UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

CARLOS FILIPE BENEVIDES

FERRAMENTA PARA ENUMERAÇÃO DE MODELOS DE GRAFOS ARCO-CIRCULARES E RECONHECIMENTO DE GRAFOS ARCO-CIRCULARES NORMAIS

RIO DE JANEIRO

2013

FERRAMENTA PARA ENUMERAÇÃO DE MODELOS DE GRAFOS ARCO-CIRCULARES E RECONHECIMENTO DE GRAFOS ARCO-CIRCULARES NORMAIS

CARLOS FILIPE BENEVIDES

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Informática.

Apresentado por:

_____

Carlos Filipe Benevides

Aprovado por:

_____

Prof. Vinícius Gusmão Pereira de Sá

_____

Prof. Mitre Costa Dourado

_____

Prof. João Antônio Recio da Paixão

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2013

AGRADECIMENTOS

Expresso a minha gratidão a todos aqueles amigos, companheiros e colegas de classe que direta ou indiretamente contribuíram para as pesquisas, desenvolvimento e elaboração do meu projeto final de curso.

Em especial, agradeço a minha família pelo apoio e incentivo na preparação deste trabalho e ao meu professor e orientador Vinícius Gusmão Pereira de Sá, que se dedicou intensamente ao projeto, tirando dúvidas, fazendo sugestões e correções, discutindo assuntos e indicando os melhores caminhos.

RESUMO

FERRAMENTA PARA ENUMERAÇÃO DE MODELOS DE GRAFOS ARCO-CIRCULARES E RECONHECIMENTO DE GRAFOS ARCO-CIRCULARES NORMAIS

CARLOS FILIPE BENEVIDES

Orientador: Vinícius Gusmão Pereira de Sá

Grafos arco-circulares são grafos de interseção de arcos em uma circunferência. Um grafo arco-circular é dito normal quando admite um modelo geométrico em que a união de dois arcos nunca cobre toda a circunferência. Determinar se um grafo arco-circular é normal constitui um problema ainda em aberto, com aplicações em escalonamento e otimização de tarefas, entre outras áreas. O presente trabalho propõe uma ferramenta em Java para auxiliar futura pesquisa no tema. O programa aqui apresentado gera modelos geométricos para um grafo arco-circular dado, verificando se o mesmo é ou não é normal.

ABSTRACT

A TOOL FOR ENUMERATING CIRCULAR-ARC GRAPH MODELS AND RECOGNIZING NORMAL CIRCULAR-ARC GRAPHS

CARLOS FILIPE BENEVIDES

Advisor: Vinícius Gusmão Pereira de Sá

Circular-arc graphs are intersection graphs of arcs in a circle. A circular-arc graph is said to be normal if it admits a geometric model in which the union of two arcs never subtends the whole circle. Determining whether a circular-arc graph is normal is still an open problem, with applications in task optimization and scheduling, among other areas. This monograph introduces a Java tool to aid in future research on the subject. The proposed program generates geometric models for a given circular-arc graph, verifying whether or not it is normal.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACN - Grafo arco-circular normal

ACNN - Grafo arco-circular não-normal

cc - Cobre o círculo

ct - Contém

cd - Contido

di - Disjunto

NAC - Grafo não arco-circular

sp - Sobrepõem

SUMÁRIO

1. INTRODUÇÃO

Grafos arco-circulares são uma classe de grafos de interseção. Trabalhos sobre essa classe de grafos foram primeiramente escritos por Hadwiger, Debrunner e Klee em 1964 [6] e por Klee em 1969 [10]. Tucker escreveu sua tese de doutorado sobre esta classe em 1969 [18], e em 1970 publicou uma série de artigos [19-23] contendo propriedades fundamentais sobre grafos arco-circulares. Estes grafos tem recebido considerável atenção, em razão de existirem diversas aplicações em áreas como genética, controle de tráfego, escalonamento, otimização de tarefas e muitas outras.

Para exemplificar, os grafos arco-circulares podem ser usados para representar a utilização de recursos em compiladores. Em um *loop* utilizado frequentemente durante um programa, algumas variáveis devem ser alocadas em registradores de índice. Nesse caso, o tempo entre o momento que uma dessas variáveis é alocada pela primeira vez, dentro de uma iteração do *loop*, até o momento em que a mesma é utilizada pela última vez, pode ser visto como um arco sobre uma circunferência. Ver as referências [1-3,8] para mais exemplos de aplicações.

Uma classe de grafos arco-circulares são os grafos arco-circulares normais, que são estudados já há algum tempo. Um dos primeiros textos a mencionar a nomenclatura "normal" foi o artigo [11], assinalando que os mesmos, geralmente, são muito complicados de caracterizar. Entre as suas aplicações, o algoritmo de reconhecimento dos grafos arco-circulares próprios utilizam como passo intermediário transformar o modelo dado em um que seja normal. Para tal, prova-se que todo grafo arco-circular próprio admite um modelo normal. Outra aplicação é com os arco-circulares Helly, que não necessariamente admitem modelos normais, mas quando os admitem se tornam mais facilmente manipuláveis [13].

Neste trabalho, exploramos a questão do reconhecimento de grafos arco-circulares normais, que é um problema ainda em aberto. Nossa contribuição é uma ferramenta computacional que indica se um grafo possui um modelo arco-circular normal. Esse resultado pode ser usado por outros pesquisadores, em alguma das etapas de investigação (que geralmente envolvem a verificação de normalidade de grafos pequenos, para validar ou invalidar conjecturas) para a solução do problema.

Para o perfeito entendimento do tema, no Capítulo 2 apresentaremos o conceito e a definição de grafo e algumas subclasses. O Capítulo 3 é dedicado à definição e descrição das propriedades básicas dos grafos arco-circulares, bem como das classes relevantes dos mesmos: Normal, Helly, Unitário, Próprio [12]. O Capítulo 4 trata do reconhecimento de

Grafos Arco-Circulares e de Grafos Arco-Circulares Normais.

Finalmente, no Capítulo 5 propomos uma ferramenta computacional para o auxílio no reconhecimento dos grafos arco-circulares. No Capítulo 6, exibimos os resultados obtidos em testes do algoritmo para reconhecimento dos grafos arco-circulares normais e, no Capítulo 7, apresentamos um resumo do conteúdo deste projeto.

2. GRAFOS

Nesse trabalho só iremos considerar grafos finitos e não-direcionados. Denotamos um grafo por $G = (V, E)$, onde V é o conjunto dos vértices de G, e E é o conjunto de arestas de G (Figura 1). Denotamos também $|V| = n$ e $|E| = m$. Para uma aresta $e \in E$, podemos escrever $e = v_iv_j$, onde $v_i,v_j \in V$ são os extremos de e. Dizemos que v_i,v_j são vértices adjacentes.

Escrevemos $N(v_i)$ para representar o conjunto de vizinhos de v_i, e $N[v_i] = \{v_i\} \cup N(v_i)$. O complemento de G é o grafo $\underline{G}$, onde o conjunto de vértices é o mesmo e dois vértices são adjacentes em $\underline{G}$ se eles não são adjacentes em G. Costuma-se empregar o prefixo **co** para referir ao complemento de um grafo. Consequentemente, um grafo G é co-conexo quando o seu complemento está conexo; G é co-bipartido quando o seu complemento é bipartido, e assim por diante.

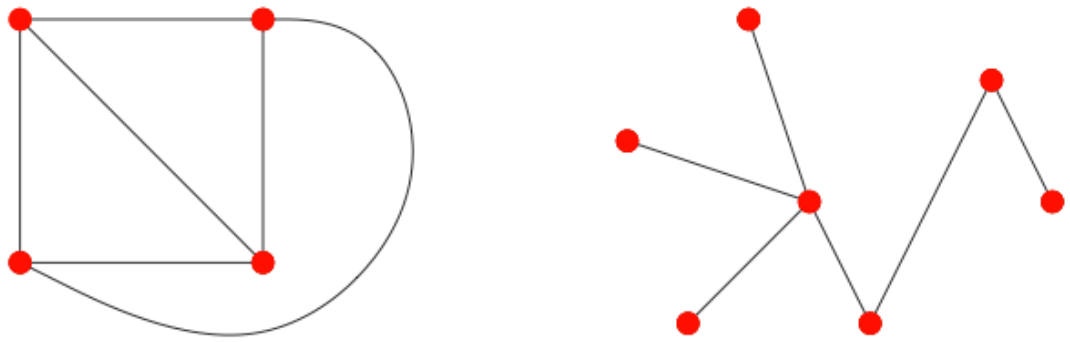

Figura 1: Exemplos de grafos

2.1. GRAFO DE INTERSEÇÃO

O **Grafo de Interseção** é um grafo no qual é possível relacionar cada vértice a um conjunto, de forma que dois vértices são adjacentes se, e somente se, os conjuntos correspondentes possuem interseção não vazia. Este mapeamento de vértices a conjuntos é denominado *modelo*.

Dado um modelo M, denotamos por $\Omega(M)$ o grafo de interseção de M, isto é, ao grafo $G = (V,E)$ com exatamente um vértice em V para cada conjunto de M e $v_iv_j \in E$ se, e somente se, os conjuntos correspondentes a v_i e v_j em M possuem interseção não vazia.

2.2. **GRAFOS DE INTERVALO**

Os **Grafos de Intervalo** são grafos de interseção de intervalos de uma reta real. Assim, $G = (V, E)$ é um grafo de intervalo quando existe um modelo $M = (R, I)$ tal que R é a reta real e, para cada intervalo $I_i \in I$, existe exatamente um vértice $v_i \in V$ correspondente. Ao modelo M chamamos de *modelo de intervalo*.

3. **GRAFOS ARCO-CIRCULARES**

Grafos Arco-Circulares são grafos de interseção onde consideramos arcos de uma circunferência. Obtemos o que chamamos de *modelo de arcos*.

Um grafo *G* é arco-circular se existe um modelo de arcos *M* = (*C*,*A*), onde *C* é a circunferência e *A* é o conjunto de arcos em *C*, tal que um vértice em *G* está relacionado a um arco em *M*, e se dois vértices são vizinhos, então há interseção entre os seus respectivos arcos (Figura 2). Nós sempre percorremos *C* no sentido horário, a menos que dito o contrário.

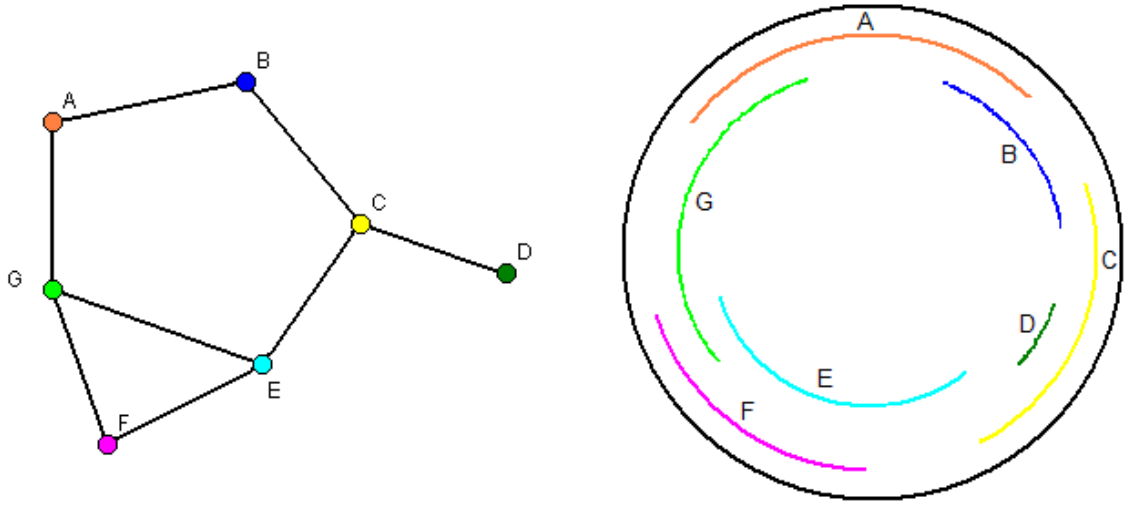

Figura 2: Exemplo de um grafo e um modelo arco-circular do grafo

O arco $A_i \in A$ é escrito como $A_i = (s_i, t_i)$. Dizemos que A_i *cobre* a circunferência *C*, no sentido horário, do ponto s_i ao ponto t_i, também chamados de extremos de A_i. Ao extremo s_i damos o nome de *ponto inicial* de A_i, enquanto que t_i é chamado de *ponto final* de A_i. Os extremos de *A* são os extremos de todos os arcos $A_i \in A$. Nós assumimos em geral que nenhum arco de *A* cobre *C*, que todos os arcos de *A* são abertos e que os extremos de *A* são todos distintos.

Uma *ordenação circular* $S = (p_1, p_2, ..., p_{2n})$ é uma permutação $p_1, p_2, ..., p_{2n}$ dos extremos dos arcos de A. Os pontos p_{2n} e p_1 são considerados consecutivos numa ordenação circular. Por exemplo, a ordenação circular da Figura 2 é (s_A, t_G, s_B, t_A, s_C, t_B, s_D, t_D, s_E, t_C, s_F, s_G, t_E, t_F). Uma ordenação circular define completamente um modelo arco-circular, e portanto as adjacências dos vértices do grafo correspondente. De fato, dois vértices são adjacentes se, e somente se, os arcos a eles associados se intersectam; na ordenação circular, a interseção de dois arcos A_i e A_j ocorre quando seus extremos aparecem (possivelmente intercalados com

extremos de outros arcos) em uma das seguintes permutações circulares: (s_i, s_j, t_j, t_i), (s_i, t_j, s_j, t_i), (s_i, s_j, t_i, t_j), (s_i, t_j, t_i, s_j), (s_i, t_i, t_j, s_j). Em outras palavras, a única possível permutação circular dos extremos de A_i e A_j em que os dois arcos não se intersectam é (s_i, t_i, s_j, t_j).

Os pares ordenados de arcos de *A* podem ser classificados de acordo com a sua posição relativa em *C*. Existem 5 tipos de pares de arcos $A_i, A_j \in A$: disjunto (di), quando $A_i \cap A_j = \emptyset$; contido (cd), quando $A_i \subset A_j$; contem (ct), quando $A_i \supset A_j$; cobre o circulo (cc), quando $A_i \cup A_j = C$; se sobrepoem (sp), caso contrário.

Existem alguns modelos especiais que são interessantes conhecer. Se a união de todos os arcos de *A* não cobrir o círculo *C* inteiro, então se trata de um modelo de intervalo. Se *M* não possui arcos cc, então o modelo é normal (Figura 3). Se o modelo não contém arcos cd, então o modelo é próprio. Se todos os arcos de *A* tem o mesmo comprimento, então o modelo é unitário. Se os arcos de *A* satisfazem a propriedade Helly, então o modelo é Helly. Os arcos de *A* satisfazem a propriedade Helly quando todo subconjunto de *A* cujos membros possuem dois a dois interseção não vazia possuem um elemento comum a todos os seus membros.

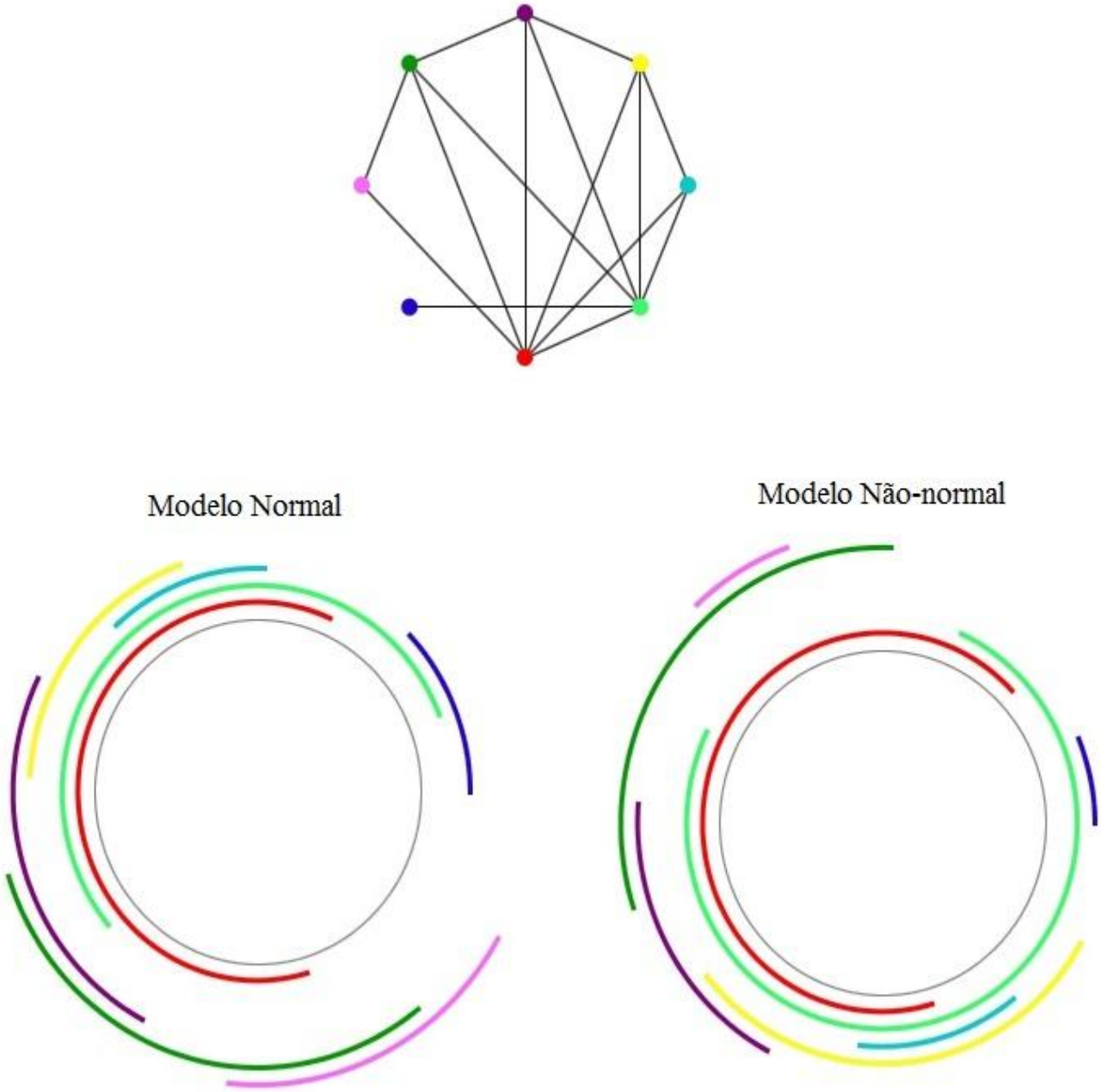

Figura 3: Exemplo de um grafo, um modelo arco-circular normal e um modelo arco-circular não-normal

As classes de grafos arco-circulares são denominadas com base nos modelos que elas aceitam, por exemplo: se um grafo arco-circular admite um modelo normal então o grafo é arco-circular normal. O mesmo é válido para os outros modelos.

Tucker caracterizou grafos arco-circulares em termos da ordenação circular dos seus vértices. Resumidamente o que ele mostrou em [19] é que um grafo G é arco-circular se e somente se existe uma ordenação circular $v_1,...,v_n$ dos seus vértices, tal que para $i < j$, se $v_iv_j \in E$ então ou $v_{i+1},..,v_j \in N(v_i)$ ou $v_{j+1},..,v_i \in N(v_j)$.

4. **RECONHECIMENTO DE GRAFOS ARCO-CIRCULARES**

O texto a seguir é inteiramente baseado no Capítulo 3 de [12], com o propósito de dar uma ideia sobre o estudo do reconhecimento de grafos arco-circulares ao longo dos anos.

A questão de reconhecimento de grafos arco-circulares foi primeiramente proposta por Hadwiger, Debrunner e Klee, em 1964 [6]. Tucker descreveu o primeiro algoritmo de tempo polinomial para esse problema [23]. A complexidade desse algoritmo é $O(n^3)$.

O conceito chave do algoritmo de Tucker pode ser descrito em termos de matrizes, como se segue: Seja G um grafo. A matriz de interseção de G é uma matriz λ ($n \times n$), onde cada entrada λ_{ij} informa o tipo de interseção, se houver, entre N[v_i] e N[v_j], para v_i, $v_j \in V$. Isso quer dizer:

$$\lambda_{ij} = \begin{cases} \text{di, se } v_iv_j \notin E \text{ , ou} \\ \text{cd, se N}[v_i] \subset \text{N}[v_j], \text{ ou} \\ \text{ct , se N}[v_i] \supset \text{N}[v_j], \text{ ou} \\ \text{cc, se N}[v_i] \cup \text{N}[v_j] = V, \text{ e} \\ \quad \text{para cada } v_k \in \text{N}[v_i] \setminus \text{N}[v_j], \text{N}[v_k] \in \text{N}[v_i], \text{ e} \\ \quad \text{para cada } v_k \in \text{N}[v_j] \setminus \text{N}[v_i], \text{N}[v_k] \in \text{N}[v_j], \text{ ou} \\ \text{sp.} \end{cases}$$

O propósito é estabelecer uma similaridade entre relações de vizinhança dos vértices do grafo G, e relações entre os arcos que vão corresponder aos vértices de G, assumindo que G é arco-circular.

Seja $M = (C,A)$ um modelo arco-circular. A matriz de interseção de M é uma matriz μ ($n \times n$), onde μ_{ij} expressa a relação entre A_i e A_j , onde A_i,$A_j \in A$, de tal forma:

$$\mu_{ij} = \begin{cases} \text{di, se } A_i, A_j \text{ são disjuntos , ou} \\ \text{cd, se } A_i \subset A_j, \text{ ou} \\ \text{ct , se } A_i \supset A_j, \text{ ou} \\ \text{cc, se } A_i \cup A_j = C, \text{ ou} \\ \text{sp, caso contrário.} \end{cases}$$

Então, dado um grafo G, a base dos métodos de reconhecimento descritos em [4,7,9,14,23] é construir uma matriz de interseção λ de G, e usá-la para guiar a construção de um modelo arco-circular de G, se existir um. Isso é: o modelo procurado é um cuja matriz de

interseção é exatamente λ.

É demonstrado, também, em [7,23], que, considerando um grafo G sem vértice universal e sem vértices gêmeos, a matriz de interseção de G é a matriz de interseção de algum modelo arco-circular de G. Consequentemente, a etapa inicial do algoritmo de reconhecimento para grafos arco-circulares seria calcular a matriz de interseção do grafo de entrada G.

O algoritmo de Tucker [23] considera dois casos para G: se ele é co-bipartido ou não. Cada um dos casos é tratado separadamente.

Quando G não é co-bipartido, o método consiste em procurar por um conjunto independente de G. De novo existem dois casos a se considerar: se o conjunto tem tamanho igual a 3 ou tamanho maior que 3. Em ambos os casos, o principio básico seria alocar, em pares, um conjunto de arcos disjuntos para corresponder a um conjunto independente. Subsequentemente, outros arcos são postos no modelo, o qual é refinado a cada etapa. O algoritmo de Tucker [23] faz essas operações em tempo $O(n^3)$.

Eschen e Spinrad [4] descreveram melhorias nesse método, que diminuíram o tempo para $O(n^2)$. Recentemente Kaplan e Nussbaum [9] melhoraram a alocação dos arcos disjuntos, para que o algoritmo rode em tempo linear.

Spinrad [17] descreveu um método para tratar do caso onde G é co-bipartido. Seja $V_1 \cup V_2 = V$ uma co-bipartição de G. Definimos um dígrafo D_G, onde $V(D_G) = V(G)$ e tem uma aresta direcionada de v_i para v_j , $v_i \neq v_j$, sempre que:

se $v_i, v_j \in V_1$ e $N[v_i] \subset N[v_j]$, ou

se $v_i, v_j \in V_2$ e $N[v_i] \supset N[v_j]$, ou

se $v_i \in V_1$, $v_j \in V_2$ e $v_i v_j \notin E(G)$, ou

se $v_i \in V_2$, $v_j \in V_1$ e $N[v_i] \cup N[v_j] = V(D_G)$.

Spinrad provou que se G for um grafo co-bipartido, sem gêmeos, então G é um grafo arco-circular se e somente se D_G tem ordenação parcial de ordem 2. Isso nos leva a um algoritmo de reconhecimento de grafos arco-circulares co-bipartidos de complexidade $O(n^3)$.

Eschen e Spinrad [4] reduziram a complexidade para tempo linear, usando um algoritmo para calcular as matrizes de interseção em tempo $O(n^2)$ e empregando um grafo bipartido cordal.

O algoritmo de Hsu [7] formalizou a base dos algoritmos de reconhecimento de grafos arco-circulares existentes até aquela data, empregando a ideia de matrizes de interseção para construir um modelo canônico para o grafo G.

Tal modelo é então transformado em um grafo círculo, que é um grafo de interseção

de cordas de um círculo. O problema de reconhecer grafos arco-circulares então é reduzido ao problema de reconhecer grafos círculos. O algoritmo é descrito em [7] e é usado no processo de reconhecimento de grafos arco-circulares. A complexidade geral é de O(*nm*).

O algoritmo proposto por McConnell [14] também emprega a redução para grafos bipartidos cordais [4] para construir a matriz de interseção. Porém, McConnell usa uma análise melhor para mostrar que a computação pode ser feita em tempo linear.

O algoritmo emprega a troca dos arcos pelos seus complementos e, também, aplica as matrizes de interseção, da seguinte forma: Seja μ uma matriz de interseção $n \times n$ e $1 \leq i \leq n$. O complemento de μ, relativo a i, denotado por $\underline{\mu_i}$ é a matriz obtida de μ mudando cada elemento da linha i e da coluna i, da seguinte forma:

ANTES		DEPOIS
di	→	ct
cd	→	cc
ct	→	di
cc	→	cd
sp	→	sp

Claramente, $\underline{(\mu_i)_i} = \mu$. Observamos que o complemento de μ, em relação a i, reflete as mudanças em um possível modelo arco-circular $M=(C,A)$ produzidas pela complementação do arco $A_i \in A$. A noção do complemento relativo a uma linha é mais generalizado em subconjuntos, da seguinte forma. Seja $I = \{i_1,...,i_k\} \subseteq \{1,...,n\}$, então o complemento de μ, relativo a I, denotado por $\underline{\mu_I}$ é a matriz obtida de μ iterando as operações de complementação, para cada $j \in I$. Isso é, primeiro considere o complemento de μ, relativo a i_1. Então pegue o complemento de $\underline{\mu_{i_1}}$, relativo a i_2, e assim por diante, até i_k ser considerado. De novo, $\underline{(\mu_I)_I} = \mu$.

McConnell mostrou em [14] que, dada uma matriz de interseção μ de um grafo arco-circular, é sempre possível achar um subconjunto de índices I, tal que $\underline{\mu_I}$ é uma matriz de interseção de um modelo ou grafo de intervalo. Essa matriz é chamada *matriz de intervalo*.

Em síntese, o algoritmo de McConnell pode ser esquematizado da seguinte forma: Seja G um grafo sem vértices universais e nem gêmeos, então :

1. Construir uma matriz de interseção μ de G;
2. Achar o complemento $\underline{\mu_I}$, relativo a um específico $I \subseteq \{1,...,n\}$, tal que $\underline{\mu_I}$ é

uma matriz de intervalo;

3. Construir um modelo *M*' para $\underline{\mu_I}$, tal que *M*' é um modelo de intervalo, sempre que *G* for um grafo arco-circular;
4. Construir o modelo *M*, de *M*', complementando todos os arcos A_i, tal que $i \in I$; e
5. Verificar se *M* é um modelo de *G*. Se sim, então *G* é arco-circular. Se não, então *G* não é arco-circular.

McConnell mostrou que todos os passos acimas podem ser implementados em tempo $O(n+m)$.

4.1. **RECONHECIMENTO DE GRAFOS ARCO-CIRCULARES NORMAIS**

O modelo arco-circular normal é tal que quaisquer dois arcos não cobrem inteiramente o círculo de referência. Logo, podemos dizer que o grafo é arco-circular normal quando possui pelo menos um modelo normal.

O problema, ainda em aberto para grafos gerais, consiste em determinar se um grafo é arco-circular normal. Quando o grafo considerado é co-bipartido, determinar se o mesmo é arco-circular normal pode ser feito por um algoritmo apresentado por Müller [15], cuja complexidade é $O(n^5m^6\log n)$.

Nosso programa computacional foi criado com o intuito de auxiliar os pesquisadores na busca da solução desse problema.

5. **FERRAMENTA PROPOSTA**

Nosso programa foi feito com o propósito de, a partir de um grafo, gerar todos os modelos arco-circulares possíveis.[1] A diferença principal entre os modelos é a posição relativa dos arcos no círculo, e não o tamanho deles. E depois, com base nesses dados, se testa todos os modelos gerados, determinando se o grafo é arco-circular normal ou não.

O programa se apresenta em duas partes: uma janela onde são colocados os dados do grafo (Figura 4) e; um conjunto de janelas que mostram os modelos arco-circulares e a representação do grafo, podendo escolher mostrar só os normais ou os não-normais (Figuras 6, 7 e 8). Os arcos dos modelos apresentados nesse programa percorrem o círculo no sentido anti-horário.

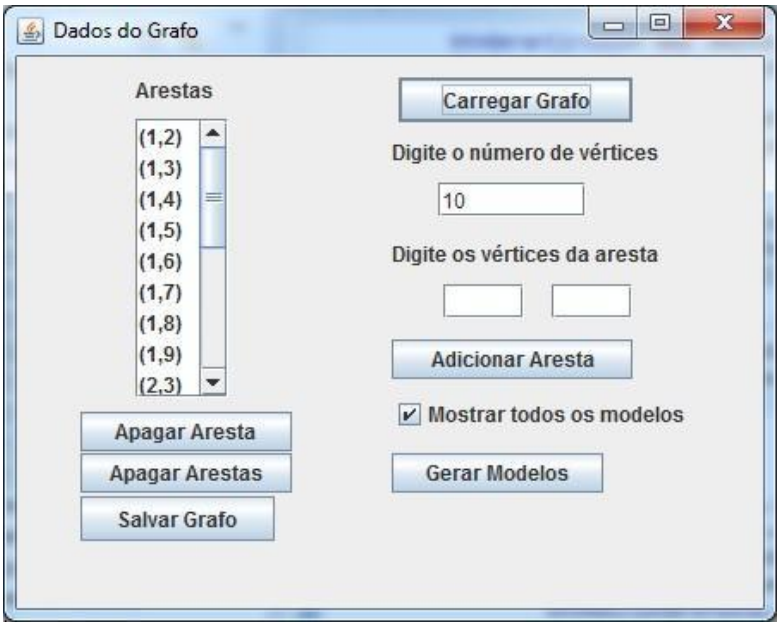

Figura 4: Janela para inserir os dados do grafo

Na janela onde são colocados os dados do grafo (Figura 4), pode-se inserir os dados manualmente ou carregá-los a partir de um arquivo XML.

O arquivo XML tem que seguir o seguinte padrão:

```
<grafo>
  <vertices>número de vertices</vertices>
  <arestas>
    <aresta>
```

[1] O código do programa, em linguagem Java, encontra-se disponível em https://github.com/cfb90/FERRAMENTA_PARA_ENUMERACAO_DE_MODELOS_DE_GRAFOS_ARCO-CIRCULARES.git

```
      <vertice1>um vértice da aresta</vertice1>
      <vertice2>o outro vértice da aresta</vertice2>
    </aresta>
   </arestas>
</grafo>
```

Como exemplo, para um grafo com 5 vértices e as aresta v_1v_2, v_3v_4 e v_2v_5, o XML é:

```
<grafo>
  <vertices>5</vertices>
  <arestas>
    <aresta>
      <vertice1>1</vertice1>
      <vertice2>2</vertice2>
    </aresta>
    <aresta>
      <vertice1>3</vertice1>
      <vertice2>4</vertice2>
    </aresta>
    <aresta>
      <vertice1>2</vertice1>
      <vertice2>5</vertice2>
    </aresta>
  </arestas>
</grafo>
```

Para inserir os dados manualmente coloque o número dos vértices e coloque as arestas do grafo. Se o número de vértices não condizer com as arestas inseridos o programa não irá conseguir gerar os modelos. É possível selecionar uma aresta e apagá-la, se for digitada errada. Os dados inseridos podem ser salvos num arquivo XML. Quando todos os dados tiverem sido inseridos, basta apertar no botão *Gerar Modelos* para gerar os modelos arco-circulares.

Se *Mostrar todos os modelos* estiver marcado, o programa irá mostrar todos os modelos válidos para o grafo inserido. Caso contrário o programa irá só mostrar um modelo normal e um modelo não-normal, se existirem, do grafo inserido. Pode ser necessário

desmarcar, pois alguns grafos podem ter tantos modelos que o programa irá parar de funcionar. Ele não conseguirá guardar todos os modelos.

Enquanto os modelos são gerados e testados na sua validade, a janela vai mostrar uma barra com a porcentagem dos modelos gerados e testados e mostrar quantos modelos foram gerados e testados do total de possíveis (Figura 5).

Figura 5: Janela enquanto os modelos estão sendo gerados e testados

Depois que os modelos são gerados e testados, o programa abre três janelas (Figuras 6, 7 e 8) para mostrar os modelos e para poder analisá-los. Se o grafo não for arco-circular aparece uma janela com um aviso e é possível mudar o que foi inserido.

Figura 6: Janela com os controles dos modelos gerados

A primeira janela (Figura 6) diz se o grafo é arco-circular normal ou não e tem os controles para as outras janelas. Podemos escolher o modelo a ser mostrado e marcar para mostrar só modelos normais, os não-normais ou todos. Podemos, também, marcar o início e fim de cada arco (Figura 9), lembrando que os arcos estão orientados no sentido anti-horário.

A cor de cada vértice é escolhida automaticamente, contudo podemos mudar individualmente, clicando na cor do lado do vértice ou podemos escolher a cor do primeiro vértice e do último e clicar no botão *Mudar Cores* para mudar a cor de todos os vértices entre as cores escolhidas, usando um gradiente degradê. Consequentemente, mudaria, também, a cor do arco relacionado ao vértice (Figura 7) e no desenho do grafo (Figura 8). O botão *Voltar* serve para voltar à janela da Figura 4.

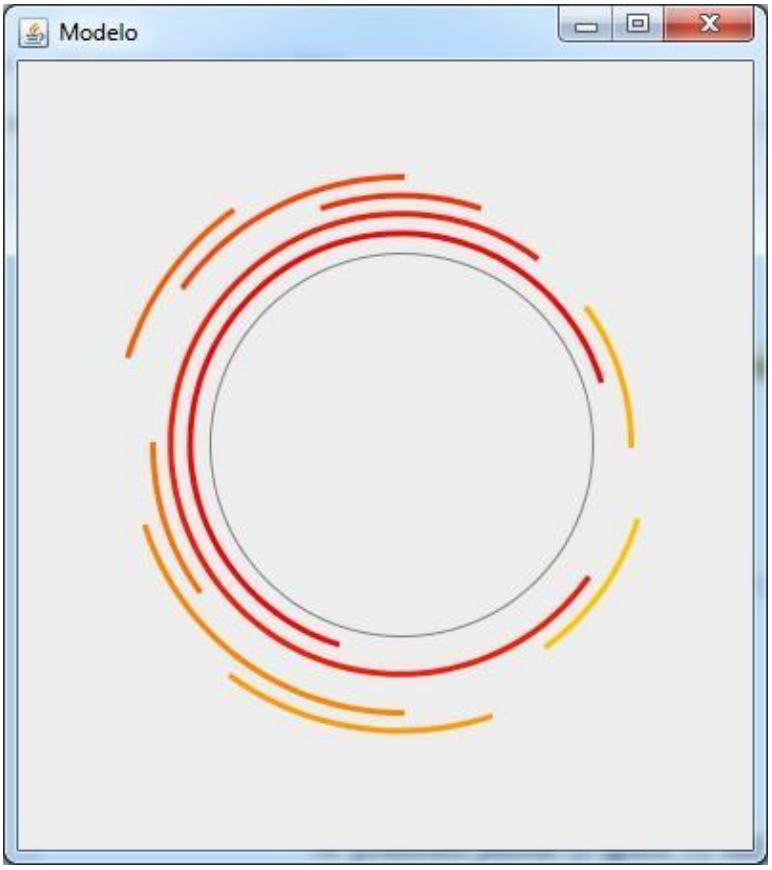

Figura 7: Janela que mostra o modelo arco-circular selecionado

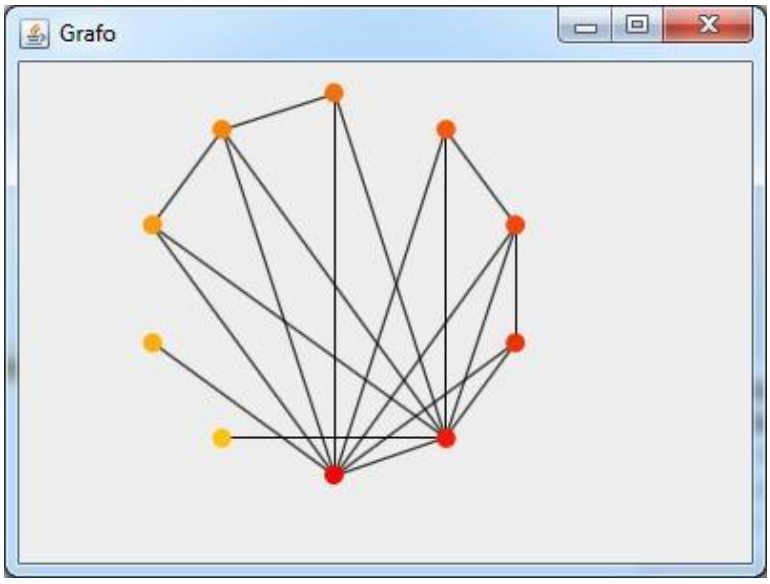

Figura 8: Janela que mostra um desenho do grafo inserido

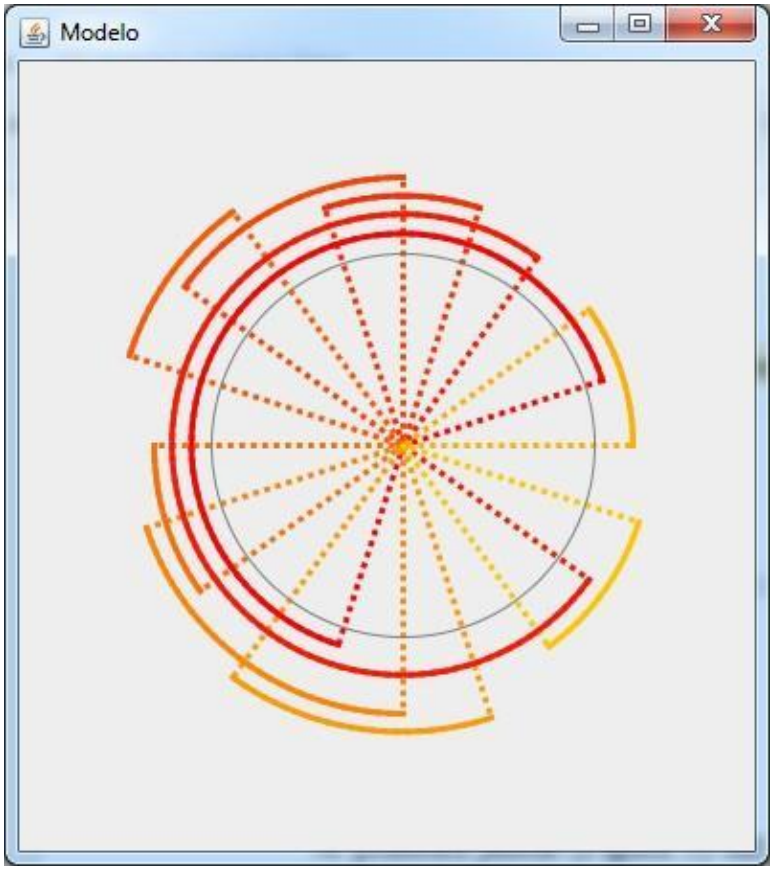

Figura 9: Janela do modelo com os inícios e fins dos arcos marcados

Para encontrar todos os modelos válidos para o grafo, geramos todas as ordenações circulares dos extremos dos arcos e depois testamos se elas são uma representação válida para o grafo. Por exemplo, a ordenação circular do modelo da Figura 7 é: (9, 1, -9, 2, 3, 4, -3, 5, -4, -5, 6, 7, -6, 8, -1, -7, -8, 10, -2, -10), onde o número positivo representa o início do arco relacionado ao vértice de mesmo número e o número negativo representa o fim do respectivo arco. Depois testamos cada modelo válido para descobrir se existe um modelo normal ou não.

Como dissemos, este programa foi feito com o intuito de ajudar pesquisadores nessa área a testar conjecturas em relação a grafos arco-circulares normais, possibilitando a análise dos modelos normais e não-normais e a geração rápida (para grafos pequenos) dos mesmos, quando existirem.

A seguir iremos mostrar os algoritmos usados no programa, para encontrar os modelos arco-circulares válidos e para testar se o modelo é normal. Caso exista um modelo normal, então o grafo é arco-circular normal.

5.1 **Código de Inicialização**

O seguinte algoritmo inicializa as variáveis necessárias para gerar os modelos arco-circulares. Os dados do grafo, vértices e arestas, podem ser inseridos manualmente ou por meio de um arquivo *XML*.

Carregamento do grafo (*G*)

Entrada: Um grafo $G = (V,E)$.

Saída: Um conjunto com todas as ordenações circulares válidas.

1. Conjunto dos extremos dos arcos $K = (1,2, ..., n,-1,-2, ...,-n)$;
2. Lista *Início* = (vértice com menor |N(*v*)|);//Se dois vértices estão empatados escolher o de maior número
3. Conjunto *Permutações*=();// Esse conjunto vai guardar os modelos válidos
4. HashMap[2] *Estado*;// Representa o estado de cada arco no Início
5. Conjunto *Teste*;// Esse conjunto irá conter as arestas representadas no Início
6. Para $i \leftarrow 1$ até *n* faça
7. *Estado*.add(*i*, não começado);
8. *Estado*.add(*Início*(0), começou);
9. Gerar_Permutações (*Permutações,G,Início,K,Estado,Teste)*;
10. Retorna *Permutações*;

5.2 **Algoritmo e Código para Geração de Ordenação Circular Válida**

Quando todos os dados já tiverem sido inseridos, os modelos serão gerados e testados na sua validade.

O algoritmo possui uma melhoria sutil em relação a se gerar, aprioristicamente, todas as ordenações circulares. Ele começa testando alguns extremos numa lista e, apenas se a ordenação circular parcial assim obtida for válida, adiciona outro extremo, e assim por diante, até que todos os extremos entrem na lista ou até que a lista seja não válida. Nesse caso, remove o último extremo, adicionando outro no lugar (*backtracking*).

Por exemplo, o programa testa a lista (1, 9, 2) para verificar se essa lista pode gerar uma ordenação válida, e se o programa detectar que essa lista é inválida, ele ignora todas as

[2] HashMap é uma implementação em Java para tabelas de dispersao, isto é, arrays associativos formados por pares (chave, valor) com endereçamento por hashing, provendo assim operações de dicionário em tempo médio constante.

ordenações circulares que começariam com (1, 9, 2) e testa a lista seguinte (1, 9, -2) para verificar se essa pode gerar uma ordenação válida e assim por diante. O presente algoritmo implementa essa idéia:

Gerar Permutações (*Permutações, G, Início, K,Estado,Teste*)

Entrada: Conjunto Permutações é o conjunto das ordenações circulares válidas; grafo $G = (V,E)$; Início é uma lista dos extremos válidos; K é o conjunto dos extremos dos arcos; *Estado* é um HashMap com os estados dos arcos de Início; e *Teste* contém as arestas que existem no Início.

Saída: Vazia.

1. Se |*Início*| igual |*K*| então
2. *Permutações* ← *Permutações* ∪ *Início*;
3. Retorna;
4. Para $i \leftarrow 0$ até $2n - 1$ faça
5. se $K(i) \in$ *Início* então vá para 4;
6. Senão
7. *Início* ← *Início* ∪ *K*(*i*);
8. HashMap *E* ← *Estado*;// O *E* serve para não perder o *Estado* anterior
9. Conjunto *T* ← *Teste*;// O *T* serve para não perder o *Teste* anterior
10. se Testar_Arco (*G*,*Início*,*E*,*T*) = válido então
11. Gerar_Permutações (*Permutações*,*G*,*Início*,*K*,*E*,*T*);
12. *Início* ← *Inicio*/*K*(*i*);
13. senão *Início* ← *Inicio*/*K*(*i*);
14. Retorna;

5.3 **Algoritmo e Código de Teste da Ordenação Circular**

O algoritmo seguinte testa a lista de extremos dos arcos. Verifica se a lista gera uma ordenação circular válida, na medida em que acrescentamos novos extremos.

Para testar a lista, o algoritmo verifica quais interseções entre os arcos existem no modelo. Se ele achar uma interseção que não é válida no grafo *G*, então o algoritmo diz que a lista passada não pode gerar uma ordenação circular válida. Caso contrário, a lista passada pode gerar uma ordenação circular válida.

Testar Arco (*G, Início,Estado,Teste*)

Entrada: grafo $G = (V,E)$; *Início* é uma lista dos extremos válidos; *Estado* é um HashMap com

os estados dos arcos de Início; e *Teste* contém as arestas que existem no Início. Saída: válido, se o *Início* não for uma representação válida de *G*, ou não válido, caso contrário.

1. Se *Início*(|*Início*|-1) > 0 então
2. Para $j \leftarrow 1$ até n faça
3. se *Estado*.get(j) = fechado então
4. Aresta t = nova Aresta(vértice j, vértice -*Início*(|*Início*|-1));
5. se $t \in E$ e $t \notin$ *Teste* então retorna não válido;
6. se *Estado*.get(j) = começou então
7. Aresta t = nova Aresta(vértice j, vértice *Início*(|*Início*|-1));
8. se $t \notin E$ então retorna não válido;
9. senão *Teste* $\leftarrow$ *Teste* $\cup$ t;
10. se *Estado*.get(j)=não começado e *Estado*.get(*Início*(|*Início*|-1))=invertido então
11. Aresta t = nova Aresta(vértice j, vértice *Início*(|*Início*|-1));
12. se $t \notin E$ então retorna não válido;
13. *Estado*.add(*Início*(|*Início*|-1), começou);
14. Retorna válido;
15. Senão se *Início*(|*Início*|-1) < 0 e *Estado*.get(arco -*Início*(|*Início*|-1)) = começou então
16. Para $j \leftarrow 1$ até n faça
17. se *Estado*.get(j) = invertido então
18. Aresta t = nova Aresta(vértice j, vértice *Início*(|*Início*|-1));
19. se $t \in E$ então
20. Para $i \leftarrow 1$ até n faça
21. se *Estado*.get(i) $\neq$ não começado então
22. Aresta *t1* = nova Aresta(vértice j, vértice i);
23. se $t1 \notin E$ então retorna não válido;
24. senão se ($i \neq j$) então
25. Aresta *t1* = nova Aresta(vértice *Início*(|*Início*|-1), vértice i);
26. Aresta *t2* = nova Aresta(vértice j, vértice i);
27. se $t1 \in E$ e $t2 \notin E$ então retorna não válido;
28. se *Estado*.get(j) = não começado então
29. Aresta t = nova Aresta(vértice j, vértice *Início*(|*Início*|-1));
30. se $t \in E$ e $t \notin$ *Teste* então retorna não válido;

31. *Estado*.add(-*Início*(|*Início*|-1), fechado);
32. Retorna válido;
33. Senão
34. Para $j \leftarrow 1$ até n faça
35. se *Estado*.get(j) $\neq$ não começado então
36. Aresta t = nova Aresta(vértice j, vértice -*Início*(|*Início*|-1));
37. se $t \notin E$ então retorna não válido;
38. senão *Teste* $\leftarrow$ *Teste* $\cup$ t;
39. *Estado*.add(-*Início*(|*Início*|-1), invertido);
40. Retorna válido;

5.4 **Algoritmo e Código de Teste do Grafo Arco-Circular Normal**

Após gerar todos os modelos válidos, o algoritmo verifica se existe algum modelo que é normal. Ele faz isso testando, em cada modelo, se existe algum par de arcos do tipo cc. Esse teste é feito submetendo-se cada um dos pares de arcos de um dado modelo ao algoritmo "Testar_Cobre_Círculo" da próxima seção.

Testar Normalidade (*Permutações*, *n*)

Entrada: *Permutações*, que é um conjunto com todas as ordenações circulares válidas, e *n*, onde *n* é |*V*| do grafo *G*.

Saída: Normal, se alguma ordenação for normal, ou Não-Normal, caso contrário.

1. Para $i \leftarrow 0$ até |*Permutações*|-1 faça
2. Lista *atual* $\leftarrow$ *Permutações*(i);
3. Para $j \leftarrow 1$ até n faça
4. Para $k \leftarrow j$ +1 até n faça
5. se Testar_Cobre_Circulo(*j*,*k*,*atual*) = falso então retorna Normal;
6. Retorna Não-Normal;

5.5 **Algoritmo e Código de Teste de Dois Arcos serem CC**

No caso do modelo ser não-normal, então ele possuirá dois arcos que cobrem o círculo (cc).

Para verificar se dois arcos cobrem o círculo todo, o algoritmo procura alguns padrões na ordenação circular que só aparecem se os arcos forem cc. Como pode ser facilmente verificado, esses padrões garantem que um arco cobrirá toda a parte do círculo não coberta pelo outro arco, logo os dois arcos são cc. Para demonstrar esses padrões, usaremos o exemplo a seguir:

Se o arco dois e o arco três cobrem o círculo, então a ordenação circular vai conter um dos seguintes padrões: (..., 3, ..., -2, ..., 2, ..., -3, ...), (..., 2, ..., -3,..., 3, ..., -2, ...), (..., -3, ..., 3,..., -2, ..., 2, ...) ou (..., -2, ..., 2,..., -3, ..., 3, ...). O algoritmo seguinte testa se dois arcos são cc no modelo.

Testar Cobre Círculo (*a1*,*a2*,*ordenação*)

Entrada: *a1* e *a2* são os números dos arcos que serão testados para checar se são cc e *ordenação* é a ordenação circular do modelo a ser testado.

Saída: verdadeiro, se os arcos forem cc, ou falso, caso contrário

```
1.  Para i ← 0 até |ordenação|-1 faça
2.    se ordenação(i ) = a1 então
3.      Para j ← i +1 até |ordenação|-1 faça
4.        se ordenação(j ) = -a1 ou ordenação(j ) = a2 então retorna falso;
5.        senão se ordenação(j ) = -a2 então
6.          Para k ← j +1 até |ordenação|-1 faça
7.            se ordenação(k ) = -a1 então retorna falso;
8.            senão se ordenação(k) = a2 retorna verdadeiro;
9.    senão se ordenação(i ) = -a1 então
10.     Para j ← i +1 até |ordenação|-1 faça
11.       se ordenação(j ) = -a2 ou ordenação(j ) = a2 então retorna falso;
12.       senão se ordenação(j ) = a1 então
13.         Para k ← j +1 até |ordenação|-1 faça
14.           se ordenação(k ) = a2 então retorna falso;
15.           senão se ordenação(k ) = -a2 retorna verdadeiro;
16.   senão se ordenação(i ) = a2 então
17.     Para j ← i +1 até |ordenação|-1 faça
```

18. se *ordenação*(*j*) = -*a2* ou o*rdenação*(*j*) = *a1* então retorna falso;
19. senão se *ordenação*(*j*) = -*a1* então
20. Para *k* ← *j* +1 até |*ordenação*|-1 faça
21. se *ordenação*(*k*) = -*a2* então retorna falso;
22. senão se *ordenação*(*k*) = *a1* retorna verdadeiro;
23. senão se *ordenação*(*i*) = -*a2* então
24. Para *j* ← *i* +1 até |*ordenação*|-1 faça
25. se *ordenação*(*j*) = -*a1* ou o*rdenação*(*j*) = *a1* então retorna falso;
26. senão se *ordenação*(*j*) = *a2* então
27. Para *k* ← *j* +1 até |*ordenação*|-1 faça
28. se *ordenação*(*k*) = *a1* então retorna falso;
29. senão se *ordenação*(*k*) = -*a1* retorna verdadeiro;
30. retorna falso;

6. **RESULTADOS COMPUTACIONAIS**

Nesta seção, apresentaremos o tempo médio que leva para o algoritmo verificar se um grafo é arco-circular normal. Para tal desenvolvemos um algoritmo de teste.

Os grafos são gerados a partir do número *n* de vértices e escolhemos uma probabilidade *p*, que é a probabilidade de cada aresta possível do grafo estar no grafo. Trata-se do modelo G(n,p) de grafos aleatórios, preconizado por Erdös e Rényi em [5]. Para cada par (n,p), geramos 50 instâncias aleatórias e as submetemos ao algoritmo de teste.[3]

O algoritmo foi alterado para em vez de gerar todos os modelos válidos para um grafo, como no programa, ele termina ao achar o primeiro modelo normal, pois então o grafo será normal.

Os números de vértices foram escolhidos de cinco a dez, porque abaixo de quatro os grafos são muito simples. E, acima de dez, o teste levaria um tempo de processamento excessivo com o equipamento disponível.

As probabilidades foram escolhidas na faixa de 10 a 90%, pois com 100% e 0% os grafos são todos iguais e são arco-circulares normais.

Seguem abaixo duas tabelas: uma contém as classificações dos grafos escolhidos aleatoriamente para os testes. Os grafos foram classificados pelo algoritmo em: ACN, se o grafo é arco-circular normal; ACNN, se o grafo é arco-circular não-normal; e NAC, se o grafo não é arco-circular; e na outra apresentamos o tempo médio computacional e o desvio padrão dos testes. O tempo foi calculado em milissegundos (ms).

Tabela de Resultados: Tabela 1.a

P / N	10%			20%			30%			40%			50%		
	ACN	ACNN	NAC	ACN	ACNN	NAC	ACN	ACNN	NAC	ACN	ACNN	NAC	ACN	ACNN	NAC
5	50	00	00	50	00	00	50	00	00	50	00	00	46	00	04
6	50	00	00	48	00	02	47	00	03	45	00	05	42	00	08
7	49	00	01	45	00	05	34	00	06	28	00	22	24	00	26
8	49	00	01	38	00	12	33	00	17	21	00	29	17	00	33
9	49	00	01	31	00	19	15	00	35	12	00	38	05	00	45
10	48	00	02	25	00	25	03	00	47	01	00	49	00	00	50

[3] O computador usado para rodar os testes usa o processador Intel core i5-3570K de 3.40GHz.

Tabela de Resultados: Tabela 1.b

P / N	60%			70%			80%			90%		
	ACN	ACNN	NAC	ACN	ACNN	NAC	ACN	ACNN	NAC	ACN	ACNN	NAC
5	48	00	02	50	00	00	50	00	00	50	00	00
6	45	00	05	47	00	03	50	00	00	50	00	00
7	33	00	17	42	00	08	45	00	05	50	00	00
8	18	00	32	29	00	21	38	00	12	50	00	00
9	06	00	44	20	00	30	36	00	14	46	00	04
10	03	00	47	09	00	41	25	00	25	48	00	02

Tabela de Tempos: Tabela 2.a

P / N	10%		20%		30%		40%		50%	
	Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão
5	0.58ms	2.13ms	0.60ms	2.16ms	0.60ms	2.13ms	0.64ms	2.15ms	0.68ms	2.30ms
6	0.82ms	3.24ms	0.78ms	2.55ms	0.83ms	2.46ms	0.98ms	3.03ms	1.34ms	4.25ms
7	0.9ms	3.52ms	0.92ms	3.23ms	1.06ms	2.67ms	1.5ms	2.8ms	1.9ms	3.27ms
8	0.8ms	2.42ms	1.08ms	2.79ms	1.7ms	4.42ms	2.94ms	4.8ms	5.18ms	9.95ms
9	0.96ms	3.96ms	2.2ms	5.51ms	3.12ms	7.09ms	4.68ms	8.51ms	10.94ms	15.83ms
10	1.0ms	2.84ms	3.18ms	5.29ms	6.1ms	6.92ms	8.02ms	9.48ms	32.9ms	67.4ms

Tabela de Tempos: Tabela 2.b

P / N	60%		70%		80%		90%	
	Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão
5	0.68ms	2.70ms	0.70ms	2.28ms	0.68ms	2.12ms	0.68ms	2.27ms
6	1.58ms	4.47ms	1.72ms	4.7ms	2.24ms	5.69ms	2.46ms	6.72ms
7	2.14ms	4.82ms	2.14ms	4.34ms	3.28ms	7.75ms	3.7ms	8.08ms
8	18.72ms	66.72ms	25.9ms	123.62ms	28.98ms	86.62ms	80ms	246.6ms
9	24.66ms	38.34ms	274.96ms	1105ms	470.96ms	1705ms	2.5s	11.1s
10	675.9ms	4380ms	1.8s	6.3s	12.34s	35.37s	254s	1243s

6.1. **ANÁLISE DOS DADOS**

Pela tabela de tempos, observamos que o tempo de processamento aumenta exponencialmente (como esperado) na medida em que cresce o número de vértices. Notamos que depois de 8 vértices, a quantidade de arestas influencia mais no tempo do que a quantidade de vértices. Isso se dá, possivelmente, porque estamos fazendo com que o algoritmo seja interrompido tão logo ele encontra o primeiro modelo normal para o grafo. Mais arestas significam mais restrições, portanto maior dificuldade de se encontrar um modelo.

Vemos que o desvio padrão é sempre maior do que o tempo médio. Isso indica que o tempo para analisar os grafos varia drasticamente, de forma que alguns grafos levam menos tempo do que um décimo da média para serem analisados, enquanto outros levam mais tempo do que média mais desvio juntos.

Notamos, com base na tabela de resultados, que a quantidade de grafos arco-circulares normais decresce na medida em que cresce o número de vértices. A quantidade de grafos arco-circulares normais decresce na medida em que cresce o número de arestas até certo ponto e depois volta a crescer quando a quantidade de arestas aumenta.

Não foi gerado nenhum grafo arco-circular não-normal nos testes. Esse resultado mostra a dificuldade de se encontrar subgrafos induzidos minimais que impeçam grafos arco-circulares de serem normais.

6.2 **GRAFO ARCO-CIRCULAR NÃO-NORMAL**

Um resultado importante é que com a ajuda do programa conseguimos achar um grafo arco-circular não-normal. Para achar esse grafo começamos com um grafo arco-circular normal e fizemos alterações no grafo até ele se tornar arco-circular não-normal.

Aqui foi muito importante termos a ferramenta. A cada grafo que conjecturávamos ser arco-circular não-normal, podíamos recorrer à ferramenta. Então, rodando o programa, víamos que estávamos errados. Até que, em dado momento, estávamos certos.

Abaixo mostramos um desenho do grafo de melhor visualização e um de seus modelos não-normais.

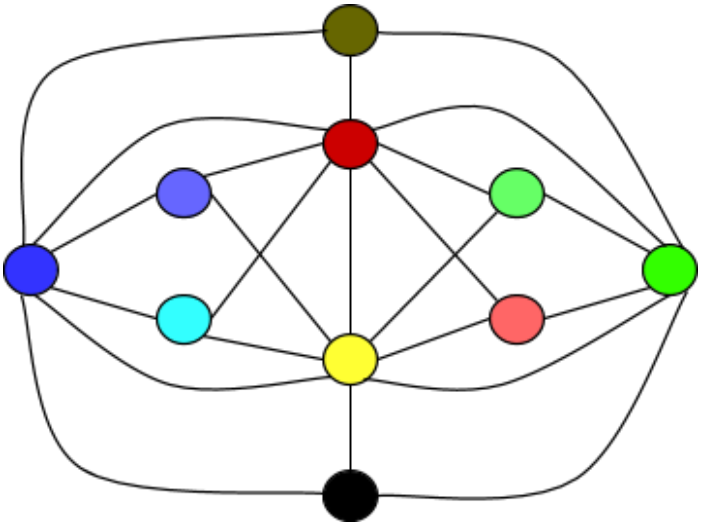

Figura 10: Grafo arco-circular não-normal

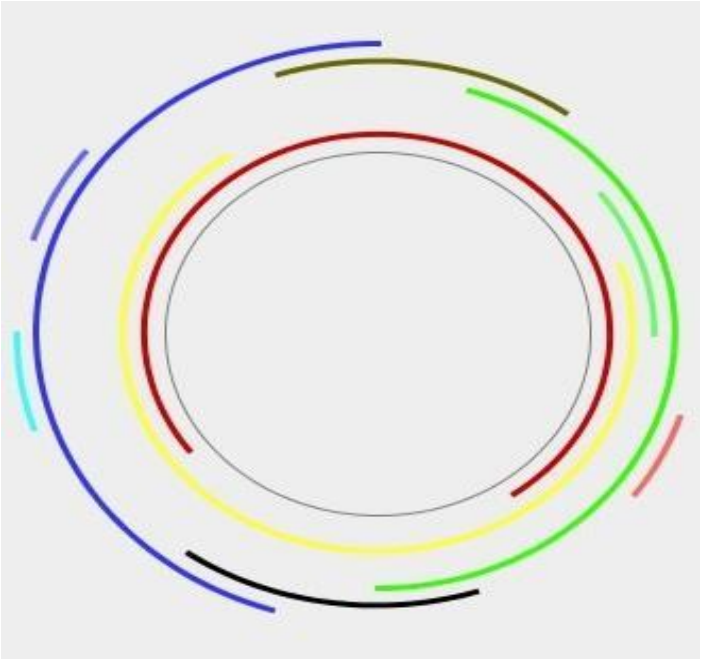

Figura 11: Modelo não-normal do grafo

7. **CONCLUSÃO**

Este trabalho teve como objetivo principal apresentar os problemas em aberto relacionados aos grafos arco-circulares normais e propor uma ferramenta computacional para ajudar a resolver esses problemas.

Até o Capítulo 4, apresentamos os principais trabalhos em relação aos grafos arco-circulares gerais. Mostramos e explicamos, resumidamente, os algoritmos principais para reconhecimento de grafos arco-circulares.

No Capítulo 5 apresentamos o programa que desenvolvemos, o qual poderá ser usado para testar conjecturas relacionadas aos grafos arco-circulares normais, como mencionado no Capítulo 4.1.

Esse programa usa algoritmos para testar, por força bruta, se um grafo é arco-circular normal. Ele também mostra os modelos arco-circulares do grafo para ajudar os pesquisadores a refinar suas teorias relacionadas a essas questões. Porém o tempo desses algoritmos aumenta rapidamente quando aumentamos o número de vértices, como mostrado no Capítulo 6.

Este trabalho não tenta solucionar os problemas, mas apresenta uma ferramenta para ajudar a resolver esses problemas e outros possíveis problemas que podem ser ajudados pela existência dessa ferramenta.

Acreditamos que tal ferramenta possa ser útil, sobretudo por considerarmos que, se a caracterização de grafos arco-circulares normais puder ser feita por subgrafos proibidos, então os subgrafos induzidos minimais que dela constarão terão, com boa chance, números de vértices não muito grandes.

Um outro ponto que gostaríamos de destacar é que a ferramenta pode ser utilizada imediatamente para gerar modelos para uma outra classe importante de grafos: os grafos de intervalo. Dado um grafo de intervalo G, basta acrescentar a G um novo vértice v que seja isolado, isto é, com grau zero. Dessa forma, se excluirmos o arco correspondente a v de qualquer modelo arco-circular encontrado para G pela ferramenta, o modelo assim obtido será um modelo de intervalo para G.

8. **REFERÊNCIAS BIBLIOGRÁFICAS**

1. BENZER, S., **On the topology of genetic structure**, Proceedings of the National Academy of Sciences, v. 45, p. 1607–1620, 1959.

2. CONFESSORE, G.; DELL'OLMO, P.; GIORDANI, S., **An approximation result for a periodic allocation problem**, Discrete Applied Mathematics, v. 112, p. 53–72, 2001.

3. CONITZER, V.; DERRYBERRY, J.; SANDHOLM, T., **Combinatorial auctions with structured item graphs**, in: Proceedings of the 19th National Conference on Artificial Intelligence, p. 212–218, 1994.

4. ESCHEN, E.M.; SPINRAD, J.P., **An O(n2) Algorithm for circular-arc graph recognition**, in: Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 128–137, 1993.

5. ERDÖS, P.; RÉNYI, A.,(1959). **On Random Graphs. I**, Publicationes Mathematicae v. 6, p. 290–297.

6. HADWIGER, H.; DEBRUNNER, H.; KLEE, V., **Combinatorial Geometry in the Plane**, New York, Holt Rinehardt and Winston, 1964.

7. HSU, W.L., **O(mn) algorithms for the recognition and isomorphism problems on circular-arc graphs**, SIAM Journal on Computing, v. 24, p. 411–439, 1995.

8. HUBERT, L., **Some applications of graph theory and related non symmetric techniques to problems of approximate seriation: The case of symmetric proximity measures**, British Journal of Mathematical and Statistical Psychology, v. 27, p. 133–153, 1974.

9. KAPLAN, H.; NUSSBAUM, Y., **A simpler linear-time recognition of circular-arc graphs**, in: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, v. 4059 p. 41–52, 2006.

10. KLEE, V., **What are the intersection graphs of arcs in a circle?**, American Mathematical Monthly, v. 76 ,p. 810–813, 1976.

11. LIN, M.C.; SZWARCFITER, J.L., **Unit circular-arc graph representations and feasible circulations**, SIAM Journal on Discrete Mathematics, v. 22, p. 409-423, 2008.

12. LIN, M.C.; SZWARCFITER, J.L., **Characterizations and recognition of circular-arc graphs and subclasses: a survey**, Discrete Mathematics, v.309, n. 18, p. 5618–5635, 2008.

13. LIN, M.C.; SOULIGNAC, F.; SZWARCFITER, J.L., **Normal Helly Circular-Arc Graphs and its Subclasses**, Discrete Applied Mathematics, v. 161, p. 1037-1059, 2013.

14. MCCONNELL, R.M., **Linear-time recognition of circular-arc graphs**, Algorithmica v. 37 p. 93–147, 2003.

15. MÜLLER, H., **Recognizing interval digraphs and interval bigraphs in polynomial time**, Discrete Applied Mathematics, v. 78 , p. 189-205, 1997.

16. SPINRAD, J.P., **Circular-arc graphs with clique cover number two**, Journal of Combinatorial Theory B, v. 44 ,p. 300–306, 1988.

17. SPINRAD, J.P., **Efficient Graph Representation**, Toronto, Fields Institute Monographs, 2003.

18. TUCKER, A., **Two characterizations of proper circular-arc graphs**, Ph.D. Thesis, California, Stanford University, Standford Operation Research Department, 1969.

19. TUCKER, A., **Characterizing circular-arc graphs**, American Mathematical Society, v. 76, p. 1257–1260, 1970.

20. TUCKER, A., **Matrix characterizations of circular-arc graphs**, Pacific Journal of Mathematics, v. 39, p. 535–545, 1971.

21. TUCKER, A., **Structure theorems for some circular-arc graphs**, Discrete Mathematics, v. 7 ,p. 167–195, 1974.

22. TUCKER, A., **Circular-arc graphs: New uses and a new algorithm**, in: Theory and Application of Graphs, Lecture Notes in Mathematics, v. 642, p. 580–589, 1978.

23. TUCKER, A., **An efficient test for circular-arc graphs**, SIAM Journal on Computing, v. 9, p. 1–24, 1980.