LIII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2021

UM ALGORITMO PARA DETERMINAR UM LIMITE SUPERIOR PARA O PROBLEMA DA CONTAGEM DE INTERVALO

Lívia Salgado Medeiros
Instituto de Matemática e Estatística – Universidade do Estado do Rio de Janeiro (UERJ)
CEP 20550-900 – Rio de Janeiro – RJ – Brazil
liviasmedeiros@gmail.com

Fabiano de Souza Oliveira
Instituto de Matemática e Estatística – Universidade do Estado do Rio de Janeiro (UERJ)
CEP 20550-900 – Rio de Janeiro – RJ – Brazil
fabiano.oliveira@ime.uerj.br

Jayme Luiz Szwarcfiter
COPPE, NCE, IM – Universidade Federal do Rio de Janeiro (UFRJ)
CEP 21941-600 – Rio de Janeiro – RJ – Brazil
jayme@nce.ufrj.br

RESUMO

O emprego de técnicas de otimização combinatória no problema da contagem de intervalo foi explorado por Joos et al. [2014], resolvendo de forma eficiente uma versão restrita do problema da contagem de intervalo 2. Para o problema de determinar a contagem de intervalo geral, nenhuma técnica semelhante é conhecida atualmente. Elaboramos formulações de programação quadrática, levando a um algoritmo pseudopolinomial, para fornecer um limite superior na contagem de intervalo geral de ordens, que é empiricamente mostrado como próximo aos respectivos valores reais. Para a classe de semiordens, o limite superior é mostrado ser justo.

PALAVRAS CHAVE. Contagem de Intervalo. Limite superior algorítmico. Programação quadrática.

Tópico: OC – Otimização Combinatória; TAG – Teoria e Algoritmos em Grafos.

ABSTRACT

The employment of combinatorial optimization techniques in the interval count problem was explored by Joos et al. [2014], efficiently solving a restricted version of the interval count 2 problem. For the problem of determining the general interval count, no similar technique is currently known. We devise quadratic programming formulations, leading to a pseudopolynomial algorithm, for providing an upper bound on the general interval count of orders, which is empirically shown to be close to the respective actual values. For the class of semiorders, the upper bound is shown to be tight.

KEYWORDS. Algorithmic upper bound. Interval count. Quadratic programming.

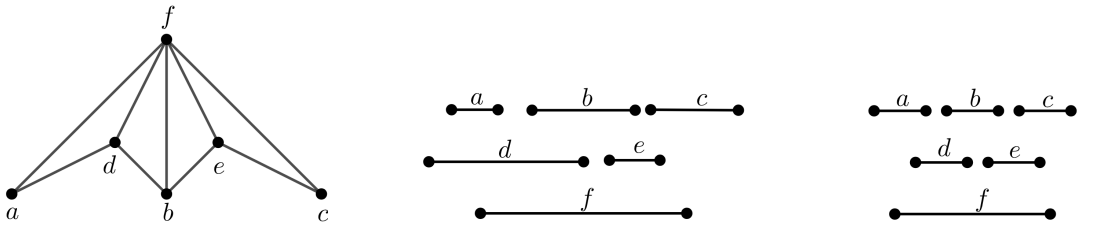
Topic: CO - Combinatorial Optimization; TAG - Theory and Algorithms in Graphs.

O segundo e terceiro autores são parcialmente financiados por FAPERJ e CNPq.

1. Introdução

O *grafo de intervalo* de uma família $\mathcal{R}$ de intervalos fechados na reta real é o grafo G tal que $V(G) = \mathcal{R}$ e, para todo $I, J \in V(G)$, $(I, J) \in E(G)$ se e somente se $I \cap J \neq \emptyset$. Chamamos $\mathcal{R}$ de um *modelo* de G. Uma *ordem* $P = (X, \prec)$ é uma relação binária $\prec$ no conjunto X que é irreflexiva e transitiva. Além disso, P é uma *ordem de intervalo* se existe um modelo de intervalo $\mathcal{R} = \{[\ell_x, r_x] \mid x \in X\}$ tal que $x \prec y$ se e somente se $r_x < \ell_y$. A *notação* $x \prec y$ será usada para denotar que $(x, y) \in \prec$ e neste caso, dizemos que x e y são *comparáveis*. Caso contrário, se x não precede y e y não precede x, x e y são ditos *incomparáveis*. O menor número $IC(G)$ (resp. $IC(P)$) de tamanhos distintos de intervalos necessário em um modelo de G (resp. P) é chamado *contagem de intervalo* (do inglês, *interval count*) de G (resp. P). Um grafo de intervalo G é *unitário* se $IC(G) = 1$. Já uma ordem de intervalo P é *unitária* ou *semiordem* se $IC(P) = 1$. O problema da contagem de intervalo foi inicialmente sugerido por Ronald Graham na década de 80 e, até o momento, a complexidade de reconhecer se $IC(G) \leq k$ (resp. $IC(P) \leq k$) para todo $k \geq 2$ é um problema em aberto [Fishburn, 1985; Klavík et al., 2019]. Por outro lado, o problema de reconhecer $IC(G) = 1$ ou $IC(P) = 1$ para dado grafo de intervalo G ou ordem de intervalo P é bem conhecido, admitindo diversos algoritmos eficientes [Cornell et al., 1995].

A Figura 1(a) apresenta um exemplo de um grafo G e dois modelos com diferentes números de tamanhos distintos. O modelo de G da Figura 1(b) apresenta quatro tamanhos distintos e aquele da Figura 1(c), também modelo de G, apresenta dois tamanhos distintos. Como o intervalo f é centro de $K_{1,3}$, o grafo não é de intervalo unitário. Portanto, a Figura 1(b) realiza a contagem de intervalo de G, que é igual a 2.



(a) Um grafo G.

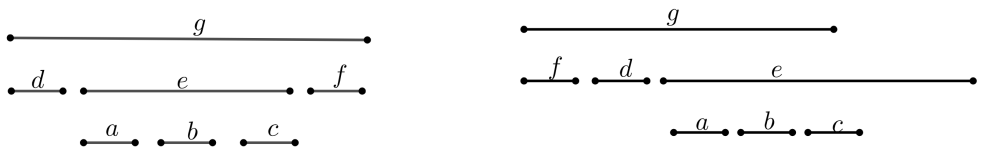
(b) Modelo de G.

(c) Modelo de G com dois tamanhos.

Figura 1: Um grafo G e dois modelos com diferentes números de tamanhos distintos.

A contagem de intervalo pode diferir para ordens e grafos associados a um mesmo modelo. Devido à relação de precedência das ordens, a Figura 2(a) realiza a contagem de intervalo da ordem P associada ao modelo, que é igual a 3. Se considerarmos o problema da contagem de intervalo ao grafo associado ao mesmo modelo, é possível reordenar os intervalos de modo que a Figura 2(b) realiza a contagem de intervalo de G, que é igual a 2. De maneira geral, temos que se certo modelo é associado tanto a uma ordem P quanto a um grafo G, então $IC(G) \leq IC(P)$.

Em [Joos et al., 2014] foi desenvolvido um algoritmo polinomial, usando programação linear, para determinar se, dados um grafo conexo G e uma bipartição dos seus vértices em dois conjuntos A e B, existe um modelo $\{I_v \mid v \in V(G)\}$ de G tal que $|I_v| = a$ para todo $v \in A$ e $|I_v| = b$ para todo $v \in B$, para algum par de constantes $a < b$. Os autores consideraram portanto o problema da contagem de intervalo restrito às classes de contagem igual a dois. Além deste trabalho, nenhuma outra técnica de otimização combinatória para o problema é conhecida, em especial para o problema geral. Neste trabalho, apresentamos formulações de programação quadrática que podem



(a) $IC(P) = 3$. (b) $IC(G) = 2$.

Figura 2: Contagem de intervalo para ordens e grafos.

ser resolvidas em tempo pseudopolinomial [Kozlov et al., 1981] fornecendo um limite superior para a contagem de intervalo de uma dada ordem de intervalo. Apesar de não haver garantias sobre o quão justo é o limite, os experimentos com exemplos da literatura relevantes para certos aspectos do problema tiveram bons resultados na prática. Na Seção 2, apresentamos o algoritmo que fornece o limite superior baseado em formulações de programação quadrática. Na Seção 3, apresentamos os experimentos realizados e as observações finais na Seção 4.

2. O algoritmo para o limite superior

Um problema com formulação de programação quadrática tem por objetivo encontrar um vetor x n-dimensional que satisfaça um conjunto de restrições enquanto minimiza certa função objetivo, com formulação geral

$$\begin{array}{ll} \textbf{minimizar} & f(x) = \frac{1}{2}x^T F x + q^T \\ \textbf{sujeito a} & Ax = b \\ & Qx \leq h \end{array} \tag{1}$$

onde F, A, Q são matrizes e q, b, h são vetores constantes, todos de dimensões apropriadas. Uma matriz M de dimensão n é *semidefinida positiva* se $x^T M x \geq 0$ para todo $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^{\mathbf{n}}$. Quando a matriz F de (1) é semidefinida positiva, o método solucionador elipsóide resolve a programação quadrática em tempo fracamente polinomial [Kozlov et al., 1981].

Na aplicação ao problema de contagem de intervalo, o vetor variável x representa um modelo de intervalo, consistindo de

$$x = [s_1, s_2, \ldots, s_n, \ell_1, r_1, \ell_2, r_2, \ldots, \ell_n, r_n, \epsilon]$$

onde ℓ_i, r_i, s_i representam respectivamente a extremidade esquerda, a extremidade direita e o tamanho do i-ésimo intervalo deste modelo e ϵ o mínimo distanciamento entre dois intervalos que não se interseptam. Consideramos duas propostas de função objetivo, nomeadamente as funções $f_1(x)$ e $f_2(x)$ definidas como

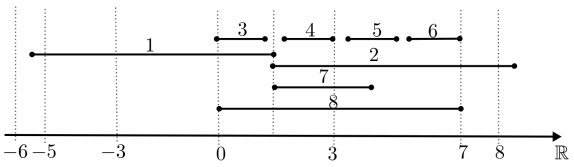
$$f_1(x) = \sum_{i=1}^{n}(s_i - 1)^2 \qquad f_2(x) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(s_i - s_j)^2 + \sum_{i=1}^{n} s_i \tag{2}$$

O conjunto de restrições faz com que uma solução viável x represente um modelo:

$$\begin{cases} r_i - \ell_i = s_i & \text{para todo } i = 1, 2, \ldots, n \\ r_u \geq \ell_v \text{ e } r_v \geq \ell_u & \text{para todo } u, v \in X,\ u \not\prec v \text{ e } v \not\prec u \\ r_u + \epsilon \leq \ell_v & \text{para todo } u, v \in X \text{ e } u \prec v \\ \ell_i \geq 0,\ r_i \geq 0 \text{ e } s_i \geq 1 & \text{para todo } i = 1, 2, \ldots, n \\ \epsilon = 0{,}01 & \end{cases} \tag{3}$$

O valor de $\epsilon = 0{,}01$ empregado é arbitrário, e de fato, qualquer valor $0 < \epsilon < 1$ seria igualmente satisfatório. Sabe-se que, sem perda de generalidade, existe um modelo que realiza a contagem de intervalo de certo grafo ou ordem no qual todos os extremos de intervalo são inteiros [Cerioli et al., 2011].

Para executar a otimização com as formulações definidas acima, utilizamos o CVXPY que é uma biblioteca de domínio especifico para otimização convexa e pode ser importada para Python. O algoritmo recebe como entrada um modelo $\mathcal{R}$ de uma ordem P e define as matrizes A, Q e vetores b, h que correspondem, em formato matricial, às restrições definidas em (3). Em seguida, define a matriz F e o vetor q correspondente a $f_1(x)$ ou $f_2(x)$ em (2), e executa o pacote CVXPY com o solucionador ECOS, obtendo o resultado da otimização. Seja $\mathcal{R}'$ o modelo associado a esse resultado. A partir de $\mathcal{R}'$, o algoritmo obtém o modelo $\mathcal{R}$ pelo aumento de todos os intervalos que se encontram na primeira e última cliques de $\mathcal{R}'$ (cliques mais à esquerda e mais à direta, respectivamente) até que os tamanhos destes intervalos sejam iguais ao maior presente em $\mathcal{R}'$. Por fim, o algoritmo retorna como contagem de intervalo de P o número de tamanhos distintos de $\mathcal{R}$. A Figura 3 ilustra o resultado da execução do algoritmo. Naturalmente, $IC(P)$ é no máximo o número de tamanhos presentes em $\mathcal{R}$. Portanto, o retorno do algoritmo é de fato um limite superior para $IC(P)$.



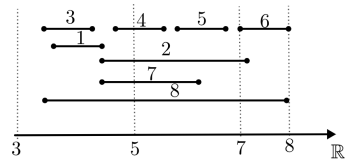
$\{I_1 = [-5.5, 1.5],\ I_2 = [1.5, 8.5],\ I_3 = [0, 1],\ I_4 = [2, 3],$
$I_5 = [4, 5],\ I_6 = [6, 7],\ I_7 = [1.5, 4.5],\ I_8 = [0, 7]\}$

(a) Ordem P de entrada, correspondente ao modelo acima.

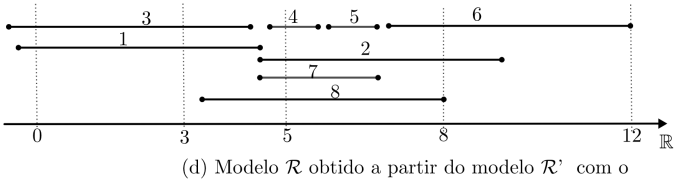
minimizar $f(x)$
sujeito a

$$\begin{cases} (\dots) \\ r_5 - \ell_5 = s_5 \\ r_2 \geq \ell_5,\ r_5 \geq \ell_2 \\ r_7 \geq \ell_5,\ r_5 \geq \ell_7 \\ r_8 \geq \ell_5,\ r_5 \geq \ell_8 \end{cases} \quad \begin{cases} r_1 + \epsilon \leq \ell_5 \\ r_3 + \epsilon \leq \ell_5 \\ r_4 + \epsilon \leq \ell_5 \\ r_5 + \epsilon \leq \ell_6 \\ \ell_5 \geq 0,\ r_5 \geq 0,\ s_5 \geq 0 \\ (\dots) \end{cases}$$

(b) Programação quadrática formulada (restrições associadas ao intervalo 5).



(c) Modelo $\mathcal{R}$' obtido a partir do resultado da otimização.



(d) Modelo $\mathcal{R}$ obtido a partir do modelo $\mathcal{R}$' com o aumento dos intervalos da primeira e última cliques.

Figura 3: Para a ordem de entrada, o algoritmo fornece o limite superior 3 (tamanhos distintos em $\mathcal{R}$).

Descrevemos, a seguir, os detalhes da implementação do algoritmo. Uma *forma quadrática de* $\mathbb{R}^n$ são funções da forma

$$f(x_1, x_2, \cdots, x_n) = \sum_{i \in \{1,\dots,n\}} a_i x_i^2 + \sum_{i \in \{1,\dots,n\}} \sum_{j \in \{1,\dots,n\}, j \neq i} b_{ij} x_i x_j \tag{4}$$

os termos da forma $b_{ij}x_ix_j$ são denominados de *produto misto*. Costuma-se combinar os termos envolvendo x_ix_j com os termos envolvendo x_jx_i. Se x denotar o vetor coluna das variáveis, podemos escrever (4) em formato matricial e então dizemos que x^TFx é a *forma quadrática associada a* $f(x_1, x_2, \cdots, x_n)$.

Lema 1. *A matriz F associada à função $f_1(x)$ ou $f_2(x)$ em (2) é semidefinida positiva.*

Demonstração. Seja $x^T Fx$ a forma quadrática associada a $f_1(x)$, onde

$$x = [s_1, s_2, \ldots, s_n, \ell_1, r_1, \ell_2, r_2, \ldots, \ell_n, r_n, \epsilon]$$

Seja $F_{i,j}$ o elemento de F da i-ésima linha, j-ésima coluna. Assim,

$$F_{i,j} = \begin{cases} 2, & \text{se } i = j \text{ e } i \leq n \\ 0, & \text{caso contrário} \end{cases}$$

para todo $1 \leq i, j \leq 3n + 1$. Logo, para todo $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^{\mathbf{3n+1}}$, temos que

$$x^T Fx = \sum_{i=1}^{n} x_i^2 = 4n \geq 0$$

e, portanto, F é semidefinida positiva.

Do mesmo modo, seja $x^T Fx$ a forma quadrática associada a $f_2(x)$ e seja $F_{i,j}$ o elemento de F da i-ésima linha, j-ésima coluna. Assim,

$$F_{i,j} = \begin{cases} 2(n-1), & \text{se } i = j \text{ e } i \leq n \\ -2, & \text{se } i \neq j \text{ e } i, j \leq n \\ 0, & \text{caso contrário} \end{cases}$$

para todo $1 \leq i, j \leq 3n + 1$. Logo, para todo $\mathbf{0} \neq \mathbf{x} \in \mathbb{R}^{\mathbf{3n+1}}$, temos que

$$x^T Fx = 2(n-1)\sum_{i=1}^{n} x_i^2 - 4\sum_{i=1}^{n}\sum_{j=i+1}^{n} x_i x_j = 2(n-1)n - 2n(n-1) = 0 \geq 0$$

e, portanto, F é semidefinida positiva. □

O Algoritmo 1 recebe como entrada um modelo $\mathcal{R}$ com pares referentes às extremidades esquerda e direita de cada intervalo do modelo. Cada par consiste de um vetor de duas posições. Inicialmente, define-se a matriz $\mathcal{M}$ quadrada nula de ordem n, onde n representa o número de intervalos de $\mathcal{R}$. Para cada intervalo I_i, o algoritmo compara a extremidade direita deste intervalo com a extremidade esquerda de todos os outros intervalos J_j. Caso a extremidade direita seja menor, então define-se $\mathcal{M}[i][j] = 1$ e $\mathcal{M}[j][i] = 0$ indicando que o intervalo I_i precede J_j. Se os intervalos são incomparáveis, $\mathcal{M}[i][j] = 0$ é mantido no final do processo.

Para modelar a função $f_1(x)$, os valores de F e q serão descritos adiante. Reescrevemos $f_1(x)$ da seguinte forma

$$\begin{aligned} f_1(x) = \textstyle\sum_{i=1}^{n}(s_i - 1)^2 &= s_1^2 - 2s_1 + 1 + s_2^2 - 2s_2 + 1 + \cdots + s_n^2 + 2s_n + 1 \\ &= (s_1^2 + s_2^2 + \cdots + s_n^2) - 2(s_1 + s_2 + \cdots + s_n) + n \end{aligned}$$

A matriz F guarda os valores dos coeficientes de s_i^2, para todo $1 \leq i \leq n$, assim como aqueles dos produtos mistos. Como os valores de s_i estão nas n primeiras posições do vetor x e como há um coeficiente $\frac{1}{2}$ na formulação padrão da programação quadrática conforme se vê em (1), define-se na matriz F o valor 2 nas n primeiras posições da diagonal principal, devendo ser nulo em todas as demais posições pela ausência de produtos mistos.

Algoritmo 1 Obtendo ordem P associada ao modelo $\mathcal{R}$ de entrada

```
procedimento DEFINIR_P(Modelo[])
    para i ← 1 até |Modelo| faça
        para j ← 1 até |Modelo| faça
            I = Modelo[i]
            J = Modelo[j]
            se I[1] < J[0] então
                M[i][j] ← 1
                M[j][i] ← 0
            senão
                se J[1] < I[0] então
                    M[j][i] ← 1
                    M[i][j] ← 0
                senão
                    M[j][i] ← 0
                    M[i][j] ← 0
```

Algoritmo 2 Função $f_1(x)$

```
procedimento DEFINIR_F(n)
    para i ← 1 até n faça
        F[i][i] ← 2
procedimento DEFINIR_q(n)
    para i ← 1 até n faça
        q[i] ← −2
```

O vetor q guarda os valores dos coeficientes de s_i, para todo $1 \leq i \leq n$. Como os valores de s_i estão nas n primeiras posições do vetor x, define-se no vetor q o valor -2 nas n primeiras posições. O Algoritmo 2 retrata a construção de F e q associado ao emprego da função $f_1(x)$.

Com respeito à função $f_2(x)$, os valores de F, q, serão descritos conforme a seguir. Como

$$f_2(x) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(s_i - s_j)^2 + \sum_{i=1}^{n} s_i = (n-1)\sum_{i=1}^{n} s_i^2 - 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} s_i s_j + \sum_{i=1}^{n} s_i$$

A matriz F guarda os valores dos coeficientes de s_i^2, para todo $1 \leq i \leq n$. Como os valores de s_i estão nas n primeiras posições do vetor x e como há um coeficiente $\frac{1}{2}$ na formulação padrão da programação quadrática conforme se vê em (1), define-se na matriz F o valor $2(n-1)$ nas n primeiras posições da diagonal principal. Nas posições do termo com produto misto, define-se na matriz F o valor -2 nas n primeiras posições.

O vetor q guarda os valores dos coeficientes de s_i, para todo $1 \leq i \leq n$. Como os valores de s_i estão nas n primeiras posições do vetor x, define-se no vetor q o valor 1 nas n primeiras posições, que equivale aos valores do segundo somatório de $f_2(x)$. O Algoritmo 3 retrata a construção de F e q associada ao emprego da função $f_2(x)$.

A matriz A e o vetor b são descritos pelas restrições formuladas em (3). O número de equações será igual a $n+1$, que corresponde a n restrições geradas na primeira linha de (3) e mais uma equação com respeito a ϵ. Este número de equações representa as linhas da matriz A.

Reescrevendo a primeira restrição (3) como

$$r_i - \ell_i = s_i \iff r_i - \ell_i - s_i = 0 \tag{5}$$

desse modo, a partir de $n+1$ até $3n$, estão posicionados no vetor x sequencialmente os valores $\ell_1, r_1, \ell_2, r_2, \ldots, \ell_n, r_n$ e a última posição $3n+1$ é relativa ao valor de ϵ. Assim, existe na matriz

Algoritmo 3 Função $f_2(x)$

procedimento DEFINIR_F(n)
  para $i \leftarrow 1$ **até** n **faça**
    para $j \leftarrow 1$ **até** n **faça**
      $F[i][j] \leftarrow -2$
      $F[j][i] \leftarrow -2$
      $F[i][i] \leftarrow 2(n-1)$
procedimento DEFINIR_q(n)
  para $i \leftarrow 1$ **até** n **faça**
    $q[i] \leftarrow 1$

A, n linhas que correspondem as n equações do tipo (5). Na i-ésima linha, define-se na matriz A o valor -1 na posição $n+2i-1$ correspondendo ao coeficiente de ℓ_i, o valor 1 na posição $n+2i$ que corresponde ao r_i e o valor -1 na posição i correspondendo ao coeficiente de s_i. Todas as demais posições são nulas. Desse modo, as n equações do tipo (5) estarão preenchidas na matriz A. Na $(n+1)$-ésima linha de A, define-se o valor 1 na posição $3n+1$ que corresponde à posição de ϵ, para que possa preencher a última equação das restrições (3).

O vetor b terá todas as n primeiras posições zeradas, em conformidade com (5), exceto a última posição que fixamos um valor arbitrário para ϵ.

A matriz Q e o vetor h são construídos a partir da ordem de entrada, representada pela matriz $\mathcal{M}$. Inicialmente, para atender as restrições

$$\begin{cases} r_i \geq 0 & \text{para todo } i = 1, \ldots, n \\ \ell_i \geq 0 & \text{para todo } i = 1, \ldots, n \\ s_i \geq 1 & \text{para todo } i = 1, \ldots, n \end{cases} \tag{6}$$

transformamos essas restrições em

$$\begin{cases} r_i \geq 0 \iff -r_i \leq 0 \\ \ell_i \geq 0 \iff -\ell_i \leq 0 \\ s_i \geq 1 \iff -s_i \leq -1 \end{cases}$$

e, assim, há 3 desigualdades associada a cada valor $1 \leq i \leq n$, totalizando $3n$ linhas na matriz Q. A diagonal principal corresponde a cada desigualdade. Mais especificamente, faz-se

$$Q[i][i] = -1 \quad \text{para todo } 1 \leq i \leq 3n$$

Quanto ao vetor h, define-se o i-ésimo elemento, $1 \leq i \leq n$, igual a -1 que corresponde à desigualdade $-s_i \leq -1$ e o restante do vetor igual a 0.

A partir da linha $3n+1$, seguem as demais inequações na matriz Q como descrito adiante.

Caso a matriz $\mathcal{M}$ indique que os intervalos são comparáveis, temos que inequações especificadas pelas as restrições abaixo conforme em (3):

$$r_i + \epsilon \leq \ell_j \iff r_i + \epsilon - \ell_j \leq 0, \text{ para todo } i, j \in X \text{ e } i \prec j \tag{7}$$

Mais especificamente, se $k-1$ é a última posição de Q onde uma restrição foi formulada (ou seja, k é a próxima linha onde uma restrição deve ser formulada), faz-se

$Q[k][n+2j-1] = -1$ (que corresponde ao coeficiente de ℓ_j da equação (7))
$Q[k][n+2i] = 1$ (que corresponde ao coeficiente de r_i da equação (7))
$Q[k][3n+1] = 1$ (que corresponde ao coeficiente de ϵ da equação (7))

No caso dos intervalos serem incomparáveis de acordo com as restrições (3),

$$\begin{aligned} r_i \geq \ell_j \iff \ell_j - r_i \leq 0, \text{ para todo } i, j \in X, i \not\prec j \text{ e } j \not\prec i \\ r_j \geq \ell_i \iff \ell_i - r_j \leq 0, \text{ para todo } i, j \in X, i \not\prec j \text{ e } j \not\prec i \end{aligned} \tag{8}$$

Note que, para cada ocorrência de (8) serão definidas duas linhas da matriz Q.

$Q[k][n+2j-1] = 1$ (que corresponde ao coeficiente de ℓ_j da equação (8))
$Q[k][n+2i] = -1$ (que corresponde ao coeficiente de r_i da equação (8))
$Q[k+1][n+2j] = -1$ (que corresponde ao coeficiente de r_j da equação (8))
$Q[k+1][n+2i-1] = 1$ (que corresponde ao coeficiente de ℓ_i da equação (8))

O Algoritmo 4 constrói a matriz A, o vetor b, a matriz Q e o vetor h.

Algoritmo 4 Conjunto de Restrições

procedimento DEFINIR_A(n)
  para $i \leftarrow 1$ **até** n **faça**
    $A[i][j] \leftarrow -1$
    $A[i][n+2i-1] \leftarrow -1$
    $A[i][n+2i] \leftarrow 1$
  $A[n][3n+1] \leftarrow 1$
procedimento DEFINIR_b(n)
  para $i \leftarrow 1$ **até** n **faça**
    $b[i] \leftarrow 0$
    $b[n+1] \leftarrow \epsilon$
procedimento DEFINIR_Q($\mathcal{M}$,n)
  para $i \leftarrow 1$ **até** $3n$ **faça**
    $Q[i][i] \leftarrow -1$
  $k \leftarrow 3n+1$
  para $i \leftarrow 1$ **até** $|\mathcal{M}|$ **faça**
    para $j \leftarrow 1$ **até** $|\mathcal{M}|$ **faça**
▷ Intervalos comparáveis
      se $\mathcal{M}[i][j] \leftarrow 1$ **então**
        $Q[k][n+2i] \leftarrow 1$
        $Q[k][n+2j-1] \leftarrow -1$
        $k \leftarrow k+1$
      senão
▷ Intervalos incomparáveis
        se $i < j$ **e** $\mathcal{M}[j][i] = 0$ **então**
          $Q[k][n+2i-1] \leftarrow 1$
          $Q[k][n+2j] \leftarrow -1$
          $Q[k+1][n+2i] \leftarrow -1$
          $Q[k+1][n+2j-1] \leftarrow 1$
          $k \leftarrow k+2$
procedimento DEFINIR_h(n)
  para $i \leftarrow 1$ **até** n **faça**
    $h[i] = -1$

Em diversos problemas testados, faz-se necessário que os intervalos da primeira e da última clique assumam o maior tamanho do modelo, caso contrário, como a formulação (1) tende a

aproximar os tamanhos do valor unitário, é provável que estes intervalos que pertencem à primeira e à última clique adicionem um tamanho extra no caso de que os (potencialmente vários) tamanhos distintos na primeira e na última clique não possam ser todos compatibilizados com algum dos tamanhos já presentes no modelo. Corrigimos este problema com o Algoritmo 5. O Algoritmo 5 recebe como entrada o vetor $x.valor$, que equivale ao resultado da otimização. Determina-se o maior tamanho do modelo, a a maior extremidade esquerda dentre todos os intervalos bem como a menor extremidade direita. Todos os intervalos que possuem extremidade esquerda menor que a menor direita pertencem a primeira clique e estes receberão o maior tamanho do modelo. Do mesmo modo, todos os intervalos que possuem a extremidade direita maior que a maior extremidade esquerda pertencem a última clique e estes receberão o maior tamanho do modelo.

Algoritmo 5 Maior Tamanho

procedimento MAIORTAMANHO($x.valor[]$)
  $menordireita \leftarrow x.valor[n+2]$
  para $i \leftarrow n+4$ **até** $3n$ **faça**
    se $x.valor[i] < menordireita$ **então**
      $menordireita \leftarrow x.valor[i]$
      $i \leftarrow i+2$
  $maioresquerda \leftarrow x.valor[n+1]$
  para $i \leftarrow n+3$ **até** $3n-1$ **faça**
    se $x.valor[i] > maioresquerda$ **então**
      $maioresquerda \leftarrow x.valor[i]$
      $i \leftarrow i+2$
  $maioretamanho \leftarrow x.valor[n+2] - x.valor[n+1]$
  para $i \leftarrow n+2$ **até** $3n-2$ **faça**
    se $(x.valor[n+2] - x.valor[n+1]) > maiortamanho$ **então**
      $maiortamanho \leftarrow x.valor[n+2] - x.valor[n+1])$
      $i \leftarrow i+2$
  para $i \leftarrow n+1$ **até** $3n-1$ **faça**
    se $x.valor[i] \leq menordireita$ **e** $(x.valor[n+2] - x.valor[n+1]) < maiortamanho$ **então**
      $x.valor[i] \leftarrow x.valor[i] - (maiortamanho - (x.valor[n+2] - x.valor[n+1]))$
      $it \leftarrow \lfloor \frac{it-n}{2} \rfloor$
      $x.valor[i] \leftarrow maiortamanho$
  para $i \leftarrow n+2$ **até** $3n$ **faça**
    se $x.valor[i] \geq maioresquerda$ **e** $(x.valor[n+1] - x.valor[n]) < maiortamanho$ **então**
      $x.valor[i] \leftarrow x.valor[i] + (maiortamanho - (x.valor[n+1] - x.valor[n]))$
      $it \leftarrow \lfloor \frac{it-n-1}{2} \rfloor$
      $x.valor[i] \leftarrow maiortamanho$

O Algoritmo 6 resolve o problema de acordo com a formulação de programação quadrática em (1) com a função $cp.Problem$. A função $prob.solve(solver = cp.ECOS)$ usa o solucionador ECOS e retorna em $x.valor$ os valores das variáveis correspondendo a uma solução ótima do problema.

Algoritmo 6 Construção do Problema

x = Variavel(n)
prob = cp.Problem(cp.Minimize($(1/2)$cp.quadform$(x, F) + q^T x$),
$Qx \leq h, Ax = b$))
▷ O valor ótimo da função objetivo é retornado por prob.solve().
result = prob.solve(solver=cp.ECOS)
▷ O valor ótimo de x é armazenado em x.valor.
retornar x.valor

3. Experimentos

O objetivo é comparar os resultados de cada função objetivo, que fornece um limite superior para a contagem de intervalo, aos valores reais. O limite superior fornecido pelo algoritmo é justo para ordens de intervalo unitário (semiordens), de acordo com o Lema 2.

Lema 2. *Seja k^* o resultado obtido pelo algoritmo com respeito a alguma das funções objetivos formuladas em (2) para dada semiordem P. Então $k^* = 1 = IC(P)$.*

Demonstração. Claramente, cada parcela $(s_i - 1)^2$ da função objetivo $f_1(x)$ é maior ou igual a zero. Desse modo, se todos os tamanhos do modelo são iguais a 1, então cada parcela se anula e certamente $f_1(x)$ atinge o menor valor possível, sobre todas as soluções viáveis. Como conclusão, para dada ordem de entrada P, se $IC(P) = 1$, então a solução viável que possui o valor mínimo é aquela que corresponde a um modelo unitário e, portanto, o limite k* que será encontrado pelo algoritmo será igual a k*$= 1 = IC(P)$.

Para a função objetivo $f_2(x)$, cada parcela $(s_i - s_j)^2$ é maior ou igual a zero. Desse modo, se todos os tamanhos do modelo são iguais a 1, então cada parcela se anula e certamente $f_2(x)$ atinge o menor valor possível, dentre todas as soluções viáveis. Analogamente, conclui-se que o limite k* que será encontrado pelo algoritmo será igual a $1 = IC(P)$. □

Descreveremos algumas classes de grafos e ordens a seguir. Todas as instâncias usadas no experimento se relacionam a ordens.

Classe 1. A Figura 4(a) ilustra um grafo G com $IC(G) = 2$ para o qual um vértice (c) que não é centro de $K_{1,3}$ não pode possuir o menor tamanho. Isto contraria a intuição ingênua que o problema de contagem de intervalo 2 se resume a atribuir o maior tamanho exclusivamente àqueles vértices que são centro de $K_{1,3}$.



(a) Classe 1. (b) Classe 2.

Figura 4: Ordens definidas por seus modelos.

Classe 2. Leibowich estudou a classe de grafos quase livres de $K_{1,3}$, definidos como aqueles que possuem um vértice cuja remoção torna-o livre de $K_{1,3}$ e provou que $IC(G) \leq 2$ para grafos G nesta classe [Leibowitz et al., 1982]. A Figura 4(b) é um exemplo de um grafo nesta classe.

Classe 3. Fishburn introduziu o problema extremal de determinar

$$\sigma_{\mathcal{C}}(k) = \min\{|X| \mid P = (X, \prec) \text{ pertence à classe } \mathcal{C} \text{ e } IC(P) \geq k\}$$

e $\tilde{\sigma}_{\mathcal{C}}(k)$ que é definido analogamente para grafos [Fishburn, 1985]. Fishburn conjecturou que $\sigma_{\mathcal{C}}(k) = 3k - 2$ para $\mathcal{C}$ como a classe geral de ordens de intervalo, tendo provado a conjectura para todo $k \leq 7$. No entanto, nenhum valor de $\tilde{\sigma}_{\mathcal{C}}(k)$ é conhecido. Um grafo de intervalo G é *trivialmente perfeito* (na classe TP) se G é livre de P_4. Um grafo de intervalo G é um grafo *split* (na classe SPLIT) se existe uma partição (K, I) de $V(G)$ na qual K é uma clique e I é um conjunto independente. Estudamos $\sigma_{\mathcal{C}}(k)$ e $\tilde{\sigma}_{\mathcal{C}}(k)$ para $\mathcal{C} \in \{\text{TP}, \text{SPLIT}\}$ Medeiros et al. [2019]. Entre outros resultados, provamos que $\sigma_{TP}(k) = 3k - 2$, $\sigma_{\text{SPLIT}}(k) = 3k - 2$, $\tilde{\sigma}_{TP}(k) = \frac{(3^k-1)}{2}$, $\tilde{\sigma}_{\text{SPLIT}}(k) = 3k - 1$.

Classe 4. Uma família P_k, com $k \geq 2$, de *ordens de caminho longo (CL)* é uma família de ordens construída da seguinte forma. O modelo da Figura 5(a) ilustra um caminho em 6 vértices acrescido de um universal. A ordem P_2 é aquela definida pela Figura 5(a). Para $k > 2$, P_k é obtido a partir da ordem P_{k-1} através da seguinte transformação: acrescenta-se três novos elementos (correspondentes a intervalos) tal que o primeiro compartilha sua extremidade direita com a menor extremidade esquerda no modelo P_{k-1}, o segundo compartilha sua extremidade esquerda com a maior extremidade direita no modelo P_{k-1} e o terceiro consiste de um universal. A Figura 5(b) ilustra um modelo da ordem P_3.



(a) Modelo (CL) com $k = 2$ e $n = 6$ (Classe 4).

(b) Modelo (CL) com $k = 3$ e $n = 9$ (Classe 4).

Figura 5: Ordens da classe CL definidas por seus modelos.

Com exceção da Classe 3, é desconhecido na literatura famílias de ordens cuja contagem de intervalo é conhecida. Portanto, esta família P_k foi construída com o objetivo de testar mais uma família em que é conhecido o valor de k variando o valor de n, para que pudéssemos ampliar os testes com diferentes instâncias.

Classe 5. Fishburn define a família de ordens $\mathcal{D}_2$, cada uma com cadeia de inclusão própria entre intervalos limitada a 2 [Fishburn, 1985], provando que esta família requer um número de tamanhos distintos arbitrariamente grande, estabelecendo o seguinte teorema.

Teorema 1. *[Fishburn, 1985] Para todo natural $k \geq 1$, existe uma ordem de intervalo $P_k \in \mathcal{D}_2$ tal que $IC(P) \geq k + 1$.*

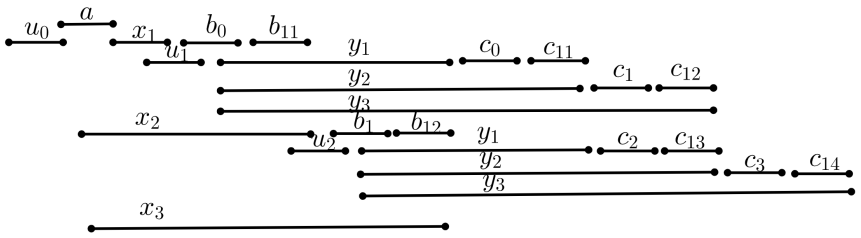


Figura 6: Modelo de P_2 da família $\mathcal{D}_2$ (Classe 4).

A Tabela 1 sumariza os resultados dos experimentos. Os resultados descritos aqui são simplificados devido a limitações de espaço. Ao todo, foram testadas 54 ordens. Nos resultados obtidos, o algoritmo com função objetivo $f_1(x)$ reportou 92% dos limites superiores que coincidem com a contagem de intervalo real e 8% dos limites superiores maiores que o valor da contagem de intervalo real. A média da diferença entre o valor reportado e o valor real foi de $1{,}5$ com o uso de $f_1(x)$. O algoritmo com função objetivo $f_2(x)$ reportou 28% dos limites superiores que coincidem com a contagem de intervalo real, 48% das ordens testadas com o uso de $f_2(x)$ não foram computados por nenhum solucionador e 24% dos limites superiores maiores que o valor da

contagem de intervalo real. A média da diferença do valor reportado e o real foi de 1,2 com o uso de $f_2(x)$.

Tabela 1: Resultados parciais sumarizados.

			Limite Superior Algorítmico		Tempo de Execução (s)	
Classe de P	n	$IC(P)$	$f_1(x)$	$f_2(x)$	$s_{f_1(x)}$	$s_{f_2(x)}$
1	13	2	3	4	0,031	0,047
2	9	2	2	2	0,031	0,062
3 (TP)	7	3	3	3	0,015	0,031
3 (TP)	43	15	15	15	0,150	0,310
3 (TP)	40	4	4	4	0,093	0,109
3 (TP)	121	5	5	6	0,921	1,249
3 (SPLIT)	14	5	5	6	0,015	0,031
3 (SPLIT)	74	25	25	26	0,340	1,658
3 (SPLIT)	16	6	6	7	0,031	0,032
3 (SPLIT)	28	10	10	10	0,059	0,046
4 (CL)	12	4	5	4	0,037	0,066
4 (CL)	18	6	6	6	0,037	0,045
4 (CL)	30	10	10	10	0,078	0,117
5	25	≥ 3	7	8	0,104	0,060

4. Conclusão

Elaboramos um limite superior algorítmico para determinar a contagem de intervalo de ordens. Observa-se que, em geral, os valores encontrados pela função objetivo $f_1(x)$ aproximam-se dos resultados reais (estes se encontram na tabela na coluna denotada por $IC(P)$). O tempo de execução do algoritmo para a função objetivo $f_2(x)$ é, em média, o dobro do tempo de execução do algoritmo para a função objetivo $f_1(x)$. Como trabalho futuro, investigaremos outras funções objetivo e se o limite superior aqui proposto é justo para classes restritas de ordens de intervalo.

Referências

Cerioli, M. R., de S. Oliveira, F., e Szwarcfiter, J. L. (2011). On counting interval lengths of interval graphs. *Discrete Applied Mathematics*, 159(7):532–543.

Cornell, D. G., Kim, H., Natarajan, S., Olariu, S., e Sprague, A. P. (1995). Simple linear time recognition of unit interval graphs. *Information Processing Letters*, 55(2):99–104.

Fishburn, P. (1985). *Interval Orders and Interval Graphs*. John Wiley and Sons.

Joos, F., Lowenstein, C., Oliveira, F. S., Rautenbach, D., e Szwarcfiter, J. L. (2014). Graphs of interval count two with a given partition. *Inf. Process. Lett.*, 114:542–546.

Klavík, P., Otachi, Y., e Sejnoha, J. (2019). On the classes of interval graphs of limited nesting and count of lengths. *Algorithmica*, 81:1490–1511.

Kozlov, M. K., Tarasov, S. P., e Khachiyan, L. G. (1981). The polynomial solvability of convex quadratic programming. *Comput. Maths. Math. Phys*, 20(5):223–228.

Leibowitz, R., Assmann, S. F., e Peck, G. W. (1982). The interval count of a graph. *SIAM J. Alg. Discr. Meth.*, 3(4):485–494.

Medeiros, L. S., Oliveira, F. S., e Szwarcfiter, J. L. (2019). Two problems on interval counting. *Electron. Notes Theor. Comput. Sci.*, 346:625–643.