MATEUS RAVEDUTTI LUCIO MACHADO

ALGORITMOS PARAMETRIZADOS PARA O PROBLEMA DE COBERTURA POR VÉRTICES EM GRAFOS

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Renato Carmo.

CURITIBA PR

2019

RESUMO

Algorítmos de problemas pertencentes à classe NP-Difícil tendem a não ser viáveis na prática. Existem algoritmos que podem deixar o problema tratável (isto é, que não possui um tempo de execução exponencial em relação ao tamanho da entrada), desde que sejam impostas algumas condições. Para isso são utilizadas algumas técnicas.

Neste trabalho serão apresentadas algumas destas técnicas utilizadas em algorítmos para o problema da cobertura por vértices, além disso, será apresentado o algorítmo de cobertura por vértices visto em (J. Chen, 2010), que explora restrições do problema em relação aos parâmetros de entrada.

Palavras-chave: grafo, cobertura por vértices, np-dificil, npt, kernelização.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
FPT	Fixed-Parameter Tractable
VC	Vertex Cover
NPC	Non-determnistic Polynomial-time Complete

LISTA DE SÍMBOLOS

O	Big O Notation

SUMÁRIO

1 INTRODUÇÃO

Na prática, um algoritmo que busca a solução de um problema pertencente a classe NP-difícil, utilizando força bruta, é inviável para entradas com tamanho muito grande. Para isso, existem abordagens que visam melhorar o desempenho de algoritmos para um problema específico.

Um exemplo de abordagem é a classificação dos problemas em outras classes, que dentro de certas condições acabam por ser tratáveis (tempo de execução não é exponencial de acordo com tamanho da entrada). Um exemplo é a classe Fixed-Parameter Tractable (FPT). A classe FPT é a classe de problemas que podem ser polinomiais em relação ao tamanho da entrada se for considerado fixo o valor de um dos parâmetros desta entrada.

Um dos focos neste trabalho será apresentar esta abordagem utilizando o problema de cobertura por vértices (VC) que pertence à classe FPT. Dados um grafo G = (V, E) e um inteiro k, o **Problema 1** é o problema da cobertura por vértices. Se assumirmos k como um valor constante, então é possível resolver o problema da cobertura por vértices em tempo polinomial, como veremos adiante.

Problema 1.
Entrada: Grafo $G = (V, E)$ e um inteiro k
Saída:
SE ($\exists$ C $\subset$ V / { |C| $\leq$ k **and** $\forall$ (u,v) $\in$ E, u $\in$ C **or** v $\in$ C}) : Retorna *VERDADEIRO*
SENÃO: Retorna *FALSO*

A Tabela 1.1 mostra um breve histórico de algumas soluções algorítmicas para o problema da cobertura por vértices, que são tratáveis se o valor de k é dado como constante.

Algoritmo	**Complexidade**
(Buss e Goldsmith, 1993)	$O\ (kn + 2^k k^{2k+2})$
(Downey e Fellows, 1995)	$O\ (kn + 2^k k^2)$
(R. Balasubramanian, 1998)	$O\ (kn + 1.324718^k k^2)$
(R. G. Downey, 1999)	$O\ (kn + 1.31951^k k^2)$
(Niedermeier e Rossmanith, 1999)	$O\ (kn + 1.29175^k k^2)$
(Stege e Fellows, 1999)	$O\ (kn + max(1.25542^k k^2, 1.2906^k k))$
(J. Chen, 2001)	$O\ (kn + 1.286^k k^2)$
(J. Chen, 2010)	$O\ (1.2738^k + kn)$

Tabela 1.1: Alguns algoritmos propostos para o problema de cobertura de vértices, onde n = |G| e k é um inteiro tal que VC(G, k) é verdade se e somente se existe cobertura por vértices de tamanho k em G.

1.1 OBJETIVO

O intuito deste trabalho é analisar o algoritmo apresentado em (J. Chen, 2010) e algumas técnicas empregadas. O algoritmo possui o melhor resultado até a data de elaboração deste documento, obtendo complexidade de tempo $O\ (1.2738^k + kn)$ e complexidade de espaço polinomial.

Serão apresentadas algumas técnicas para reduzir o tamanho do problema, como a inserção de vértices no conjunto solução baseada apenas na checagem do grau destes vértices, além da exclusão da possibilidade de alguns vértices pertencerem à solução, analisando apenas o vértice e seus vizinhos.

Outra abordagem a ser utilizada será a ramificação, que divide o problema em subproblemas (instâncias diferentes) complementares, e escolhe a melhor solução entre eles.

2 CONCEITOS FUNDAMENTAIS

Neste capítulo serão vistas duas técnicas: kernelização e árvores de busca limitada (Bounded Search Trees). Também será apresentado um algoritmo que mostra que se a instância (G, k) do problema VC restringir G a um grafo bipartido, o problema pode ser resolvido em tempo linear.

2.1 KERNELIZAÇÃO

Dada uma instância (G, k) do problema de cobertura por vértices, procurar por um grafo G' e um inteiro k' tal que existe cobertura por vértices de tamanho k' em G' se e somente se existe cobertura por vértices de tamanho k em G.

2.1.1 Kernelização de Buss

De acordo com (Buss e Goldsmith, 1993) existe uma kernelização para o problema da cobertura por vértices onde é possível reduzir uma instância (G, k) do problema para uma outra instância (G', k') tal que k' <= k e G' tem no máximo $(k')^2$ arestas (ou $2(k')^2$ vértices) que pode ser implementada em tempo $O(kn + 2^k k^{2k+2})$ onde n = |V(G)|. O **Algoritmo 1** descreve essa kernelização.

Lema: Dada uma instância (G, k) do problema VC, se $v \in V(G)$ e $\delta(v) > k$, então v pertence a uma cobertura mínima com entrada (G, k). O conjunto U no Algoritmo 1 contém todos os vértices com essa propriedade.
Prova: Sejam C uma cobertura por vértices mínima da instância (G, k) e $v \notin C$ um vértice com grau maior que k. Temos que $N(v) \subset C$, porém, como $| C | > k$, então não existe cobertura por vértices de tamanho k em G. Logo, ou $v \in C$, ou não existe cobertura mínima de tamanho k em G.

Ao final de cada execução da função *BussProc* no algoritmo 1, se existe uma cobertura por vértices de tamanho k em G, então G' tem no máximo kk' arestas. Como na última iteração temos $k = k'$, então existe cobertura por vértices de tamanho k' em G' se e somente se $| E(G') | \leq (k')^2$.
Prova: Como VC(G, k) = VC(G', k') e G' exclui qualquer vértice v tal que $\delta(v) >$ k, então se há cobertura mínima por vértices em G' de tamanho k', é possível cobrir todas as arestas com k' vértices de G'. Logo há no máximo kk' arestas ao final da execução em *BussProc*.

Entrada: Um grafo G e um inteiro k
Saída: Uma kernelização (G', k') de (G, k), onde $|E(G')| \leq k'^2$, ou null se não houver cobertura por vértices de tamanho k em G

```
1  Buss(G, k):
2      W ← { v / v ∈ V(G) e grau(v) = 0};
3      // Remove vértices isolados em G (não pertencem a
          cobetura mínima por vértices pois não cobrem
          aresta)
4      G' ← G – W;
5      return BussProc(G',k)

6  BussProc(G, k):
7      U ← { v / v ∈ V(G) e grau(v) > k};
8      if (|U| = 0) then
9          return (G, k);
10     end
11     if (|U| > k) or (|E(G')| > k k') then
12         return null;
13         // Não existe cobertura de tamanho k em G
14     end
15     G' ← G – U;
16     k' ← k – |U|;
17     return BussProc(G', k');
```

Algoritmo 1: Kernelização de (Buss e Goldsmith, 1993)

2.2 ÁRVORES DE BUSCA LIMITADAS

O método de árvores de busca limitadas é descrito como sendo uma busca a partir de uma raiz de uma árvore cuja altura é limitada por um valor. Problemas FPT utilizam este método para limitar a busca em relação ao parâmetro que se assume fixo na análise de complexidade. O algoritmo a seguir mostra que o problema VC pertence a classe FPT e utiliza esse método.

2.2.1 Algoritmo de Mehlhorn

Dada uma instância (G, k) do problema VC, o algoritmo de Mehlhorn resolve o problema em tempo $O(2^k|V(G)|)$ (Downey, 2013). O **Algoritmo 2** descrito é o algoritmo de Mehlhorn.

```
   Entrada: Um grafo G e um inteiro k
   Saída: VC(G,k)
1  Mehlhorn(G, k):
2      R ← E(G);
3      // R é o cojunto de arestas que restam ser
          verificadas
4      C ← {};
5      // C é o conjunto de vértices que fazem parte da
          cobertura parcial
6      return ( MehlhornProc(G,k,R,C) )

7  MehlhornProc(G,k,R,C):
8      if k == 0 then
9          return False;
10     end
11     e ← { {u, v} ∈ R / u ∉ C and v ∉ C };
12     // e é uma aresta qualquer em R não coberta por C
          ou NULL caso não exista
13     if e == NULL then
14         return True;
15     end
16     R ← R − e;
17     return ( MehlhornProc(G,k − 1,R,C + u) or
18     MehlhornProc(G,k − 1,R,C + v) );
```

Algoritmo 2: Algoritmo de Mehlhorn para calcular VC

Seja C uma cobertura mínima por vértices de um grafo G. O algoritmo parte do princípio que para cada aresta $\{u, v\} \in V(G)$ se u não pertence a C, então v deve pertencer a C, caso contrário a aresta não é coberta.
Cria-se uma árvore binária (Figura 2.1) da seguinte forma: cada nó na árvore de busca possui um conjunto de vértices adicionados à cobertura, chamado de cobertura parcial; cada aresta na árvore de busca se refere a adição de um vértice de G à cobertura parcial.

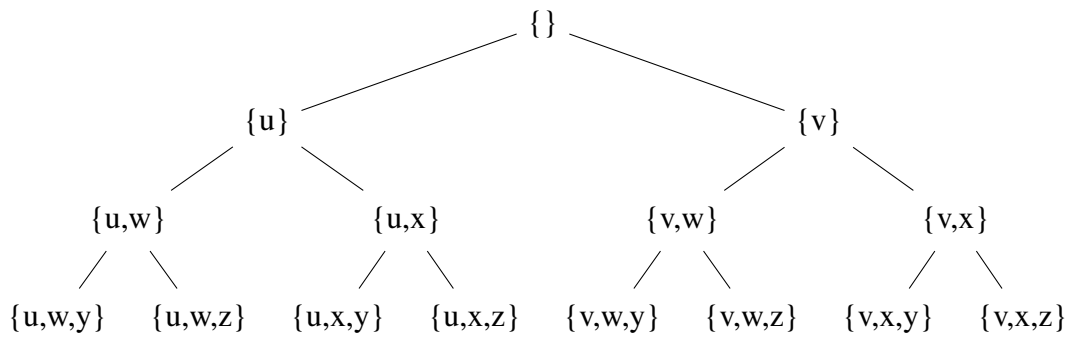

Figura 2.1: Árvore de busca do algoritmo de Mehlhorn. São adicionadas as seguintes arestas: {u, v} {w, x} {y, z}.

A cobertura parcial de um nó é o conjunto de vértices adicionados pelas arestas do caminho da raiz até ele. Em cada nó é selecionado uma aresta *{u, v}* em G ainda não coberta pela cobertura parcial e são adicionadas duas arestas na árvore de busca que dão origem a outros dois nós: em um nó da árvore é inserido o vértice *u* em sua cobertura parcial, e no outro é adicionado *v*.
A árvore resultante da busca se limita a uma árvore binária de altura máxima k, pois cada aresta adiciona 1 vértice à cobertura parcial e o problema permite desconsiderar conjuntos com mais de k vértices, logo, o algoritmo constrói uma árvore de no máximo 2^k nós.

2.3 RESTRIÇÃO DO PROBLEMA

Uma restrição de um problema é um problema que devolve a mesma resposta, porém o conjunto de instâncias da restrição é um subconjunto das instâncias do problema original. Um exemplo é considerar instâncias (G, k) do problema VC onde G é um grafo bipartido.

Definição: Um emparelhamento em um grafo **G** é um conjunto de arestas **M** tal que se *(u,v)* $\in M$, então, para uma aresta *(p,q)* $\in \{M - (u, v)\}$, *p* $\notin$ *{u,v}* e *q* $\notin$ *{u,v}*, ou seja, um vértice só pode pertencer a no máximo uma aresta em **M**.

Definição: Dado um grafo **G** e um emparelhamento **M** em **G**, um caminho alternado em **G** é um caminho **P** tal que:
Se $e_1 \in P$ e $e_2 \in P$ são arestas consecutivas em P, então $\{e_1 \in M \text{ e } e_2 \notin M\}$.

Observação: Seja G um grafo bipartido, (Hopcroft, 1973) apresenta um algoritmo $O(\sqrt{n}m)$ para calcular o emparelhamento máximo em G.

Teorema 1 ((Hall, 1935)) *Seja G um grafo bipartido com bipartição (L, R). G contém um emparelhamento máximo que inclui todos os vértices em L se e somente se* $|N(S)| \geq |S| \ \forall \ S \subseteq L$.

Teorema 2 ((Kőnig, 1931)) *Em um grafo bipartido, o número de arestas em um emparelhamento máximo é igual ao número de vértices em uma cobertura mínima por vértices.*

Dado G um grafo bipartido, a prova se dá pelo seguinte:

– Não há uma cobertura por vértices de tamanho menor que o número da arestas em um emparelhamento.
Prova: Seja M o conjunto de arestas de um emparelhamento máximo em G e C o conjunto de vértices de uma cobertura por vértices em G. Se $v \in C$ cobre uma aresta $(u,v) \in M$, então $v \notin M - \{u, v\}$, pois por definição, um vértice pode pertencer a no máximo uma aresta de um emparelhamento. Como C deve cobrir todas as arestas em M temos |C| ≤ |E(M)|.

– Existe uma cobertura por vértices de tamanho |M|.
Prova: Dados um grafo bipartido G onde L e R são conjuntos dos vértices da esquerda e direita em G, respectivamente, e os seguintes conjuntos:

  – U: vértices em L que não pertencem ao emparelhamento M
  – Z: vértices em G que possuem um caminho alternado até um vértice em U
  – S: vértices em L que pertencem a Z
  – T: vértices em R que pertencem a Z.

  Pelo Teorema 1 temos que T = N(S) e todo vértice em T pertence a M.
  Seja K = (L \ S) ∪ T:
  K é uma cobertura mínima por vértices em G, pois, se existe uma aresta (u, v) ∈ G tal que u ∈ S e v ∈ (R \ T), então N(S) ≠ T. Além disso |K| = |M|.
  A Figura 2.2 ilustra estes conjuntos para um grafo bipartido G = X ∪ Y.
  O Algoritmo 3 devolve esta cobertura em tempo $O(\sqrt{n}m)$.

O Algoritmo 4 calcula a cobertura por vértices mínima de um grafo bipartido G em tempo $O(\sqrt{n}m)$.

2.4 CONCLUSÃO

A técnica de kernelização permite a redução do tamanho de uma instância do problema, pois reduz o escopo a ser verficado através da identificação de elementos que definitivamente pertencem à solução (vértices com grau maior que k no algoritmo de Buss) e elementos que não pertencem à solução (vértices isolados no algoritmo de Buss).
As árvores de busca limitada estão intimamente ligadas com a classe dos problemas FPT, fazendo com que a busca se limite em um parâmetro da entrada a ser considerado fixo (criação de uma

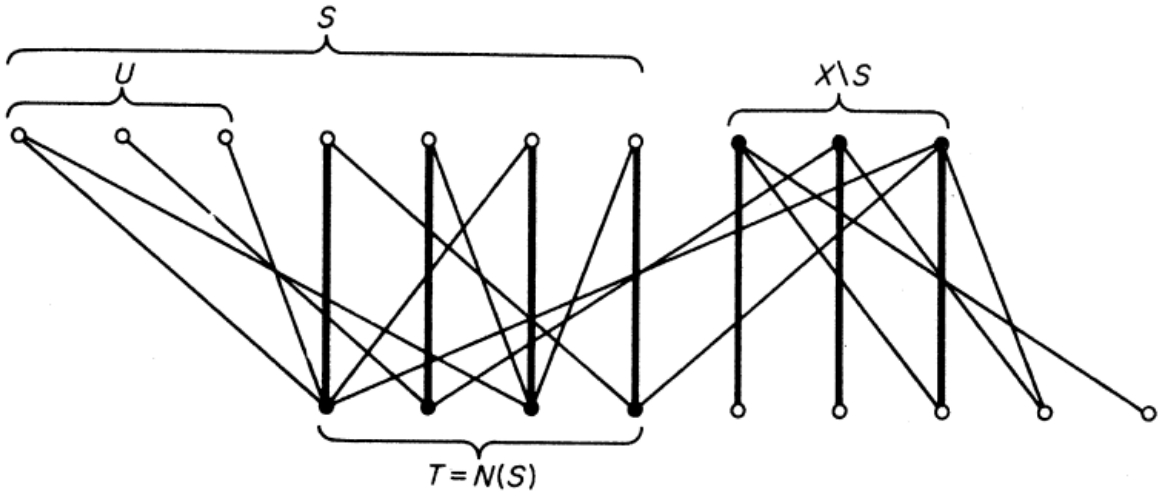

Figura 2.2

Fonte: (J. A. Bondy, 1976)

Entrada: Um grafo bipartido G onde L e R são os conjuntos de vértices à esquerda e direita em G, respectivamente, e um emparelhamento máximo M em G

Saída: Uma cobertura por vértices VC de tamanho | M | onde VC é cobertura mínima em G

```
1 Konig(G, M):
2     U ← { v / v ∈ L and v ∉ V(M) }
3     Z ← { v / v ∈ U or existe um caminho alternado em G de v até u, onde u ∈ U }
4     return (L \ Z) ∪ (R ∪ Z)
5     // Remove vértices isolados em G (não pertencem a
         cobetura por vértices pois não cobrem aresta)
```

Algoritmo 3: Cobertura por vértices dado um grafo bipartido e um emparelhamento máximo

```
1 VC(G, k):
      Entrada: Um grafo G bipartido
      Saída: Uma cobertura por vértices mínima em G
2     M ← matching(G);
3     // (Hopcroft, 1973) apresenta um algoritmo O(√VE)
         para calcular o emparelhamento maximo em um
         grafo bipartido
4     return ← Konig(B,M);
```

Algoritmo 4: Algoritmo para calcular cobertura mínima por vértices em grafos bipartidos

árvore de altura máxima k com fator de ramificação limitado (2) no algoritmo de Mehlhorn).
A restrição mostra que VC se torna um problema solúvel em tempo polinomial se limitarmos o conjunto de instâncias do problema (se G é sempre um grafo bipartido, encontrar um emparelhamento máximo em G e utilizar o algoritmo baseado no teorema de Kőnig para resolver o problema VC pode ser feito em tempo polinomial).
Todas as técnicas vistas aqui são complementares e serão utilizadas no algoritmo VC apresentado em (J. Chen, 2010).

3 KERNELIZAÇÃO

O capitulo anterior descreveu o conceito de kernelização e apresentou a kernelização proposta por (Buss e Goldsmith, 1993).

Neste capítulo serão abordados outros métodos que utilizam a técnica de kernelização.

Definição: Dado um grafo G, dois conjuntos U, W $\subset$ V(G) e um emparelhamento M em G. M mapeia U em W se e somente se $\forall\, u \in$ U existe uma aresta {u, w} $\in$ M tal que w $\in$ W.

Definição: Dado um grafo G = (V,E), (I, N(I)) é uma estrutura coroa em G se satisfaz as seguintes condições:

- I é um conjunto independente em G, onde |N(I)| $\leq$ |I|. Se |N(I)| < |I| então (I, N(I)) é uma coroa *estrita* em G, senão (I, N(I)) é uma coroa *equivalente*.
- Existe um emparelhamento em G que mapeia N(I) em I.

Definição: Dado um grafo G = (V,E), (I, N(I)) é uma estrutura quase-coroa em G se satisfaz as seguintes condições:

- G possui um conjunto independente I, onde |N(I)| = |I| + 1.
- Para cada S $\subset I$, |N(S)| $\geq$ |S| + 1.

3.1 KERNELIZAÇÃO UTILIZANDO TEOREMA NT

De acordo com (Nemhauser e Trotter, 1975), dado um grafo G = (V,E) onde n = |V(G)| e m = |E(G)|, é possivel obter um algoritmo $O(\sqrt{n}m)$ que decompõe G em dois subconjuntos C, H $\subset V(G)$, disjuntos entre si, satisfazendo:

- Se há um subconjunto D em H onde D é cobertura por vértices em G[H], então D U C é cobertura por vértices em G.
- Existe cobertura mínima por vértices em G contendo C.
- A cobertura mínima por vértices para G[H] tem pelo menos $\frac{|H|}{2}$ vértices.

A partir destes subconjuntos, cria-se uma kernelização (G', k') onde G' = G[H] e k' = k - |C|. Uma decomposição utilizando o algoritmo de kernelização NT é trivial se k' = k.

```
1 NT (G, k) :
    Entrada: Um grafo G e um inteiro k
    Saída: Uma kernelização (G', k') de (G, k), onde k' ≤ |G'|/2
2   V(B)← { v' / v ∈ V(G)} ∪ V(G);
3   E(B)← { {u, v'}, {u', v } / v ∈ N(u)};
4   // B é um grafo bipartido onde L e R sao os
       conjuntos de esquerda e direita em B,
       respectivamente.  Temos ∀ v ∈ V(G), v ∈ L(B) e v'
       ∈ R(B)
5   S ← VC (B) ;
6   C ← { v ∈ V(G) / v ∈ S ∧ v' ∈ S};
7   H ← { v ∈ V(G) / v ∈ S ∨ v' ∈ S} - C;
8   G' ← G [ H ];
9   k' ← k – | C |;
```

Algoritmo 5: Kernelização utilizando o Teorema NT (Nemhauser e Trotter, 1975)

O Algoritmo 5 apresenta uma kernelização em um grafo G utilizando o Teorema NT.

Lema: Dado um grafo G, o Algoritmo 5 executa em tempo $O(\sqrt{n}m)$ onde n = |V(G)| e m = |E(G)|.

3.2 A OPERAÇÃO "FOLDING"

Lema: Um grafo G não possui coroa se e somente se a decomposição NT de G é trivial (J. Chen, 2010).

Lema: Seja G um grafo sem coroa. G possui uma estrutura quase coroa se e somente se existe um vértice v ∈ $V(G)$ tal que G - v possui uma coroa equivalente (J. Chen, 2010).

O algoritmo em (J. Chen, 2010) utiliza um método chamado folding, sendo este uma generalização de um algoritmo em (J. Chen, 2001). A entrada do método consiste em um grafo G sem coroa e uma estrutura quase-coroa (I, N(I)) em G.

Lema: Dado um grafo G e um inteiro k, existe um algoritmo que em tempo $O(k^3\sqrt{k})$ reduz G a um grafo G', onde G' é uma kernelização de G e G' não possui coroa. Além disso se G' possui uma estrutura quase-coroa, o algoritmo devolve esta estrutura.

Prova: O algoritmo 6 cumpre estes requisitos. Como o algoritmo NT executa em tempo $O(\sqrt{n}m)$ onde n = |V(G)| e m = |E(G)| e o número de reduções não triviais em uma instância (G, k) qualquer é $O(k)$, temos:

n = $O(k)$

m = $O(k^2)$

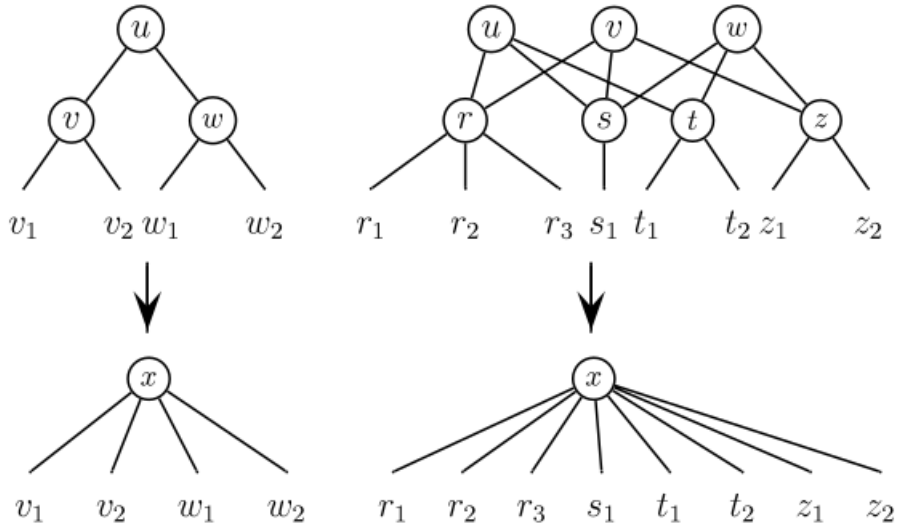

Figura 3.1

Fonte: (J. Chen, 2010)

$O(k)\ O(\sqrt{n}m) = O(k)\ O(\sqrt{k}k^2) = O(\sqrt{k}k^3)$

Dado um grafo G sem coroa e uma estrutura quase-coroa (I, N(I)) em G, a operação folding parte das seguintes observações:

– Como |N(I)| = |I| + 1, para cobrir as arestas adjacentes aos vértices em I, devem pertencer a cobertura por vértices I ou N(I).

– Se o grafo induzido por N(I) não é um conjunto independente, então existe uma cobertura mínima por vértices em G que inclui N(I) e exclui I.
Justificativa: Isso ocorre pois como há pelo menos uma aresta em G[N(I)], uma cobertura por vértices teria que incluir um conjunto C ⊆ N(I) onde |C| ≥ 1, e consequentemente |I| + |C| ≥ |N(I)|, onde I ∪ C e N(I) cobrem as mesmas arestas. Logo N(I) é adicionado a cobertura mínima por vértices e I é desprezado.

– Seja G' o grafo *G - (I ∪ N(I))*. Se o grafo induzido por N(I) é um conjunto independente, então:
Adicione um vértice v com as seguintes arestas em G': se u é um vértice em G' e u é vizinho de um vértice em N(I), adicione a aresta (u, v) em G'. Existe cobertura por vértices de tamanho k em G se e somente se existe cobertura k' = k - |I| em G'.
Justificativa: É escolhido I para cobrir as arestas (w, z) onde w ∈ I e z ∈ N(I), e é adicionado um vértice novo u para cobrir as arestas (u, v) onde u ∈ N(I) e v ∉ I.

A Figura 3.1 e o Algoritmo 7 ilustram esta operação.

```
1  RemoveEstruturasCoroa(G, k):
       Entrada: Um grafo G e um inteiro k
       Saída: Uma redução (G', k') de (G, k), onde k' ≤ k e G' não possui um
              conjunto de vértices que formam uma coroa. Uma estrutura
              quase-coroa H ⊂ G, caso exista.
2      (G',k')←(G,k);
3      do
4          old_k' ←k';
5          (G',k')←NT(G', k') ;
6          // Se a reducao NT for trivial, entao nao ha
               coroa em G'
7      while k' < old_k';
8      foreach v ∈ V(G) do
9          if (G − v) possui coroa equivalente (I, H) then
10             S ← (I, H ∪ v) ;
11             // S é uma estrutura quase-coroa em G
12         end
13     end
14     // Se S não tiver valor atribuido, então não
           existe uma estrutura quase-coroa em G
15     return (G', k', S);
```

Algoritmo 6: Redução de G em um grafo sem coroa

3.3 A OPERAÇÃO STRUCTION

Definição: Dado um grafo G = (V,E), uma anti-aresta em G é uma aresta (u, v) tal que (u, v) $\notin$ E(G).

O algoritmo em (J. Chen, 2010), utiliza um método chamado "struction", sendo este uma especialização de um algoritmo em (Ch. Ebenegger, 1984). O algoritmo recebe como entrada um grafo G = (V, E) e um vértice v tal que v tem menos anti-arestas que vizinhos. O algoritmo segue os seguintes passos:

1. O vértice v e o conjunto N(v) = {v_1, v_2, ... , v_n} (onde n = |N(v)|) são removidos do grafo G.

2. São adicionados os vértices v_{ij} em G, para cada anti-aresta (v_i, v_j) em G.

3. São adicionadas as seguintes arestas em G: (v_{ir}, v_{is}) / r ≠ s e (v_r, v_s) ∈ E(G).

4. São adicionadas as seguintes arestas em G: (v_{ir}, v_{js}) / i ≠ j.

5. São adicionadas as seguintes arestas em G: (v_{ij}, u) / u $\notin$ N[v] e (v_i, u) ∈ E(G) ou (v_j, u) ∈ E(G).

A Figura 3.2 e o Algoritmo 8 ilustram este método.

```
1  Folding(G, k):
       Entrada: Um grafo G e um inteiro k
       Saída: Uma kernelização (G', k') de (G, k), onde k' ≤ k
2      (G', k', S) ← RemoveEstruturasCoroa(G, k);
3      if S ≠ null then
4          (I, H) ← S;
5          // (I, H) é uma estrutua quase-coroa em G, onde I
               é um conjunto independente e H = N(I)
6          if O grafo induzido por H não é um conjunto independente then
7              G' ← G – (I ∪ N(I));
8              k' ← k – | N(I)|;
9              // Existe uma cobertura mínima de vértices em
                   G que inclui H e exclui I
10         end
11         else
12             V(G')← (V(G) – I ∪ H) + u;
13             // u é um novo vértice
14             E1 ← {(u,v) / v ∈ N(N(I)) };
15             E(G')← E(G) ∪ E1;
16             k' ← k – | I |;
17         end
18     end
19     return G', k';
```

Algoritmo 7: Operação folding

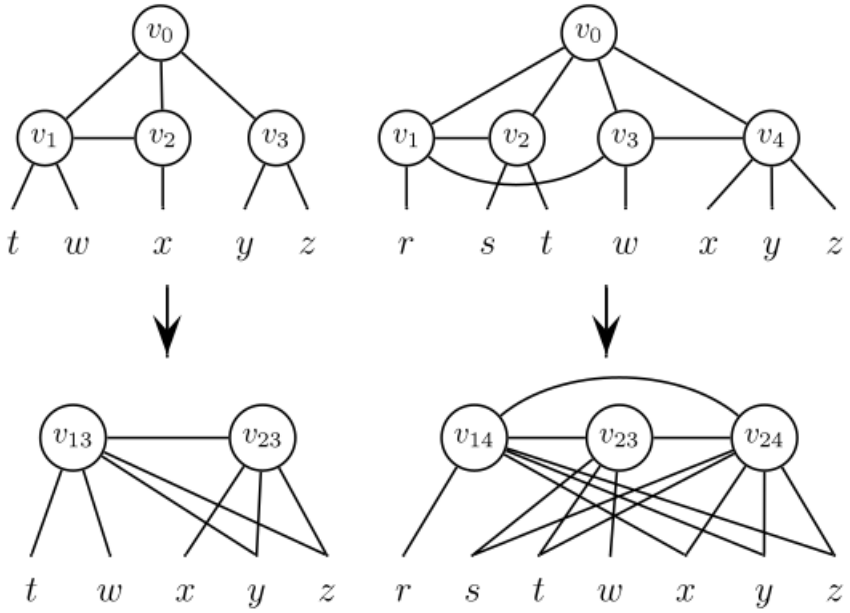

Figura 3.2

Fonte: (J. Chen, 2010)

```
1 Struction(G, v):
    Entrada: Um grafo G e um vertice v em G tal que existem no máximo
             (| N(v) | −1) anti-arestas em N(v)
    Saída: Uma kernelização (G', k') de (G, k), onde k' ≤ k
2   G' ← G;
3   V(G') ← ( V(G') − N[v] ) ∪
4   { (v_ij) / v_i,v_j ∈ N(v) e (v_i, v_j) é anti-aresta em G } ;
5   E1 ← { (v_ir, v_js) / i = j, r ≠ s e (v_r, v_s) ∈ E(G)};
6   E2 ← { (v_ir, v_js) / i ≠ j};
7   E3 ← {(v_ij, u) / u ∉ N[v] e (v_i, u) ∈ E(G) ou (v_j, u) ∈ E(G)};
8   E(G') ← E(G') ∪ (E1 ∪ E2 ∪ E3);
9   k' ← k – (|G | – |G' | −1);
```

Algoritmo 8: Operação struction

3.4 CONCLUSÃO

Métodos de kernelização mais sofisticados podem reduzir o tamanho do problema consideravelmente, buscando estruturas específicas (um conjunto de vértices/arestas ou um subgrafo) na instância do problema que tenham relação com a resposta.

O método da decomposição NT utiliza do algoritmo visto no capítulo anterior. Cria-se um grafo bipartido B a partir de G onde há uma relação entre as coberturas por vértices de G e B.

Os métodos seguintes só funcionam para uma instância que tem uma propriedade específica: no método struction é necessário que haja um vértice v onde o grafo induzido por N[v] possui um número grande de arestas; no método folding é necessário um grafo que contenha uma estrutura quase-coroa, e para isso foi apresentado um algoritmo que localiza esta estrutura em G, se houver.

4 APRESENTAÇÃO DO ALGORITMO

Este capítulo descreve o algoritmo de cobertura por vértices visto em (J. Chen, 2010), utilizando algoritmos e técnicas vistas nos capítulos anteriores. Primeiramente serão descritas as estruturas que auxiliam na escolha do fluxo de execução do algoritmo (ramificação).

4.1 ESTRUTURAS

4.1.1 Tuplas

Dado um grafo G = (V,E), uma *tupla* é um par *(S, n)* tal que S é um conjunto de vértices em V(G) e n um inteiro tal que é possivel procurar uma cobertura mínima por vértices em G removendo no mínimo *n* vértices em *S*.
Uma *tupla-2* é uma tupla (S, n) para S = {u, v} onde as seguintes condições são satisfeitas:

- $n = 1$
- $d(u) \geq 1$ e $d(v) \geq 1$
- *u* e *v* não são adjacentes

Uma tupla-2 é forte se satisfaz uma das seguintes condições:

- $d(u) \geq 4$ e $d(v) \geq 4$
- $2 \leq d(u) \leq 3$ e $2 \leq d(v) \leq 3$

O conjunto de tuplas no algoritmo VC armazena apenas tuplas-2.

Lema: Seja G um grafo e v ∈ V(G). Existe uma cobertura mínima por vértices em G que contém N(v) ou no máximo |N(v)| - 2 vértices em N(v).
A partir desse lema, se escolhermos um caminho no algoritmo que adiciona um vértice v na cobertura por vértices, então é possível

Proposta: Seja G um grafo e (u, v) ∈ E(G). Existe uma cobertura mínima por vértices em G que inclui v ou que exclui v e um outro vizinho de u.

Observação: Uma tupla impõe a restrição de que é possível remover um número de vértices de um conjunto. Além da criação de tuplas, outras operações no algoritmo estabelecem restrições, como as operações “struction“ e “folding“. Dada uma estrutura quase-coroa (I, N(I)), se a operação de folding resultar na escolha de inserir o conjunto I na cobertura por vértices, é possível que seja removido um vértice v ∈ N(I), onde v é vizinho de um vértice u pertencente a uma tupla S. Esta operação pode gerar conflito, pois o vértice u (dado que toda tupla é tupla-2 no

algoritmo) deve ter grau maior que 1. Além disso, se um vizinho de u é removido e não inserido na cobertura por vértices, então u obrigatoriamente deve fazer desta cobertura.
No algoritmo, os conflitos não são resolvidos atualizando as tuplas, ao invés disso, tomam-se as seguintes decisões:

1. Após uma operação de struction ou uma operação de folding, a estrutura que armazena as tuplas correntes no algoritmo é esvaziada.

2. Ao se gerar novas tuplas, estas novas tuplas podem ter conflito com as tuplas correntes, nesse caso escolhe-se uma tupla e remove-se as tuplas restantes seguindo a seguinte prioridade: se houver uma tupla-2 forte, então o conjunto de tuplas se restringe a uma tupla-2 forte. Caso contrário o conjunto se mantém apenas com uma tupla-2.
   Esta operação é feita implicitamente no algoritmo descrito.

4.1.2 Prioridades

Definição: Dado um grafo G = (V, E) e dois vértices *u, v* ∈ G, o vértice *u* é dominado pelo vértice *v* em G se *(u, v)* é uma aresta em G e *N(u) ⊆ N[v]*.

Definição: Dado um grafo G = (V, E) e dois vértices *u, v* ∈ G, o vértice *u* é quase dominado pelo vértice *v* se não existe aresta *(u, v)* e |N(u) - N(v)| ≤ 1.

Definição: Dado um grafo G = (V, E) e um vértice v ∈ G de grau n, o rótulo do vértice v sera um vetor T(v) = $\{d_1, d_2, ..., d_n\}$ onde d_1 = maior grau em N(v), d_2 = segundo maior grau em N(v), ... , d_n = menor grau em N(v).
Exemplo: Seja v ∈ G e N(v) = {u, w, y, z} tal que δ(u) = 3, δ(w) = 3, δ(y) = 5, δ(z) = 2. Então T(v) = {5, 3, 3, 2}.

Definição: Dado um grafo G = (V, E) e um vértice v ∈ G, definimos como L(v) o inteiro representado pela concatenação em ordem lexicográfica dos elementos de T(v).
Exemplo: Seja v ∈ G tal que T(v) = {5, 3, 3, 2}. Então L(v) = 5332.

Definição: Dado um grafo G = (V, E) e um vértice de grau mínimo em G, um par de vértices {u, z} é chamado par útil se a escolha dos vértices satisfaz os seguintes critérios:

- Escolha do vértice u (por ordem de prioridade):

- u tem grau mínimo em G.
- L(u) ≥ L(v) ∀ v ∈ G tal que d(v) = d(u).
- Se G é regular, então N(u) tem o maior número de pares x, y onde y é quase dominado por x.

- O subgrafo induzido por N(u) tem o maior número de arestas.

- Escolha do vértice z (por ordem de prioridade):

1. Se existem dois vértices v, w ∈ N(u) tal que v é quase dominado por w, então z é quase dominado por um vizinho de u.
2. z é o vértice com maior grau que satisfaz a Condição (1) em G.
3. z é o vértice com menor grau que satisfaz as condições (1) e (2) em um subgrafo induzido por N(u).
4. z tem maior número de vizinhos em N(u) que satisfazem as Condições (1), (2) e (3).

A ordem de prioridade das estruturas no algoritmo é a seguinte (menor número indica maior prioridade):

1 **T** é uma tupla-2 forte.
2 **T** é uma tupla-2.
3 **T** é um par útil (u, z) tal que d(u) = 3 e para qualquer v ∈ N(u), d(v) = 5 e, N(v) e N(w) só possuem u de vizinho em comum, para qualquer outro w ∈ N(u).
4 **T** é um par útil (u, z) tal que d(u) = 3 e d(z) ≥ 5.
5 **T** é um par útil (u, z) tal que d(u) = 3 e d(z) ≥ 4.
6 **T** é um par útil (u, z) tal que d(u) = 4, u tem pelo menos 3 vizinhos de grau 5 e o grafo induzido por N[u] tem pelo menos 1 aresta.
7 **T** é um par útil (u, z) tal que d(u) = 4 e para qualquer v ∈ N(u), d(v) = 5 e, N(v) e N(w) só possuem u como vizinho comum, para qualquer outro w ∈ N(u).
8 **T** é um vértice z tal que d(z) ≥ 8.
9 **T** é um par útil (u, z) tal que d(u) = 4 e d(z) ≥ 5.
10 **T** é um par útil (u, z) tal que d(u) = 5 e d(z) ≥ 6.
11 **T** é um vértice z tal que d(z) ≥ 7.
12 **T** é um par útil que não se encaixa em nenhuma regra acima.

4.2 REDUÇÃO

Um método utilizado no algoritmo VC é o método de redução. No capítulo anterior foram apresentados os algoritmos para os métodos de *Folding* e *Struction* que serão utilizados neste método.

Lema: Seja um grafo G = (V, E) tal que para cada v ∈ $V(G)$, d(v) ≥ 2. Se G contém 3 vértices u, v e w onde d(u) = (v) = d(w) = 2 e {u, v, w} é um conjunto independente em G, então se aplicarmos (G', k') = Folding(G, k), temos k' ≤ k - 2. (J. Chen, 2010)

Lema: Seja um grafo G = (V, E) tal que para cada v $\in G$, $1 \leq d(v) \leq 3$. Se G contém 4 vértices com grau no máximo 2, então se aplicarmos (G', k') = Folding(G, k), temos k' $\leq$ k - 2. (J. Chen, 2010)

Lema: Seja um grafo G = (V, E) tal que para cada v $\in G$, $d(v) \geq 3$. Se há um conjunto independente I em G tal que $2 \leq I \leq 3$ e $|N(I)| \leq |I| + 1$, então o método "folding" é aplicável.

O Algoritmo 9 e o Algoritmo 10 apresentam os casos em que se aplica o método de *Folding* e *Struction* respectivamente.

```
1  Folding_Condicional(G,k,τ):
       Entrada: Um grafo G, um inteiro k e um conjunto de tuplas τ
       Saída: True se Folding é aplicado, False caso contrário
2      if Existe uma tupla-2 forte (u,z, 1) ∈ τ then
3          if A aplicação repetida de Folding reduz o parâmetro k em pelo menos
             2 unidades then
4              do
5                  old_k ← k;
6                  (G,k) ← Folding(G,k);
7              while k < old_k;
8              return True;
9          end
10         else if A aplicação de Folding reduz o parâmetro k em uma unidade e
             d(u) < 4 then
11             do
12                 old_k ← k;
13                 (G,k) ← Folding(G,k);
14             while k < old_k;
15             return True;
16         end
17         return False;
18     end
19     else
20         do
21             old_k ← k;
22             (G,k) ← Folding(G,k);
23         while k < old_k;
24         return True;
25     end
```

Algoritmo 9: Operação de folding condicional

O algoritmo de redução utiliza um conjunto de tuplas-2 denotado por τ, que é atualizado ao longo da execução. Tuplas que representam novas informações são incluidas apenas em ramificações (chamadas recursivas).

```
1  Struction_Condicional(G, k, τ):
       Entrada: Um grafo G, um inteiro k e um conjunto de tuplas τ
       Saída: True se Struction é aplicado, False caso contrário
2      if Existe uma tupla-2 forte (u,v, 1) ∈ τ then
3          if Existe w ∈ u, v, tal que d(w) = 3 e a operação Struction é aplicavel em
             w then
4              (G, k) ← Struction(G, k);
5              return True;
6          end
7      end
8      else if Existe um vértice u ∈ G tal que d(u) = 3 ou d(u) = 4 e Struction é
         aplicável em u then
9          (G, k) ← Struction(G, k);
10         return True;
11     end
```

Algoritmo 10: Operação de struction condicional

O método é divido em 3 partes que são executadas na ordem a seguir:

1. Dado o conjunto de tuplas τ:
   - Verifica se existe alguma tupla (S, q) com q > |S|. Se sim, não existe cobertura de tamanho k em G, pois não existe cobertura mínima por vértices excluindo no mínimo q vértices de S.
   - Para cada tupla (S, q) são criadas novas tuplas (S - v, q - 1) $\forall$ v $\in$ S.
   - Se existe alguma aresta (u, v) entre os vértices em S, obrigatoriamente um dos vértices pertence à cobertura e o outro é coberto por este, logo, podemos criar uma nova tupla (S - {u,v}, q - 1).
   - Se um vértice tem uma vizinhança com mais vértices em S que |S| - q, este vértice pode ser adicionado a cobertura, pois este vértice supera a regra estipulada pela tupla. Ao adicionar o vértice, retorna chamada recursiva de VC com nova instância.
2. Verifica se há condições favoráveis para executar *Folding*, se não houver, verifica se há condições para executar *Struction*. Se algum dos métodos for executado, termina-se a redução resetando o conjunto de tuplas.
3. Se há um vértice v que domina u, então adiciona v na cobertura por vértices e retorna chamada recursiva de VC com nova instância.

```
1  Reducao(G, τ):
       Entrada: Um grafo G e um conjunto de tuplas τ
2      foreach (S, q) ∈ τ do
3          if | S | < q then
4              // Retorna erro:  não há cobertura em G de
                  tamanho k
5          end
6          foreach v ∈ S do
7              τ ← τ ∪ {(S − { v }, q − 1)};
8          end
9          if ∃ (u, w) ∈ E(G) / u ∈ S & w ∈ S then
10             τ ← τ ∪(u, v) ∈ E(G) /u ∈ S & w ∈ S;
11         end
12         if ∃ v ∈ G / |N(v) ∩ S | ≥ | S − q + 1| then
13             return (1 + VC(G- v, τ, k- 1));
14         end
15     end
16     if Folding_Condicional (G, k, τ) or Struction_Condicional
        (G, k, τ) then
17         τ ← ∅;
18         return;
19     end
20     if Existem u,v ∈ V(G) tal que o vértice v domina o vértice u then
21         return (1 + VC(G- v, τ, k- 1));
22     end
```

Algoritmo 11: Operação de redução

4.3 VC

O algoritmo 12 retorna o tamanho da cobertura mínima por vértices em um grafo G se este tamanho for menor que k, caso não exista cobertura de tamanho k em G, é reportada falha. O algoritmo apresenta uma árvore de busca onde o número de folhas possui limite superior de 1.2738^k. O algoritmo executa em tempo $O\ (1.2738^k + kn)$ (J. Chen, 2010).

```
1  VC (G, τ) :
       Entrada: Um grafo G, um inteiro k e um conjunto de tuplas τ, vazio na
                primeira chamada
       Saída: VC(G,k)
2      if k ≤ 7 then
3          // Resolva VC(G,k) por força bruta
4      end
5      Reducao (G, k, τ) ;
6      T ← {Estrutura em G de maior prioridade} ;
7      if T é uma tupla-2 ({u,z}, 1) or T é um par útil (u,z) tal que z é quase
        dominado por um vértice v ∈ N(u) or T é um vértice z tal que d(z) ≥ 7 then
8          return min { (1 + VC (G- z, τ ∪ (N(z),2), k- 1) ),
            (d(z) + VC (G- N[z], τ, k- d(z)) ), } ;
9      end
10     else
11         return min { (1 + VC (G- z, τ, k- 1) ),
            (d(z) + VC (G- N[z], τ ∪ (N(u),2), k- d(z)) ), } ;
12     end
```

Algoritmo 12: Algoritmo do cálculo da cobertura por vértices

5 CONCLUSÃO

Neste trabalho foi apresentado o algoritmo visto em (J. Chen, 2010), que resolve o problema da cobertura mínima por vértices em um grafo G.
Para isso foram utilizadas técnicas e estruturas para: Encontrar uma instância equivalente a instância original do problema (kernelização); Limitar a busca de acordo com um parâmetro específico da instância (árvores de busca limitada, tuplas); Apresentar algoritmo que resolve o problema para um conjunto específico de instâncias (restrição).
O resultado final é um algoritmo que reduz o fator exponencial do problema consideravelmente.

REFERÊNCIAS

Buss, J. F. e Goldsmith, J. (1993). Nondeterminism within p. *SIAM J. Comput.*, 22(3):560–572.

Ch. Ebenegger, P.L. Hammer, D. d. W. (1984). Pseudo-boolean functions and stability of graphs. *North-Holland Mathematics Studies*, 95:83–97.

Downey, Rod G.; Fellows, M. R. (2013). *Fundamentals of Parameterized Complexity*. Springer.

Downey, R. G. e Fellows, M. R. (1995). Parameterized computational feasibility. *Feasible Mathematics*, 2:219–244.

Hall, P. (1935). "on representatives of subsets". *J. London Math*.

Hopcroft, John E.; Karp, R. M. (1973). "an n5/2 algorithm for maximum matchings in bipartite graphs". *SIAM Journal on Computing*, 2(4):225–231.

J. A. Bondy, U. S. R. M. (1976). *Graph Theory with Applications*. NewYork: Elsevier.

J. Chen, I. Kanj, W. J. (2001). Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):208–301.

J. Chen, I. Kanj, W. J. (2010). Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756.

Kőnig, D. (1931). "gráfok és mátrixok". *Matematikai és Fizikai Lapok*, 38:116–119.

Nemhauser, G. L. e Trotter, L. E. (1975). Vertex packing: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248.

Niedermeier e Rossmanith, P. (1999). Upper bounds for vertex cover further improved. *Lecture Notes in Computer Science*, 1563:561–570.

R. Balasubramanian, M. R. Fellows, V. R. (1998). An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65:3736–3756.

R. G. Downey, M. R. Fellows, U. S. (1999). Parameterized complexity: A framework for systematically confronting computational intractability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 49:49–99.

Stege, U. e Fellows, M. (1999). An improved fixed-parameter-tractable algorithm for vertex cover.