Um algoritmo polinomial para construção de um modelo de intervalo para grafos de intervalo

Ana Carolyne Pereira de Souza[1] **Julliano Rosa Nascimento**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74001-970 – Goiânia – GO – Brasil

ana.carolyne@discente.ufg.br, jullianonascimento@ufg.br

Abstract. *We explored some fundamental properties of interval graphs, such as the absence of cycles with four or more vertices and asteroidal triples. Based on these properties, given an interval graph G, we developed a polynomial algorithm that uses the perfect elimination scheme characteristic of chordal graphs, obtained through the lexicographic breadth-first search algorithm, to construct an interval model for an interval graph G.*

Resumo. *Exploramos algumas propriedades fundamentais de grafos de intervalo, como a ausência de ciclos com quatro ou mais vértices e triplas asteroidais. Com base nessas propriedades, dado um grafo de intervalo G, desenvolvemos um algoritmo polinomial que usa o esquema de eliminação perfeita próprio dos grafos cordais, obtido através do algoritmo de busca em largura lexicográfica, para criar um modelo de intervalo para um grafo de intervalo G.*

Palavras chave: Teoria dos Grafos, grafos de intervalo, modelo de intervalo, algoritmo.

1. Introdução

A Teoria dos Grafos é uma área fundamental da matemática que tem aplicabilidade em diversos campos, como a ciência da computação, biologia, redes de comunicação, entre outros. Grafos são capazes de representar relações entre objetos e o estudo dessas estruturas pode possibilitar a resolução de problemas complexos de maneira eficiente.

Um *grafo* G é um par ordenado $G = (V(G), E(G))$, que consiste de um conjunto $V(G)$ de vértices e um conjunto $E(G)$ de arestas de forma que $E(G) \subseteq [V(G)]^2$. Denotamos uma *aresta* que liga um vértice u a um vértice v por (u,v). Dois vértices são ditos *adjacentes* se são incidentes à mesma aresta. Um *subgrafo* de um grafo $G = (V(G), E(G))$ é um grafo $G' = (V'(G'), E'(G'))$ tal que $V' \subseteq V$ e $E' \subseteq E$. Se G' contém todas as arestas $(u,v) \in E$ com $u,v \in V'$, então G' é um *subgrafo induzido* de G, denotado por $G[V']$. Dizemos que V' *induz* ou gera G' em G. Uma *clique* C de um grafo G é um subconjunto de vértices de G em que cada par de vértices em C é adjacente.

Um grafo G é um *grafo de intervalo* se $V(G)$ pode ser representado através de um modelo de interseção de intervalos na reta real. Grafos de intervalo podem modelar contextos que requerem a construção de uma linha do tempo onde cada evento a corresponde um intervalo representando a sua duração [4]. Sendo assim, o estudo dos grafos de intervalo possui diversas aplicações práticas em problemas de escalonamento, alocação de recursos e análise de bioinformática, onde a interseção de intervalos representam um

conceito central. Um grafo de intervalo pode ser representado por uma estrutura de dados conhecida como PQ-tree [2].

Um *conjunto independente* em um grafo G é um subconjunto de vértices tal que nenhum dos vértices desse subconjunto está conectado entre si por uma aresta. O conjunto independente será *maximal* se não é possível adicionar nenhum vértice a I de forma que se mantenha a propriedade de independência. I será *máximo* se for o maior conjunto independente possível de G. A cardinalidade de um conjunto independente máximo em um grafo G é o *número de independência* de G, denotado por $\alpha(G)$.

Embora encontrar o conjunto independente máximo seja bastante útil em diversos contextos, a complexidade computacional do problema de decisão é NP-completa para grafos arbitrários [6]. Devido à complexidade inerente de se determinar um conjunto independente máximo em grafos, os pesquisadores têm focado em restringir as características desses grafos, buscando determinar $\alpha(G)$ quando G pertence a subclasses específicas, o pode ter complexidade diferente de acordo com o problema. Por exemplo, existem algoritmos que resolvem o problema em tempo linear quando G é uma árvore [9] ou quando G é um grafo de intervalo [7].

O *grafo ciclo*, conhecido como C_n, é um grafo com n vértices, onde $n \geq 3$, cujos vértices podem ser arranjados em uma sequência cíclica de tal forma que dois vértices são adjacentes se eles são consecutivos na sequência e não são adjacentes caso contrário. Um grafo G é *cordal* se todo ciclo de G com 4 ou mais vértices possui uma aresta entre vértices não consecutivos do ciclo. Seja G um grafo com n vértices. Um vértice v é *simplicial* se $N_G(v)$ for uma clique. Um *esquema de eliminação perfeita* (EEP) para G é uma sequência de vértices $\sigma = v_1, v_2, \ldots, v_n$ com a propriedade de que v_i é um vértice simplicial em $G[\{v_i, \ldots, v_n\}]$, para todo $i \in \{1, \ldots, n\}$. Grafos cordais admitem esquemas de eliminação perfeita. Quando G é um grafo cordal, também é possível determinar o maior conjunto independente em tempo linear [7]. O algoritmo que resolve esse problema é baseado no esquema de eliminação perfeita e da orientação transitiva das arestas próprios de caracterizações de grafos cordais. A partir desse algoritmo, como grafos de intervalo formam uma subclasse dos grafos cordais, temos também outra forma de resolver o problema para grafos de intervalo.

Neste trabalho, apresentamos um algoritmo que, dado um grafo de intervalo, utiliza o EEP obtida pelo algoritmo Lex BFS para obter a sequência utilizada para posicionar cada intervalo na reta real através das operações de expansão, compressão ou translação desses intervalos.

Este texto está dividido em duas seções principais: a primeira, intitulada "Conceitos Básicos" (Seção 2), aborda definições detalhadas sobre grafos, grafos de intervalo, cliques e conjuntos. A segunda seção, "Resultados" (Seção 3), apresenta um teorema da literatura referente às condições para que um grafo seja de intervalo. Em seguida, mostramos o algoritmo apresentado em [8] utilizado para obter o EEP. Finalmente, apresentamos um algoritmo polinomial que, dado um grafo de intervalo, retorna seu modelo de intervalo na reta real.

2. Conceitos Básicos

Nesta seção, apresentaremos uma fundamentação teórica necessária para o entendimento do restante do trabalho. Algumas notações e definições presentes neste artigo

foram retiradas de Bondy e Murty [1] e Cormen et al [3].

Um *caminho em um grafo* $G = (V(G), E(G))$ é uma sequência de vértices $v_1, v_2, \ldots, v_k$ tal que, para cada i (onde $1 \leq i < k$), existe uma aresta $(v_i, v_{i+1}) \in E$. Um *grafo caminho* P_n é um tipo de grafo no qual os vértices formam uma sequência linear, sem ciclos, e cada vértice está conectado a, no máximo, dois outros vértices. Um grafo é considerado *conexo* se existe um caminho entre qualquer par de vértices, caso contrário ele é chamado *desconexo*. Um *grafo ciclo* C_n é um grafo no qual os vértices formam uma sequência fechada, ou seja, é uma sequência de vértices $v_1, v_2, \ldots, v_n$ tal que cada vértice v_i (para $1 \leq i < n$) está conectado a v_{i+1}, e v_n está conectado de volta a v_1. Em um grafo ciclo, cada vértice é adjacente a exatamente dois outros vértices, formando uma estrutura circular sem arestas múltiplas nem vértices isolados.

É importante ressaltar que um grafo simples não possui *laços* (arestas que contenham vértices não-distintos) e nem *multi-arestas* (arestas idênticas), por definição. Utilizaremos apenas a notação *grafo* quando formos nos referir a um grafo simples. Considere (u, v) uma aresta de um grafo. Diremos que u e v são *adjacentes* caso estejam conectados pela aresta (u, v). Ainda, considere que (u, v) é considerado *incidente* a u e a v se a aresta (u, v) está conectada a ambos os vértices u e v. A *vizinhança aberta* (ou, simplesmente, *vizinhança*) de $v \in V(G)$ é o conjunto de todos os vértices adjacentes a esse vértice, excluindo o vértice de referência, ou seja, $N(v) = \{w \mid (v, w) \in E(G)\}$. Denominamos *grau* $d_G(v)$ ou $d(v)$ de um vértice $v \in V(G)$ o número de arestas incidentes a este vértice.

Seja $r \geq 2$. Diremos que um grafo G é *r-partido* se $V(G)$ admitir uma partição em r conjuntos independentes tal que cada aresta possui seus extremos em partições diferentes. Chamamos o grafo 2-partido de *bipartido*. Um *grafo completo* é um grafo com n vértices no qual há uma aresta entre cada par de vértices distintos. Um grafo *bipartido completo*, representado por $G[X, Y]$, é composto por dois conjuntos de vértices, X e Y, onde cada vértice em X está conectado a todos os vértices em Y e vice-versa. Se o conjunto X tem m vértices e o conjunto Y tem n vértices, o grafo é denotado como $K_{m,n}$. Um *grafo estrela* S_n é um grafo bipartido do tipo $K_{1,n}$.

Uma *clique* C é um subgrafo induzido de G que induz um grafo completo. Uma clique *maximal* é um subconjunto C de vértices tal que não existe nenhum vértice que possa ser adicionado a G de modo que se mantenha a a propriedade de completude. Uma clique será *máxima* se C é o maior conjunto possível de G. A *cardinalidade* deste conjunto é é denotada por $\omega(G)$.

Um conjunto independente de três vértices em um grafo formam uma *tripla asteroidal* se, para quaisquer dois desses vértices, existir um caminho que os conecta sem passar pela vizinhança do terceiro vértice. Um grafo que não contém nenhuma tripla asteroidal é denominado grafo *sem tripla asteroidal* (STA). A seguir, considere o grafo C_6 composto por $V(G) = \{v_1, v_2, \ldots, v_6\}$ e $E(G) = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6)\}$. Um conjunto possível que satisfaz as propriedades de tripla asteroidal é $\{v_1, v_3, v_5\}$, uma vez que é possível ter um caminho entre quaisquer 2 vértices v_i e v_j que não passe por $N(v_k)$, para todo $i, j, k \in \{v_1, v_3, v_5\}$ distinto. Veja na Figura 1 o exemplo descrito acima:

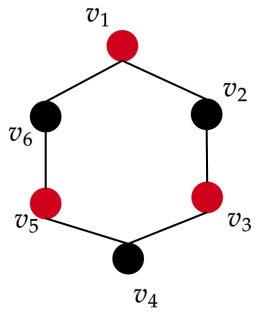

Figura 1. Uma tripla asteroidal no grafo C_6.

Um grafo G é *livre* de um grafo específico H se e somente se G não contém nenhum subgrafo induzido que seja isomorfo a H.

Um *intervalo real fechado* entre dois pontos a e b (onde $a \leq b$) é o conjunto de todos os números reais x tais que $a \leq x \leq b$. Esse intervalo inclui os pontos extremos a e b. É denotado por $[a,b]$. Ou seja:

$$[a,b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$$

Um *intervalo real aberto* entre dois pontos a e b (onde $a < b$) é o conjunto de todos os números reais x tais que $a < x < b$. Esse intervalo não inclui os pontos extremos a e b. É denotado por (a,b). Logo:

$$(a,b) = \{x \in \mathbb{R} \mid a < x < b\}$$

Dois conjuntos de vértices V_1 e V_2 são considerados sobrepostos se $V_1 \cap V_2 \neq \emptyset$, indicando que há vértices comuns a ambos os conjuntos. Analogamente, dois conjuntos de arestas E_1 e E_2 se sobrepõem se $E_1 \cap E_2 \neq \emptyset$, indicando a presença de arestas comuns a ambos os conjuntos.

Seja $\mathcal{F}$ uma família de conjuntos. O *grafo de interseção* de $\mathcal{F}$ é o grafo obtido representando-se por um vértice distinto cada conjunto de $\mathcal{F}$ e fazendo dois de tais vértices adjacentes precisamente quando os conjuntos correspondentes têm interseção não-vazia. Note que, dada uma família $\mathcal{F}$, o grafo de interseção associado é bem definido, porém o contrário não é verdade: um mesmo grafo pode ser o grafo de interseção de diferentes famílias. Se G é o grafo de interseção de uma família $\mathcal{F}$, dizemos que $\mathcal{F}$ é um *modelo* de G.

Um grafo G é um *grafo de intervalo* se G é o grafo de interseção de uma família $\mathcal{R}$ de intervalos da reta real. Em outras palavras, um grafo G é um grafo de intervalo precisamente quando existir uma correspondência entre $V(G)$ e uma família de intervalos $\mathcal{R} = \{I_v \mid v \in V(G)\}$ da reta real tal que, para todo $u,w \in V(G)$ distintos, $I_u \cap I_w \neq \emptyset$ se e somente se $(u,w) \in E(G)$. Chamamos $\mathcal{R}$ um *modelo de intervalo* de G. Um exemplo de grafo de intervalo pode ser visto na Figura 2.

Assumimos que todos os extremos de intervalo são distintos e denotamos os extremos esquerdo e direito de um intervalo I_v respectivamente por $\ell(I_v)$ e $r(I_v)$. Quando

$\ell(I_v) = r(I_v)$, diremos que I_v é *trivial*. O *tamanho* de I_v é representado por $|I_v|$ e é calculado através da seguinte fórmula : $|I_v| = r(I_v) - \ell(I_v)$. Para qualquer modelo de intervalo $\mathcal{R}$, assumiremos que $\ell(I_v) > 0$, para todo $I_v \in \mathcal{R}$. Por conveniência, dado um intervalo I_v de um modelo de intervalo e o vértice correspondente v, podemos usar indistintamente I_v ou v quando o contexto não criar ambiguidades.

Definimos, a seguir, as operações de expansão, compressão e translação de intervalos. Sejam $\mathcal{R}$ um modelo de intervalo e $I = [a,b] \in \mathcal{R}$.

- A *expansão* de I é obtida por $(\mathcal{R} \setminus I) \cup I'$ onde $I' = [a-d, b+e]$, com $d, e \in \mathbb{R} \setminus I$. Intuitivamente, essa operação aumenta o comprimento do intervalo.
- A *compressão* de I é obtida por $(\mathcal{R} \setminus I) \cup I'$ onde $I' = [a+d, b-e]$, com $d, e \in I$, $d < e$. A compressão reduz o comprimento do intervalo dentro dos limites originais.
- A *translação à direita* de I é obtida por $(\mathcal{R} \setminus I) \cup I'$ onde $I' = [a+c, b+c]$ para $c \in \mathbb{R}$, $c > a$. A *translação à esquerda* de I é obtida por $(\mathcal{R} \setminus I) \cup I'$ onde $I' = [a-c, b-c]$ para $c \in \mathbb{R}$, $c < b$. Intuitivamente, as translações consistem em deslocar a o intervalo para uma nova posição na reta real, sem alterar seu comprimento.

Considere o grafo G com $V(G) = \{v_1, v_2, \ldots, v_9\}$ e $E(G) = \{(v_1, v_3), (v_1, v_9), (v_1, v_8), (v_1, v_7), (v_1, v_4), (v_1, v_5), (v_5, v_6), (v_4, v_2)\}$. Ilustraremos a criação de um modelo de intervalo para G. Para criarmos um modelo de intervalo de G, precisamos assegurar que os intervalos correspondentes aos vértices adjacentes tenham uma interseção. Portanto, criaremos uma interseção a todos vértices adjacentes a v_1. Temos $N(v_1) = \{v_3, v_4, v_5, v_7, v_8, v_9\}$. Note que $v_2, v_6 \notin N(v_1)$, portanto, estes não irão sobrepor o intervalo referente à v_1, mas vão sobrepor os intervalos referentes a v_4 e v_5.Veja na Figura 2:

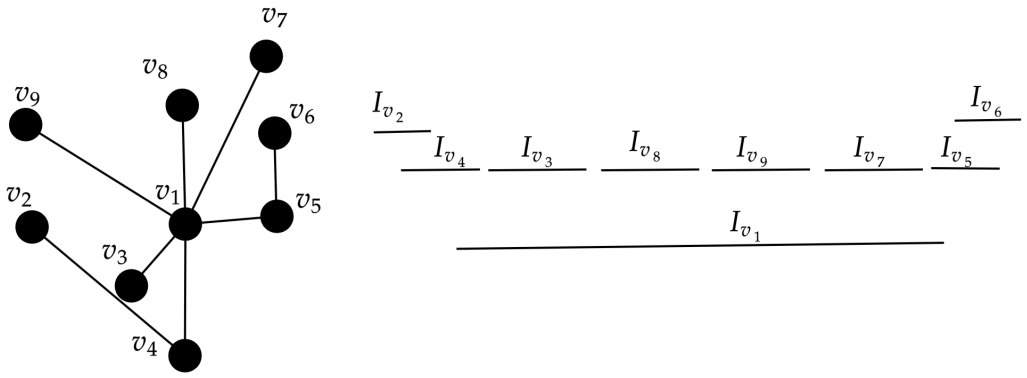

Figura 2. À esquerda, o grafo G e, à direita, um de seus modelos de intervalo.

A *complexidade computacional* é um ramo da teoria da computação que se concentra em classificar problemas computacionais de acordo com sua dificuldade inerente e relacionar essas classes entre si. Problemas de complexidade *polinomial* podem ser resolvidos por algoritmos que rodam em tempo no máximo $c \cdot n^k$, para constantes reais positivas c, k e n o tamanho da entrada. Se $k = 1$, diremos que o problema tem complexidade *linear*. Problemas de complexidade polinomial são considerados *eficientes*, pois podem ser resolvidos de forma prática na maioria dos casos. No entanto, há problemas classificados como *NP-completos*, que são considerados difíceis de serem resolvidos computacionalmente, pois ainda não se conhece um algoritmo eficiente que possam resolvê-los.

3. Resultados

Esta seção se destina à apresentação dos nossos resultados. Inicialmente, considere o seguinte teorema, demonstrado por Lekkeikerker e Boland [10].

Teorema 1 *[10] Seja G um grafo. G é um grafo de intervalo se e somente se G não possui ciclo induzido com 4 ou mais vértices e G não possui tripla asteroidal.*

Algoritmo 1: ÉGRAFODEINTERVALO(G)

Entrada: Grafo $G = (V(G), E(G))$
Saída: `SIM`, se G é grafo de intervalo, ou `NÃO`, caso contrário

1 **para cada** *ciclo induzido* $C \subseteq V(G)$ *;* `// Checa se G é cordal.`
2 **faça**
3 | **se** $|C| \geq 4$ **então**
4 | | **retorna** `NÃO`
5 | **fim**
6 **fim**
7 **para cada** *tripla de vertices* $u, v, w \in V(G)$*;* `// Checa se G não possui triplas asteroidais.`
8 **faça**
9 | **se** *existe um caminho entre u e v em* $G - N(w)$
10 | ***e*** *existe um caminho entre u e w em* $G - N(v)$
11 | ***e*** *existe um caminho entre v e w em* $G - N(u)$
12 | **então**
13 | | **retorna** `NÃO`
14 | **fim**
15 **fim**
16 **retorna** `SIM`

Dado um grafo G qualquer, utilizando a caracterização apresentada no Teorema 1 podemos decidir se G é um grafo de intervalo por meio do Algoritmo 1.

Considere que o grafo G de entrada para o Algoritmo 1 possua n vértices e m arestas. As linhas 1 a 3 checam se G possui ciclos induzidos com 4 ou mais vértices. Tal checagem pode ser executada com uma modificação do algoritmo de busca em profundidade, que executa em em $O(n+m)$.

As linhas 4 a 6 testam se G possui uma tripla asteroidal. Dado que o número de vértices de G é n, temos um total de $\binom{n}{3}$ triplas possíveis. Isso resulta em uma complexidade de $O(n^3)$.

Por fim, na linha 7 o algoritmo simplesmente retorna que de fato G é um grafo de intervalo. A complexidade total do Algoritmo 1 é então da ordem de $O(n^3)$, que é polinomial.

Uma vez obtido o retorno `SIM` do Algoritmo 1, prosseguimos para a construção do modelo de intervalo. Para tal, utilizaremos busca em largura lexicográfica [8], discutida a seguir.

O *Algoritmo de Busca em Largura Lexicográfica (Lex BFS)* atribui rótulos aos

vértices de um grafo de maneira ordenada, priorizando os vértices que possuem adjacências entre si. Inicialmente, o algoritmo rotula os vértices com $\emptyset$, indicando que nenhum vértice tem prioridade de escolha inicial. Em seguida, o algoritmo processa os vértices em ordem decrescente, começando com um valor i igual ao número de vértices do grafo e, a cada iteração, reduz i até 1.

Sendo assim, ao final do algoritmo, obtemos uma ordem linear, onde cada vértice recebe um número a partir do maior valor disponível. Em cada iteração, o algoritmo seleciona o vértice não numerado que possui o maior rótulo acumulado. Ao atribuir ao vértice v o número i, indicamos a posição desse vértice na ordem final. O algoritmo, em seguida, atualiza os rótulos dos vértices adjacentes a v, incrementando o rótulo de cada vizinho não numerado com o valor de i, permitindo que os vértices mais próximos de v se tornem candidatos prioritários a serem rotulados nas próximas iterações.

Ao final do algoritmo, obtemos uma ordenação dos vértices de G correspondente a um esquema de eliminação perfeita. O algoritmo pode ser descrito conforme as seguintes etapas:

Algoritmo 2: LEXBFS(G)

Entrada: Grafo $G = (V(G), E(G))$ com n vértices
Saída: Uma ordem σ dos vértices de G

1 **para cada** $v \in V(G)$ **faça**
2 | $r(v) \leftarrow \emptyset$; // Inicializa o rótulo de cada vértice como vazio
3 **fim**
4 **para** $i \leftarrow n$ **até** 1 **faça**
5 | **Selecione:** Escolha um vértice v nao numerado com o maior rótulo $r(v)$;
6 | $\sigma(v) \leftarrow i$; // Atribui o número i ao vértice v
7 | **Atualize: para cada** $w \in N(v)$ *não numerado* **faça**
8 | | $r(w) \leftarrow r(w) \cup \{i\}$; // Adiciona i ao rótulo do vértice w
9 | **fim**
10 **fim**
11 **retorna** σ ; // Retorna a ordem dos vértices

Considere novamente que o grafo G de entrada para o Algoritmo 2 possua n vértices. A linha 1 é executada para cada vértice de G, o que resulta em um tempo de execução $O(n)$. A linha 2 é repetida de n até 1, o que totaliza $O(n)$ passos. A linha 3 requer encontrar um vértice não numerado com maior rótulo. Podemos encontrar tal vértice simplesmente fazendo uma busca linear em cada vértice, em tempo $O(n)$. A linha 4 é simplesmente uma atribuição, que usa tempo $O(1)$. A linha 5 atualiza rótulos dos vizinhos, logo requer $O(\Delta)$ passos, onde Δ é o grau máximo de G. Assim, o laço de repetição do bloco das linhas 2 a 5 tem complexidade $O(n^2)$. O passo 5 é apenas um retorno, que executa em tempo constante $O(1)$. A complexidade total então é a soma das complexidades de todos os passos, sendo da ordem de $O(n^2)$, que é polinomial.

A partir do Lex BFS, podemos obter um esquema de eliminação perfeita, uma vez que todo grafo de intervalo é um grafo cordal. Veja na Figura 3 o EEP referente ao grafo S_7.

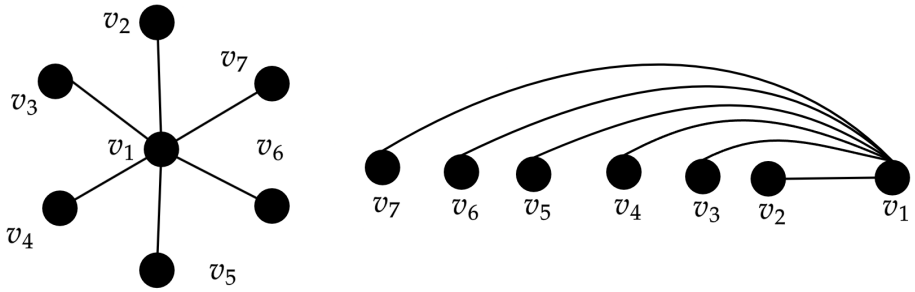

Figura 3. À esquerda, o grafo S_7 e, à direita, seu EEP.

A partir da ordenação obtida do Lex BFS, dado um grafo G de intervalo, utilizaremos esta ordem para construir um modelo de intervalo de G. Veja abaixo o algoritmo de construção de um modelo de intervalo.

Algoritmo 3: GERAMODELOINTERVALO(G)

Entrada: Grafo de intervalo $G = (V(G), E(G))$
Saída: Modelo de intervalo $\mathcal{R}$ de G

1 Seja $\sigma = (v_1, v_2, \ldots, v_n)$ um EEP retornado pelo algoritmo LEXBFS(G);
2 $I_n \leftarrow [0,1]$;
3 **para** $i \leftarrow n-1$ **até** 1 **faça**
4 $\quad G' \leftarrow G[v_{i+1}, \ldots, v_n]$;
5 $\quad \mathscr{I}_A \leftarrow \{j \in \{i+1, \ldots, n\} \mid v_j \in N_{G'}(v_i)\}$;
6 $\quad \mathscr{I}_{\overline{A}} \leftarrow \{i+1, \ldots, n\} \setminus \mathscr{I}_A$;
7 $\quad I_C \leftarrow \bigcap_{j \in \mathscr{I}_A} I_j$;
8 $\quad I_F \leftarrow I_C \setminus \left(\bigcup_{j \in \mathscr{I}_{\overline{A}}} I_j \right)$;
9 $\quad$ **se** $I_F = \emptyset$ **então**
10 $\quad\quad$ Sem criar novas sobreposições, expanda I_j, para todo $j \in \mathscr{I}_A$ ou comprima $I_{j'}$, para todo $j' \in \mathscr{I}_{\overline{A}}$ até $I_F \neq \emptyset$;
11 $\quad$ **fim**
12 $\quad$ Posicione I_i em alguma porção contígua mais à esquerda de I_F.
13 **fim**
14 **retorna** $\mathcal{R} = \{I_1, I_2, \ldots, I_n\}$; `// Retorna o modelo de intervalo`

Teorema 2 *Seja G um grafo de intervalo. O Algoritmo 3 quando executado sobre G produz corretamente um modelo de intervalo $\mathcal{R}$ para G.*

Seja $\sigma = (v_1, v_2, \ldots, v_n)$ um EEP retornado pelo algoritmo LEXBFS(G). O Algoritmo 3 percorre a sequência σ de v_n até v_1, para determinar o intervalo I_i correspondente ao vértice v_i. O passo base ocorre na Linha 2, onde I_n recebe $[0,1]$, o que é um modelo de intervalo válido para $G[v_n]$. Por hipótese de indução, considere que $\mathcal{R}' = \{I_{i+1}, \ldots, I_n\}$ seja um modelo de intervalo para $G' = G[v_{i+1}, \ldots, v_n]$. Vamos mostrar que $R = \{I_i, \ldots, I_n\}$ é um modelo de intervalo para $G[v_i, \ldots, v_n]$.

Na Linha 5 definimos o conjunto $\mathscr{I}_A$ de índices j tais que v_j é adjacente a v_i em G' e na Linha 6 definimos o conjunto $\mathscr{I}_{\overline{A}}$ de índices j tais que v_j não é adjacente a v_i em G'. A

ideia é criar interseção entre I_i e I_j para todo $j \in \mathscr{I}_A$ e não criar interseção entre I_i e I_j para todo $j \in \mathscr{I}_{\overline{A}}$. Para tal, na Linha 7 é definido I_C pela interseção dos intervalos dos vizinhos de v_i como um intervalo candidato a posicionar I_i. Sabemos que isso é possível, já que no EEP σ, o conjunto de vizinhos de v_i em G' forma uma clique. Na Linha 8, obtemos um intervalo final I_F a partir do intervalo candidato I_C, onde são removidas as porções dos intervalos dos vértices que v_i não deve intersectar. Observe que I_F pode ser vazio. Assim, na Linha 10 ajustamos o modelo expandindo os intervalos dos vizinhos de v_i para que I_F tenha possibilidade de intersectar I_i, cujo posicionamento é feito na Linha 12. Há uma expansão possível à esquerda ou à direita, graças a ausência de ciclos induzidos com 4 ou mais vértices e triplas asteroidais. Observe também que I_F pode ser composto por um ou mais intervalos disjuntos, por isso na Linha 12 escolhemos uma porção contígua de I_F para posicionar I_i. Desta maneira, como I_C garante as interseções necessárias entre vizinhos de v_i e I_F garante que não hajam interseções entre não vizinhos de v_i, concluímos o desejado.

Dado o Algoritmo 3 acima, Veja um exemplo na Figura 4 de um modelo de intervalo criado para o grafo estrela S_7.

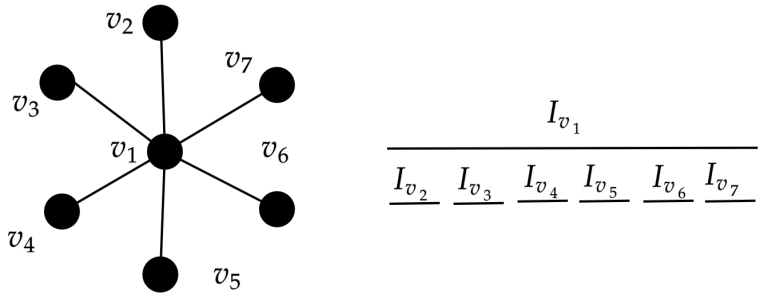

Figura 4. À esquerda, o grafo S_7 e, à direita, uma possível representação de modelo de intervalo.

Teorema 3 *Seja G um grafo de intervalo. O Algoritmo 3 quando executado sobre G produz em tempo polinomial um modelo de intervalo $\mathcal{R}$ para G.*

A Linha 1 chama o Algoritmo 2 como subrotina, que usa tempo $O(n^2)$. A Linha 2 realiza apenas uma atribuição dos extremos de I_n, o que requer tempo constante. O laço de repetição da Linha 3 é executado para $n-1$ vértices. Vamos analisar a complexidade de cada uma dessas iterações. As atribuições das Linhas 4 a 6 dependem de $n-i$ vértices, tendo complexidade $O(n)$. As Linhas 7 e 8, embora calculem uniões e interseções de intervalos também requerem tempo $O(n)$ uma vez que precisamos apenas dos pontos extremos de cada intervalo. Para as Linhas 9 a 11 devem ser expandidos $\Theta(|\mathscr{I}_A|)$ intervalos, uma vez que temos que produzir um intervalo I_F com a interseção de todos os intervalos vizinhos de v_i não vazia para posicionar I_i. Para expandir cada um deles sem criar novas sobreposições precisa-se checar as extremidades de cada outro intervalo, o que requer $O(n)$ operações, logo as Linhas 9 a 11 requerem tempo $O(n^2)$. Após isso, a Linha 12 determina as extremidades de I_i a fim de posicioná-lo em I_F o que requer tempo constante. Em suma, cada bloco das Linhas 4 a 12 gasta tempo $O(n^3)$, repetidos $O(n)$ vezes temos um tempo total de execução do Algoritmo 3 da ordem de $O(n^4)$.

4. Considerações Finais

Neste artigo, desenvolvemos um algoritmo que, dado um grafo de intervalo, utiliza o EEP obtida pelo algoritmo Lex BFS para obter a sequência utilizada para posicionar cada intervalo na reta real através das operações de expansão, compressão ou translação desses intervalos.

Além disso, em outro trabalho, implementamos um algoritmo, proposto por Feofiloff [5], em linguagem *python* que tem como objetivo identificar e extrair intervalos disjuntos a partir de uma lista de intervalos.Para isso, o algoritmo cria uma lista vazia *SCD*, ordena os intervalos em ordem crescente e adiciona, inicialmente, sempre o menor intervalo à lista. Após isso, o algoritmo sempre verifica se o último intervalo adicionado possui interseção com o próximo intervalo a ser adicionado, o que equivale a descobrir o conjunto independente máximo em um grafo de intervalo, uma vez que todos os intervalos da lista *SCD* são disjuntos. A implementação pode ser encontrada em: https://github.com/AnaCarolynePds/Descobrindo-alpha-G-em-um-grafo-de-intervalo.

Como trabalho futuro, sugerimos a implementação de algoritmos para outros problemas em grafos, como encontrar a clique máxima e uma k-coloração para um grafo de intervalo.

Referências

[1] J. A. Bondy and U. S. R. Murty. *Graph theory*. Springer Publishing Company, Incorporated, 2008.

[2] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of computer and system sciences*, 13(3):335–379, 1976.

[3] T. H. Cormen. Algorithms unlocked. *Editora Campus*, 2013.

[4] F. de Souza Oliveira. *Sobre Ordens e Grafos de Intervalo*. PhD thesis, Universidade Federal do Rio de Janeiro, 2011.

[5] P. Feofiloff. Minicurso de análise de algoritmos. *Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo*, 2017. Disponível em: `https://www.ime.usp.br/~pf/livrinho-AA/`.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, USA, 1979.

[7] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

[8] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[9] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education India, 2006.

[10] C. Lekkeikerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 1962.