de valores a *rp* quanto a comparação e posterior inserção de *rp* à lista *L* irá ocorrer $\lceil \frac{n}{2} \rceil$ vezes. Portanto, a complexidade do Algoritmo 2 é de $O(n \log^2 n)$.

```
1: ListaRPrimos(n)
2: J ← 1
3: para i ← 2 até ⌈n/2⌉ faça
4:     rp ← Euclides(n,i)
5:     se rp = 1 então
6:         L[J] ← rp
7:         J ← J+1
8:     fim se
9: fim para
10: Retorna L
```

Algoritmo 2: Calcula uma lista de números *m* relativamente primos a *n*.

O cálculo do máximo divisor comum - Para a execução do Algoritmo 2, é utilizado um procedimento, chamado Euclides(n,i) que calcula o "máximo divisor comum" entre dois números inteiros *a* e *b*. Caso o máximo divisor comum entre esses números seja igual à 1, então esses números são relativamente primos. Esse procedimento utiliza o algoritmo de Euclides, de complexidade $O(\log_2 n)$, demonstrado em [Knuth, 1973]. Apenas para fins ilustrativos, o algoritmo de Euclides é exibido no Algoritmo 3

```
1: Euclides(a,b)
2: se b=0 então
3:     retorna a
4: senão
5:     Euclides (b, mod(a,b))
6: fim se
```

Algoritmo 3: Verifica se dois números *a* e *b* são relativamente primos.

3.2.2. Particionando o conjunto dos *offsets*

Nesse algoritmo, o conjunto dos *offsets* do grafo *G* original é separado em partições disjuntas, usando como base a seqüência *S*, gerada através do Algoritmo 5, que será explicado a seguir. Para sua execução, ele recebe como parâmetros o grafo *G* original e um dos números *mi* do conjunto $\mathcal{M}$.

Calculando a seqüencia *S* - O Algoritmo 5 recebe como entrada o número *n*, que é o número de vértices do grafo *G*, e *m*, um número inteiro relativamente primo a *n*, gerado pelo Algoritmo 2, estando no intervalo de 1 até $\lceil \frac{n}{2} \rceil$. O resultado desse algoritmo é uma seqüência ordenada segundo a Definição 4.

Complexidade do Algoritmo 5 - Na linha 3 do Algoritmo 5 tanto a multiplicação $i * m$ quanto o cálculo do modulo $mod(i * m, n)$ tem complexidade $O(1)$. A comparação da linha 4 também tem complexidade $O(1)$. Sabendo que a inserção do elemento *S* à lista S [*i*] é de complexidade $O(1)$, e sabendo que os passos das linhas 3 até 7 irão executar $\lceil \frac{n}{2} \rceil$ vezes, concluímos que a complexidade do Algoritmo 5 é $O(n)$.

```
1: Particao(G, m)
2: SequenciaS ← GeraSeqS(G.n, m)
3: para i ← 1 ate ⌈n/2⌉ faça
4:     J ← 1
5:     Resp ← Busca(SequenciaS[i], G.A)
6:     se Resp=1 então
7:         ListaParticoes[J] ← SequenciaS[i]
8:         J ← J + 1
9:     fim se
10: fim para
11: Retorne ListaParticoes
```

Algoritmo 4: Gera um particionamento dos *offsets* do grafo G, em partições máximas de distância *m*.

```
1: GeraSeqS(n,m)
2: para i ← 1 ate ⌈n/2⌉ faça
3:     s ← mod(i*m, n)
4:     se s > ⌈n/2⌉ então
5:         s ← n - s
6:     fim se
7:     S[i] ← s
8: fim para
9: Retorna S
```

Algoritmo 5: Gera uma seqüência de números de 1 até $\lceil \frac{n}{2} \rceil$, ordenados segundo a Definição 4.

Podemos perceber que a operação elementar do Algoritmo 4 é a busca seqüencial executada na linha 5, que irá pesquisar se o elemento SequenciaS[i] existe na lista contendo o conjunto $\mathcal{A}$ dos offsets. Sabendo que o tempo de busca seqüencial de um elemento em uma lista não ordenada é de ordem $O(n)$, e que no nosso caso $n = |A|$, temos então um tempo de busca de ordem $O(|A|)$.

Como o esta busca será executada dentro de um laço que irá repetir $\lceil \frac{n}{2} \rceil$, a complexidade do Algoritmo 4 é $O(n|A|)$.

3.2.3. Encontrando sua forma em bloco

O Algoritmo 6 é usado para encontrar a forma em bloco do grafo $(G_m[a_1, a_2, \ldots, a_i; n])$, isto é, encontrar o Grafo $BL(G_m[a_1, a_2, \ldots, a_i; n])$.

Para sua execução, são passados como parâmetros uma lista de partições geradas pelo Algoritmo 4, e os números *n* e *m*, relativamente primos.

```
 1: GeraGrafoBloco(ListaParticoes,m,n)
 2: i_m ← 1/m
 3: tamanho ← Tamanho(ListaParticoes)
 4: para i ← 1 ate tamanho faça
 5:     offset ← i_m * ListaParticoes[i]
 6:     se Nao_Inteiro(offset) então
 7:         offset ←n - ListaParticoes[i]
 8:         offset ←offset * i_m
 9:         se Se Nao_Inteiro(offset) então
10:             offset ←n +ListaParticoes[i]
11:             offset ←offset * i_m
12:         fim se
13:     fim se
14:     Grafo.offset[i] ←offset
15: fim para
16: Retorne Grafo
```

Algoritmo 6: A partir de uma lista de números pertencentes a uma partição $P(\mathcal{A}, n, m)$, do número m_i e do tamanho do grafo G original, encontra sua forma em bloco.

Como é possível ver, o passo principal do Algoritmo 6 é o laço **para** da linha 4. Como a complexidade de todos os outros passos internos a esse laço é igual a $O(1)$, e esse laço irá executar $|A|$ vezes (devido ao tamanho máximo de ListaParticoes ser $|A|$), então a complexidade desse algoritmo é $O(|A|)$.

3.2.4. Encontrando a solução k-tolerante a falhas de cada $BL(G_m[a_1, a_2, ..., a_{\lfloor \frac{n}{2} \rfloor}]$

Após particionar o conjunto de *offsets* em conjuntos disjuntos e encontrar sua forma em bloco, esse algoritmo usa a técnica do Teorema 1 para gerar a solução k-tolerante a falhas do Grafo $BL(G_m[a_1, a_2, ..., a_i: n]$.

```
 1: GeraKFT(Grafo, k)
 2: A ← G.A
 3: tamanho ← Tamanho(A)
 4: para i ← 1 até tamanho faça
 5:     Hash_Insere(hash_solucao,n,A[i])
 6:     para L ← 1 até k faça
 7:         Hash_Insere(hash_solucao,n,A[i] + L)
 8:     fim para
 9: fim para
10: retorne hash_solucao
```

Algoritmo 7: Gera a solução k-tolerante a falhas para um dado grafo G e para um dado número k de vértices falhos a serem tolerados.

Na linha 5 do Algoritmo 7, utilizamos um procedimento chamado Hash_Insere. Esse procedimento utiliza uma função Hash para calcular a posição onde deverá ser inserido o *offset* encontrado. Utilizaremos apenas um vetor estático, sem utilizar a técnica conhecida como *chaining* para resolver a colisão. Isso porque devemos armazenar apenas uma vez o número encontrado, utilizando, então, uma função Hash apenas porque a complexidade de inserção será de ordem $O(1)$. Assim, como esse procedimento Hash_Insere é executado dentro do laço **para** da linha 6, que será executado k vezes, sendo k o número de tolerância a falhas desejadas, e esse laço **para** da linha 6 será executado $|A|$ vezes, então a complexidade desse algoritmo é de $O(|A|k)$.

3.3. A complexidade final

Agora, que já temos a complexidade de cada passo do Algoritmo 1, podemos ver qual será a complexidade do Algoritmo como um todo.

Como vimos acima, o passo da linha 2 (a execução do Algoritmo 2), tem complexidade $O(n \log_2 n)$. Já a execução do procedimento Tamanho(Lista_de_Primos) na linha 3 tem complexidade $O(1)$, como dito no inicio da seção 3.1. Como resultado, este procedimento retorna a quantidade de números m_i relativamente primos ao tamanho do grafo G original, que será no máximo igual a n.

Por isso, o laço **para** da linha 5 será executado no máximo $\lfloor \frac{n}{2} \rfloor$ vezes. Na linha 6, a execução do procedimento Partição é da ordem $O(n|A|)$, como demonstrado na seção 3.2.2.

A seguir, na linha 7 é executado o procedimento GeraGrafoBloco e na linha 8 o algoritmo GeraKFT, de complexidades $O(|A|)$ e $O(|A|k)$, respectivamente.

Assim, o conteúdo interno do laço da linha 5 tem complexidade $O(n|A| + |A|k)$. Como esse laço será executado n vezes, a complexidade final do Algoritmo 1 é de ordem $O(n^2 |A| + n\, k\, |A|)$.

4. CONCLUSÃO

Este estudo do trabalho publicado em Farrag, [1996] foi muito útil para pôr em prática as teorias apresentadas por Farrag. Atingimos com sucesso a expectativa de explicar e implementar a técnica de particionamento em distancias m para projeto de extensões k-tolerante a falhas proposta.

Apesar de ser um método relativamente simples de implementar, sua conceituação teórica é complicada, envolvendo um razoável nível de abstração matemática.

Foi possível na implementação perceber pequenos detalhes que facilitam a compreensão do procedimento como um todo, como a idéia de gerar a partição $P(\mathcal{A}, n, m)$, para, em seguida, encontrar a forma em bloco deste grafo.

Podemos, então, sugerir como uma continuação do trabalho uma pesquisa mais aprofundada em um método que permitisse prever quais seriam os grafos que iriam gerar as melhores soluções. Aparentemente, são os grafos em bloco com o menor número de subconjuntos, porém não foi possível a comprovação desta hipótese.

Se for possível prever quais os grafos que geram as melhores soluções, seria útil, então, sugerir a utilização de um algoritmo como o *Union/Find*, por exemplo, para armazenar os subconjuntos de *offsets* da partição $BL(P(\mathcal{A}, n, m))$, pois a inserção de novos *offsets* poderia gerar a união de dois subconjuntos.

REFERÊNCIAS BIBLIOGRÁFICAS

BANÂTRE, J. P.; BANÂTRE, M.; MULLER, G. Architecture of fault-tolerant multiprocessor workstations. In IEEE, editor, **Workstation operating systems: proceedings of the Second Workshop on Workstation Operating Systems** (WWOS-II), September 27-29, Pacific Grove, CA, pages 20-24, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA. IEEE Computer Society Press, 1989.

DUTT, S. Designing and reconfiguring fault-tolerant multiprocessor systems. **Technical Report CSE-TR-73-90**, University of Michigan, Ann Arbor, 1990.

DUTT, S.; HAYES, J. P. Designing fault-tolerant systems using automorphisms. **Journal of Parallel and Distributed Computing**, 12(3):249-268, 1991.

FARRAG, A. Algorithm for constructing fault-tolerant solutions of the circulant graph configuration. In **Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation**, Los Alamitos, California. IEEE Computer Society Press. Dalhousie University, 1995.

FARRAG, A. A. New algorithm for constructing fault-tolerant solutions of the circulant graph configuration. **Parallel Computing**, 22(9):1239-1253 (or 1239-1254??), 1996.

KNUTH, D. E. **The Art of Computer Programming, Vol. 2: Seminumerical Algorithms**. Addison-Wesley, Reading, MA, second edition, 1973.

LEE, I. Y.-Y.; WANG, S.-D. Ring-connected networks and their relationship to cubical ring connected cycles and dynamic redundancy networks. **IEEE Transactions on Parallel and Distributed Systems**, 6(9):988-996, 1995.

WEISSTEIN, E. Eric Weisstein's world of mathematics. http://mathworld.wolfram.com, 2000.

WU, J. A fault-tolerant adaptive and minimal routing scheme in n-D meshes. In **Proceedings of 2000 International Conference on Parallel Processing (29th ICPP'00)**, Toronto, Canada. Ohio State Univ, 2000.