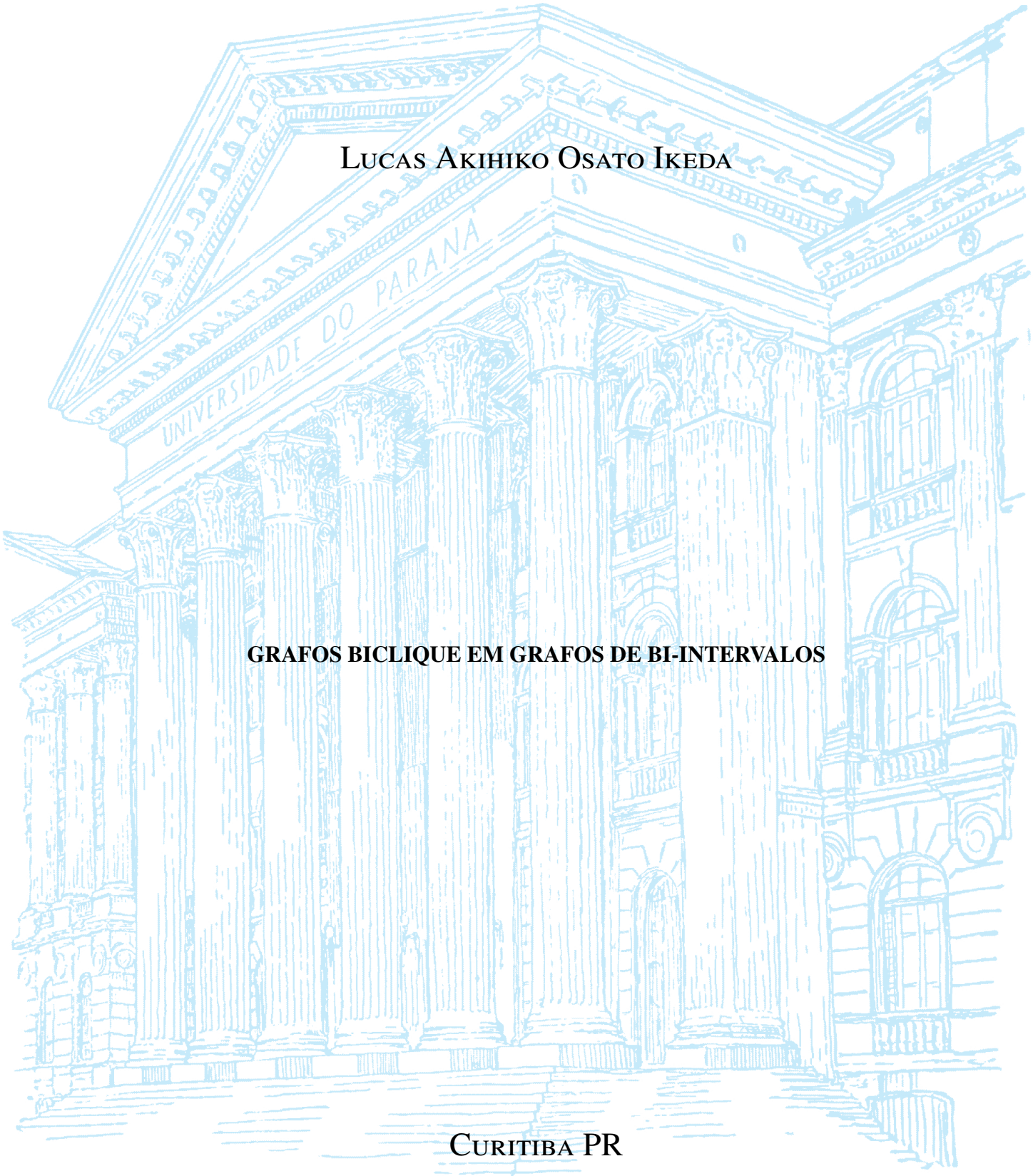
Universidade Federal do Paraná

Lucas Akihiko Osato Ikeda



GRAFOS BICLIQUE EM GRAFOS DE BI-INTERVALOS

Curitiba PR
2018

Lucas Akihiko Osato Ikeda

GRAFOS BICLIQUE EM GRAFOS DE BI-INTERVALOS

Monografia apresentada ao curso de Ciência da Computação do Departamento de Informática, da Universidade Federal do Paraná, como requisito para obter o grau de Bacharel em Ciência da Computação.

Área de concentração: *Ciência da Computação*.

Orientador: André Guedes.

Curitiba PR
2018

Resumo

A partir de um número finito de intervalos numa reta real, uma bipartição de intervalos é a repartição destes intervalos em duas partes. Um bigrafo de intervalos é um grafo de intervalos em um modelo bipartido no qual cada vértice corresponde a um intervalo e dois vértices são vizinhos se e somente se os intervalos correspondentes se intersectam e não pertencem à mesma parte. Neste trabalho é estudado as propriedades dessa subclasse de grafos e proposto um algoritmo de busca extensiva, que percorre todos os vértices do grafo, para encontrar todas as bicliques maximais de um bigrafo de intervalos com uma complexidade de tempo de $O(N^3)$.

Palavras-chave: grafos, grafos biclique, grafos de bi-intervalos, algoritmo.

Abstract

From a finite number of intervals on a line, a bipartition of intervals is a division of those intervals in two parts. An interval bigraph is an intersection graph of a bipartite model wherein to each vertex corresponds an interval and two vertices are neighbours if and only if both corresponding intervals intersect and do not belong to the same part. In this work it is studied the proprieties of this graph subclass and proposed a search algorithm to find all maximal bicliques in a interval bigraph with a time complexity of $O(N^3)$.

Keywords: graphs, biclique graphs, bi-intervals graphs, algorithm.

Lista de Figuras

Lista de Símbolos

O	Limitante assintótico superior
Ω	Limitante assintótico inferior
Θ	Limitante assintótico superior e inferior
K_n	Grafo completo de n vértices
$K_{n,m}$	Grafo bipartido completo com partes de n e m vértices, respectivamente
$\mathcal{B}_G$	Conjunto das bicliques maximais do grafo G

Sumário

1 Introdução

Este documento contém a descrição de estudos das propriedades de bicliques maximais de bi-intervalos e como encontrar grafos-biclique em bigrafos de intervalos, inspirado nos estudos realizados por da Cruz (2018) e no trabalho de Groshaus e Szwarcfiter (2010).

A caracterização e estudos de uma classe específica, como grafos-biclique em bigrafos de intervalos nos permite uma melhor compreensão de uma classe mais genérica, como a superclasse de grafos-biclique de todos os grafos. E assim, contribuir para um maior entendimento dentro da teoria de grafos em geral.

Algumas das aplicações de bicliques maximais são estudas por Bu et al. (2010) e Li et al. (2007) no ramo da biologia em grafos de interação proteína-proteína. Kumar et al. (1999) apresentaram uma outra aplicação do problema para encontrar comunidades web em redes sociais. No ramo da medicina aplicações de enumeração de bicliques maximais foram apresentadas por Nagarajan e Kingsford (2012), e Atluri et al. (1988) apresentam uma aplicação no ramo da genética. Haemers (2001) apresenta uma aplicação do problema no ramo da teoria da informação.

1.1 Caracterização do problema

O problema em questão pode ser solucionado com um algoritmo de busca por todas as bicliques de um grafo, que são os subgrafos bipartidos completos maximais, ou seja, dado um grafo G com duas partes $\{A, B\}$, devemos encontrar todos os pares de conjuntos de vértices $V \subseteq A$ e $V' \subseteq B$ onde todo vértice $v \in V$ é vizinho de todos os vértices $v' \in V'$ e vice-versa.

Para resolver o problema proposto, propomos um algoritmo que recebe como entrada os vértices de um bigrafo de intervalos e as duas partes pertencentes a este grafo. E, a partir disso, construir uma estratégia que busca por intersecções para todos os possíveis conjuntos de vértices de uma das partes do grafo e verificar se as intersecções formam um conjunto V de vértices maximais, ou seja, não existe nenhum outro vértice que intersecta com os vértices pertencentes a V.

A busca por intersecções será feita com base nas definições a serem descritas no próximo capítulo, onde a ideia será procurar todos os possíveis pares de eventos de começo e fim de cada intervalo de uma das partes do grafo e encontrar todos os intervalos da outra parte que intersectam com esse grupo. Como listar todos os pares de eventos de início e fim de um conjunto de N vértices tem um custo de tempo $O(N^2)$, pois cada vértice possui um par de eventos, e depois deve-se realizar uma verificação linear $O(N)$ para garantir que este par de eventos resulta numa biclique maximal, temos um algoritmo com custo $O(N^3)$.

Um bigrafo de intervalos é um modelo bipartido de intervalos, onde cada intervalo pertence a um de dois possíveis conjuntos, os quais denominamos partes. Para esse problema será considerado como um vértice cada intervalo onde cada extremo deste intervalo pode ser interpretado como seu respectivo começo ou fim, e uma aresta existe se e somente se o intervalo de um vértice a intersecta o intervalo de um vértice b, resultando em uma aresta $\{a, b\}$. Para

simplificar o problema utilizaremos apenas o conceito de intervalos fechados, e com apenas um evento, seja começo ou fim, por unidade de tempo.

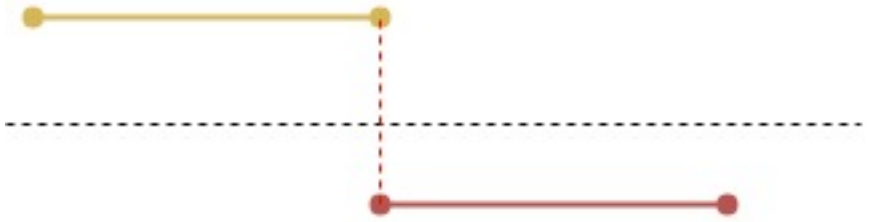


Figura 1.1: Exemplo de intervalos fechados, intervalos intersectam

Portanto, dentro do conjunto de vértices do bigrafo de intervalos podemos classificar como biclique todo subconjunto de $S' \subseteq \{A \cup B\}$ onde todos os vértices do subconjunto S' que pertencem a parte A intersectam com todos os vértices do subconjunto S' que pertencem a parte B, e como cada intersecção de intervalos corresponde a uma aresta entre seus respectivos vértices, temos assim um grafo bipartido completo, ou seja, uma biclique.

Resumindo, temos que um bigrafo de intervalos é um grafo que contém dois conjuntos de vértices distintos, onde cada vértice representa um intervalo e portanto possuem um evento de começo e outro de fim com valor único dentro do grafo. Uma intersecção ocorre quando dois vértices possuem um intervalo de tempo onde já aconteceu o evento de começo de ambos mas ainda não ocorreu o evento de fim para nenhum, caracterizando portanto uma intersecção e, consequentemente, aresta entre eles. Caso todos os vértices de um subconjunto de vértices da parte A do bigrafo de intervalos possua intersecção com todos os vértices de um subconjunto de vértices da parte B desse mesmo grafo, temos uma biclique.

2 Definições

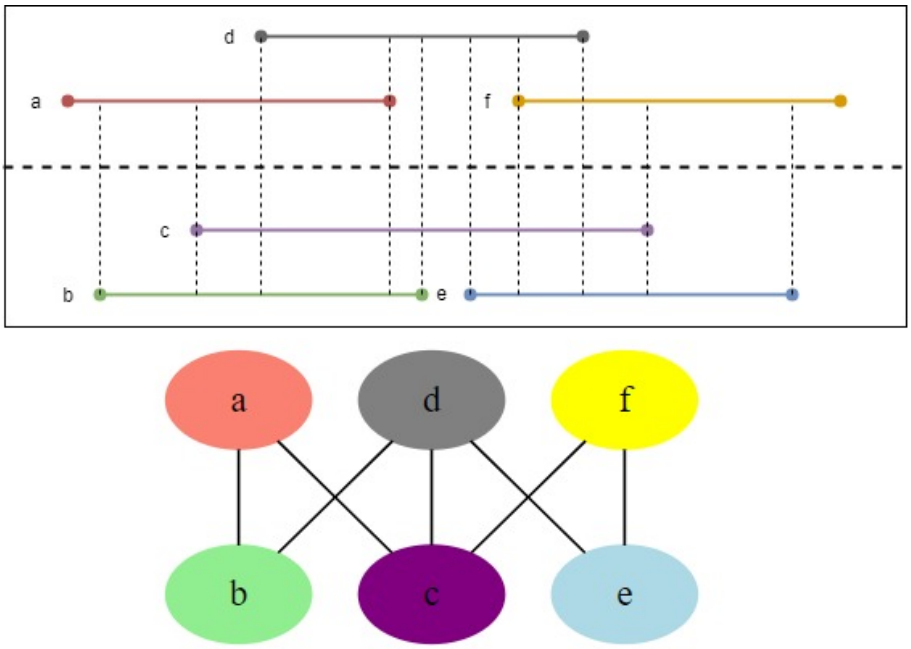


Figura 2.1: Grafo gerado a partir das intersecções dos intervalos

As definições a seguir foram retiradas do trabalho de da Cruz (2018).

Definição 1: Um intervalo é um conjunto que contém cada número real entre dois extremos indicados.

Neste trabalho será abordado um intervalo como um conjunto onde os extremos serão pontos numa reta denominados de eventos, e o evento mais a esquerda na reta é o evento de início do intervalo, e o mais a direita o evento de fim, dado uma reta real com orientação a direita.

Com isso temos a garantia de que para todos os intervalos, seu evento de início ocorre antes de seu evento de fim.

Definição 2: Um modelo bipartido de intervalos M é uma tripla (A, B, E) onde A e B são conjuntos finitos disjuntos de intervalos entre si, as partes, e E é uma ordenação total do conjunto de eventos dos intervalos de A e B e para todo intervalo, seu evento de início ocorre antes de seu evento de fim. Denotamos as componentes A, B e E do modelo M por, respectivamente, $A(M)$, $B(M)$ e $E(M)$.

Para as seguintes definições, considere um modelo bipartido de intervalos M qualquer.

Definição 3: Os elementos do conjunto $\{0, 1\} \times (A(M) \cup B(M))$ são chamados de eventos. Eventos que pertencem a $\{0\} \times (A(M) \cup B(M))$ são chamados de eventos de início e eventos que pertencem a $\{1\} \times (A(M) \cup B(M))$ são chamados de eventos de fim.

Para um intervalo $d \in (A(M) \cup B(M))$, denotamos por s_d o evento de início de d e por f_d o evento de fim de d, isto é, $s_d = (0, d)$ e $f_d = (1, d)$.

Podemos então, inferir que uma intersecção entre dois intervalos $\{a, b\}$ ocorre se e somente se o começo do vértice a ocorre antes do fim do vértice b, ou seja, $s_a < f_b$, e o fim desse mesmo vértice a também ocorre depois do começo do vértice b, $f_a > s_b$. Com isso existe uma garantia que há uma fatia de tempo compartilhada entre ambos os vértices, a e b.

Figura 2.2: Exemplo de intervalos que intersectam

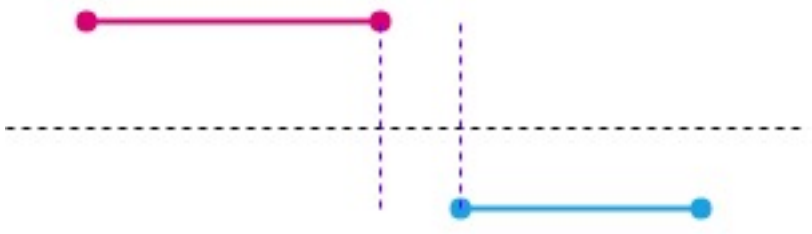


Figura 2.3: Exemplo de intervalos que não intersectam

Como existem duas partes no bigrafo de intervalos e todos os eventos ocorrem em ordem, podemos assumir, usando a mesma regra que foi vista para intersecção de intervalos singulares, que todos os vértices de uma parte A intersectam todos os vértices de uma parte B, se, e somente se, o começo do último vértice a começar na parte A, ocorre antes do fim do primeiro vértice a terminar na parte B, e o fim do primeiro vértice a terminar na parte A, ocorre antes do começo do último vértice a começar na parte B.

Definição 4: Sejam dois eventos e e e' de $\{0,1\} \times (A(M) \cup B(M))$, denotamos o fato de que e ocorre antes de e' na ordenação $E(M)$ por $e <_{E(M)} e'$ e de que e ocorre depois de e' em $E(M)$ por e $>_{E(M)} e'$. Denotamos por $e \leq_{E(M)} e'$ o fato de e ocorrer antes de e' ou ambos e e e' se referirem ao mesmo evento. De maneira similar, denotamos por $e \geq_{E(M)} e'$ o fato de e ocorrer depois de e' ou ambos e e e' se referirem ao mesmo evento.

Definição 5: Seja um grafo não-direcionado $G = (V, E)$. Denotamos por $V(G)$ o conjunto de vértices V de G e denotamos por $E(G)$ o conjunto de arestas E de G.

Definição 6: Dizemos que G é bigrafo de intervalos se G admite um modelo bipartido de intervalos como modelo de intersecção.

Definição 7: Sejam um grafo G e o conjunto $S \subseteq V(G)$. Denotamos por $G[S]$ o subgrafo de G induzido pelo subconjunto de vértices S, isto é, o subgrafo de G composto pelos vértices em S e todas as arestas em G que conectam tais vértices.

Definição 8: Uma biclique de um grafo G é um conjunto $S \subseteq V(G)$ onde $G[S]$ é bipartido completo, isto é, o conjunto S pode ser particionado em dois conjuntos independentes X e Y tais que $\{x, y\} \in E(G[S])$ para todo $x \in X$ e $y \in Y$.

Definição 9: Dizemos que a biclique S é maximal se não existe $v \in (V(G) \setminus S)$ tal que $S \cup \{v\}$ é biclique.

Definição 10: Denotamos o conjunto de todas bicliques maximais de G por B_G.

Definição 11: Chamamos de grafo-biclique de G o grafo de intersecção das bicliques maximais de G, isto é, um grafo H tal que existe uma bijeção $\pi : B_G \to V(H)$ onde $\{\pi(S_1), \pi(S_2)\} \in E(H)$ se, e somente se, a intersecção $S1 \cap S2$ não for vazia, para todo $S1, S2 \in B_G$. Denotamos o grafo-biclique de G por $K\mathcal{B}(G)$.

Embora a motivação deste trabalho seja o estudo da classe dos grafos-biclique de bigrafos de intervalos, estudar diretamente grafos de uma classe com caracterizações pouco triviais torna mais difícil a análise de suas bicliques maximais e, portanto, sobre seu grafo-biclique. Como alternativa, definimos as classes dos grafos-biclique de modelos bipartidos de intervalos, que possuem a vantagem de dar mais informações de natureza topológica do que estudar seus respectivos grafos de intersecção.

As bicliques dos modelos são conjuntos de intervalos onde todo intervalo possui intersecção não-vazia com todos os intervalos da parte contrária. Da mesma forma que nos grafos, podemos definir as bicliques maximais e, por fim, os grafos-biclique dos modelos.

Definição 12: Uma biclique de um modelo bipartido de intervalos M é um subconjunto de intervalos S do modelo tal que todo intervalo de $S \in A(M)$ possui intersecção não-vazia em M com todos os intervalos de $S \in B(M)$.

Definição 13: Dizemos que uma biclique S de um modelo bipartido de intervalos M é maximal se não existe um intervalo d que não pertence a S tal que $S \cup \{d\}$ é biclique.

Definição 14: O conjunto de todas as bicliques maximais de um modelo bipartido de intervalos M é denotado por B_M.

3 Algoritmo

3.1 Ideia

Podemos então criar uma estratégia para um algoritmo de busca que tenha como objetivo encontrar todas as bicliques maximais em um bigrafo de intervalos percorrendo todos os vértices pertencentes a este grafo.

Esta estratégia será baseada nos seguintes conceitos. Dado dois vértices $a(i, f)$ e $b(i, f)$, onde i corresponde ao tempo de início do vértice, e f o tempo em que ele finaliza, eles intersectam se e somente se $a(i) < b(f)$ e $a(f) > b(i)$.

Logo, quando temos dois conjuntos $A(i, f)$ e $B(i', f')$, onde $\{i, i'\}$ representam os inícios dos últimos vértices a começarem em seus respectivos conjuntos, e o $\{f, f'\}$ os fins dos primeiros vértices a terminarem, temos que todos os vértices desses conjuntos se intersectam somente no caso de $A(i) < B(f')$ e $A(f) > B(i')$.

Para garantir que uma biclique é maximal dentro deste contexto, podemos utilizar um algoritmo de busca em que dado um conjunto não vazio pertencente à parte $A(i, f)$, ele devolva todos os vértices pertencentes a parte B que intersectam todos os vértices de $A(i, f)$. Caso o conjunto B seja não vazio, o passo seguinte será executar o mesmo algoritmo, agora para $B(i', f')$, e se for obtido um conjunto igual ao $A(i, f)$ inicial, temos então que essa biclique é maximal, visto que não pode ser adicionado qualquer vértice novo que intersecta todos os demais nesses conjuntos. Caso o algoritmo devolva um conjunto $A'(i'', f'')$ com um número maior de vértices que o conjunto inicial $A(i, f)$, temos então que $A(i, f)$ não era maximal, então repetimos o processo até não ser possível mais inserir vértices nesses conjuntos.

3.2 Pseudocódigo

O algoritmo se baseia então no seguinte princípio: dada uma sequência $T(e_1, e_2, e_3, \ldots, e_{2n})$ de eventos, onde n é o número de intervalos, e e_i é $v(i)$ ou $v(f)$ de um intervalo em um bigrafo de intervalos, então para cada par (i, j) que pertencem a $[1..2n]$:

Algoritmo 1 busca-bicliques:

```
1: for all i, f ∈ E_A do
2:     i', f' ← acha-interseccoes(i, f, B)
3:     i'', f'' ← acha-interseccoes(i', f', A)
4:     if i = i'' E f = f'' then
5:         bicliques ← conjunto(i,f)
6:     end if
7: end for
```

Este primeiro algoritmo utiliza o conceito de que uma biclique é maximal quando dado dois conjuntos $\{A, B\}$ todos os vértices que fazem intersecção com os vértices do conjunto A pertencem ao conjunto B, e mesmo ocorre para os vértices do conjunto B, ou seja, todos fazem intersecção com os vértices do conjunto A.

Portanto, ao procurar por todos os vértices que fazem intersecção com um conjunto de vértices qualquer pertencentes a uma das partes do bigrafo de intervalos, se todos os vértices do conjunto resultante fizerem intersecção com todos os vértices do conjunto original e nenhum a mais, temos uma biclique máximal.

Dessa maneira, ao executar este método por todos os possíveis pares de eventos de início e fim de uma parte A de um bigrafo de intervalos G, temos consequentemente todos os possíveis subconjuntos de vértices $a \subseteq A$, e portanto obtemos como resultado todas as bicliques maximais pertencentes a G.

Algoritmo 2 acha-interseccoes(i, f, cjt):

```
    i' ← 0
 2: f' ← ∞
    for all v ∈ cjt do
 4:   if inicio(v) < f E fim(v) > i then
        if i' < inicio(v) then
 6:       i' ← inicio(v)
        end if
 8:     if f' > fim(v) then
          f' ← fim(v)
10:     end if
      end if
12: end for
    return i', f'
```

Para encontrarmos as intersecções, partimos da ideia de que é necessário saber apenas o primeiro fim e o último começo do conjunto de vértices, e a partir dessa informação encontrar todos os vértices da parte contrária que fazem intersecção com esse par de eventos.

Algoritmo 3 conjunto(i, f):

```
    for all v ∈ {A ∪ B} do
      if inicio(v) < f E fim(v) > i then
 3:     vertices ← inicio(v)
      end if
    end for
 6: return vertices
```

Este terceiro pseudocódigo representa um algoritmo que devolve todos os vértices que fazem intersecção com um determinado par de eventos de começo e fim.

4 Análise do algoritmo

O número máximo de bicliques de um grafo desta classe é quadrático, ou seja, $O(N^2)$, pois cada biclique tem um par $\{i, f\}$ de início e fim associado, e cada par está associado a uma biclique, ou nenhuma. Portanto o número de bicliques é menor ou igual ao número de pares $\{i, f\}$. Temos que o número de intervalos na parte a ser processada é sempre menor que o total de intervalos do bigrafo de intervalos. Portanto, se o grafo tem N intervalos, temos que a parte terá no máximo $N - 1$ intervalos, o que gera $(N - 1)^2$ pares, ou seja, o número de pares $\{i, f\}$ e, consequentemente o número de bicliques, é $O(N^2)$.

Para emparelhar todos os possíveis pares de começo e fim dos intervalos da menor parte, temos que no pior caso essa parte tem tamanho $\frac{N}{2}$, e o custo então é de $\left(\frac{N}{2}\right)^2$, ou seja, $\frac{N^2}{4}$ que é $O(N^2)$.

Para cada par de começo e fim é necessário varrer a outra parte, que no caso da parte contrária ter $\frac{N}{2}$ também possui $\frac{N}{2}$ vértices, em busca de todos os intervalos que fazem intersecção, o que retorna o par $\{i, f\}$ referente ao conjunto C que intersecta a primeira parte. Deve-se garantir que a biclique é maximal, portanto varre-se a primeira parte em busca de todos os vértices que intersectam com o conjunto C, o que custa $\frac{N}{2}$. Portanto o custo total de encontrar o conjunto C e depois averiguar que é uma biclique maximal é $\frac{N}{2} + \frac{N}{2}$, ou seja, N, o que nos leva ao resultado de um custo de $\frac{N^2}{4} \times N$, o que implica em $\frac{N^3}{4}$, ou seja, $O(N^3)$.

No código isso é visto na forma de dois *loops*:

```
for key1, value1 in start_end[0].items():
   if key1 in min_gpo[0]:
      for key2, value2 in start_end[1].items():
```

E cada iteração chama no máximo duas vezes a função intersection:

```
i, f = intersection(value1, value2, min_gpo[1])
if (i != None) & (f != None):
   i, f = intersection(i, f, min_gpo[0])
```

Que por sua vez tem um *loop*, que vai rodar no máximo N vezes, como visto anteriormente:

```
def intersection(i, f, group):
   ii = None
   ff = None
   for v in group:
```

5 Conclusão

Para este projeto foi proposto um estudo, juntamente com um algoritmo de busca, sobre bicliques maximais dentro da classe de bigrafos de intervalos. Foram demonstradas as características importantes dessa classe de grafos que permitem a busca por bicliques maximais, e como utiliza-las para fazer um algoritmo com custo $O(N^3)$.

Embora o algoritmo faça o que foi proposto, não há garantia dele ser ótimo, visto que o número de bicliques de um bigrafo de intervalos é $O(N^2)$ e o algoritmo é $O(N^3)$. O algoritmo depende também de uma propriedade que ocorre apenas em bi-intervalos: a de que todo evento de início de um intervalo ocorre antes de seu respectivo evento de fim na ordem de eventos.

Referências

Atluri, G. et al. (1988). The meaning of synchronous distributed algorithms run on asynchronous distributed systems. Em *3rd International Symposium on Computer and Information Sciences*, páginas 307–316, Izmir - Turkey.

Bu, D. et al. (2010). Discovering coherent value bicliques in genetic interaction data.

da Cruz, E. P. (2018). Grafos biclique de grafos de bi-intervalos e bi-arco-circulares. Dissertação de Mestrado, Universidade Federal do Paraná.

Groshaus, M. e Szwarcfiter, J. L. (2010). Biclique graphs and biclique matrices. *Journal of Graph Theory*, 63(1):1–16.

Haemers, W. H. (2001). Bicliques and eigenvalues. *Journal of Combinatorial Theory, Series B*, 82(1):56–66.

Kumar, R. et al. (1999). Trawling the web for emerging cyber-communities. *Computer Networks*, 31(11-16):1481–1493.

Li, J. et al. (2007). Maximal biclique subgraphs and closed pattern pairs of the adjacency matrix: A one-to-one correspondence and mining algorithms.

Nagarajan, N. e Kingsford, C. (2012). Uncovering genomic reassortments among influenza strains by enumerating maximal bicliques.

Apêndice A: Implementação do algoritmo para encontrar bicliques maximais em bigrafos de intervalos

A partir dos pseudocódigos apresentados na seção 3.2, foi escrito um algoritmo em Python com as ideias apresentadas.

A.1 main

```
if __name__ == '__main__':
    import argparse

    start_end = a_gp = b_gp = []

    graph_path = 'placeholder.txt'
    parser = argparse.ArgumentParser()
    parser.add_argument("--path", help="path to graph")
    args = parser.parse_args()

    if args.path:
        graph_path = args.path

    try:
        read_graph(path=graph_path)
    except:
        print('Caminho invalido para grafo.')
        exit()

    bicliques = find_biclique()

    for b in bicliques:
        aux_a = []
        aux_b = []
        for v in b:
            if v in a_gp:
                aux_a.append(v)
            else:
                aux_b.append(v)
        print([aux_a, aux_b])
```

A função *main* possue duas partes importantes, a leitura do grafo, e a impressão da biclique.

A.2 read_graph

```
def read_graph(path='tt.txt'):
   with open(path, 'r') as f:
      a = f.readline()[:-1]
      seq = f.readline()
   a_grupo = []
   b_grupo = []
   for item in a.split(','):
      a_grupo.append(item)
   i = 0
   start = {}
   end = {}
   for item in seq.split(','):
      if item[1] not in a_grupo:
         if item[1] not in b_grupo:
            b_grupo.append(item[1])
      if item[0] == '+':
         start[item[1]] = i
      else:
         end[item[1]] = i
      i += 1
   global start_end, a_gp, b_gp
   start_end = [start, end]
   a_gp = a_grupo
   b_gp = b_grupo
```

Função para leitura do grafo. Ela aceita arquivos *.txt* no formato:

- Primeira linha contém um conjunto de vértices separados por vírgula, esse conjunto será considerado como a parte *A* do grafo de bi-intervalos.
- Segunda linha contém os eventos ordenados em ordem crescente, com sinais de *positivo* ou *negativo* para começo e fim de um intervalo, respectivamente.

Exemplo:

```
a,c,i,e,g
+a,+b,-a,+h,+c,-b,+i,+d,-h,-c,+e,+j,-i,-d,+f,-e,-j,+g,-f,-g
```

A função, a partir dessa entrada, aloca dois conjuntos, *a_gp* e *b_gp* como as duas partes do grafo, e guarda os eventos em ordem em um dicionário *start_end*, que salva os começos e fins ordenados dos vértices.

A.3 start

```
def start(v):
   return start_end[0][str(v)]
```

Devolve o *timestamp* do começo de determinado vértice.

A.4 end

```
def end(v):
   return start_end[1][str(v)]
```

Devolve o *timestamp* de fim de determinado vértice.

A.5 intersection

```
def intersection(i, f, group):
   ii = None
   ff = None
   for v in group:
      vi = start(v)
      vf = end(v)
      if (vi < f) & (i < vf):
         if ii == None:
            ii = vi
         else:
            if ii < vi:
               ii = vi
         if ff == None:
            ff = vf
         else:
            if vf < ff:
               ff = vf
   return ii, ff
```

Recebe um *timestamp* de começo i e outro de fim f e um conjunto de vértices. Varre todos os vértices pertencentes ao conjunto dado, e verifica se o vértice faz intersecção com o intervalo $\{i, f\}$, ou seja, ele intersecta todos os vértices dentro desse intervalo.

Retorna o começo do último intervalo que começa no conjunto dado e o fim do primeiro intervalo a terminar no conjunto dado.

A.6 find_group

```
def find_group(i, f, div_gp = False, a = True):
   vs = []
   for key, value in start_end[0].items():
      if value == i:
         if key in a_gp:
            grupo = b_gp
            grupo2 = a_gp
         else:
            grupo = a_gp
            grupo2 = b_gp
         break
   for v in grupo:
      if ((start(v) < f) & (end(v) > i)) | (end(v) == f) | (start(v) == i):
         if v not in vs:
            vs.append(v)
   for v in grupo2:
      if ((start(v) < i) & (end(v) > f)) | (end(v) == f) | (start(v) == i):
         if v not in vs:
            vs.append(v)
   if div_gp:
      if a:
         return [x for x in vs if x in a_gp]
      else:
         return [x for x in vs if x in b_gp]
   else:
      return vs
```

Recebendo no formato $\{i, f\}$, onde i representa o último início de um conjunto de intervalos, e f representa o primeiro término desse mesmo conjunto, essa função busca por todos os vértices que fazem parte ou intersectam com esse intervalo.

A.7 find_biclique

```
def find_biclique():
   bicliques = []

   min_gpo = [a_gp, b_gp] if len(a_gp) < len(b_gp) else [b_gp, a_gp]

   for key1, value1 in start_end[0].items():
      if key1 in min_gpo[0]:
         for key2, value2 in start_end[1].items():
            if key2 in min_gpo[0]:
               i, f = intersection(value1, value2, min_gpo[1])
               if (i != None) & (f != None):
                  i, f = intersection(i, f, min_gpo[0])
                  if (i == value1) & (f == value2):
                     ngp = sorted(find_group(i, f))
                     if ngp not in bicliques:
                        bicliques.append(ngp)
   return bicliques
```

Varre todos os pares possíveis de $\{i, f\}$ dentro do conjunto de vértices pertencentes a parte com o menor número de vértices do grafo. Como o algoritmo já executa a verificação utilizando a outra parte, percorrer novamente no segundo conjunto é redundante, portanto não é feito.

Tal ação é possível devido a propriedade das bicliques serem simétricas, pois como todos os vértices da parte A são vizinhos de todos os vértices da parte B e vice-versa, encontrar os vértices pertencentes a parte qualquer uma das partes da biclique permite encontrar por inferência os vértices da parte oposta.

Se o início e fim dos intervalos limitantes são os mesmos depois de procurar as intersecções tanto para A quanto para B, temos então uma biclique dentro desses intervalos.

A.8 biclique_graphs

```
def biclique_graphs(bicliques, return_list = False):
   from graphviz import Graph
   lst_nodes = []
   lst_edges = []

   grafo_biclique = Graph(format='png')

   for bcl in bicliques:
      for no in bcl:
         if no not in lst_nodes:
            grafo_biclique.node(no)
            lst_nodes.append(no)
         if no in a_gp:
            for n in [x for x in bcl if x in b_gp]:
               if (no+n) and (n+no) not in lst_edges:
```

```
16                  grafo_biclique.edge(no, n)
17                  lst_edges.append(n+no)
18
19      if return_list:
20          lst_bcl = []
21          for bcl in bicliques:
22              g_aux = Graph(format='png')
23              for no in bcl:
24                  g_aux.node(no)
25                  if no in a_gp:
26                      for n in [x for x in bcl if x in b_gp]:
27                          g_aux.edge(no, n)
28              lst_bcl.append(g_aux.source)
29
30      return (grafo_biclique, lst_bcl) if return_list else grafo_biclique
```

Essa função retorna uma variável no formato do Graphviz, que pode ser desenhada para vizualização do grafo biclique. Ela também tem a funcionalidade de retornar uma lista com todos as bicliques maximais.