Processos Irreversíveis:
$IRR_f\ Conversion\ Sets$

Rodrigo Lamblet Mafort

Instituto de Computação
Universidade Federal Fluminense

24 de junho de 2016

Sumário

1 Introdução

Um processo em um grafo corresponde à uma sequência de configurações associadas aos vértices. Uma configuração corresponde à uma atribuição de rótulos 0 e 1 para cada vértice do grafo. Uma nova configuração é obtida pela aplicação de uma determinada regra na configuração anterior.

Uma regra especifica as condições necessárias para que um vértice com rótulo 0 seja convertido, isto é, passe a possuir rótulo 1. Quando esta regra não permite que um vértice com rótulo 1 retorne ao rótulo 0 esse processo é determinado irreversível. Do contrário, este processo é dito reversível.

Este trabalho abordará processos irreversíveis nos quais um vértice será convertido para o rótulo 1 em uma determinada configuração se na configuração anterior ele possuir um número pré-estabelecido de vizinhos com rótulo 1. Para especificar o número de vizinhos necessários para que a conversão ocorra uma função *Threshold* $f : V(G) \rightarrow \mathbb{Z}$ é utilizada. Neste caso, este processo pode ser chamado de $IRR_f - Process$. Desta forma, o rótulo $c'(v)$ de cada vértice v pode ser obtido a partir da configuração anterior $c(v)$ pela aplicação da seguinte regra:

$$\forall\, v \in V(G) : (c(v) = 1 \vee |\{u \in Adj(v) \mid c(u) = 1\}| \geq f(v)) \Leftrightarrow c'(v) = 1 \qquad (1)$$

Como pode ser observado, na regra apresentada sempre que um vértice recebe o rótulo 1 ele nunca retornará ao rótulo 0, caracterizando um processo irreversível. Além disso, quando o valor de $f(v) = k$ para todo vértice $v \in V(G)$ este processo pode ser descrito como $IRR_k - Process$.

Um processo converge para 1 quando em um determinado momento todos os vértices possuem rótulo 1. A configuração inicial deste processo é conhecida como $Hull\ Set$. Quando o $Hull\ Set$ é mínimo este pode ser chamado de $IRR_f - Conversion\ Set$.

Considerando que a função f é um fator importante para a definição do $Hull\ Set$, este trabalho utilizará a notação S é um $Hull\ Set$ de (G, f), onde G e f correspondem, respectivamente, ao grafo e a função *Threshold* para os quais S é um $Hull\ Set$. Qualquer modificação em G ou f pode fazer com que S não seja mais um $Hull\ Set$.

Decidir se em dado um grafo G existe um IRR_2 de tamanho inferior à um número inteiro positivo d é um problema NP-Completo (Centeno et al., 2011). Desta forma, considerando que o problema $IRR_2\ Conversion\ Set$ é uma redução do problema $IRR_f\ Conversion\ Set$ onde $\forall v \in V(G) : f(v) = 2$, pode-se afirmar este problema também é NP-Completo em sua versão de decisão.

Este trabalho tem como objetivo estudar aspectos algorítmicos relacionados aos $IRR_f - Process$ e a obtenção de $Hull\ Sets$ mínimos em tempo polinomial, quando possível.

2 Algoritmo Dreyer e Roberts (2009)

O primeiro algoritmo estudado foi proposto por (Dreyer e Roberts, 2009) e tem como objetivo calcular o tamanho do $IRR_2\ Conversion\ Set$ para árvores. Seu funcionamento é baseado em concatenações e na avaliação de subárvores. Para isso um novo conceito precisou ser estabelecido: Um $a - IRR_2\ Conversion\ Set$ é o menor conjunto de vértices da subárvore capaz de converter todos os demais para o estado 1, com exceção da raiz, que pode ou não ser convertida.

Para cada subárvore três informações são mantidas: T, r e S que representam, respectivamente, o conjunto de vértices da subárvore, sua raiz, e um $a - IRR_2\ Conversion\ Set$.

Supondo duas subárvores $T_1 = (T_1, r_1, S_1)$ e $T_2 = (T_2, r_2, S_2)$, tais que $T_1 \cap T_2 = \emptyset$. A concatenação de T_1 e T_2 se dá pela adição de uma aresta entre r_1 e r_2. O resultado desta operação é uma nova subárvore $T = (T_1 \cup T_2, r_1, S_1 \cup S_2)$, de maior tamanho e enraizada em r_1.

Para construir o algoritmo, é necessário caracterizar todas as possíveis subárvores $T = (T, r, S)$. Supondo que o problema é obter o tamanho do $IRR_2\ Conversion\ Set$, existem quatro possíveis classes de T.

$$Classe\ (T, r, S)\ = \begin{cases} 0, & \text{se} \quad r \in S, \\ 1, & \text{se} \quad r \notin S \wedge deg(r) \geq 2, \\ 2, & \text{se} \quad r \notin S \wedge deg(r) = 1, \\ 3, & \text{se} \quad r \notin S \wedge deg(r) = 0. \end{cases}$$

As classes apresentadas são específicas para o problema $IRR_k\ Conversion\ Set$, onde $k = 2$. Para resolver problemas nos quais k é superior a 2 são necessárias mais k classes do que as definidas.

Ao concatenar duas subárvores cujas classes são previamente conhecidas, a classe da subárvore resultante pode ser calculada de forma simples. A Figura 1 apresenta todas as possíveis concatenações e a classe resultante. As arestas pontilhadas representam a aresta que une as duas subárvores. Os nós pretos, brancos e cinzas são, respectivamente, os vértices pertencentes ao conjunto S, os não pertencentes e os vértices adicionados ao conjunto S para converter r_2 (em alguns casos o próprio r_2 foi incluído em S).

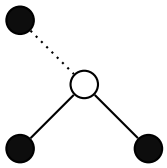
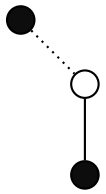
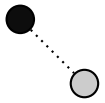
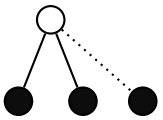
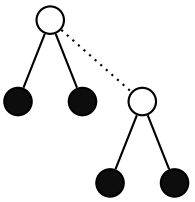
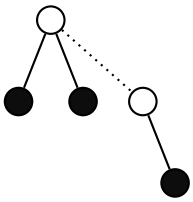
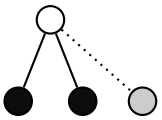
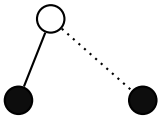
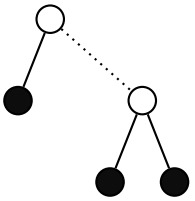
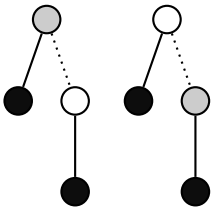
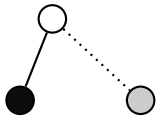
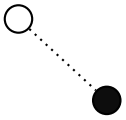
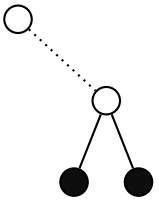
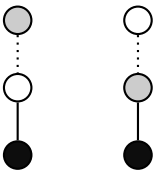
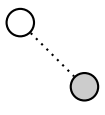
	[0]	[1]	[2]	[3]
[0]	 [0]	 [0]	 [0]	 [0] $(r_2 \in S)$
[1]	 [1]	 [1]	 [1]	 [1] $(r_2 \in S)$
[2]	 [1]	 [1]	 [0]$(r_1 \in S)$ [1]$(r_2 \in S)$	 [1] $(r_2 \in S)$
[3]	 [2]	 [2]	 [0]$(r_1 \in S)$ [2]$(r_2 \in S)$	 [2] $(r_2 \in S)$

Figura 1: Classes Resultantes Após Concatenação

Como dito anteriormente, o algoritmo é baseado na concatenação de vértices e na avaliação das possíveis classes da subárvore formada. Sempre que uma concatenação é efetuada o algoritmo avalia a nova subárvore em busca do menor conjunto S possível.

Inicialmente, o algoritmo efetua uma busca em largura na árvore T, a partir de um vértice arbitrário v_1, rotulando os vértices a medida em que são alcançados pela busca. Este procedimento garante que cada vértice possuirá um rótulo maior do que seus ancestrais na árvore.

Em seguida, o algoritmo passará a concatenar subárvores. No primeiro momento, cada vértice constituí uma subárvore. Para estes casos, o tamanho dos $a - IRR_2\ Conversion\ Sets$ é previamente conhecido para todas as classes possíveis:

Classe 0 O vértice pertence ao conjunto, logo seu tamanho é 1;

Classe 1 Impossível, pois o vértice está isolado. Neste caso, considera-se seu tamanho infinito;

Classe 2 Idem Classe 1;

Classe 3 O vértice não pertence ao conjunto, que ainda assim é um $a-IRR_2Conversion\ Set$. Desta forma, seu tamanho é 0.

A partir deste ponto inicial, o algoritmo passará a concatenar as subárvores duas-a-duas, seguindo uma ordenação estabelecida de forma decrescente pelos rótulos atribuídos na busca em largura. A cada iteração o algoritmo unirá uma subárvore, de raiz r com o vértice pai de r. Desta forma, na primeira iteração a última folha será concatenada ao seu pai. Este processo será repetido até que reste apenas uma subárvore, que conterá todos os vértices.

Ao concatenar duas subárvores deve-se analisar para cada classe diversas combinações possíveis para localizar a melhor opção, isto é, a opção que minimize o conjunto S resultante. Na Figura 1 podem ser vistos os resultados de cada combinação possível.

As expressões abaixo apresentam como o tamanho do conjunto de cada classe possível para a subárvore resultante pode ser obtido. Pode-se notar que não é possível obter a classe 3 em uma concatenação, pois a adição da aresta entre as raízes impossibilita a raiz possuir grau 0. Desta forma, assume-se tamanho infinito para o conjunto desta classe.

$$[0] = Min \begin{cases} [0] \circ [0] \\ [0] \circ [1] \\ [0] \circ [2] \\ [0] \circ [3]^2 \\ [2] \circ [2]^1 \\ [3] \circ [2]^1 \end{cases} \quad (2) \qquad [1] = Min \begin{cases} [1] \circ [0] \\ [1] \circ [1] \\ [1] \circ [2] \\ [1] \circ [3]^2 \\ [2] \circ [0]^2 \\ [2] \circ [1] \\ [2] \circ [2] \\ [2] \circ [3]^2 \end{cases} \quad (3)$$

$$[2] = Min \begin{cases} [3] \circ [0] \\ [3] \circ [1] \\ [3] \circ [2]^2 \\ [3] \circ [3]^2 \end{cases} \quad (4) \qquad [3] = \infty \quad (5)$$

Uma vez detalhado o princípio de funcionamento, pode-se apresentar o algoritmo para a obtenção do tamanho do $IRR_2\ Conversion\ Set$ para árvores. Para armazenar o tamanho do $Hull\ Set$ obtido por cada classe foi utilizada uma matriz de dimensões n por 4, onde n representa o número

[1] Foi necessário incluir r_1 em S

[2] Foi necessário incluir r_2 em S

de vértices da árvore. Além disso, um vetor, denominado Pai foi utilizado para indicar o pai de cada vértice. O algoritmo referencia cada vértice da árvore através do rótulo a ele associado pela busca em largura, iniciada na raiz da árvore.

O tamanho do $Hull\ Set$ corresponde ao menor conjunto encontrado na última concatenação, realizada entre a raiz da árvore e um de seus filhos, dentre os obtidos nas classes 0 e 1.

Algoritmo 1: $IRR_2\ ConversionSet$ - Árvores

Entrada: Árvore $T(V, E)$, Vetor Pai
Saída: Tamanho do $IRR_2\ Conversion\ Set$

Aplicar Busca em Largura em T, rotulando os vértices conforme este são alcançados
para cada $v \in V(T)$ **faça**
    $Classe[v] = (1, \infty, \infty, 0)$
fim
para $j = 0$ **até** $n - 2$ **faça**
    $v = v_{n-j}$
    $pai = Pai[v]$
    Combinar($Classe$,pai,v);
fim
retorne `Min` $(Classe[1, 0], Classe[1, 1])$

Procedimento Combinar($Classe$,pai,v)

Entrada: Vetor de Classes $Classe$, Vértices pai e v
Saída: Vetor de Classes $Classe$

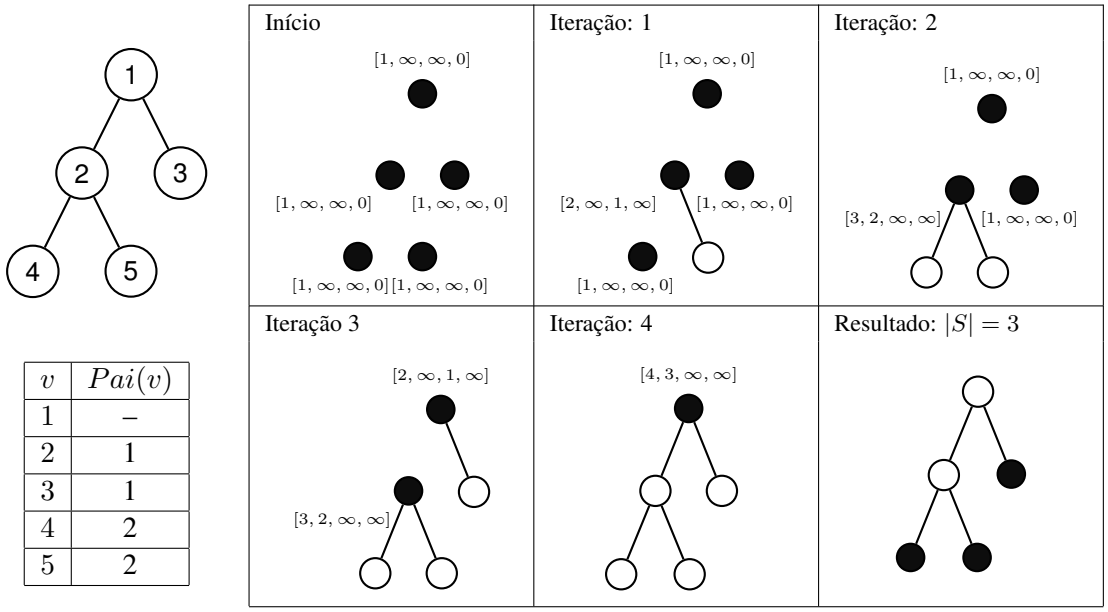
$Classe'[0] =$ `Min` $(Classe[pai, 0] + Classe[v, 0], Classe[pai, 0] + Classe[v, 1],$
$Classe[pai, 0] + Classe[v, 2], Classe[pai, 0] + Classe[v, 3] + 1,$
$Classe[pai, 2] + Classe[v, 2] + 1, Classe[pai, 3] + Classe[v, 2] + 1)$
$Classe'[1] =$ `Min` $(Classe[pai, 1] + Classe[v, 0], Classe[pai, 1] + Classe[v, 1],$
$Classe[pai, 1] + Classe[v, 2], Classe[pai, 1] + Classe[v, 3] + 1,$
$Classe[pai, 2] + Classe[v, 0], Classe[pai, 2] + Classe[v, 1],$
$Classe[pai, 2] + Classe[v, 2] + 1, Classe[pai, 2] + Classe[v, 3] + 1)$
$Classe'[2] =$ `Min` $(Classe[pai, 3] + Classe[v, 0], Classe[pai, 3] + Classe[v, 1],$
$Classe[pai, 3] + Classe[v, 2] + 1, Classe[pai, 3] + Classe[v, 3] + 1)$
$Classe'[3] = \infty$
para $j = 0$ **até** 3 **faça**
    $Classe[pai, j] = Classe'[j]$
fim

A complexidade deste algoritmo é $O(n)$, tendo em vista que cada vértice é considerado apenas uma vez e que o procedimento Combinar possui complexidade $O(1)$. Seu comportamento é ilustrado na Figura 2.

2.1 Obtenção do $IRR_2\ Conversion\ Set$

Como pode ser observado, o Algoritmo 1 retorna apenas o tamanho do $Hull\ Set$, sem fornecelo. Para obter este conjunto, algumas modificações devem ser implementadas no algoritmo original.

A primeira alteração visa armazenar a cada concatenação entre um vértice v e seu pai qual a classe de v que minimizou cada classe resultante do vértice pai. Esta estrutura será utilizada para recompor as decisões tomadas pelo algoritmo, retornando o $Hull\ Set$.



v	$Pai(v)$
1	–
2	1
3	1
4	2
5	2

Figura 2: Exemplo do Algoritmo 1

O Procedimento *CombinarMod* corresponde ao procedimento original acrescido das alterações necessárias. Duas novas estruturas de dados foram introduzidas. A primeira foi utilizada para armazenar o identificador das classes dos vértices pai e v. Esta estrutura possui 3 atributos: $Classe_{Pai}$, $Classe_v$ e $Valor$. A função *Min* obtém o elemento que possui o menor atributo $Valor$. Já a segunda estrutura corresponde a uma lista que manterá o histórico de todas as concatenações realizadas. Cada elemento desta lista possui 5 atributos: V, Pai, Min_{C0}, Min_{C1} e Min_{C2}.

Para construir o IRR_2 $Conversion\ Set$ basta percorrer o histórico de concatenações, construído durante a execução do Algoritmo 1, da raiz até as folhas. A cada recursão um vértice é analisado. Para isso, o momento em que este vértice foi concatenado com seu pai deve ser recuperado do histórico. É importante notar que a classe do vértice pai já é conhecida, tendo em vista que o procedimento percorre a árvore de cima para baixo.

Uma vez que as informações da concatenação do vértice v com seu pai foram recuperadas e que a classe do vértice pai é conhecida, basta encontrar qual classe de v minimizou a classe resultante de pai. Seja c essa classe. Se $c = 0$, então v deve ser incluído no conjunto resultante. Do contrário, v não pertence ao $Hull\ Set$. Em seguida, o algoritmo deve ser chamado recursivamente para todos os filhos de v, utilizando c como classe do vértice pai.

O Algoritmo 2 apresenta a versão modificada do Algoritmo 1 para retornar o $Hull\ Set$.

Histórico Construído na Figura 2				
Pai	V	Min_{C0}	Min_{C1}	Min_{C2}
2	5	0	0	0
2	4	0	0	0
1	3	0	0	0
1	2	1	1	0

$IRR_2\ Conversion\ Set = \{3, 4, 5\}$

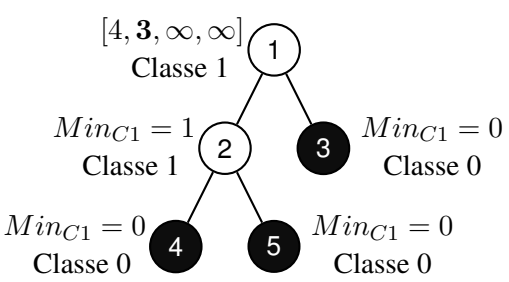


Figura 3: Exemplo da Obtenção do $IRR_2\ Conversion\ Set$

Procedimento CombinarMod($Classe$,pai,v)

Entrada: Vetor de Classes $Classe$, Vetor $Historico$ e Vértices pai e v
Saída: Vetor de Classes $Classe$, Vetor $Historico$

$Classe'[0] =$ `Min` $((0, 0, Classe[pai, 0] + Classe[v, 0]),$
$(0, 1, Classe[pai, 0] + Classe[v, 1]),\ (0, 2, Classe[pai, 0] + Classe[v, 2]),$
$(0, 3, Classe[pai, 0] + Classe[v, 3] + 1),\ (2, 2, Classe[pai, 2] + Classe[v, 2] + 1),$
$(3, 2, Classe[pai, 3] + Classe[v, 2] + 1))$
$Classe'[1] =$ `Min` $((1, 0, Classe[pai, 1] + Classe[v, 0]),$
$(1, 1, Classe[pai, 1] + Classe[v, 1]),\ (1, 2, Classe[pai, 1] + Classe[v, 2]),$
$(1, 3, Classe[pai, 1] + Classe[v, 3] + 1),\ (2, 0, Classe[pai, 2] + Classe[v, 0]),$
$(2, 1, Classe[pai, 2] + Classe[v, 1]),\ (2, 2, Classe[pai, 2] + Classe[v, 2] + 1),$
$(2, 3, Classe[pai, 2] + Classe[v, 3] + 1))$
$Classe'[2] =$ `Min` $((3, 0, Classe[pai, 3] + Classe[v, 0]),$
$(3, 1, Classe[pai, 3] + Classe[v, 1]),\ (3, 2, Classe[pai, 3] + Classe[v, 2] + 1),$
$(3, 3, Classe[pai, 3] + Classe[v, 3] + 1))$
$Classe'[3] = (-1, -1, \infty)$
para $j = 0$ **até** 3 **faça**
  $Classe[pai, j] = Classe'[j].Valor$
fim
$X = (pai, v, Classe'[0].Classe_v, Classe'[1].Classe_v, Classe'[2].Classe_v)$
`Inserir` ($Historico$, X)

Função ConstruirConversionSet(v, Cp, H)

Entrada: Vértice v, Classe do pai de v Cp, Histórico de Concatenações H
Saída: $IRR_2\ Conversion\ Set\ \ S$

$X =$ `Buscar` (H, $V = v$)
$c = X.Min_{Cp}$
$S = \emptyset$
se $c = 0$ **então**
  $S = S \cup \{v\}$
fim
para cada $m \in$ `Buscar` *(H, $Pai = v$)* **faça**
  $S = S \cup$ ConstruirConversionSet($m.V$, c, S)
fim
retorne S

Algoritmo 2: $IRR_2\ Conversion\ Set$ - Árvores - Modificado

Entrada: Árvore $T(V, E)$, Vetor Pai
Saída: $IRR_2\ Conversion\ Set\ \ S$

Aplicar Busca em Largura em T, rotulando os vértices conforme este são alcançados
para cada $v \in V(T)$ **faça**
    $Classe[v] = (1, \infty, \infty, 0)$
fim
para $j = 0$ **até** $n - 2$ **faça**
    $v = v_{n-j}$
    $pai = Pai[v]$
    `CombinarMod` ($Classe$, pai, v, H);
fim
$S = \emptyset$
se $Classe[1, 0] < Classe[1, 1]$ **então**
    $ClasseRaiz = 0$
    $S = S\ \cup \{v_1\}$
senão
    $ClasseRaiz = 1$
fim
para cada $v \in \{V(T) | Pai(v) = v_1\}$ **faça**
    $S = S \cup$ ConstruirConversionSet(v, $ClasseRaiz$, H, S)
fim
retorne S

2.2 Contraexemplo do Algoritmo

Centeno et al. (2011) apresentou um contraexemplo para os Algoritmos 1 e 2. Para algumas árvores a aplicação deste procedimento resulta em um resultado incorreto. A Figura 4 ilustra uma destas árvores.

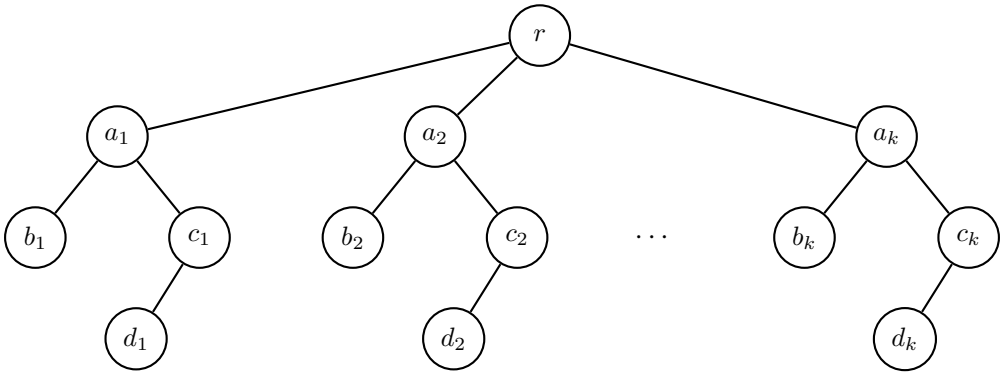


Figura 4: Contra-Exemplo Algoritmos 1 e 2

Os Algoritmos 1 e 2 quando executados tendo como entrada a árvore apresentada na Figura 4 retornam 4, quando $k = 1$, e $3k$, quando $k \geq 2$. Contudo a solução ótima é $2k + 1$. Supondo $k \geq 2$, o Algoritmo 2 retornará o conjunto $\{a_1, b_1, d_1, a_2, b_2, d_2, \cdots, a_k, b_k, d_k\}$, enquanto a solução ótima corresponde ao conjunto $\{r, b_1, d_1, b_2, d_2, \cdots, b_k, d_k\}$.

Uma vez demonstrado que este algoritmo não funciona para alguns casos, tornou-se necessário o estudo de outras soluções para a obtenção de $Hull\ Sets$ mínimos.

3 Algoritmo Centeno et al. (2011)

A próxima solução estudada para obtenção de $Hull\ Sets$ mínimos foi proposta por (Centeno et al., 2011) e se baseia na divisão do grafo em blocos folha. Um bloco folha é um bloco que contém no máximo uma articulação do grafo. Seu funcionamento se baseia no seguinte lema:

Lema 3.1 (Centeno et al., 2011)**.** *Seja G um grafo que não é 2-conexo e seja $f : V(G) \to \mathbb{Z}$ uma função. Seja B um bloco folha de G e seja u a única articulação de G em B. Seja $G' = G - (V(B) \{u\})$. Seja $c_B = irr_{f|V(B)}(B)$.*

i *Se não existir nenhum $IRR_f - Conversion\ Set\ C_B$ de B tal que $u \in C_B$ e $|C_B| = c_B$ então*

$$irr_f(G) = irr_{f'}(G') + c_B,$$

onde $f' : V(G') \to \mathbb{Z}$ é tal que

$$f'(x) = \begin{cases} 0, & se \quad x = u, \\ f(x), & se \quad x \in V(G'). \end{cases}$$

Além disso, se C' for um $IRR_{f'}\ Conversion\ Set$ de G' e C_B for um $IRR_{f|V(B)}\ Conversion\ Set$ de B então $C' \cup\ C_B$ é um $IRR_f\ Conversion\ Set$ de G.

ii *Se C_B for um $IRR_{f|V(B)}\ Conversion\ Set$ de B com $u \in C_B$ e $|C_B| = c_B$ tal que, dentre todos estes conjuntos, C_B foi escolhido de forma que $C_B \setminus \{u\}\ IRR_{f|V(B)} - converta$ o maior número possível d_B de vizinhos de u em B, então*

$$irr_f(G) = irr_{f''}(G') + (c_B - 1),$$

onde $f'' : V(G') \to \mathbb{Z}$ é tal que

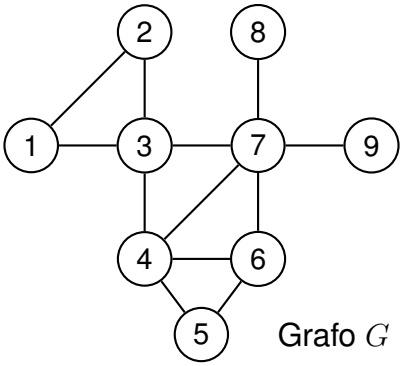
$$f''(x) = \begin{cases} f(x) - d_B, & se \quad x = u, \\ f(x), & se \quad x \in V(G') \setminus \{u\}. \end{cases}$$

Além disso, se C'' for um $IRR_{f''}\ Conversion\ Set$ de G', então $C'' \cup (C_B \setminus \{u\})$ é um $IRR_f\ Conversion\ Set$ de G.

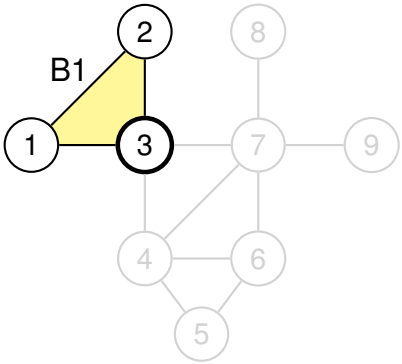
O Lema 3.1 apresenta uma forma se obter o $Hull\ Set$ mínimo dividindo o grafo em blocos folha e solucionando um bloco de cada vez. Além disso, demonstra também como mesclar os conjuntos resultantes em um único conjunto. O Algoritmo 3 resolve o problema baseado neste princípio.

O Algoritmo 3 é linear assumindo que o tamanho do maior bloco é limitado por uma constante n_0. Para tal, basta considerar que as linhas 2, 10 e 17 podem ser realizadas em $O(2^{n_0})$, que é um valor constante e que a decomposição do grafo em blocos pode ser realizada em $O(n + m)$, por um procedimento de busca em profundidade. A Figura 5 apresenta um exemplo da aplicação deste algoritmo.

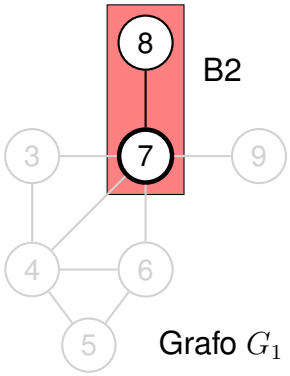
Existem casos em que encontrar a solução para cada bloco pode ser otimizada, utilizando conhecimentos sobre a estrutura destes blocos. Um exemplo desta afirmação são as árvores, onde cada bloco possui no máximo 2 vértices. Considerando este aspecto, foram desenvolvidos algoritmos para determinados tipos de blocos e também para algumas famílias de grafos.



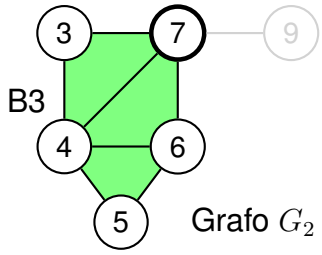
v	1	2	3	4	5	6	7	8	9
$f(v)$	2	2	2	3	1	3	2	1	1



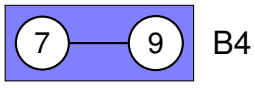
$S = \{1, 3\}$ Caso (ii) do Lema 3.1
$f'(3) = 1$ $G_1 = G - (B1 \setminus \{3\})$



$S = \{7\}$ Caso (ii) do Lema 3.1
$f''(7) = 2$ $G_2 = G_1 - (B2 \setminus \{7\})$

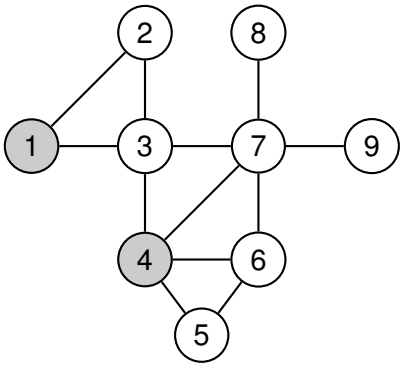


$S = \{4\}$ Caso (i) do Lema 3.1
$f'(7) = 0$ $G_3 = G_2 - (B3 \setminus \{7\})$



Grafo G_3

Grafo 2-Conexo $S = \{\}$
Caso Base da Recusão



$IRR_f(G) = \{1, 4\}$

Figura 5: Exemplo da Aplicação do Algoritmo 3.

Algoritmo 3: IRR-Converter(G,f)

Entrada: Grafo G e Função *Threshold* f
Saída: $IRR_f\ Conversion\ Set\ \ S$

se *G é 2-Conexo* **então**
  Determine o menor $IRR_f\ Conversion\ Set\ \ C$ de G
  retorne C
fim
Sejam B, u e G' conforme definido no Lema 3.1
Determine $c_B = irr_{f|V(B)}(B)$
se *não existe nenhum $IRR_f\ Conversion\ Set\ C_B$ de B tal que $u \in C_B$ e $|C_B| = c_B$* **então**
  Seja f' como definido no Lema 3.1 - (i)
  Seja C' = IRR-Converter(G',f')
  Determine o menor $IRR_{f|V(B)}\ Conversion\ Set\ \ C_B$ de B
  retorne $C' \cup C_B$
fim
se *existe algum $IRR_f\ Conversion\ Set\ C_B$ de B tal que $u \in C_B$ e $|C_B| = c_B$* **então**
  Seja f'' como definido no Lema 3.1 - (ii)
  Seja C'' = IRR-Converter(G',f'')
  Determine d_B conforme apresentado no Lema 3.1 - (ii)
  Determine o menor $IRR_{f|V(B)}\ Conversion\ Set\ \ C_B$ de B tal que
   $C_B \setminus \{u\}\ IRR_{f|V(B)}\ converta\ d_B$ vizinhos de u em B
  retorne $C' \cup (C_B \setminus \{u\})$
fim

4 Algoritmos Desenvolvidos

Para algumas famílias de grafos a obtenção de $IRR_f\ Conversion\ Sets$ pode ser otimizada ao incorporar nos algoritmos algumas características destas famílias. Esta seção apresentará algoritmos obtidos para árvores, grafos completos, ciclos e grafos cactus. Além disso conjectura um algoritmo para obter $Hull\ Sets$ mínimos de grafos de intervalo. Alguns destes algoritmos foram inspirados no Algoritmo 3.

4.1 Árvores

Considerando que o algoritmo apresentado por Dreyer e Roberts, 2009 não identifica corretamente um $HullSet$ mínimo em alguns casos, um novo procedimento foi desenvolvido, tendo como base o Algoritmo 3.

Conforme exposto anteriormente, o algoritmo definido por Centeno et al., 2011 é baseado na decomposição do grafo em blocos folha, isto é, blocos que contém no máximo uma articulação do grafo. Esta decomposição é elementar em árvores: Cada vértice folha unido ao seu ancestral direto constitui um bloco folha. Além disso, ao remover todos os blocos folha que contém esse ancestral, ele próprio unido ao seu ancestral passa a constituir um novo bloco folha.

Além disso, cada bloco folha de uma árvore é formado por no máximo dois vértices: um vértice folha v e seu pai u, caso este exista. Para encontrar o $Hull\ Set$ mínimo deste bloco, a solução trivial é inicialmente considerada, isto é, $S = \{u, v\}$. Em seguida, analisa-se a possibilidade de remover o vértice v do conjunto. Para que isso seja possível é necessário que $f(v) \leq 1$ e $u \in S$, isto é, v não é necessário pois será convertido pelo vértice u. Finalmente, o algoritmo verifica a possibilidade de remover o vértice u do conjunto. Essa remoção só será possível se um dos seguintes casos ocorra:

(i) $(v \in S) \wedge (f(u) \leq 1)$, isto é, o vértice u será convertido pelo vértice v; (ii) $(v \notin S) \wedge (f(u) = 0)$, isto é, o vértice u é convertido automaticamente e na iteração seguinte, converterá o vértice v; (iii) $(v \notin S) \wedge (f(v) = 0) \wedge (f(u) = 1)$, o que implica em v ser auto-convertido, logo não pertence ao $Hull\ Set\ S$, e que u será convertido por v.

Função $HSK2(f, B, v, u)$

Entrada: Função *Threshold* f, Bloco Folha B com dois vértices, Vértice folha v e Articulação u

Saída: Conjunto $IRR_f\ S$

```
1  S = {u, v}
2  se f(v) ≤ 1 então
3  |  S = S \ {v}
4  fim
5  se f(u) ≤ 1 então
6  |  se (v ∈ S) ∨ (f(u) = 0) ∨ (f(v) = 0) então
7  |  |  S = S \ {u}
8  |  fim
9  fim
10 retorne S
```

Uma vez que seu princípio de funcionamento foi detalhado, o algoritmo para obtenção de $IRR_f\ Conversion\ Set$ para árvores pode ser apresentado. Conforme dito anteriormente, este procedimento é baseado no Algoritmo 3.

A complexidade do Algoritmo 4 é $O(n)$, pois cada vértice é considerado apenas uma vez e a obtenção do $Hull\ Set$ mínimo de cada bloco é realizada em $O(1)$ operações. Para provar a corretude do Algoritmo 4 é necessário primeiro provar que o algoritmo $HSK2$ está correto.

Lema 4.1. *O Algoritmo $HSK2$ encontra o $Hull\ Set$ mínimo para $(B, f_{V(B)})$, onde B e $f_{V(B)}$ correspondem respectivamente a um bloco com dois vértices para o qual se busca o $Hull\ Set$ mínimo e a função Threshold restrita aos vértices de B. Além disso, sempre que possível o $Hull\ Set$ mínimo que contenha a articulação do bloco será retornado.*

Prova. Considerando que o bloco contém apenas dois vértices, seja u a articulação e v o outro vértice. Existem seis casos distintos aos analisar os *Thresholds* dos vértices:

Caso 1 $f(u) \geq 2 \wedge f(v) \geq 2$. Neste caso, o $Hull\ Set$ para $(B, f_{V(B)})$ deverá conter ambos os vértices, já que ambos não podem ser convertidos por nenhum $Hull\ Set$ que não os contenha. Neste caso, o algoritmo ao analisar os *Thresholds* dos vértices notará que ambos são superiores a 1 e não efetuará a remoção de nenhum do conjunto inicial, que contém os dois vértices. Provando a corretude do algoritmo neste caso.

Caso 2 $f(u) \geq 2 \wedge f(v) \leq 1$. Neste caso, o $Hull\ Set$ para $(B, f_{V(B)})$ deverá conter o vértice u, uma vez que ele não pode ser convertido por nenhum $Hull\ Set$ que não o contenha. Além disso, o vértice v será convertido por u. Ao analisar o comportamento do algoritmo para este caso, nota-se que o vértice v será removido do conjunto pois possuí *Threshold* menor ou igual a 1. Em seguida, ao perceber que o *Threshold* de u é maior que 2, o algoritmo encerrará sua execução, retornando o conjunto $\{u\}$, que é o $Hull\ Set$ mínimo de $(B, f_{V(B)})$.

Caso 3 $f(u) \leq 1 \wedge f(v) \leq 0$. Neste caso, o $Hull\ Set$ para $(B, f_{V(B)})$ é vazio, uma vez que os vértice v é auto-convertido e também é suficiente para converter o vértice v. Neste caso, primeiramente o algoritmo efetuará a remoção do vértice v. Em seguida, ao perceber

Algoritmo 4: $HSArvore$

Entrada: Árvore $T(V, E)$, Função *Threshold* f
Saída: $Hull\ Set\ \ S$

```
se V(T) = ∅ então
    retorne ∅
fim
Seja v uma folha de T
se deg(v) = 0 então
    se f(v) > 0 então
        retorne {v}
    senão
        retorne ∅
    fim
fim
Seja u o pai de v
se u ∉ S' então
    f' = f
    f'(u) = 0
    retorne S' ∪ HSArvore(T − {v}, f')
senão
    f'' = f
    se (v ∈ S') ∨ (f(v) = 0) então
        f''(u) = f(u) − 1
    fim
    retorne (S' \ {u}) ∪ HSArvore(T − {v}, f'')
fim
```

que o *Threshold* de u é menor ou igual a 1, analisará se é possível remover este vértice do conjunto. Essa remoção acontece quando uma das seguintes afirmações é verdadeira: $(v \in S)$, $(f(u) = 0)$ e $(f(v) = 0)$. A terceira afirmativa é verdadeira, uma vez $f(v) \leq 0$, logo o vértice u também será removido do conjunto, que passará a ser vazio, provando que o algoritmo neste caso encontrou o $Hull\ Set$ mínimo.

Caso 4 $f(u) = 1 \wedge f(v) = 1$. Existem dois $Hull\ Sets$ mínimos possíveis: $\{u\}$ e $\{v\}$. Contudo, conforme definido no Lema 3.1, o $Hull\ Set$ mínimo que contenha a articulação deve ser retornado. Logo, o $Hull\ Set$ deste bloco é $\{u\}$. O algoritmo neste bloco, ao perceber que o *Threshold* de v é 1, efetuará a remoção deste vértice. Em seguida, ao verificar que o *Threshold* de u é 1, analisará as seguintes afirmativas: $(v \in S)$, $(f(u) = 0)$ e $(f(v) = 0)$ e como nenhuma é verdadeira, o vértice u não será removido do conjunto, que é um $Hull\ Set$ mínimo do bloco, provando a corretude do algoritmo neste caso.

Caso 5 $f(u) \leq 0 \wedge f(v) \leq 1$. Idêntico ao caso 3.

Caso 6 $f(u) \leq 1 \wedge f(v) \geq 2$. O $Hull\ Set$ de $(B, f_{V(B)})$ deve conter o vértice v, pois v não pode ser convertido por nenhum conjunto que não o contenha. Além disso, o vértice u será convertido por v, logo não deve pertencer ao $Hull\ Set$ mínimo. O algoritmo neste bloco, ao avaliar o *Threshold* do vértice v, não o removerá do conjunto. Na etapa seguinte, ao analisar o *Threshold* de u, este vértice será removido. Ao final, o algoritmo retornará um conjunto que contém apenas o vértice v, demonstrando que o algoritmo encontrou o $Hull\ Set$ mínimo de $(B, f_{V(B)})$.

Uma vez que para todos os casos o algoritmo encontrou o $Hull\ Set$ mínimo e que sempre que possível o vértice u está contido no conjunto retornado, pode-se afirmar que o Algoritmo $HSK2$ está correto. □

Teorema 4.2. *O Algoritmo 4 retorna o $IRR_f\ Conversion\ Set$ para árvores.*

Prova. Considerando que o Lema 3.1 prova a metodologia utilizada para dividir a árvore em blocos folha e combinar os $Hull\ Sets$ mínimos em um único conjunto que é um $Hull\ Set$ mínimo para a árvore e que o Lema 4.1 prova que o $Hull\ Set$ obtido para cada bloco folha é mínimo e sempre que possível contém a articulação do bloco, a prova segue. □

4.2 Grafos Completos

Além das árvores, a obtenção de $Hull\ Sets$ mínimos para grafos completos também foi estudada. A solução proposta para obter o $Hull\ Set$ de um grafo completo é um algoritmo guloso. Inicialmente, o algoritmo adiciona ao conjunto que está sendo construído todos os vértices cuja função *Threshold* seja maior ou igual ao tamanho do grafo. Essa etapa inicial visa tratar os vértice que somente podem ser convertidos se pertencerem ao $Hull\ Set$. Uma vez que estes estes vértices foram incluídos no $Hull\ Set$ eles são removidos do grafo. Em seguida, o valor da função *Threshold* dos demais vértice deve ser atualizada, pois eles já possuem vizinhos pertencentes ao $Hull\ Set$.

Uma vez que este passo inicial tenha sido concluído, podem existir vértices que já são convertidos pelos vértices adicionados ao $Hull\ Set$. Nesta etapa de limpeza, para cada vértice nesta situação, a função *Threshold* dos demais vértices deve ser debitada em uma unidade. Além disso, estes vértices devem ser removidos do grafo. Esta operação deve ser repetida enquanto houver um vértice que pode ser convertido pelo $Hull\ Set$.

As etapas realizadas até o momento constituem um pré-processamento do grafo. Considerando que vértices foram removidos do grafo, a hipótese de que este tenha se tornado vazio deve ser considerada. Neste caso, o conjunto construído foi capaz de converter todos os vértices, isto é, este conjunto é um $Hull\ Set$ para o grafo, e nenhuma outra ação é necessária. Para os demais casos, o algoritmo adotará o comportamento guloso.

A cada iteração, o algoritmo escolhe o vértice com o maior valor de *Threshold*, o adiciona ao conjunto, debita a função *Threshold* dos demais vértices em uma unidade e executa a etapa de limpeza novamente. Este processo é repetido até que o grafo se torne vazio.

O Algoritmo 5 possui complexidade $O(n^2)$, tendo em vista que a cada escolha de vértice a função *Threshold* dos demais deve ser reajustada.

Teorema 4.3. *O Algoritmo 5 encontra o $IRR_f\ Conversion\ Set$ de um grafo completo.*

Demonstração. A prova será realizada por indução no tamanho do grafo.

Base: Seja G um grafo completo com dois vértices. Sejam v e u estes dois vértices. Considerando o *Threshold* de v e u, existem cinco casos distintos:

Caso 1 Os dois vértices possuem *Threshold* maior ou igual à dois. O único $Hull\ Set$ capaz de converter o grafo é formado pelos vértices v e u, uma vez que um único vértice não seria suficiente para induzir o outro a mudar de rótulo. Neste caso, o algoritmo guloso escolheria um dos vértices e na iteração seguinte, incluiria o outro, provando a corretude neste caso.

Caso 2 Um vértice possui *Threshold* maior ou igual à 2 enquanto o outro possui *Threshold* 1. Sem perda de generalidade, seja v o vértice tal que $f(v) \geq 2$ e u o outro vértice. Claramente o vértice v é capaz de converter o vértice u, uma vez que este necessita de apenas um vizinho de rótulo 1 para ser convertido. Contudo, o inverso não é verdadeiro, o vértice u não é suficiente

Algoritmo 5: $HSCompleto$

Entrada: Grafo Completo $K(V, E)$ de tamanho k, Função Threshold f
Saída: $Hull\ Set\ \ S$

```
1  S = {v ∈ V(K) | f(v) ≥ k}
2  S̄ = V(K) \ S
3  para cada v ∈ S̄ faça
4  |  f'(v) = f(v) − |S|
5  fim
6  enquanto S̄ ≠ ∅ faça
7  |  Z = {v ∈ S̄ | f'(v) ≤ 0}
8  |  enquanto Z ≠ ∅ faça
9  |  |  S̄ = S̄ \ Z
10 |  |  para cada v ∈ S̄ faça
11 |  |  |  f'(v) = f'(v) − |Z|
12 |  |  fim
13 |  |  Z = {v ∈ S̄ | f'(v) ≤ 0}
14 |  fim
15 |  se S̄ ≠ ∅ então
16 |  |  Seja v o vértice de maior f'(v) contido em S̄
17 |  |  S = S ∪ {v}
18 |  |  S̄ = S̄ \ {v}
19 |  |  para cada w ∈ S̄ faça
20 |  |  |  f'(w) = f'(w) − 1
21 |  |  fim
22 |  fim
23 fim
24 retorne S
```

para converter v. Desta forma, o $Hull\ Set$ mínimo é constituído pelo vértice v. Neste caso, o critério guloso do algoritmo o levaria a escolher o vértice v, debitando em uma unidade o *Threshold* do vértice u. Na iteração seguinte, ao notar que o vértice u possui *Threshold* 0, este é removido do grafo, que se torna vazio, encerrando o procedimento. Desta forma, o $Hull\ Set$ construído pelo algoritmo é igual ao ótimo, logo o algoritmo encontrou o resultado correto neste caso.

Caso 3 Ambos os vértices possuem *Threshold* 1. Ao incluir um vértice no conjunto, o outro passa a possuir o número de vizinhos de rótulo 1 necessário para que este seja convertido. Desta forma, o conjunto deve possuir apenas um vértice. O algoritmo neste caso detectaria um empate entre os dois vértices e escolheria um deles. Ao incluir um vértice no conjunto, o algoritmo reduziria o *Threshold* do outro em uma unidade. Assim como no caso anterior, na iteração seguinte este vértice de *Threshold* 0 é removido do grafo, que se torna vazio, encerrando o processo. O conjunto construído possui um único vértice, logo seu tamanho é igual ao mínimo, provando a corretude do algoritmo neste caso.

Caso 4 Um vértice possui *Threshold* 1 enquanto o outro, 0. Sem perda de generalidade, seja v o vértice tal que $f(v) = 1$ e u o outro. Neste caso u é convertido automaticamente. Em seguida, o vértice v também é convertido, uma vez que era necessário apenas um vizinho de rótulo 1. Neste caso, o algoritmo, na etapa de limpeza, removeria o vértice u e debitaria o

Threshold de v em uma unidade. Conforme descrito no algoritmo, este processo de limpeza é repetido enquanto houver um vértice com *Threshold* menor ou igual a 0. Sendo assim, na iteração seguinte da limpeza, o vértice v é removido e o grafo se torna vazio. Além disso, o conjunto resultante é vazio, demonstrando que para este caso o algoritmo funciona.

Caso 5 Ambos os vértices possuem *Threshold* 0. A solução deste caso é trivial: Os dois vértices são auto-convertidos e o $Hull\ Set$ é vazio. Ao aplicar o algoritmo neste caso, os dois vértices são removidos na etapa de limpeza e o algoritmo retorna um conjunto vazio, comprovando a corretude neste caso.

Uma vez que em todos os casos o algoritmo retornou o $Hull\ Set$ mínimo, assume-se que para grafos completos com dois vértice o algoritmo é correto.

Hipótese da Indução: O algoritmo encontra o $Hull\ Set$ mínimo para grafos com até $n - 1$ vértices.

Passo da Indução: Seja G um grafo completo com n vértices. Seja v o vértice de G com o maior *Threshold*. Seja X o conjunto de todos os $Hull\ Sets$ mínimos para o grafo G. Ao analisar a presença do vértice v nos $Hull\ Sets$ contidos em X, existem dois casos:

Caso 1 O vértice v está presente em todos os $Hull\ Sets$ contidos em X. Seja S um $Hull\ Set$ qualquer contido em X. Como G é completo, sabe-se que v é adjacente a todos os demais vértices do grafo. Para que todo $Hull\ Set$ mínimo contenha v é necessário que $f(v) >= n$. Do contrário, basta remover v e adicionar dentre seus vizinho não convertidos por $S \setminus \{v\}$ o de maior *Threshold*. Seja S' este novo conjunto. Sabe-se que $|S'| = |S|$, logo S' é mínimo. Assumindo que $f(v) < n$, v também será convertido, uma vez que todo o restante do grafo é convertido por S', contrariando a hipótese de que v pertence a todos os $Hull\ Sets$ contidos em X. Desta forma, o único caso em que v pertence a todos os $Hull\ Sets$ mínimos de G é $f(v) >= n$. Neste caso, o algoritmo incluirá v no conjunto que está sendo construído na etapa de pré-processamento do grafo. Além disso, v é removido do grafo e o *Threshold* dos demais vértices é reduzido em uma unidade. Sendo assim, seja G' o grafo obtido pela remoção de v e f' a função *Threshold* corrigida. Pela hipótese de indução, sabe-se que o algoritmo encontra o $Hull\ Set$ mínimo S'' para (G', f'). Para se obter um $Hull\ Set$ mínimo para G basta adicionar o vértice v ao conjunto S'', uma vez que v não pode ser convertido por este conjunto. Portanto, $|S'' \cup \{v\}|$ é um $Hull\ Set$ mínimo para (G, f), provando a corretude do algoritmo neste caso.

Caso 2 Existe um $Hull\ Set\ S$ tal que $v \notin C$. Existem três casos nos quais $v \notin S$ e S é um $Hull\ Set$ mínimo de (G, f).

Caso 2.1 $f(v) = 0$. Neste caso, o $Hull\ Set$ mínimo de G é vazio, pois como v possui *Threshold* máximo, todos os demais vértices também possuem *Threshold* igual a 0. O algoritmo neste caso, durante a etapa de limpeza, remove todos os vértices do grafo, que ao final se torna vazio. Este processo termina sem que nenhum vértice tenha sido adicionado ao $Hull\ Set$, portanto o algoritmo funciona neste caso.

Caso 2.2 $f(v) > 0$ e $S = \emptyset$. Para que $S = \emptyset$ é necessário que exista um conjunto de vértices cujo *Threshold* seja 0. Ao aplicar a regra do processo irreversível por um determinado número de vezes todos os vértices, incluindo v, são convertidos. Neste caso, o algoritmo durante a etapa de limpeza, removerá este conjunto de vértices do grafo e debitará seu tamanho do *Threshold* dos demais. Ao final deste procedimento, novos vértices passarão a ter *Threshold* igual à 0 e a limpeza será repetida. Ao final de várias iterações, não existirá mais nenhum vértice cujo *Threshold* seja 0. É importante

notar que o procedimento de limpeza possui o mesmo comportamento da regra do processo irreversível considerado. Desta forma, pode-se afirmar que ao final do processo de limpeza, o grafo será vazio, terminando o algoritmo com $Hull\ Set$ vazio, provando a corretude do algoritmo neste caso.

Caso 2.3 $f(v) > 0$ e $S \neq \emptyset$. Neste caso, o vértice v é convertido pelos demais vértices, logo qualquer S também é um $Hull\ Set$ de $(G - v, f_{V(G)\setminus\{v\}})$. Desta forma, seja $G' = G - v$. Seja S' o $Hull\ Set$ obtido pelo algoritmo para $(G', f_{V(G)\setminus\{v\}})$. Como G' possui $n - 1$ vértices, pela hipótese da indução, S' é mínimo. E como S' converte todos os vizinhos de v, v também será convertido, implicando que S' seja um $Hull\ Set$ mínimo de (G, f), logo $|S| = |S'|$ e o algoritmo encontrou o $Hull\ Set$ mínimo para este caso.

Considerando que em todos os casos o algoritmo encontrou o $Hull\ Set$ mínimo de G, pode-se afirmar que o Algoritmo 5 está correto.

□

4.3 Ciclos

Além dos grafos completos, a família dos grafos ciclos também foi estudada em busca de um algoritmo para a obtenção de $IRR_f\ Conversion\ Sets$ em tempo polinomial.

Inicialmente o algoritmo efetua uma fase de pré-processamento na qual todos os vértices que possuem *Threshold* maior que dois são incluídos no conjunto em construção. Assim como nos grafos completos, essa etapa visa converter os vértices que não podem ser convertidos por seus vizinhos. Para cada vértice v que tenha sido adicionado nesta etapa, o algoritmo reduz o *Threshold* de seus vizinhos em uma unidade e remove este vértice v do grafo.

Em seguida, o algoritmo buscará por vértices que já tenham sido convertidos pelos vértices adicionados na etapa anterior. Para cada vértice que já tenha sido convertido, o algoritmo também debitará em uma unidade o *Threshold* dos seus vizinhos e removerá este vértice do grafo. Este processo será repetido até que nenhum vértice seja convertido por este processo. Se ao final deste processo, o grafo for vazio, o conjunto construído corresponde à um $Hull\ Set$ mínimo deste grafo e o processo é encerrado.

Do contrário, existem dois casos possíveis. No primeiro, as etapas anteriores não converteram nenhum vértice do grafo, que continua sendo um ciclo. Já no segundo, o grafo deixou de ser um ciclo e passou a ser uma coleção de caminhos (pelo menos um). Claramente o primeiro caso pode ser convertido no segundo removendo-se um vértice do grafo. Para determinar qual vértice será removido um critério guloso será utilizado.

Para cada vértice um valor inteiro, denominado potencial, será calculado da seguinte forma: Sejam v, u e w, respectivamente, o vértice para o qual o potencial está sendo calculado e seus dois vizinhos:

$$Potencial[v] = \begin{cases} 2, & \text{se} \quad f(v) = 2, \\ 0, & \text{se} \quad f(v) < 2. \end{cases} \tag{6}$$

$$Potencial[v] += \begin{cases} 0, & \text{se} \quad f(u) = 2 \wedge f(w) = 2, \\ 1, & \text{se} \quad f(u) = 1 \vee f(w) = 1, \\ 2, & \text{se} \quad f(u) = 1 \wedge f(w) = 1. \end{cases} \tag{7}$$

O calculo do potencial corresponde no algoritmo à função *CalcularPotencial(v,u,w)*. Se um de seus vizinhos não existir este será desconsiderado na segunda equação. A motivação do uso do potencial é puramente gulosa: O vértice de maior *Threshold* com a maior capacidade de converter seus vizinhos terá maior potencial e será escolhido preferencialmente.

Uma vez que os potenciais foram calculados, o algoritmo escolherá um vértice de maior potencial, o adicionará ao conjunto, debitará o *Threshold* de seus vizinhos em uma unidade e o removerá do grafo. Este processo rompe o ciclo, criando um único caminho. A partir deste momento, apenas um caso restou: O grafo passou a ser uma coleção de caminhos.

Em seguida, enquanto o grafo contiver vértices, o algoritmo selecionará um caminho contido na coleção de caminhos e o processará. O processamento de um caminho é constituído de duas fases. Na primeira, o algoritmo analisará os extremos do caminho. Seja v um dos extremos do caminho, tal que $f(v) \leq 0 \vee f(v) = 2 \vee Adj(v) = \emptyset$. Se v for um vértice isolado, basta adicionar v ao conjunto e o remover do grafo. Caso contrário, se $f(v) = 2$ então v deve ser adicionado ao conjunto, pois seu único vizinho w não o converterá. Além disso, o *Threshold* de w deve ser debitado em uma unidade e v deve ser removido do grafo. Quando $f(v) = 0$, v deve ser removido do grafo e, assim como no caso anterior, o *Threshold* de w deve ser reduzido em uma unidade.

Ao final deste processo, ou o caminho deixou de existir ou possui dois extremos com *Threshold* igual a um. No primeiro caso, basta selecionar outro caminho ou terminar o processo se o grafo for vazio. Já no segundo caso é necessário analisar o interior deste caminho.

Para isso, o algoritmo calculará os potenciais dos vértices e escolherá o vértice de maior potencial. Sejam v, u e w, respectivamente, o vértice escolhido e seus dois vizinhos. O vértice v deve ser adicionado ao conjunto, removido do grafo e os *Threshold* de u e w debitados em uma unidade, se estes vértices existirem. Se o vértice v for um extremo do caminho, este teve seu tamanho reduzido em um vértice. Caso contrário, o caminho foi quebrado em dois novos caminhos. Em ambos os casos, o algoritmo deve retornar ao início, selecionado um outro caminho para ser analisado. Este processo será repetido enquanto o grafo contiver vértices.

A complexidade do Algoritmo 6 é $O(n^2)$, considerando que cada vértice é analisado apenas uma vez e que a complexidade da função $CalcularPotenciais$ é $O(n)$.

Tendo em vista que o Algoritmo 6 decompõe o ciclo em uma coleção de caminhos, é necessário inicialmente provar sua corretude para estes casos.

Lema 4.4. *O Algoritmo 6 obtém o IRR_f $Conversion\ Set$ de um caminho P com pelo menos dois vértices cujos extremos possuem Threshold 1 e todos os demais vértices possuem Threshold 1 ou 2.*

Prova. A prova será feita por indução no tamanho do caminho P.

Base: O caminho P possui dois vértices. O caminho possui 2 vértices que possuem *Threshold* 1. Neste caso, qualquer um destes vértices é capaz de converter o outro, caso esteja no $Hull\ Set$. Sendo assim, o $Hull\ Set$ deste caminho possui apenas um vértice. Neste caso, o algoritmo efetuaria o cálculo dos potenciais e como ambos os vértices possuem o mesmo potencial, escolheria qualquer um dos dois vértices. Ao escolher um vértice, este é adicionado ao conjunto que está sendo construído e o *Threshold* de seu vizinho, debitado em uma unidade, que passará a possui *Threshold* 0. Na iteração seguinte, durante a etapa de limpeza, o algoritmo detectaria um vértice isolado. Ao avaliar o *Threshold* deste vértice e verificar que é inferior à 1, este vértice é removido do caminho, que passa a ser vazio, encerrando o processo. Sendo assim, o algoritmo retorna um $Hull\ Set$ contendo apenas um vértice, provando a corretude do algoritmo neste caso.

Hipótese: O Algoritmo 6 obtém o $Hull\ Set$ mínimo para caminhos com até $n - 1$ vértices.

Passo da Indução: Seja P um caminho com $n \geq 3$ vértices, tal que todo vértice de P possui *Threshold* 1 ou 2, exceto os extremos cujos *Thresholds* são obrigatoriamente 1. Ao analisar o caminho, pode-se perceber que nenhum $Hull\ Set$ vazio pode converter P, uma vez que P não possui vértices de *Threshold* 0. Desta forma, pelo menos um vértice de P deve ser adicionado ao $Hull\ Set$ para converter P. A cada iteração, um vértice dentre os não convertidos pelo $Hull\ Set$ deve ser adicionado ao conjunto para garantir que ao final de um determinado número de iterações o conjunto

construído seja um $Hull\ Set$ mínimo de P. Uma estratégia interessante para definir qual vértice será adicionado ao conjunto é a análise de quantos vizinhos cada vértice é capaz de converter. Ao maximizar este valor, menos vértices são adicionados ao $Hull\ Set$ ao final do procedimento.

Sendo assim, ao analisar este número de vértices convertidos por cada novo vértice adicionado ao $Hull\ Set$ percebe-se cinco casos distintos:

Caso 1 Existe um vértice v, que possui dois vizinhos u e w, cujos *Thresholds* são 1. Este vértice no algoritmo possui potencial 3, logo será preferencialmente escolhido. Ao adicionar v ao $Hull\ Set$ no mínimo 3 vértices são convertidos e o caminho será transformado em um dos seguintes casos:

Caso 1.1 P é inteiramente convertido, encerrando o processo com um $Hull\ Set$ de tamanho 1, que é mínimo. Neste caso, o $Hull\ Set$ é mínimo, pois um $Hull\ Set$ vazio não é suficiente para converter o grafo.

Caso 1.2 P é reduzido em um único caminho com $n-3$ vértices. Seja P' esse novo caminho. Seja S' o $Hull\ Set$ mínimo de P' obtido pelo algoritmo. De acordo com a hipótese de indução, esse conjunto é mínimo. Resta provar que S' não é capaz de converter P. Seja u o extremo do caminho resultante que estava mais próximo de v em P. Sabe-se que em P esse vértice tinha *Threshold* 2, pois do contrário teria sido convertido no momento que v fosse adicionado ao $Hull\ Set$, e em P' esse vértice possui *Threshold* 1. Se $u \in S'$, todos os vértices entre u e v em P são convertidos, porém v não. Logo v precisa estar contido no $Hull\ Set$. Supondo que $u \notin S'$, então u foi convertido por algum outro vértice. Se v não estiver contido no $Hull\ Set$, então u terá *Threshold* 2 e não será mais convertido por este vértice, logo neste caso v também precisa estar contido no $Hull\ Set$. Sendo assim, o $Hull\ Set$ mínimo de P é $S = S' \cup \{v\}$, provando a corretude do algoritmo.

Caso 1.3 P é reduzido em dois caminhos, cada um com no máximo $n-4$ vértices. Este caso decorre do anterior, contudo o algoritmo deve ser aplicado duas vezes, uma para cada caminho resultante. O $Hull\ Set$ de P corresponde a união dos $Hull\ Sets$ mínimos referentes aos dois caminhos. Pela hipótese de indução estes dois $Hull\ Sets$ são mínimos. Seja S' a união destes dois $Hull\ Sets$. Sabe-se que S' não é capaz de converter P aplicando-se a mesma lógica do caso anterior, isto é, os novos extremos dos caminhos possuíam *Threshold* 2 e não seriam convertidos por S', logo os vértices situados entre esses dois vértices também não seriam convertidos. Sendo assim, para que S' seja um $Hull\ Set$ de P é necessário adicionar o vértice v, provando a corretude do algoritmo.

Caso 2 Existe um vértice v tal que $f(v) = 2$ que possui dois vizinhos u e w, tal que, sem perda de generalidade, $f(u) = 1$ e $f(w) = 2$. Neste caso, ao adicionar v ao $Hull\ Set$ no mínimo 2 vértices são convertidos. Ao adicionar este vértice ao $Hull\ Set$,

Caso 2.1 P é reduzido em um caminho com $n-2$ vértices. A prova deste caso é idêntica à prova do Caso 1.2.

Caso 2.2 P é reduzido em dois caminhos, cada um com no máximo $n-3$ vértices. A prova deste caso é segue a prova do Caso 1.3.

Caso 3 Existe um vértice v tal que $f(v) = 1$ que possui dois vizinhos u e w, cujos *Thresholds* são 2. Neste caso, basta assumir, sem perda de generalidade, que $w = v$ e $v = u$, e analisar o outro vizinho do novo v. Este vizinho existe pois u possuia *Threshold* 2, logo não era extremo de P. Caso este vértice possua *Threshold* 1, basta aplicar o caso 1. Do contrário, aplicar o caso 2.

Caso 4 Existe um vértice v tal que $f(v) = 1$ que possui dois vizinhos u e w, tal que, sem perda de generalidade, $f(u) = 1$ e $f(w) = 2$. Neste caso, basta assumir que $u = v$ e $v = w$, e analisar o outro vizinho do novo v. Este vizinho existe pois w possuia *Threshold* 2, logo não era extremo de P. Caso este vértice possua *Threshold* 1, basta aplicar o caso 1. Do contrário, aplicar o caso 2.

Caso 5 Existe um vértice v tal que $f(v) = 2$ que possui dois vizinhos u e w, cujos *Thresholds* são 2. Neste caso, ao adicionar v ao $Hull\ Set$ somente v é convertido. Este caso não é interessante e deve ser evitado, uma vez que é garantida a existência dos casos anteriores.

Considerando que em todos os casos o Algoritmo 6 obteve o $Hull\ Set$ mínimo, pode-se afirmar que o Algoritmo 6 está correto.

□

Uma vez que o algoritmo encontra o $Hull\ Set$ mínimo de um caminho cujos extremos possuem *Threshold* 1 e todos os demais vértices possuem *Threshold* 1 ou 2, é necessário generalizar esta prova para caminhos cujos *Thresholds* são desconhecidos.

Lema 4.5. *O Algoritmo 6 obtém o IRR_f $Conversion\ Set$ de um caminho P.*

Prova. Como P é um caminho, todo vértice possui no máximo dois vizinhos, logo um vértice v de *Threshold* maior que 2, não pode ser convertido a menos que v esteja contido no $Hull\ Set$. Seja $T_3 = \{v \in V(P) | f(v) \geq 3\}$. Claramente todos os vértices de T_3 não podem convertidos por nenhum $Hull\ Set$ que não os contenha. Sendo assim, estes vértices devem ser adicionados ao $Hull\ Set$ de P. Além disso, estes vértices devem ser removidos do grafo e os *Thresholds* de seus vizinhos debitados em uma unidade.

Existem também vértices que são auto-convertidos ou foram convertidos ao adicionar os vértices de T_3 ao $Hull\ Set$. Seja $T_0 = \{v \in V(P) | f(v) \leq 0\}$. Claramente todos os vértices de T_0 não devem participar de nenhum $Hull\ Set$ pois são auto-convertidos ou já foram convertidos pelo $Hull\ Set$ em construção. Desta forma, estes vértices devem ser removidos do grafo e os *Thresholds* de seus vizinhos debitados em uma unidade. Uma vez que alguns vértices tiveram os *Thresholds* debitados, podem existir novos vértices de *Threshold* 0. Sendo assim, este processo deve ser repetido até que não existam mais vértice nesta situação.

Vale destacar que o $Hull\ Set$ construído até o momento consiste de vértices que não podem ser convertidos a menos que estejam contido neste conjunto. Ao final deste processo, todos os vértices possuem *Threshold* 1 ou 2.

Uma vez que vértices podem ter sido removidos de P, já não é mais possível afirmar que P não é vazio. Neste caso, o $Hull\ Set$ construído é mínimo pois contém apenas os vértices que não podem ser convertidos a menos que estejam contidos no $Hull\ Set$. Além disso, também não é possível assegurar que P é conexo, isto é, podem existir vários caminhos com pelo menos um vértice cada. Os vértices isolados não podem ser convertidos, pois não possuem vizinhos, logo devem ser incluídos no conjunto em construção e removidos do grafo. Sendo assim, seja C o conjunto de todos os caminhos formados pelos vértices de P. Se $P = \emptyset$, o $Hull\ Set$ construído é mínimo. Do contrário, cada caminho deve ser analisado individualmente.

Seja P' um caminho pertencente a C. Sejam u e w os extremos de P'. Como u e w possuem apenas um vizinho, se algum deles possuir *Threshold* 2, este vértice obrigatoriamente deve ser incluído no $Hull\ Set$ de P', pois não será possível converte-lo. Ao adicionar este vértice ao $Hull\ Set$, ele também deve ser removido do caminho e o *Threshold* de seu vizinho debitado em uma unidade. Não é possível que este vizinho também possui *Threshold* 2, pois todos os vértices possuem *Threshold* 1 ou 2. Contudo, se este vértice passar a ter *Threshold* 0, ele próprio deve ser removido do grafo e o *Threshold* de seu vizinho também debitado de uma unidade. Este processo deve ser repetido até $P' = \emptyset$ ou que os extremos de P' tenham *Threshold* 1. No primeiro caso, o

processo termina e o $Hull\ Set$ construído é mínimo, pois contém apenas vértices que obrigatoriamente devem estar incluídos no $Hull\ Set$. No segundo caso, seja P'' o caminho resultante deste processo. Claramente, P'' possui apenas vértices de *Threshold* 1 ou 2, exceto seus extremos que obrigatoriamente possuem *Threshold* 1. Seja S' o conjunto construído até o momento. Além disso, seja S'' o $Hull\ Set$ de P'', obtido pelo Algoritmo 6, pelo Lema 4.4 S'' é mínimo. Como S' contém apenas vértices que obrigatoriamente devem estar contidos no $Hull\ Set$ e S'' é o $Hull\ Set$ mínimo de P'', que já contemplava as atualizações demandadas pelos vértices de S' o conjunto resultante $S = S' \cup S''$ também é mínimo. Provando que o Algoritmo 6 obtém o $Hull\ Set$ mínimo de um caminho P qualquer. □

Vale destacar que o Lema 4.4 corresponde as linhas 36 até 41 do Algoritmo 6 e o Lema 4.5, as linhas 27 até 33. As demais linhas correspondem a quebra do ciclo em caminhos e união dos $Hull\ Sets$ mínimos de cada um. O Teorema 4.6 prova que a união destes processos obtém o $Hull\ Set$ mínimo de um ciclo qualquer.

Teorema 4.6. *O Algoritmo 6 obtém o $IRR_f\ Conversion\ Set$ de grafos ciclo.*

Prova. Seja $G(V, E)$ um grafo ciclo com $n = |V(G)|$ vértices. Ao analisar os *Thresholds* dos vértices de G, pode-se identificar três casos distintos:

Caso 1 Existe pelo menos um vértice cujo *Threshold* seja 0. Seja v um destes vértices. Como $f(v) = 0$, v é auto-convertido, logo v não participa de nenhum $Hull\ Set$. Neste caso, basta remover v do ciclo, debitar uma unidade dos *Thresholds* dos seus vizinhos. Ao final deste processo, o ciclo passou a ser um caminho com no máximo $n - 1$ vértices.

Caso 2 Existe pelo menos um vértice cujo *Threshold* seja maior ou igual a 3. Seja v um destes vértices. Como $f(v) \geq 3$ e $|Adj(v)| = 2$, v não pode ser convertido por nenhum $Hull\ Set$ que não o contenha, logo para converter o ciclo v pertence a todo $Hull\ Set$. Neste caso, basta adicionar v ao conjunto, remover v do ciclo e debitar uma unidade dos *Thresholds* dos seus vizinhos. Ao final deste processo, o ciclo passou a ser um caminho com no máximo $n - 1$ vértices.

Caso 2 Todo vértice possui *Threshold* 1 ou 2. Como nos casos anteriores, um vértice deve escolhido para ser removido, reduzindo o ciclo em um caminho. A escolha destes vértice deve visar a maior redução possível do tamanho do caminho resultante, uma vez que isso levará a um $Hull\ Set$ mínimo. Desta forma, seja v um vértice do grafo, escolhido respeitando a seguinte sequência de prioridades. Sejam u e w os vizinhos de v.

1. $f(v) = 2 \wedge f(u) = 1 \wedge f(w) = 1$
2. $f(v) = 1 \wedge f(u) = 1 \wedge f(w) = 1$
3. $f(v) = 2 \wedge f(u) = 1 \vee f(w) = 1$
4. $f(v) = 1 \wedge f(u) = 1 \vee f(w) = 1$
5. $f(v) = 2 \wedge f(u) = 2 \wedge f(w) = 2$

Nos dois primeiros casos, ao escolher o vértice v, os vértices u e w passarão a ter *Threshold* 0 e serão removidos do grafo pelo algoritmo ao analisar o caminho. Já nos dois casos seguintes, um dos vizinhos passará a ter *Threshold* 0 e também será removido ao analisar o caminho resultante. No último caso, somente o vértice v será convertido, sendo esse o pior caso possível. Sempre que um vértice v for escolhido, seguindo a lista de prioridades definida, ele deve ser adicionado ao conjunto em construção, removido do grafo e os *Thresholds* de seus vizinhos debitado em uma unidade. Ao final deste processo, o ciclo foi reduzido em um caminho com no máximo $n - 1$ vértices.

Em todos os casos, pelo menos um vértice foi removido do ciclo, que passou a ser um caminho com no máximo $n-1$ vértices. Conforme provado pelo Lema 4.5, o Algoritmo 6 obtém o $Hull\ Set$ mínimo de um caminho P qualquer. Seja S' o $Hull\ Set$ mínimo retornado pelo Algoritmo 6 para o caminho obtido pela a remoção de um vértice do ciclo, conforme descrito nos três casos.

Como no Caso 1, o ciclo foi reduzido em um caminho sem que nenhum vértice tenha sido incluído no $Hull\ Set$ então S' é um $Hull\ Set$ mínimo para G. Já no Caso 2, um vértice v de *Threshold* maior ou igual a 3 foi detectado e, conforme descrito anteriormente, todo $Hull\ Set$ de G obrigatoriamente deve conter este vértice, logo seja $S = S' \cup \{v\}$. Notoriamente, S é um $Hull\ Set$ de G. Além disso, S é um $Hull\ Set$ mínimo de G, pois ao remover v do ciclo, os *Thresholds* dos vizinhos de v foram debitados em uma unidade, logo ao obter o $Hull\ Set$ mínimo S' os *Thresholds* dos dois únicos vértices que v poderia influenciar já haviam sido atualizados para contemplar a futura inclusão de v ao $Hull\ Set$ do caminho resultante. Em relação ao terceiro caso, a prova segue idêntica ao Caso 2, uma vez que o vértice v foi escolhido, os *Thresholds* de seus vizinhos foram debitados em uma unidade antes de se obter o $Hull\ Set$ do caminho.

É importante destacar que nos três casos, um vértice qualquer que respeitasse as restrições do caso foi escolhido, o que não impossibilita a existência de outros vértices que também respeitem a mesma restrição. Estes vértices serão analisados como vértices do caminho, conforme o Lema 4.5. □

4.4 Cactus

A próxima família estudada em busca de um algoritmo polinomial para obter $Hull\ Sets$ mínimos foram os grafos cactus. Um grafo cactus é formado por blocos, que podem ser cliques, ciclos ou árvores. O algoritmo construído se baseia nos métodos já apresentados para solucionar estes blocos isoladamente e no Algoritmo 3 para tratar da união das soluções obtidas para cada bloco.

A enumeração dos blocos de um grafo e a avaliação do tipo de cada bloco é simples e possui complexidade $O(n + m)$. Este processo consiste de identificar os blocos utilizando uma busca em profundidade e avaliar as arestas entre os vértices do bloco. Uma vez que o bloco foi identificado basta aplicar o método correto para obter o $Hull\ Set$.

Contudo, o Algoritmo 3 possui uma exigência que não foi atendida nos algoritmos apresentados. É desejável que a solução mínima contenha a articulação sempre que possível. Desta forma, é necessário implementar uma alterações nos Algoritmos 5 e 6.

No Algoritmo 5, duas modificações são necessárias. A primeira deve ser realizada na linha 16 e tem como objetivo resolver os casos de empate entre *Threshold* de vértices. Sempre que houver um empate entre quaisquer vértices e a articulação, esta deve ser escolhida. A segunda modificação visa garantir ao método guloso a capacidade de arrependimento. Neste arrependimento, caso a articulação não tenha sido incluída no $Hull\ Set$ construído deve-se analisar a possibilidade de trocar o vértice de menor *Threshold* pela articulação. Se este novo conjunto também for um $Hull\ Set$ para a clique, o algoritmo o retornará. Caso contrário, o algoritmo retornará o conjunto original. Claramente este processo não aumenta o tamanho do $Hull\ Set$, que continua mínimo, conforme o Teorema 4.3.

As modificações do Algoritmo 6 são bastante similares às aplicadas no Algoritmo 5. A primeira modificação é idêntica, isto é, em caso de empate entre a articulação e qualquer outro vértice, a articulação deve ser escolhida. Esta alteração deve ser realizada nas linhas 19 e 37 do Algoritmo 6. Já a segunda modificação envolve testar todas as possibilidades de troca entre a articulação e os vértices do $Hull\ Set$ construído, uma de cada vez, caso a articulação não esteja contida no conjunto. Essa busca exaustiva é necessária pois, diferentemente das cliques onde todos os vértices são adjacentes, nos ciclos existe a possibilidade de ao trocar um vértice pela articulação um dos vizinhos deste não seja convertido pelo novo conjunto.

Uma vez que o $Hull\ Set$ mínimo de cada bloco pode ser obtido pelos algoritmos já apresentados, resta apresentar como o grafo cactus em si deve ser analisado. Este algoritmo é baseado no método proposto por Centeno et al., 2011 e detalhado na Seção 3.

Algoritmo 7: $HullSetCactus$

Entrada: Grafo Cactus $G(V, E)$, Função Threshold f
Saída: Conjunto IRR_f S

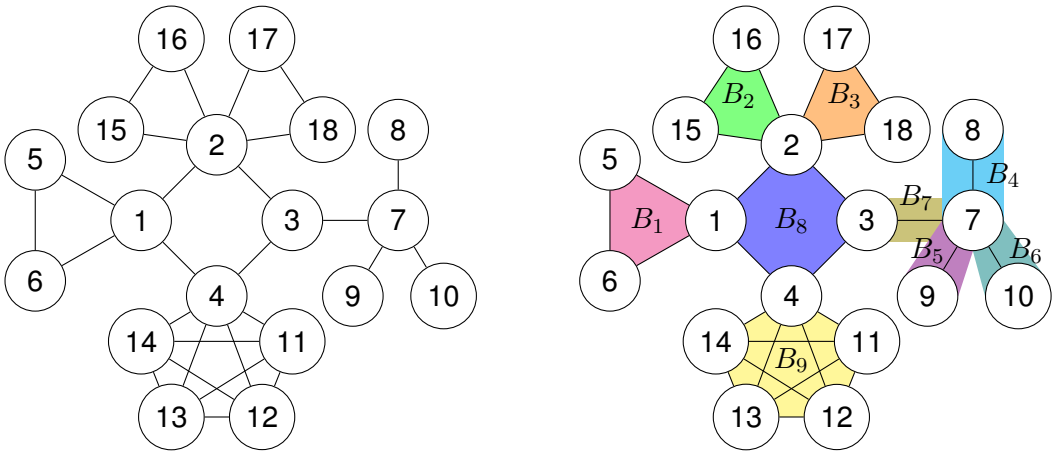
```
1  S = {}
2  S̄ = V(G)
3  enquanto S̄ ≠ ∅ faça
4      Seja B um bloco folha contido em S̄
5      Seja u a articulação que liga B ao restante do grafo, caso exista
6      se B é Ciclo então
7          S' = HSCiclo(B, u, f)
8      senão
9          se |V(B)| = 2 então
10             S' = HSK2(B, u, f)
11         senão
12             S' = HSCompleto(B, u, f)
13         fim
14     fim
15     se u ∈ S então
16         d_B = CalcularIRR(K, u, S' \ {u}, f)
17         f(u) = f(u) − d_B
18         S = S ∪ (S' \ {u})
19     senão
20         se u existe então
21             f(u) = 0
22         fim
23         S = S ∪ S'
24     fim
25     S̄ = S̄ \ (V(B) \ {u})
26 fim
27 retorne S
```

O algoritmo apresentado faz uso de uma função, denominada *CalcularIRR*, cujo objetivo é calcular quantos vizinhos da articulação de um determinado bloco folha são convertidos por um determinado conjunto. Esta função tem complexidade $O(n)$. A complexidade do Algoritmo 7 é $O(n^3)$. A Figura 6 apresenta um exemplo do processo de obtenção de um $IRR_2\ Conversion\ Set$ de um grafo cactus.

Teorema 4.7. *O Algoritmo 7 obtém o $IRR_f\ Conversion\ Set$ de um grafo cactus, cujas componentes são ciclos ou cliques.*

Prova. Tendo em vista que o algoritmo utiliza a metodologia proposta por (Centeno et al., 2011), que consiste de dividir o grafo em blocos folha e combinar os $Hull\ Sets$ mínimos obtidos em um $Hull\ Set$ mínimo para o grafo, e provada no Lema 3.1, resta analisar os resultados obtidos para cada bloco folha.

Considerando que cada bloco folha ou é um ciclo ou uma clique, e que a corretude dos métodos empregados na obtenção dos $Hull\ Sets$ mínimos destas componentes foi provada pelos Teoremas



B. Folha	Família	Vértices	Articulação	$Hull\ Set$	Alteração *Threshold*
B_1	Completo	$\{1,5,6\}$	1	$\{1,5\}$	$f(1)=1$
B_2	Completo	$\{2,15,16\}$	2	$\{2,15\}$	$f(2)=1$
B_3	Completo	$\{2,17,18\}$	2	$\{18\}$	$f(2)=0$
B_4	Árvore	$\{7,8\}$	7	$\{7,8\}$	$f(7)=1$
B_5	Árvore	$\{7,9\}$	7	$\{9\}$	$f(7)=0$
B_6	Árvore	$\{7,10\}$	7	$\{10\}$	$f(7)=0$
B_7	Árvore	$\{3,7\}$	3	$\{3\}$	$f(3)=1$
B_8	Ciclo	$\{1,2,3,4\}$	4	$\{\}$	$f(4)=0$
B_9	Completo	$\{4,11,12,13,14\}$	–	$\{13\}$	–
$IRR_2\ Conversion\ Set = \{5,15,18,8,9,10,13\}$					

Figura 6: Exemplo de Grafo Cactus e Obtenção do $IRR_2\ Conversion\ Set$.

4.6 e 4.3, pode-se concluir que o Algoritmo 7 obtém o $Hull\ Set$ mínimo dos grafos cactus. □

4.5 Grafos de Intervalo

Outra família de grafos estudada foram os grafos de intervalo. Um grafo de intervalo é uma forma de representar um conjunto de intervalos situados na reta real. Cada intervalo corresponde a um vértice do grafo. Sempre que a interseção de dois intervalos for diferente de vazio, adiciona-se uma aresta entre os dois vértices que representam estes dois intervalos no grafo. Assume-se que todos os extremos de intervalos são distintos, isto é, nenhum intervalo se inicia ou termina junto com outro. A Figura 7 apresenta um exemplo de um conjunto de intervalos e o grafo que o representa.

Um importante teorema, apresentado por Gilmore e Hoffman, 1964, permite caracterizar um grafo de intervalo e definir uma ordem na qual as cliques do grafo serão percorridas.

Teorema 4.8 (Gilmore e Hoffman, 1964)**.** *Para um grafo não direcionado G as seguintes afirmações são equivalentes:*

i *G é um grafo de intervalo.*

ii *G não contém ciclos de quatro vértices sem uma corda.*

iii *As cliques maximais de G podem ser ordenadas linearmente de forma que, para todo vértice v de G, as cliques maximais que contém v ocorram consecutivamente.*

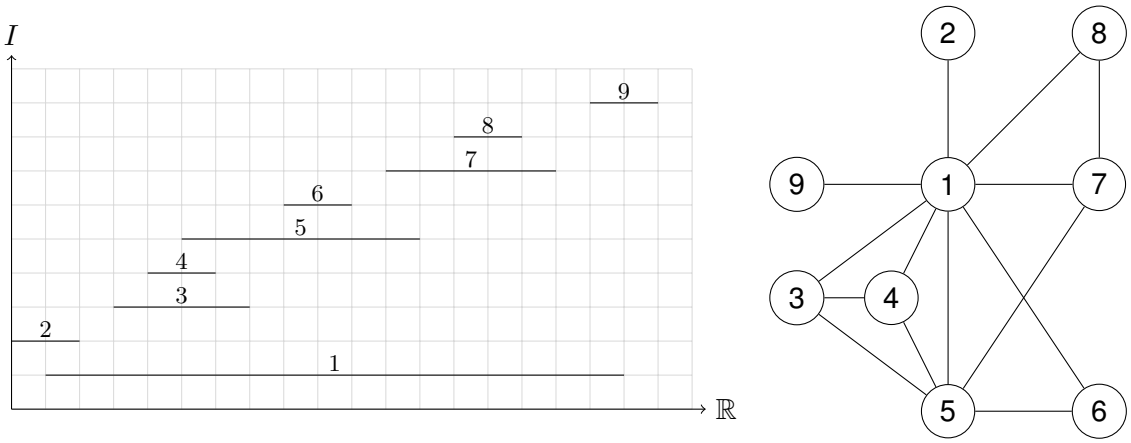


Figura 7: Exemplo de Grafo de Intervalo: Conjunto de Intervalos e Representação em um Grafo

O item (ii) garante que G é cordal, logo é formado por cliques e possui pelo menos um vértice simplicial. Um vértice é simplicial se pertence somente a uma clique do grafo, isto é, todos os seus vizinhos são adjacentes entre si. Além disso, considerando que G é cordal, ao remover um vértice simplicial pelo menos um outro vértice do grafo também será simplicial.

Para encontrar as cliques maximais de um grafo de intervalo, basta considerar um conjunto de retas verticais cortando o conjunto de intervalos. Sempre que dois intervalos são cortados por uma reta, pela construção do grafo, pode-se concluir que os vértices correspondentes aos intervalos são adjacentes entre si. Para localizar uma clique maximal do grafo basta analisar dentre as retas a que maximize a clique. A Figura 8 ilustra as cliques maximais do grafo de intervalo apresentado na Figura 7 e as retas verticais que as definem. (A área marcada com a cor laranja no grafo corresponde a uma sobreposição entre as cliques C_3 e C_4 e não possui nenhum significado distinto no exemplo.)

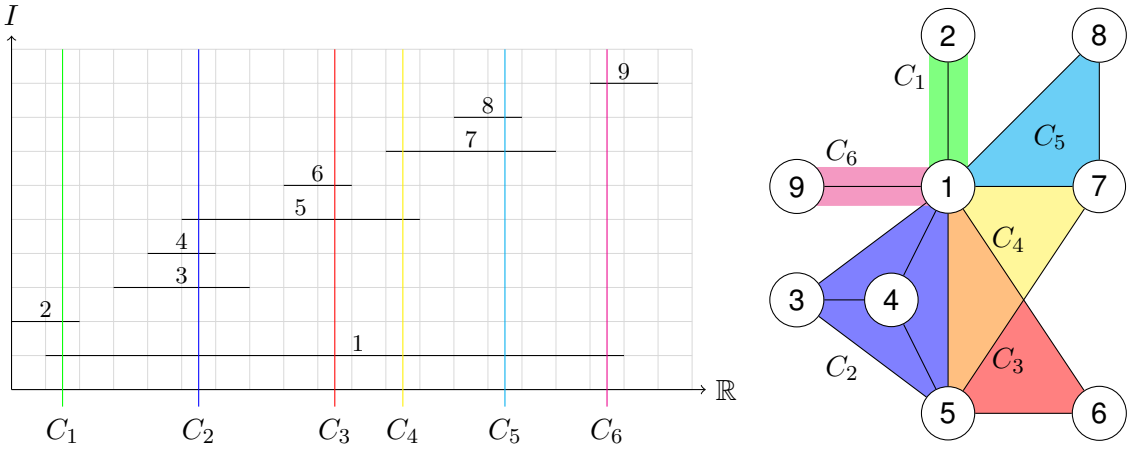


Figura 8: Exemplos de Cliques Maximais em um Grafo de Intervalo

Considerando que existe um vértice simplicial e este está contido em uma única clique e que para todo vértice v do grafo existe uma ordenação das cliques na qual as cliques maximais que contém v aparecem consecutivamente é possível estabelecer uma ordem na qual o algoritmo percorrerá as cliques maximais do grafo.

Essa sequência será iniciada pela clique K que contém o vértice simplicial do grafo. Os vértices de K podem ser particionados em dois conjuntos: Simpliciais e Não Simpliciais. Este segundo conjunto contém todos os vértices que pertencem a outras cliques maximais do grafo. Assumindo que a clique K foi tratada pelo algoritmo é possível reduzir o grafo removendo os vértices simpliciais

sem perda da conectividade do grafo, tendo em vista que os vértices não simpliciais permanecerão. Seja X o conjunto das cliques maximais do grafo que contém pelo menos um vértice v pertencente ao conjunto de vértices não simpliciais de K. Pelo Teorema 4.8, sabe-se que o conjunto X contém pelo menos uma clique maximal. Dentre o conjunto X existe uma clique que contém um vértice simplicial. Seja K' essa clique. Uma vez determinada a clique K', este processo pode ser reiniciado, considerando $K = K'$. Este procedimento deverá ser repetido até o grafo se torne vazio. Uma ordenação que pode ser facilmente obtida de um grafo de intervalo e que atende às condições estabelecidas corresponde a ordenação das linhas verticais que definem as cliques maximais da esquerda para a direita na representação gráfica dos intervalos. Esta ordenação será seguida pelo algoritmo e nos exemplos apresentados.

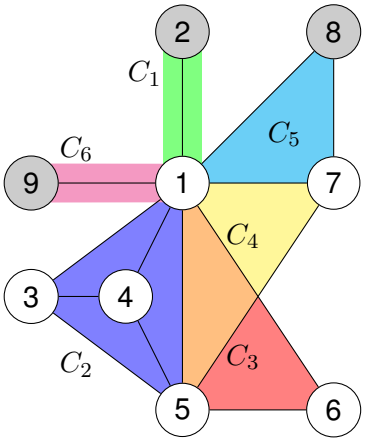
Uma vez determinada a ordem na qual as cliques do grafo de intervalo deve ser percorridas, resta apresentar o algoritmo para obter o $Hull\ Set$ de cada clique maximal. O procedimento para calcular o $Hull\ Set$ mínimo de cada clique é similar ao já apresentado para cliques em grafos cactus. Contudo, conforme visto anteriormente, podem existir vários vértices (não simpliciais) que estão conectados a outras cliques do grafo, o que inviabiliza sua aplicação imediata. Sendo assim uma nova modificação foi implementada no algoritmo já exposto.

A modificação tem como objetivo priorizar os vértices do bloco para o qual se deseja o $Hull\ Set$ mínimo que também pertencem a outras cliques do grafo. O Algoritmo 8 é baseado no Algoritmo 5 e já possui todas as modificações necessárias.

Uma vez apresentado o algoritmo que soluciona cada clique, resta apresentar o algoritmo que supostamente obtém o $Hull\ Set$ mínimo de grafos de intervalos.

O Algoritmo 9 faz uso de um procedimento *ObterIRR*, cuja função é obter dentre os vértices de uma determinada clique quais são convertidos pelo conjunto de vértices passado como parâmetro. Este procedimento possui complexidade $O(n)$.

Tendo em vista que a solução de cada clique maximal possui complexidade $O(n^2)$ e que localizar uma clique maximal adjacente que contenha um vértice simplicial pode ser realizada em $O(n)$ pode-se afirmar que o Algoritmo 9 tem complexidade $O(n^3)$. A corretude deste algoritmo é provada pelo Teorema 4.9. Além disso, a Figura 9 apresenta um exemplo da aplicação do algoritmo para obtenção de um $IRR_2\ Conversion\ Set$ de um grafo de intervalo. A ordenação das cliques corresponde à apresentada na Figura 8.



Bloco Folha	Vértices	A	$Hull\ Set$	Alteração de *Thresholds*
C_1	$\{1,2\}$	$\{1\}$	$\{1,2\}$	$f(1)=1$
C_2	$\{1,3,4,5\}$	$\{1,5\}$	$\{5\}$	–
C_3	$\{1,5,6\}$	$\{1,5\}$	$\{5\}$	–
C_4	$\{1,5,7\}$	$\{1,7\}$	$\{7\}$	–
C_5	$\{1,7,8\}$	$\{1\}$	$\{8\}$	–
C_6	$\{1,9\}$	–	$\{9\}$	–
$IRR_2\ Conversion\ Set = \{2,8,9\}$				

Figura 9: Processo de obtenção do $IRR_2\ Conversion\ Set$ de um grafo de intervalo

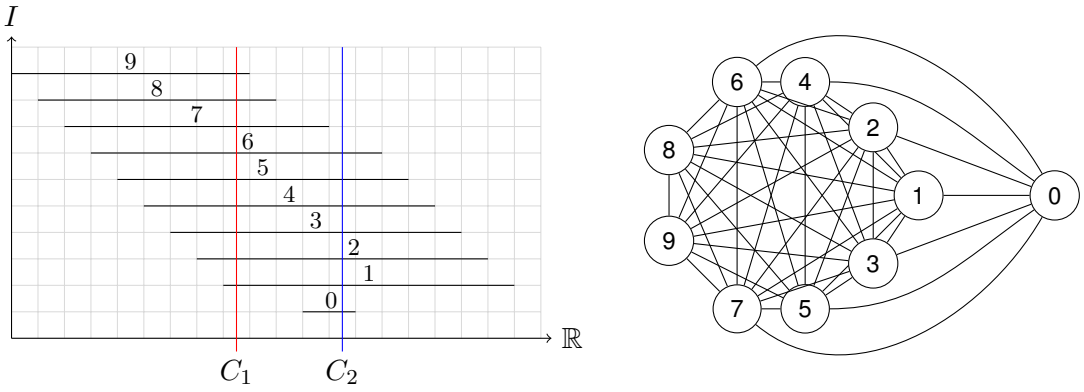
Tendo em vista o algoritmo apresentado, é possível estabelecer um conjectura em relação ao seu funcionamento.

Conjectura 4.9. *O Algoritmo 8 obtém um $IRR_f\ Conversion\ Set$ para grafos de intervalo.*

4.5.1 Contraexemplos

A corretude do algoritmo é apenas um conjectura até o momento, uma vez foram localizados contraexemplos de seu funcionamento. Para localizar estes exemplos o algoritmo proposto foi comparado com um algoritmo força bruta (detalhado no Apêndice A), limitado pelo tamanho do $Hull\ Set$ retornado pelo Algoritmo 9 com margem de um vértice a mais. Este algoritmo não constituí uma solução interessante para o problema pois possui complexidade $O(2^n)$.

A Figura 10 apresenta um contraexemplo onde o Algoritmo 9 retornou um $Hull\ Set$ de tamanho superior ao mínimo. Como pode ser observado, o grafo possui duas cliques maximais $C_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ e $C_2 = \{0, 1, 2, 3, 4, 5, 6, 7\}$. De acordo com o algoritmo conjecturado, a clique C_1 é processada primeiro. Um $Hull\ Set$ mínimo desta clique é $\{2\}$. Tendo em vista que ao remover este vértice não simplicial do $Hull\ Set$ nenhum vértice é convertido, os *Thresholds* seguem sem alteração. Ao prosseguir para a clique C_2 o algoritmo identifica que o $Hull\ Set$ mínimo desta clique é $\{2, 3, 5\}$. Este conjunto é retornado pelo algoritmo como sendo um $Hull\ Set$ mínimo. Contudo, o algoritmo força-bruta identificou que o $Hull\ Set$ mínimo deste grafo de intervalo é formado apenas pelo vértice 2, provando que para este caso o algoritmo não localizou corretamente um $Hull\ Set$ mínimo.

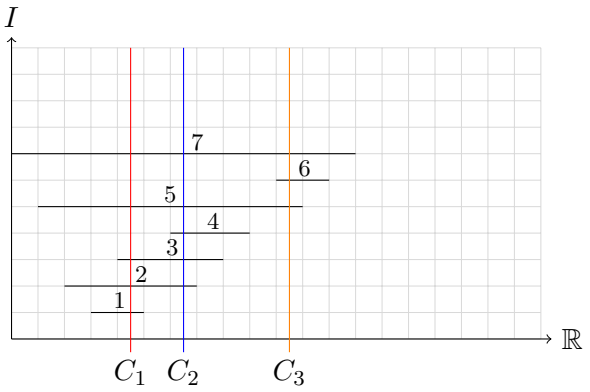


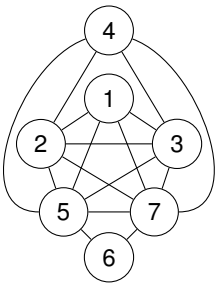
v	1	2	3	4	5	6	7	8	9	10
$f(v)$	6	7	5	4	6	1	5	5	2	2

Figura 10: Primeiro Contraexemplo do Algoritmo 9

A Figura 11 ilustra um outro contraexemplo onde o Algoritmo 9 apresenta um resultado incorreto. Ao processar a clique C_1, o algoritmo detecta que os vértices 1 e 3 são auto-convertidos e, durante o processo de limpeza, remove estes vértices do grafo e debita os *Thresholds* dos demais vértices de C_1 em duas unidades. Ao fazer isso, o vértice 2 passa a ter *Threshold* 0 e também é removido. Ao final do processo de limpeza do grafo, os vértices 5 e 7 possuem *Threshold* 1 e 2, respectivamente. Desta forma, o comportamento guloso determina que o vértice 7 deve ser escolhido, convertendo a clique. Uma vez que o $Hull\ Set$ desta clique foi definido, o algoritmo passa a corrigir os *Threshold* dos vértices seguindo as regras do algoritmo. Conforme descrito, os vértices não simpliciais são removidos do $Hull\ Set$, que se torna vazio. Em seguida, ao avaliar que mesmo com o conjunto vazio três vértices são convertidos (1, 2 e 3) os *Thresholds* dos vértices 5 e 7 são debitados em 3 unidades e os vértices 2 e 3 passam a ter *Threshold* 0. Na etapa seguinte, onde a clique C_2 é processada, os vértices 5 e 7, que possuem *Threshold* 1 e 2 respectivamente, são convertidos pelos vértices 2 e 3. Em seguida, o vértice 4 é convertido, finalizando o processo. Como nenhum vértice foi adicionado ao conjunto, o algoritmo retorna que o $Hull\ Set$ desta clique é vazio e atribui 0 aos *Thresholds* de todos vértices. Na terceira clique, os vértices 5 e 7 possuem *Threshold* 0 e convertem o vértice 6, terminando o processo com um conjunto vazio novamente. Ao final

do processo, o algoritmo mescla os conjuntos obtidos e retorna um conjunto também vazio como $Hull\ Set$ do grafo. Claramente, este conjunto não é um $Hull\ Set$ do grafo, pois os vértices 1 e 3 não são capazes de converter o grafo inteiro. Já o algoritmo força-bruta identifica que é necessário adicionar pelo menos mais um vértice do conjunto {4, 5, 6, 7} para converter o grafo.





v	1	2	3	4	5	6	7
$f(v)$	0	2	0	3	4	2	5

Figura 11: Segundo Contraexemplo do Algoritmo 9

Estes contraexemplos demonstram que a metodologia utilizada para atualizar os *Thresholds* sempre que uma clique é processada ainda não é adequada ao problema. Além disso é necessário um estudo mais aprofundado em relação à ordem na qual as cliques são processadas.

Algoritmo 6: $HSCiclo$

Entrada: Ciclo $C(V, E)$, Função *Threshold* f
Saída: $Hull\ Set\ \ S$

$S = \{v \in V(C) \mid f(v) > 2\}$
para cada $v \in S$ **faça**
    Sejam u e w os adjacentes de v no ciclo
    DebitarThreshold(u,w)
fim
$\overline{S} = V(C) \setminus S$
$Z = \{v \in \overline{S} \mid f(v) \leq 0\}$
enquanto $Z \neq \emptyset$ **faça**
    para cada $v \in Z$ **faça**
        Sejam u e w os adjacentes de v no ciclo
        $\overline{S} = \overline{S} \setminus \{v\}$
        DebitarThreshold(u,w)
    fim
    $Z = \{v \in \overline{S} \mid f(v) \leq 0\}$
fim
se $\overline{S} = \emptyset$ **então retorna** S ;
se $\overline{S}$ *é um ciclo* **então**
    CalcularPotenciais()
    Seja v o vértice de maior potencial
    Sejam u e w os vizinhos de v no ciclo
    $S = S \cup \{v\}$
    $\overline{S} = \overline{S} \setminus \{v\}$
    DebitarThreshold(u,w)
fim
enquanto $\overline{S} \neq \emptyset$ **faça**
    Seja P o primeiro caminho pertencente a $\overline{S}$
    Seja v um extremo de P, tal que $f(v) \leq 0$ ou $f(v) = 2$ ou $Adj(v) = \emptyset$
    enquanto v *existir* **faça**
        Seja w o vértice adjacente a v
        se $f(v) > 0$ **então** $S = S \cup \{v\}$;
        se w *existe* **então** $f(w) = f(w) - 1$;
        $\overline{S} = \overline{S} \setminus \{v\}$
        Seja v um extremo de P, tal que $f(v) \leq 0$ ou $f(v) = 2$ ou $Adj(v) = \emptyset$
    fim
    se $P \neq \emptyset$ **então**
        CalcularPotenciais()
        Seja v o vértice de maior potencial
        $S = S \cup \{v\}$
        Sejam u e w os vértices adjacentes a v
        DebitarThreshold(u,w)
        $\overline{S} = \overline{S} \setminus \{v\}$
    fim
fim
retorna S

Algoritmo 8: $HullSetClique - GrafoIntervalo$

Entrada: Clique $K(V, E)$ de tamanho k, Conjunto de Vértices Não Simpliciais A, Função Threshold f

Saída: $Hull\ Set\ \ S$

$S = \{v \in V(K) \mid f(v) \geq k\}$
$\overline{S} = V(K) \setminus S$
para cada $v \in \overline{S}$ **faça**
    $f'(v) = f(v) - |S|$
fim
enquanto $\overline{S} \neq \emptyset$ **faça**
    $Z = \{v \in \overline{S} \mid f'(v) \leq 0\}$
    enquanto $Z \neq \emptyset$ **faça**
        $\overline{S} = \overline{S} \setminus Z$
        para cada $v \in \overline{S}$ **faça**
            $f'(v) = f'(v) - |Z|$
        fim
        $Z = \{v \in \overline{S} \mid f'(v) \leq 0\}$
    fim
    se $\overline{S} \neq \emptyset$ **então**
        Ordenar $\overline{S}$ em ordem decrescente de $f'(v)$
        Seja L a lista de vértices de maior $f'(v)$ pertencentes a $\overline{S}$
        $L' = L \cap A$
        se $L' \neq \emptyset$ **então**
            Seja v um vértice qualquer de L'
        senão
            Seja v um vértice qualquer de L
        fim
        $S = S \cup \{v\}$
        $\overline{S} = \overline{S} \setminus \{v\}$
        para cada $w \in \overline{S}$ **faça**
            $f'(w) = f'(w) - 1$
        fim
    fim
fim
se $A \setminus S \neq \emptyset$ **então**
    para cada $v \in A \setminus S$ **faça**
        Seja w o vértice com o menor $f(w)$ contido em $S \setminus A$
        se $f(w) < k$ **então**
            $S' = (S \setminus \{w\}) \cup \{v\}$
            se *TestarIRR*$_f(S', V(K), f)$ **= Verdadeiro então**
                $S = S'$
            fim
        senão
            retorne S
        fim
    fim
fim
retorne S

Algoritmo 9: IRR_f Grafo de Intervalo()

Entrada: Grafo de Intervalo $G(V, E)$, Função Threshold f
Saída: $Hull\ Set\ S$

$S = \{\}$
$\overline{S} = V(G)$
Seja v um vértice simplicial de contido em $\overline{S}$
Seja K a clique que contém v
Seja A o conjunto dos vértices não simpliciais de K
enquanto $\overline{S} \neq \emptyset$ **faça**
    $S' =$ ConversionSet-Clique (K, A, f)
    $X =$ ObterIRR($K, S' \setminus A, f$)
    para cada $v \in A$ **faça**
        se $v \in X$ **então**
            $f(v) = 0$
        senão
            $f(v) = f(v) - |X|$
        fim
    fim
    $S = S \cup (S' \setminus A)$
    $\overline{S} = \overline{S} setminus(V(K) \setminus A)$
    Seja X o conjunto de cliques maximais de $\overline{G}$ que contém pelo menos um vértice de A
    Seja K uma clique de X que contenha um vértice simplicial
    Seja A o conjunto dos vértices não simpliciais de K
fim
retorne S

5 Conclusão

Este trabalho apresentou algoritmos polinomiais para obtenção de $IRR_f\ Conversion\ Sets$ para algumas famílias de grafos, tais como, completos, ciclos, árvores e cactus. Além disso, conjecturou um algoritmo para encontrar o $Hull\ Set$ mínimo de grafos de intervalo.

Como trabalhos futuros destacam-se a busca por um método que permita a correta propagação das atualizações dos *Thresholds* sempre que uma clique for solucionada para viabilizar o Algoritmo 9 e provar a conjectura estabelecida. Além disso, uma questão que será estudada é extensão deste algoritmo à família dos grafos cordais utilizando o esquema de eliminação perfeita como guia para o processamento de cada clique maximal do grafo.

Referências

Artigas, Danilo et al. (2011). "Partitioning a graph into convex sets". Em: *Discrete Mathematics* 311.17, pp. 1968–1977.

Benevides, Fabrício et al. (2015). "The maximum time of 2-neighbour bootstrap percolation: Algorithmic aspects". Em: *European Journal of Combinatorics* 48, pp. 88–99.

— (2016). "The maximum infection time in the geodesic and monophonic convexities". Em: *Theoretical Computer Science* 609, pp. 287–295.

Cáceres, J. et al. "Convex Sets in Graphs". Em: iii, pp. 53–56.

Centeno, Carmen C. et al. (2011). "Irreversible conversion of graphs". Em: *Theoretical Computer Science* 412.29, pp. 3693–3700.

Dourado, Mitre C., John G. Gimbel et al. (2009). "On the computation of the hull number of a graph". Em: *Discrete Mathematics* 309.18, pp. 5668–5674.

Dourado, Mitre C., Lucia Draque Penso et al. (2012). "Reversible iterative graph processes". Em: *Theoretical Computer Science* 460.September 2010, pp. 16–25.

Dourado, Mitre C., Fábio Protti et al. (2012). "On the Convexity Number of Graphs". Em: *Graphs and Combinatorics* 28.3, pp. 333–345.

Dreyer, Paul A. e Fred S. Roberts (2009). "Irreversible k-threshold processes: Graph-theoretical threshold models of the spread of disease and of opinion". Em: *Discrete Applied Mathematics* 157.7, pp. 1615–1627.

Farber, Martin e Robert E. Jamison (1987). "On local convexity in graphs". Em: *Discrete Mathematics* 66.3, pp. 231–247.

Gilmore, P. C. e A. J. Hoffman (1964). "A characterization of comparability graphs and of interval graphs". Em: *Canadian Journal of Mathematics* 16, pp. 539–548.

Golumbic, Martin Charles (1980). *Algorithmic Graph Theory and Perfect Graphs*.

Kynčl, Jan, Bernard Lidický e Tomáš Vyskočil (2016). "Irreversible 2-conversion set in graphs of bounded degree".

Penso, Lucia Draque et al. (2015). "Complexity analysis of P3-convexity problems on bounded-degree and planar graphs". Em: *Theoretical Computer Science* 607, pp. 83–95.

A Apêndice

Conforme descrito, um algoritmo força-bruta foi desenvolvido para averiguar se os conjuntos retornados pelo Algoritmo 9 eram de fato um $Hull\ Set$ mínimo para o grafo. Além desse procedimento, um outro algoritmo foi implementado para gerar grafos de intervalo e funções *Threshold* de forma aleatória.

A geração de grafos de intervalo se baseia na construção dos intervalos, respeitando a restrição de que todos os extremos de intervalos são distintos, isto é, nenhum intervalo se inicia ou termina junto com outro. Além disso, também são evitados grafos desconexos, pois os $Hull\ Sets$ destes grafos são obtidos pela união dos $Hull\ Sets$ das componentes, que possuem tamanho inferior ao grafo.

Para evitar que dois intervalos possuam extremos em comum, uma lista foi construída contendo todos os valores inteiros contidos entre 0 e $2n - 1$, onde n indica o número de vértices que o grafo deve possuir. A partir deste ponto, para cada vértice entre 1 e n, dois valores são sorteados e removidos desta lista. O menor destes dois valores será o inicio do intervalo enquanto outro, o final. Este processo é repetido até que se obtenha um grafo conexo.

Em seguida, a função *Threshold* é construída. Para isso, o grau de cada vértice é considerado. O *Threshold* de cada vértice v será escolhido dentre uma lista que varia de 0 até $grau(v) + 1$. Para garantir que casos intermediários dessa lista tenham mais chance de serem escolhidos, uma vez que esses extremos não são casos de grande interessantes, pois possuem solução trivial, uma lista foi construída de forma que quanto mais próximo de $(grau(v) + 1)/2$ maior a probabilidade deste valor ser escolhido.

Uma vez que o grafo de intervalo G e a função *Threshold* $f : V(G) \to \mathbb{Z}$ foram construídos, resta comparar os resultados obtidos pelo Algoritmo 9 com o algoritmo força-bruta. Considerando a elevada complexidade deste algoritmo, a profundidade máxima de recursões foi pré-fixada no tamanho da solução retornada pelo Algoritmo 9 acrescido de uma unidade. Caso a solução encontrada pelo algoritmo 9 esteja contida nas soluções encontradas pelo algoritmo força-bruta, pode-se afirmar que para este grafo a solução retornada está correta. Do contrário, o Algoritmo 9 não foi capaz de localizar o $Hull\ Set$ mínimo do grafo.

O algoritmo força-bruta utilizado faz uso da técnica *BackTracking* para percorrer todas as possibilidades de $Hull\ Sets$ para o grafo. Conforme descrito, o algoritmo recebe um parâmetro k que especifica o tamanho da solução encontrada pelo Algoritmo 9 acrescido de uma unidade. A função *Recursão* tem como objetivo percorrer todas as possibilidades de conjuntos. A cada iteração, um vértice v é incluído e uma recursão é chamada. Quando esta recursão retornar, este vértice v é removido e um outro vértice é considerado.

O procedimento *Recursão* utilizou uma função *ObterNãoConvertidos*, que localiza dentre todos os vértices do grafo quais ainda não são convertidos pelo conjunto passado como parâmetro. Vale destacar que se nenhum vértice foi encontrado nesta situação então todos foram convertidos e o conjunto é um $Hull\ Set$ do grafo.

O Algoritmo 11 teria complexidade $O(2^n)$ se não estivesse limitado pelo valor de k.

Algoritmo 10: Construtor de Grafo de Intervalo e Função *Threshold*

Entrada: Inteiro N
Saída: Grafo de Intervalo G e Função *Threshold* $f : V(G) \to \mathbb{Z}$

repita
  pos = { }
  para $i = 0$ **até** $2n - 1$ **faça**
    $pos = pos \cup \{i\}$
  fim
  para $i = 1$ **até** n **faça**
    ini =Aleatório(pos)
    fim =Aleatório(pos)
    $pos = pos \setminus \{ini\}$
    $pos = pos \setminus \{fim\}$
    se $fim < ini$ **então**
      SWAP(ini,fim)
    fim
    AdicionarVértice(G, i, ini, fim)
  fim
até G *ser conexo*;
para cada $v \in V(G)$ **faça**
  $pos = \{\ \}$
  $ct = 1$
  para $x = 0$ **até** $grau(v)/2$ **faça**
    para $i = 1$ **até** ct **faça**
      $pos = pos \cup \{x\}$
      se $x \neq grau(v) - x$ **então**
        $pos = pos \cup \{grau(v) - x\}$
      fim
    fim
    $ct = ct + 1$
  fim
  $f(v)$ = Aleatório(pos)
fim
retorne G e f

Algoritmo 11: Algoritmo Força Bruta para obtenção de $Hull\ Sets$ mínimos

Entrada: Grafo de intervalo $G(V, E)$, Inteiro k
Saída: Lista de $Hull\ Sets$ mínimos encontrados X

$X = \{\ \}$
$S = \{v \in V(G) | f(v) > |Adj(v)|\}$
Recursão(G,f, X,S,k)
retorne X

Procedimento Recursão(G, f, X, S, k)

Entrada: Grafo G, Função *Threshold* f, Conjunto de $Hull\ Sets\ X$, $Hull\ Set$ em construção S e inteiro k

```
1  NC =ObterNãoConvertidos(G,f,S)
2  se (|S| > k) ∧ (|NC| > 0) então
3  |  retorne
4  fim
5  se |NC| = 0 então
6  |  X = X ∪ {S}
7  senão
8  |  para cada v ∈ NC faça
9  |  |  S = S ∪ {v}
10 |  |  Recursão(G, f, X, S, k)
11 |  |  S = S \ {v}
12 |  fim
13 fim
14 retorne
```