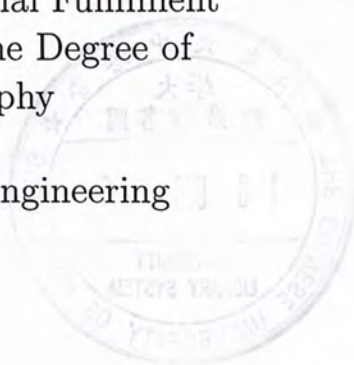


Maximum K -vertex Covers for Some Classes of Graphs

Leung Chi Wai

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering



©The Chinese University of Hong Kong
Aug 2005

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or whole of the materials in the thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.



Abstract

In this thesis we study the following MAXIMUM k -VERTEX COVER problem from the parameterized complexity point of view: find k vertices in a graph that cover a maximum number of edges.

We give uniformly polynomial-time algorithms for solving the problem on various graph classes. In particular, we devise $O(k^2n)$ algorithms for trees, partial t -trees and cographs. Furthermore, we introduce an extension method that enables us to extend a maximum $(k - 1)$ -vertex cover to a maximum k -vertex cover, and use the method to obtain a uniformly polynomial-time algorithm for planar graphs.

We also consider two related problems for some classes of perfect graphs. Our investigation produces an $O(kn^2)$ algorithm to find a maximum dominating k -set in an interval graph, and a uniformly polynomial-time algorithm to find a maximum k -vertex subgraph in a chordal graph.

摘要

在本論文中，我們從參數化複雜性的角度研究「最大 k 頂點覆蓋」的問題：從一幅圖中找出 k 個頂點去覆蓋最多的邊。

我們提出了劃一多項式時間算法以在數個圖類中解決此問題。首先，我們為樹形圖、局部 t -樹形圖及 co -圖設計了 $O(k^2n)$ 的算法。更進一步，我們引入了「擴展法」幫助我們把一個最大 $(k-1)$ 頂點覆蓋擴展成一個最大 k 頂點覆蓋，利用這個方法我們為平面圖找到劃一多項式時間算法。

我們亦在完美圖上研究兩個相關問題。我們找到 $O(kn^2)$ 的算法去在一幅區間圖上找出最大 k 頂點支配，及劃一多項式時間算法以在一幅弦圖中找出最大 k 頂點導出圖。

Acknowledgement

With a deep sense of gratitude, I wish to express my sincere thanks to my supervisor Prof. Cai Leizhen, for his support and patience in last three years. He gave me generous guidance and encouragement when I got stuck in my research. It is an invaluable experience to have my studies under his supervision.

I also want to thank my colleague Man Lam Ho, who gave me a lot of help in my studies. He really helped me to adapt to the research work in a short time.

Contents

| | |
|--|-----------|
| Abstract | i |
| Acknowledgement | iii |
| 1 Introduction | 1 |
| 1.1 Motivations | 1 |
| 1.2 Related work | 3 |
| 1.2.1 Fixed-parameter tractability | 3 |
| 1.2.2 Maximum k -vertex cover | 4 |
| 1.2.3 Dominating set | 4 |
| 1.3 Overview of the thesis | 5 |
| 2 Preliminaries | 6 |
| 2.1 Notation and definitions | 6 |
| 2.1.1 Basic definitions | 6 |
| 2.1.2 Partial t -trees | 7 |
| 2.1.3 Cographs | 9 |
| 2.1.4 Chordal graphs and interval graphs | 11 |
| 2.2 Upper bound | 12 |
| 2.3 Extension method | 14 |
| 3 Planar Graphs | 17 |
| 3.1 Trees | 17 |
| 3.2 Partial t -trees | 23 |
| 3.3 Planar graphs | 30 |

| | | |
|----------|--|-----------|
| 4 | Perfect Graphs | 34 |
| 4.1 | Maximum k -vertex cover in cographs | 34 |
| 4.2 | Maximum dominating k -set in interval graphs . . | 39 |
| 4.3 | Maximum k -vertex subgraph in chordal graphs . . | 46 |
| 4.3.1 | Maximum k -vertex subgraph in partial t -trees | 46 |
| 4.3.2 | Maximum k -vertex subgraph in chordal graphs | 47 |
| 5 | Concluding Remarks | 49 |
| 5.1 | Summary of results | 49 |
| 5.2 | Open problems | 50 |

List of Figures

| | | |
|-----|---|----|
| 2.1 | A graph G of treewidth 2 and its tree decomposition. | 8 |
| 2.2 | A regular tree decomposition of the graph G in Figure 2.1. | 10 |
| 2.3 | A cograph G and its cotree T | 10 |
| 2.4 | Relations among sets of vertices. | 13 |
| 2.5 | Example of $f(2)=2f(1)$ | 14 |
| 2.6 | S and S' | 15 |
| 3.1 | v and T_v^{i-1} in $\alpha(T_v^i, j)$ | 19 |
| 3.2 | v and T_v^{i-1} in $\beta(T_v^i, j)$ | 20 |
| 3.3 | Relation between X_v and X_{v_i} | 26 |
| 3.4 | An example on constructing G' | 31 |
| 4.1 | When v is a 0-node, the subgraphs induced by all its subtree are not connected. | 37 |
| 4.2 | When v is a 1-node, vertices in different subtrees are adjacent to each other. | 37 |
| 4.3 | Finding a maximum 2-vertex cover of the cograph in Figure 2.3 | 39 |
| 4.4 | An interval graph and its corresponding interval set. | 40 |

List of Tables

Introduction

| | | |
|-----|---|----|
| 2.1 | Treewidth of different classes of graph. | 9 |
| 4.1 | C_i , L_i , v_i and $F(v_i)$ of the interval graph in the Figure ?? | 41 |
| 4.2 | $DS(i, k')$ of interval graph in the Figure ?? | 45 |

Let us start with the following well known graph problem:

VERTEX COVER

INPUT : Graph $G = (V, E)$ and nonnegative integer k .

QUESTION : Does G contain a vertex cover of size at most k , that is, a subset V' of V of size $|V'| \leq k$ such that every edge in G is incident with at least one vertex in V' ?

The problem is one of the classical NP-complete problems [10]. However, in spite of the intractability, a lot of effort has been made [1, 29, 3] since [29], which is linear time for each fixed value of k .

Now consider the problem of finding up to k vertices that stops all road intersections. Let the profit of these vertices be proportional to the total number of roads they would intersect. We deal with equivalent graphs. We call G the road graph as a graph with each road intersection as a vertex and each road as an edge. Then the problem of finding up to k vertices that stops all road intersections is an NP-complete problem of finding k vertices that would intersect the maximum number of edges. It often happens that we have better graphs than G that we

Chapter 1

Introduction

1.1 Motivations

Let us start with the following well known graph problem.

VERTEX COVER

INPUT : Graph $G = (V, E)$ and nonnegative integer k .

QUESTION : Does G contain a vertex cover of size at most k , that is, a subset V' of at most k vertices such that every edge in G is incident with at least one vertex in V' ?

The problem is one of the classical NP-complete problems [10]. However, in spite of the intractability, it can be solved in $O(kn + 1.29^k)$ time [28], which is linear time for each fixed value of k .

Now consider the problem of setting up some convenient stores at road intersections. Let the profit of these stores be proportional to the total number of roads connected to the intersections with convenient stores. We can model the road network as a graph with each road intersection as a vertex and each road as an edge. Then the problem of setting up a minimum number of stores to gain profit from all roads corresponds to the problem of finding a minimum size vertex cover in the graph. However, it often happens that we have limited resource and thus can-

not set up enough stores to cover the whole road network. In such a situation, we usually try to make the most profit under the constraint on the number of stores. Therefore, we need to solve the problem of setting up k stores to maximize our profit, which corresponds to the MAXIMUM k -VERTEX COVER problem – find k vertices in a graph to cover a maximum number of edges.

Clearly, MAXIMUM k -VERTEX COVER is as hard as VERTEX COVER if k is a part of input, and can be solved in $O(n^k(m+n))$ time by exhaustive search. The problem has been studied in the literature, but most of the work deals with approximation algorithms. An article of Feige and Langberg (J. Alg. 41 174-211, 2001) contains a mini-survey on approximation algorithms on the problem.

In this thesis, we take the parameterized complexity point of view to study MAXIMUM k -VERTEX COVER, i.e., regard the cardinality k of the solution as a fixed parameter. We are interested in solving the problem in *uniformly polynomial time*, i.e., $O(f(k)n^c)$ time for a computable function $f(k)$ and a constant c independent of k . Recently, Cai (Parameterized complexity of cardinality constrained optimization problems, manuscript 2005) has proved that the problem is fixed-parameter intractable and is thus very unlikely to be solvable in uniformly polynomial time. Therefore in this thesis we consider the problem for special classes of graphs. In particular, we use the dynamic programming method to devise uniformly polynomial-time algorithms for trees, partial t -trees and cographs. Furthermore, we introduce an extension method that enables us to extend a maximum $(k-1)$ -vertex cover to a maximum k -vertex cover, and use the method to obtain a uniformly polynomial-time algorithm for planar graphs.

Apart from MAXIMUM k -VERTEX COVER, we also consider two related problems for some classes of perfect graphs. Our investigation produces uniformly polynomial-time algorithms for finding a maximum dominating k -set in an interval graph, and a maximum k -vertex subgraph in a chordal graph.

1.2 Related work

1.2.1 Fixed-parameter tractability

A *parameterized problem* consists of a pair (I, k) , where I is the actual input and k the parameter. Downey and Fellows[23] have introduced a theoretical framework to deal with parameterized problems. A parameterized problem (I, k) is *fixed-parameter tractable* if it admits a *uniformly polynomial-time algorithm*, i.e., an algorithm that runs in $O(f(k)|I|^c)$ time, where $|I|$ denotes the size of input, for some computable function $f(k)$ and constant c independent of k , and *fixed-parameter intractable* if it is hard for some $W[i]$ in the W -hierarchy also introduced by Downey and Fellows.

VERTEX COVER is one of most studied problems in the field of parameterized complexity. The first uniformly polynomial-time algorithm for VERTEX COVER was proposed by Buss[22] with running time $O(kn + 2^k k^{2k+2})$. Since then many researchers have tried to improve the complexity ([23], [24], [25], [26], [27]). The currently fastest algorithm is due to Chen et al.[28], which runs in $O(kn + 1.2852^k)$ time and is practical for k in a couple of hundreds.

MAXIMUM k -VERTEX COVER is an example of cardinality constrained optimization problems that ask for solutions of fixed cardinality to optimize solution values. Recently, Cai [44] has

initiated the study of the parameterized complexity of cardinality constrained optimization problems by considering twenty some fundamental graph problems.

1.2.2 Maximum k -vertex cover

Most of the existing work for MAXIMUM k -VERTEX COVER use semidefinite programming to approximate the solution. For example, [30], [31] give an approximation algorithm that find a set of k vertices covers 0.8α edges in experiment where α is the maximum number of edges that can be covered by k vertices.

Gandhi studied a related problem called PARTIAL VERTEX COVER, i.e., find the minimum number of vertex to cover k edges in a graph. He gives an approximation algorithm of approximation ratio $(2 - \theta(\frac{\ln \ln d}{\ln d}))$ in a graph of bounded degree d .

Recently, Cai[44] has studied the parameterized complexity of MAXIMUM k -VERTEX COVER and shown that the problem is W[1]-complete when restricted to regular graphs. He has also established W[1]-completeness of some special types of MAXIMUM k -VERTEX COVER for bipartite graphs(personal communication).

1.2.3 Dominating set

Finding a minimum dominating set in a general graph is NP-complete[29]. Furthermore, it was shown by Downey and Fellows that the problem is W[2]-complete[32].

For special graph classes, DOMINATING SET can be solved in polynomial time. For example, this problem is polynomial time solvable for series parallel graphs[33], outerplanar graphs[34] and interval graphs[35]. Downey and Fellows[23] showed that

the problem becomes fixed parameter tractable when restricted to planar graphs by presenting an algorithm of $O(11^k n)$ time, which has been considerably improved to $O((3^6 \sqrt{34})^{\sqrt{k}} n)$ by Alber et al. [45] using a much more sophisticated technique of tree decomposition.

1.3 Overview of the thesis

In Chapter 2, we fix notation and definitions. Then we introduce an extension method that can be used to find a maximum k -vertex cover efficiently for some special classes of graphs.

The main results of this thesis appear in Chapters 3 and 4. In Chapter 3, we first present an $O(k^2 n)$ algorithm for finding maximum k -vertex covers in trees and partial t -trees. Then we combine the “extension method” and the result on partial t -trees to derive a uniformly polynomial-time algorithm of finding a maximum k -vertex cover in a planar graph.

In Chapter 4, we consider MAXIMUM k -VERTEX COVER and two related problems for some subclasses of perfect graphs. First we give an $O(k^2 n)$ algorithm for finding a maximum k -vertex cover in a cograph. Then we present an $O(kn^2)$ algorithm for finding a maximum dominating k -set in an interval graph. We also show how to find a maximum k -vertex subgraph in a chordal graph in uniformly polynomial time.

In last chapter, we give our concluding remarks and discuss some open problems.

Chapter 2

Preliminaries

2.1 Notation and definitions

2.1.1 Basic definitions

In this thesis we follow standard notation and definition for graphs in [9]. All the graphs in this thesis are simple and undirected. The vertex set and the edge set of a graph are represented by V and E respectively. We will use n and m to denote the size of V and E respectively. For an edge $e = (u, v)$, the edge e is *covered* by u (and v). For any 2 adjacent vertices u, v , u dominates v , and vice versa.

The complement of a graph G is the graph $\overline{G} = (V, E')$, such that $uv \in E'$ iff $uv \notin E$. A graph $H = (V', E')$ is a subgraph of graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. H is an induced subgraph, denoted by $G[V']$, if H is a subgraph of G and it contains all the edges uv if $u, v \in V'$ and $uv \in E(G)$. A *path* is a list of vertices such that two vertices are consecutive in list if they are adjacent in the graph. A *cycle* is a path with identical ends. The *length* of a cycle/path is the number of edges in it. A *chord* of a cycle is an edge that is connecting two non-consecutive vertices in the cycle.

A *vertex cover* of a graph is a set of vertices $V' \subseteq V$ such that for all edge uv , at least one of $u, v \in V'$. A *dominating set* of a graph is a set of vertices $V' \subseteq V$ such that for all vertices v , $v \in V'$ or some of its neighbor is in V' . A *clique* of a graph is a complete subgraph. The *clique number* of a graph is the size of maximum clique in the graph. The *chromatic number* of a graph G is the minimum number of color to color $V(G)$ such that no two adjacent vertices share the same color.

2.1.2 Partial t -trees

Tree is one of the most well-known graph classes in graph theory. In 1983, Robertson and Seymour introduced the idea of tree-decomposition and partial t -tree to generalize the tree classes[21]. Many NP-Complete will become solvable in polynomial time when restricted to partial t -trees of some fixed t . We will use the theory of partial t -tree in Chapter 3, so we first give some basic definition and lemma about partial t -tree.

Definition 2.1.1 A complete graph of t vertices is a t -tree. Given a t -tree of n vertices, G , we can construct a t -tree of $n+1$ vertices by adding a new vertex v to G , which is made adjacent to each vertex of some clique of size t .

Based on this definition, all the trees are 1-trees.

Definition 2.1.2 A graph G is a *partial t -tree* if it is a spanning subgraph of some t -tree.

Definition 2.1.3 A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(\{X_v \mid v \in V_T\}, T = (V_T, E_T))$ with $\{X_v \mid v \in V_T\}$ a family of subsets of V , one for each node of T , and T a tree such that

- $\bigcup_{v \in V_T} X_v = V$.

- for all edges $(u, w) \in E$, there exists an $v \in V_T$ with $u \in X_v$ and $w \in X_v$.
- for any $v \in V$ and $a, b, c \in V_T$: if $v \in X_a$ and $v \in X_c$, then $v \in X_b$ for any b on the path from a to c . (That is, all the subsets containing v are connected)

Definition 2.1.4 The treewidth of a tree-decomposition is $\max_{v \in V_T} |X_v| - 1$.

Definition 2.1.5 [14] The treewidth of a graph G is the minimum treewidth of a tree-decomposition of G .

Theorem 2.1.6 (Scheffler, [12]) G has treewidth at most t if and only if G is a partial t -tree.

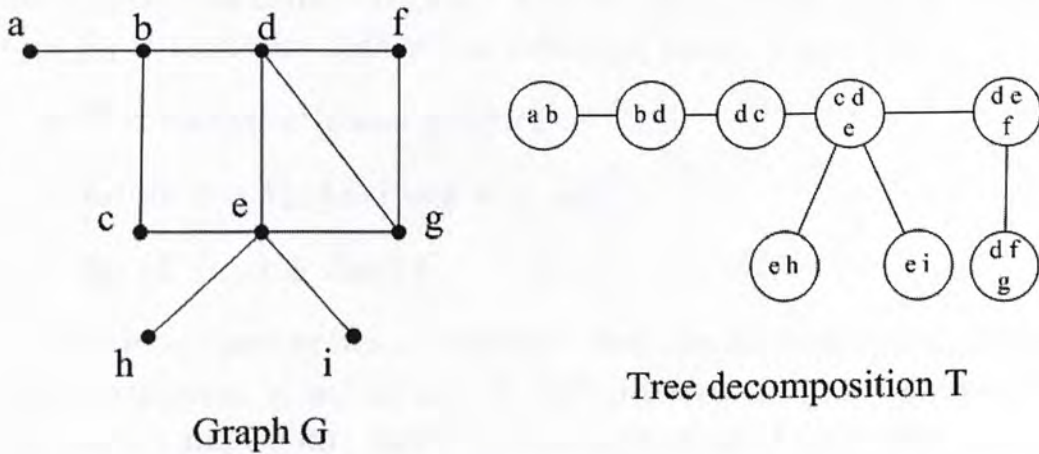


Figure 2.1: A graph G of treewidth 2 and its tree decomposition.

Determining the treewidth of a general graph G is NP-Complete. However, for certain classes of graph, there exist polynomial time algorithm to determine their treewidths. Part of these results are summarized in table 2.1.2[2].

Theorem 2.1.7 (Bodlaender, [1]) For any constant t , there is a linear time algorithm to determine if a graph G is of treewidth at most t , and if so, finds a tree-decomposition of G with treewidth at most t .

| Class | Treewidth |
|------------------------|--------------------------|
| Trees | 1 |
| Series-parallel graphs | 2 |
| Halin graphs | 3 |
| Outerplanar graphs | 2 |
| k-outerplanar graphs | $\leq 3k - 1$ |
| Chordal graphs | Size of max. clique - 1 |
| Split graphs | Size of max. clique - 1 |
| Bipartite graphs | NP-Complete to determine |

Table 2.1: Treewidth of different classes of graph.

Sometimes researchers will consider a special kind of tree-decomposition called “regular” tree-decomposition $(\{X_v | v \in V_T\}, T = (V_T, E_T))$ that also fulfills the following three properties:

- T is rooted at some node r ,
- for all $v \in V_T$, $|X_v| = t + 1$, and
- for all $(i, j) \in E_T$, $|X_i - X_j| = 1$.

The extra properties of “regular” tree-decomposition can make the description of algorithm of partial t -tree simpler. As shown by Martin and Julian, any tree-decomposition of treewidth t , can be transformed into a “regular” tree-decomposition in $O(nt)$ [13].

2.1.3 Cographs

We will consider finding maximum k -vertex covers in cographs in Section 4.1. Here we first give its definition: The class of cograph is defined recursively as follows:

- A single vertex is a cograph.
- If G_1, G_2 are vertex disjoint cographs, then $G_1 \cup G_2$ is a cograph.

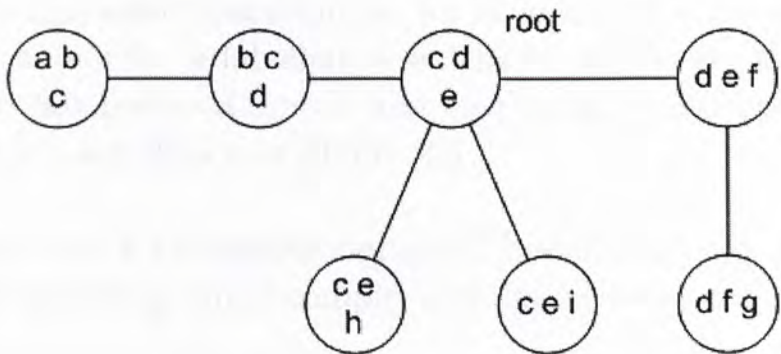


Figure 2.2: A regular tree decomposition of the graph G in Figure 2.1.

iii) If G is a cograph, then its complement \overline{G} is also a cograph.

As shown by Corneil et.al.[7], cographs are exactly the class of graphs that contain no induced P_4 , and a cograph can be represented by a tree structure, called *cotree*. Cotrees enable us to use dynamic programming to solve many difficult problems on cographs. We will also use cotrees to find a maximum k -vertex cover in cographs.

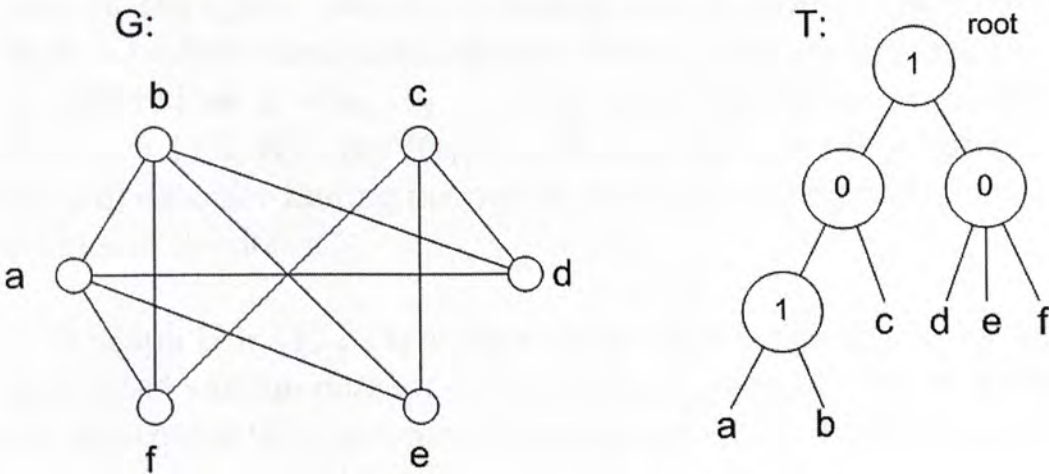


Figure 2.3: A cograph G and its cotree T

A cotree T of a cograph G is a rooted tree where each leaf represents a vertex of G and each internal node represents an

union-complement operation on its children. Furthermore the root of the cotree is labelled as a 1-node, and every child of a 1-node is labelled as a 0-node and vice versa. A cograph G and its cotree T are shown in Figure 2.3.

We can use T to construct graph G bottom-up from leaves to the root by taking union-complement operation at each internal node.

An important property of cotree is, two vertices v, u are adjacent in G if and only if, in the cotree T , the path from leaf v to root and the path from leaf u to root meet at a 1-node. For example, in Figure 2.3, vertex a and d are adjacent in G and their paths to root in T meet at the root, a 1-node.

2.1.4 Chordal graphs and interval graphs

A chordal graph is a graph where every cycle of length greater than 3 has a chord, i.e., an edge joining 2 nonconsecutive vertices in the cycle. Gavril [8] showed that a chordal graph will have a perfect elimination order. That is, the vertices set can be ordered as $\sigma = [v_1, v_2, \dots, v_n]$ such that for each v_i , the set $X_{v_i} = \{x \in N(v_i) \mid \sigma^{-1}(v_i) < \sigma^{-1}(x)\}$ is a complete subgraph. We will consider finding maximum k -vertex subgraph in chordal graphs in Section 4.3.

A graph $G = (V, E)$ is an interval graph if every vertex can be associated with an interval on the real line, such that two vertices are adjacent if their associated intervals intersect. Interval graph is a subclass of chordal graphs and we will present the algorithm for finding a maximum dominating k -set in an interval graph in Section 4.2.

2.2 Upper bound

Definition 2.2.1 For a graph G , let $f(k)$ be the maximum number of edges that can be covered by k vertices from G .

Definition 2.2.2 A maximum k -vertex cover, S_k , of a graph G is a set of k vertices from G which covers $f(k)$ edges.

Lemma 2.2.3 If a graph G needs at least k vertices to cover all edges, then for all $i \leq k$, $f(i-1) + 1 \leq f(i) \leq 2f(i-1)$.

Proof Clearly, $f(i-1) + 1 \leq f(i)$ is trivial.

To prove $f(i) \leq 2f(i-1)$, we first introduce some notation. Given two vertex covers S_{i-1} and S_i of G . We can divide V into four parts:

- $S_{i-1} \cap S_i$, this part is called B . Any edgess covered by these vertices are covered by both S_{i-1} and S_i .
- $S_{i-1} - S_i$, we call this part D .
- $S_i - S_{i-1}$, note that the size of this part is equal to $|D| + 1$. So we call this set $A \cup v$ where v is an arbitrarily vertex from this set and $|A| = |D|$.
- The vertices not in the above three parts, this part is called R .

In Figure 2.4, we use e_1 to e_8 to represent the number of edges between the vertex sets:

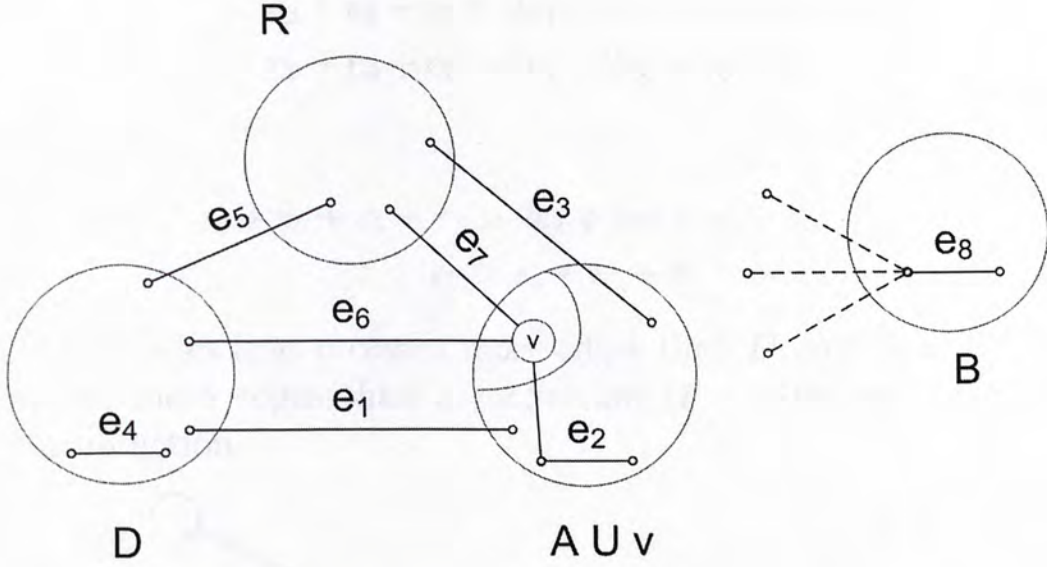


Figure 2.4: Relations among sets of vertices.

| | |
|-------|--|
| e_1 | the number of edges between A and D |
| e_2 | the number of edges in the graph induced by $A \cup v$ |
| e_3 | the number of edges between A and R |
| e_4 | the number of edges in the graph induced by D |
| e_5 | the number of edges between D and R |
| e_6 | the number of edges between v and D |
| e_7 | the number of edges between v and R |
| e_8 | the number of edges covered by B |

We now prove that $f(i) < 2f(i - 1)$. Since $B \cup D$ is a *maximum* $(i - 1)$ -vertex cover, number of edges covered by $B \cup A$ is less than or equal to that of $B \cup D$. That is,

$$\begin{aligned} e_2 + e_3 + e_1 + e_8 &\leq e_4 + e_5 + e_6 + e_1 + e_8 \\ e_2 + e_3 &\leq e_4 + e_5 + e_6 \end{aligned} \tag{2.1}$$

Suppose that $f(i) > 2f(i - 1)$. Then:

$$e_2 + e_3 + e_6 + e_7 + e_1 + e_8 > 2(e_4 + e_5 + e_6 + e_1 + e_8)$$

$$\begin{aligned}
e_2 + e_3 + e_7 &> 2e_4 + 2e_5 + e_6 + e_1 + e_8 \\
e_2 + e_3 + e_7 &> 2e_4 + 2e_5 + e_6 + e_1
\end{aligned}$$

By (2.1),

$$\begin{aligned}
e_4 + e_5 + e_6 + e_7 &> 2e_4 + 2e_5 + e_1 + e_6 \\
e_7 &> e_4 + e_5 + e_1
\end{aligned}$$

This implies that v covers more edges than D , and thus $B \cup v$ covers more edges than a *maximum* $(i - 1)$ -vertex cover, a contradiction.

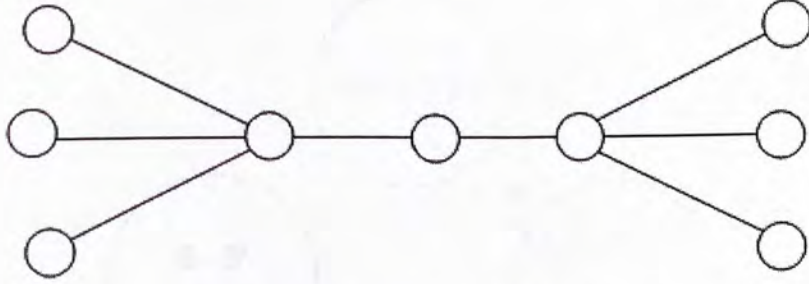


Figure 2.5: Example of $f(2)=2f(1)$

The upper bound is tight. For example, in Figure 2.4, $f(2) = 2f(1)$. □

2.3 Extension method

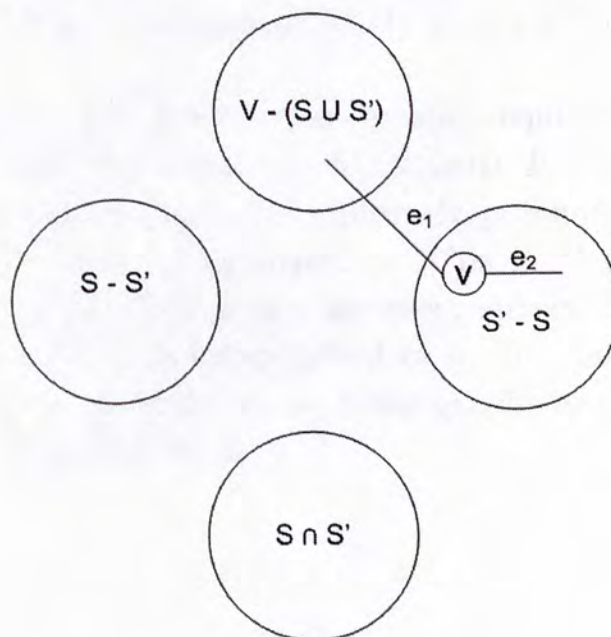
Definition 2.3.1 A maximum k -vertex cover S is **extendible** if it can be extended to a maximum $(k+1)$ -vertex cover S' by adding one more vertex.

Definition 2.3.2 For a subset V' of vertices of a graph G , let $e(V')$ denote the number of edges covered by V' .

Lemma 2.3.3 (Extension Lemma) For every maximum k -vertex cover S , $1 \leq k < n$, if S is not extendible, then for any

maximum $(k + 1)$ -vertex cover S' , every vertex in $S' - S$ is adjacent to some vertices in $S - S'$.

Proof Let v be an arbitrary vertex in $S' - S$. If v is not adjacent to any vertex in $S - S'$, then the number of edges covered by v but not by S (e_1 and e_2 in Figure 2.6) is more than or equal to the number of edges covered by v but not by $S' - v$ (e_1 in Figure 2.6). Since $e(S) \geq e(S' - v)$, we have $e(S \cup v) \geq e(S')$. Therefore $S \cup v$ is a *maximum $(i + 1)$ -vertex cover*, contrary to S being not extendible.

Figure 2.6: S and S'

□

The Extension Lemma enables us to solve MAXIMUM k -VERTEX COVER by continuously extending a solution of size i to a solution of size $i + 1$, until the solution size becomes k . This method may lead to a uniformly polynomial-time algorithm if the time spent for each extension is not too much.

Let us consider bounded degree graphs as an example. In a graph G with bounded degree d where d is a constant, any subset S of its vertices has at most $d|S|$ neighbors and there are $2^{d|S|}$ combinations of subset from the set of neighbors. Using this property, we can solve MAXIMUM k -VERTEX COVER on bounded degree graphs as follows. First we take an arbitrary vertex with maximum degree as a maximum 1-vertex cover. Then for each $i < k$, based on Lemma 2.3.3, we can extend a maximum i -vertex cover S to a maximum $(i + 1)$ -vertex cover by either adding a vertex to S ($O(n)$ possible ways) or replacing $D \subseteq S$ by $|D|+1$ vertices in $N(D)$ ($O(2^i \times 2^{id})$ possible ways).

In last paragraph, every extension step requires $O(2^{f(k)})$ time. So this method can solve the MAXIMUM k -VERTEX COVER on bounded degree graphs in uniformly polynomial time, that is, $O(n^c f(k))$ for some constant c . This method can also be applied on any graph G where for every subset S of vertices of G , the size of $N(S)$ is independent of n . In Chapter 3 we will apply the extension method on planar graphs to get a uniformly polynomial time algorithm.

Chapter 3

Planar Graphs

In this chapter, we consider MAXIMUM k -VERTEX COVER on planar graphs. In Section 3.1, we present an $O(k^2n)$ algorithm for finding a maximum k -vertex cover in a tree. Then we extend the algorithm to solve MAXIMUM k -VERTEX COVER on partial t -trees in Section 3.2. In Section 3.3, we use the algorithm on partial t -trees and the Extension Lemma in Section 2.3 to develop a uniformly polynomial-time algorithm to find a maximum k -vertex cover in a planar graph.

3.1 Trees

For trees, one can easily solve the classical VERTEX COVER problem in linear time. However, it seems much more difficult to obtain an efficient algorithm for MAXIMUM k -VERTEX COVER. In this section, we give an $O(k^2n)$ dynamic programming algorithm for finding a maximum k -vertex cover in a tree T .

When $n = k$, it is trivial that the vertices of the entire tree forms a maximum k -vertex cover, so we assume $n > k$ in this section. First, we arbitrarily choose a vertex r as the root of T and arbitrarily order the children of each internal vertex of T . Henceforth we regard T as an ordered rooted tree.

Definition 3.1.1 For each vertex v of T , $c(v)$ denotes the number of children of v , and v_i , $1 \leq i \leq c(v)$, denotes the i -th child of v .

Note that $c(v) = \deg(v)$ for $v = r$ and $c(v) = \deg(v) - 1$ for $v \neq r$.

Definition 3.1.2 For each vertex v , T_v denotes the subtree rooted at v , and T_v^i , $0 \leq i \leq c(v)$, denotes the rooted tree $T_v - \bigcup_{j=i+1}^{c(v)} T_{v_j}$, i.e., rooted tree obtained from T_v by deleting subtrees $T_{v_{i+1}}$, $T_{v_{i+2}}$, \dots , $T_{v_{c(v)}}$.

Note that T_v^0 is the tree containing the single vertex v , and $T_v = T_v^{c(v)}$. For each rooted tree T_v^i , $0 \leq i \leq c(v)$, we define two parameters $\alpha(T_v^i, j)$ and $\beta(T_v^i, j)$ for $0 \leq j \leq k$ that will be used in our dynamic programming algorithm.

Definition 3.1.3 For each $0 \leq i \leq c(v)$ and $0 \leq j \leq k$,

$\alpha(T_v^i, j)$ equals the maximum number of edges in T that can be covered by j vertices from T_v^i that include vertex v , i.e., vertex v and $j - 1$ vertices from $T_v^i - v$. For convenience, we define $\alpha(T_v^i, j)$ to be $-\infty$ if $j > |T_v^i|$.

$\beta(T_v^i, j)$ equals the maximum number of edges in T that can be covered by j vertices from T_v^i that exclude v , i.e., j vertices from $T_v^i - v$. For convenience, we define $\beta(T_v^i, j)$ to be $-\infty$ if $j > |T_v^i| - 1$.

Clearly, the maximum number of edges in T that can be covered by k vertices is equal to $\max(\alpha(T_r, k), \beta(T_r, k))$. In order to compute $\alpha(T_r, k)$ and $\beta(T_r, k)$, we establish recurrence relations for $\alpha(T_v^i, j)$ and $\beta(T_v^i, j)$ in the following lemma. Note again that $T_v = T_v^{c(v)}$ and T_v^0 is the tree containing the single vertex v .

Lemma 3.1.4 *For every $0 \leq i \leq c(v)$, $0 \leq j \leq k$, we have*

$$\begin{aligned} \alpha(T_v^i, 0) &= 0 \\ \alpha(T_v^i, 1) &= \deg(v), \text{ and} \\ \alpha(T_v^i, j) &= \begin{cases} -\infty & \text{if } j > |T_v^i| \\ \max\{\alpha(T_v^{i-1}, l) + \max[\alpha(T_{v_l}, j-l) - 1, \beta(T_{v_l}, j-l)]\} & \text{otherwise} \end{cases} \end{aligned}$$

(Note that the recurrence relation implies that $\alpha(T_v^1, j) = \deg(v) + \max[\alpha(T_{v_1}, j-1) - 1, \beta(T_{v_1}, j-1)]$.)

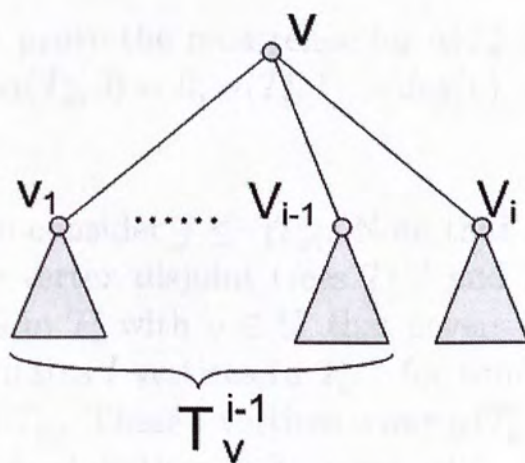
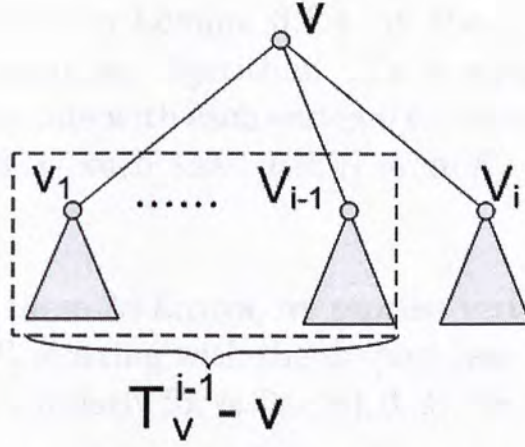


Figure 3.1: v and T_v^{i-1} in $\alpha(T_v^i, j)$

For every $0 \leq i \leq c(v)$, $0 \leq j \leq k$, we have

$$\begin{aligned} \beta(T_v^i, 0) &= 0 \\ \beta(T_v^i, j) &= \begin{cases} -\infty & \text{if } j > |T_v^i| - 1 \\ \max\{\beta(T_v^{i-1}, l) + \max[\alpha(T_{v_l}, j-l), \beta(T_{v_l}, j-l)]\} & \text{otherwise} \end{cases} \end{aligned}$$

(Note that the recurrence relation implies that $\beta(T_v^1, j) = \max[\alpha(T_{v_1}, j), \beta(T_{v_1}, j)]$.)


 Figure 3.2: v and T_v^{i-1} in $\beta(T_v^i, j)$

Proof We first prove the recurrence for $\alpha(T_v^i, j)$. It is clear by definition that $\alpha(T_v^i, 0) = 0$, $\alpha(T_v^i, 1) = \deg(v)$, and for $j > |T_v^i|$, $\alpha(T_v^i, j) = -\infty$.

It remains to consider $j \leq |T_v^i|$. Note that T_v^i can be partitioned into two vertex disjoint trees T_v^{i-1} and T_{v_i} . Let V' be a set of j vertices in T_v^i with $v \in V'$ that covers $\alpha(T_v^i, j)$ edges in T . Then V' contains l vertices in T_v^{i-1} for some $1 \leq l \leq j$ and $j - l$ vertices in T_{v_i} . These l vertices cover $\alpha(T_v^{i-1}, l)$ edges of T . The remaining $j - l$ vertices in T_{v_i} cover $\alpha(T_{v_i}, j - l)$ edges of T if vertex $v_i \in V'$, and $\beta(T_{v_i}, j - l)$ edges of T if vertex $v_i \notin V'$. Since in the former case, edge vv_i is covered by v as well, these $j - l$ vertices cover $\max(\alpha(T_{v_i}, j - l) - 1, \beta(T_{v_i}, j - l))$ edges not covered by the l vertices in T_v^{i-1} . Therefore V' covers

$$\alpha(T_v^{i-1}, l) + \max(\alpha(T_{v_i}, j - l) - 1, \beta(T_{v_i}, j - l))$$

edges, and hence $\alpha(T_v^i, j)$ equals the maximum of the above value among all possible values of l .

The proof for $\beta(T_v^i, j)$ is very similar to that of $\alpha(T_v^i, j)$, and we will omit it. \square

The recurrences in Lemma 3.1.4 lay the foundation of our dynamic programming algorithm. To compute $\alpha(T_r, k)$ and $\beta(T_r, k)$, we associate with each vertex v two arrays $a_v[0..c(v), 0..k]$ and $b_v[0..c(v), 0..k]$ such that $a_v[i, j] = \alpha(T_v^i, j)$ and $b_v[i, j] = \beta(T_v^i, j)$.

To compute these $2n$ arrays, we process vertices according to their depth in T_r starting with the deepest leaves. In computing $a_v[0..c(v), 0..k]$, similarly for $b_v[0..c(v), 0..k]$, we fill the array row by row.

For a tree with n vertices, $\sum_{v \in V} (c(v) + 1) = O(n)$. Therefore the total number of entries in arrays $a_v[0..c(v), 0..k]$ and $b_v[0..c(v), 0..k]$ is $O(kn)$. Since it takes $O(k)$ time to compute each entry of these arrays, the total time for our dynamic programming algorithm is $O(k^2n)$.

To find a maximum k -vertex cover in T_r , we need to store some extra information in the dynamic programming algorithm. For each vertex v , we use four arrays $l_v^\alpha[0..c(v), 0..k]$, $l_v^\beta[0..c(v), 0..k]$, $\lambda_v^\alpha[0..c(v), 0..k]$, $\lambda_v^\beta[0..c(v), 0..k]$ to store the extra information. As discussed in the proof of Lemma 3.1.4, a maximum j -vertex cover V' of T_v^i consists of l vertices from T_v^{i-1} and $j - l$ vertices from T_{v_i} . If V' contains vertex v , then such a value l can be obtained in the process of computing $a_v[i, j]$ and we store it in $l_v^\alpha[i, j]$. Otherwise, l can be obtained in the process of computing $b_v[i, j]$ and we store it in $l_v^\beta[i, j]$.

Furthermore, to obtain the required $j - l$ vertices from T_{v_i} , we need to know whether V' contains vertex v_i or not. Therefore, in computing $a_v[i, j]$, we put a symbol α or β in $\lambda_v^\alpha[i, j]$ depending on whether $v_i \in V'$, i.e., whether $\alpha(T_{v_i}, j - 1) - 1 \geq \beta(T_{v_i}, j - l)$. Similarly, in computing $b_v[i, j]$, we put a symbol

α or β in $\lambda_v^\beta[i, j]$ depending on whether $v_i \in V'$, i.e., whether $\alpha(T_{v_i}, j-1) \geq \beta(T_{v_i}, j-l)$.

Therefore for each $a_v[i, j] \neq -\infty$, we maintain values $l_v^\alpha[i, j]$ and $\lambda_v^\alpha[i, j]$, and for each $b_v[i, j] \neq -\infty$, we maintain values $l_v^\beta[i, j]$ and $\lambda_v^\beta[i, j]$. It is easy to see that it takes $O(1)$ extra work to compute these four values for each valid $a_v[i, j]$ and $b_v[i, j]$. We now use these four values to construct a maximum k -vertex in T as follows. Let $C(T_v^i, j, \alpha)$, $|T_v^i| \geq j$, be a maximum j -cover in T_v^i that includes vertex v , and $C(T_v^i, j, \beta)$, $|T_v^i| \geq j+1$, be a maximum j -cover in T_v^i that excludes vertex v . Then we have the following recurrences for $C(T_v^i, j, \alpha)$ and $C(T_v^i, j, \beta)$:

$$C(T_v^0, 1, \alpha) = \{v\},$$

$$C(T_v^i, j, \alpha) = C(T_v^{i-1}, l_v^\alpha[i, j], \alpha) \cup C(T_{v_i}, j - l_v^\alpha[i, j], \lambda_v^\alpha[i, j])$$

and

$$C(T_v^0, 0, \beta) = \emptyset,$$

$$C(T_v^i, j, \beta) = C(T_v^{i-1}, l_v^\beta[i, j], \beta) \cup C(T_{v_i}, j - l_v^\beta[i, j], \lambda_v^\beta[i, j]).$$

It is important to note that whenever $|T_v^i| \geq j$, $l_v^\alpha[i, j]$ is defined and thus both $C(T_v^{i-1}, l_v^\alpha[i, j], \alpha)$ and $C(T_{v_i}, j - l_v^\alpha[i, j], \lambda_v^\alpha[i, j])$ are defined. Furthermore, whenever $|T_v^i| \geq j+1$, $l_v^\beta[i, j]$ is defined and thus both $C(T_v^{i-1}, l_v^\beta[i, j], \beta)$ and $C(T_{v_i}, j - l_v^\beta[i, j], \lambda_v^\beta[i, j])$ are defined.

Since $|T_r| = n$ and $n \geq k+1$, either $C(T_r^{c(r)}, k, \alpha)$ or $C(T_r^{c(r)}, k, \beta)$ is a maximum k -vertex cover V^* of T_r , we can use the above recurrences to compute these two k -vertex covers and take the larger one as V^* . Because T_v^{i-1} and T_{v_i} are vertex disjoint, it is easy to see that the total time for recursive calls in constructing V^* is $O(n)$. Therefore the total time to construct V^* is $O(k^2n)$.

Theorem 3.1.5 *A maximum k -vertex cover of a tree T of n vertices can be found in $O(k^2n)$ time.*

3.2 Partial t -trees

In this section, we extend the idea in the last section to derive a uniformly polynomial time algorithm to solve MAXIMUM k -VERTEX COVER on partial t -trees in $O(k^2n)$ time. The algorithm will be used as a subroutine to solve MAXIMUM k -VERTEX COVER for planar graphs in the next section.

Our algorithm will use a “regular” tree-decomposition of a partial t -tree $G = (V, E)$. Recall that a “regular” tree-decomposition $(\{X_v | v \in V_T\}, T = (V_T, E_T))$ that has the following three properties:

- T is rooted at some vertex r ,
- for all $v \in V_T$, $|X_v| = t + 1$, and
- for all $(i, j) \in E_T$, $|X_i - X_j| = 1$.

For the detail please refer to Section 2.1.

Similar to the dynamic programming algorithm for trees in the previous section, we arbitrarily order the children of each internal node of T . And we regard T as an ordered tree rooted at vertex r .

To avoid confusion, we refer to vertices in T as *nodes*. Before describing the algorithm, we first give some definitions, some of them are similar to that in the previous section:

- For each node v of T , $c(v)$ is the number of children of v in T . Note that $c(v)$ equals to $\deg(v)-1$, if v is not the root of T and $\deg(v)$ if v is the root of T .
- For a node v , v_i represents the i -th child of v .

- T_v denotes the subtree rooted at node v . $T_v^i, 0 \leq i \leq c(v)$, denotes the rooted tree $T_v - \bigcup_{j=i+1}^{c(v)} T_{v_j}$, i.e., rooted tree obtained from T_v by deleting subtrees $T_{v_{i+1}}, T_{v_{i+2}}, \dots, T_{v_{c(v)}}$. Note that $T_v^0 = v$.
- For any subtree T' of T , $X(T')$ denotes the set of vertices $\{u \in X_v | v \in T'\}$.
- For $0 \leq i \leq c(v)$, $0 \leq j \leq k$ and $Z \subseteq X_v$, $\alpha(T_v^i, j, Z)$ equals the maximum number of edges that can be covered by a set S of j vertices from $X(T_v^i)$ with $S \cap X_v = Z$.

Note that for some combinations of T_v^i, j, Z , it is impossible to have a set S of j vertices from $X(T_v^i)$ with $S \cap X_v = Z$. We define $\alpha(T_v^i, j, Z)$ as $-\infty$ for such case so that it will be ignored in the computation. And based on the definition, the maximum number of edges can be covered by k vertices in G is equal to the largest $\alpha(T_r, k, Z)$ among all subsets Z of X_r .

Lemma 3.2.1 *For $0 \leq i \leq c(v)$, $0 \leq j \leq k$, $\alpha(T_v^i, j, Z)$ satisfies the following recursive formulas.*

For all node v (leave node and internal node):

$$\alpha(T_v^0, j, Z) = \begin{cases} e(Z) & \text{if } |Z| = j \\ -\infty & \text{otherwise} \end{cases}$$

For each internal node v and $1 \leq i \leq c(v)$ where $X_v - u = X_{v_i} - w$ for some vertices u, w :

$$\alpha(T_v^i, j, Z) = \begin{cases} \max_{0 \leq x \leq j} \{ \alpha(T_v^{i-1}, x, Z) + \max[\alpha(T_{v_i}, j - x + |Z - u|, Z - u), \alpha(T_{v_i}, j - x + |Z - u|, Z - u + w)] \} - e(Z - u) & \text{if } |Z| \leq j \\ -\infty & \text{otherwise} \end{cases}$$

Proof $\alpha(T_v^0, j, Z)$ means choosing j vertices only from node X_v , under the restriction of Z . If j is not equal to $|Z|$, we can't choose such a set of j vertices. So we have

$$\alpha(T_v^0, j, Z) = \begin{cases} -\infty & \text{if } |Z| \neq j \\ e(Z) & \text{otherwise} \end{cases}$$

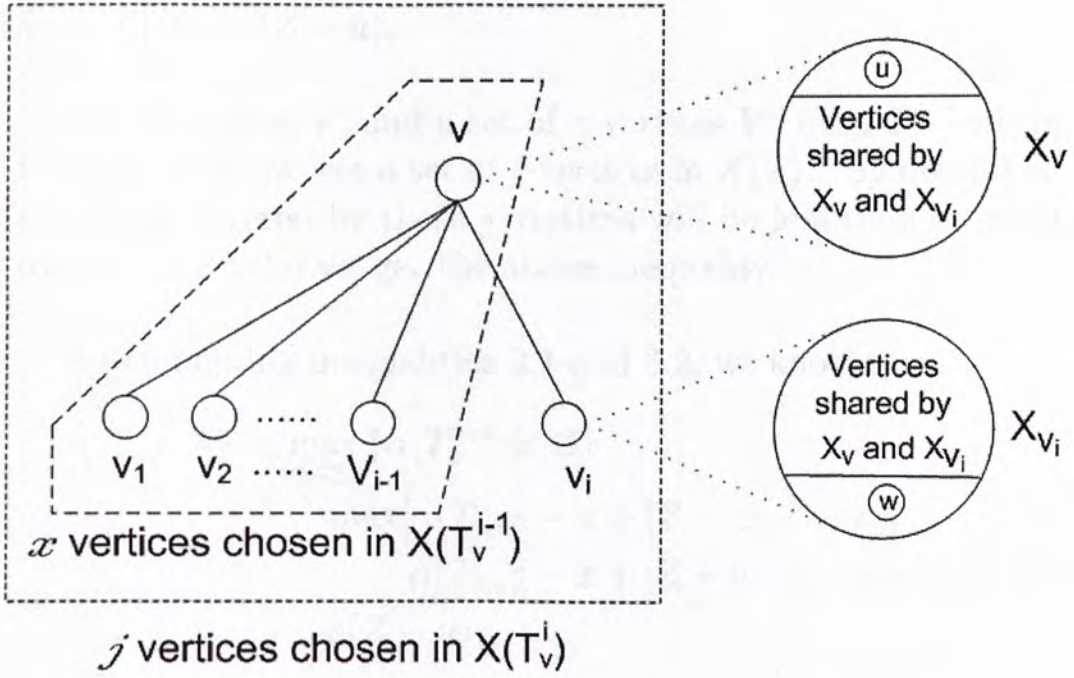
Now we consider $|Z| \leq j$. We first show that

$$\begin{aligned} \alpha(T_v^i, j, Z) \leq \max_{0 \leq x \leq j} \{ & \alpha(T_v^{i-1}, x, Z) \\ & \max[\alpha(T_{v_i}, j - x + |Z - u|, Z - u), \\ & \alpha(T_{v_i}, j - x + |Z - u|, Z - u + w)] - \\ & e(Z - u) \} \end{aligned} \quad (3.1)$$

Consider a set V' of j vertices from $X(T_v^i)$ which covers $\alpha(T_v^i, j, Z)$ edges and $V' \cap X_v = Z$. We claim that $V' \cap X_{v_i} = (Z - u)$ or $(Z - u + w)$ depending on whether $w \in V'$.

As we are given $V' \cap X_v = Z$ and $X_{v_i} = X_v - u + w$, $(Z - u) \subseteq V' \cap X_{v_i}$ (if $w \notin V'$) or $(Z - u + w) \subseteq V' \cap X_{v_i}$ (if $w \in V'$). Assume there is a vertex y , such that $y \in V' \cap X_{v_i}$ and $y \notin (Z - u)$ (or $(Z - u + w)$). Because $u \notin X_{v_i}$, $y \neq u$. Then we know that $y \notin X_v$, $y \in X_a$ for some node a in $T_v^{i-1} - v$. Consider the path $\{a, \dots, v, v_i\}$ in T , $y \in X_a, X_{v_i}$ and $y \notin X_v$. This violates the definition of tree decomposition that the subsets containing y have their nodes connected in T . So $V' \cap X_{v_i} = (Z - u)$ if $w \notin V'$, $V' \cap X_{v_i} = (Z - u + w)$ if $w \in V'$.

Since $V' \subseteq T_v^i$, V' contains x vertices in T_v^{i-1} for some $0 \leq x \leq j$, and thus V' contains $j - x + |Z - u|$ vertices in T_{v_i} . The relation is shown in Figure 3.3. The number of edges covered by these $j - x + |Z - u|$ vertices in T_{v_i} will be less than or equal to $\max[\alpha(T_{v_i}, j - x + |Z - u|, Z - u), \alpha(T_{v_i}, j - x + |Z - u|, Z - u + w)]$, so we get the above inequality.


 Figure 3.3: Relation between X_v and X_{v_i}

We now show that

$$\alpha(T_v^i, j, Z) \geq \max_{0 \leq x \leq j} \{ \alpha(T_v^{i-1}, x, Z) \max[\alpha(T_{v_i}, j - x + |Z - u|, Z - u), \alpha(T_{v_i}, j - x + |Z - u|, Z - u + w)] - e(Z - u) \} \quad (3.2)$$

Consider a set V' of $j - x + |Z - u|$ vertices from $X(T_{v_i})$, which covers $\max[\alpha(T_{v_i}, j - x + |Z - u|, Z - u), \alpha(T_{v_i}, j - x + |Z - u|, Z - u + w)]$ edges and $V' \cap X_{v_i} = (Z - u)$ or $(Z - u + w)$, depending on which choice covers more edges.

We claim that $V' \cap X_v = (Z - u)$. Assume there is a vertex y such that $y \in V' \cap X_v$ and $y \notin (Z - u)$. Because $w \notin X_v$, $y \neq w$. Then we know that $y \in X_a$ for some node a in the subtree T_{v_i} . Consider the path $\{v, v_i, \dots, a\}$ in T , $y \in X_v, X_a$ and $y \notin X_{v_i}$. This violates the definition of tree decomposition that the subsets containing y have their nodes connected in T .

So $V' \cap X_v = (Z - u)$.

By combining V' and a set of x vertices V'' from T_v^{i-1} where $V'' \cap X_v = Z$, we get a set of j vertices in $X(T_v^i)$. By definition, the edges covered by these j vertices will be less than or equal to $\alpha(T_v^i, j, Z)$. So we get the above inequality.

By combining inequalities 3.1 and 3.2, we know that,

$$\begin{aligned} \alpha(T_v^i, j, Z) = \max_{0 \leq x \leq j} \{ & \alpha(T_v^{i-1}, x, Z) \\ & \max[\alpha(T_{v_i}, j - x + |Z - u|, Z - u), \\ & \alpha(T_{v_i}, j - x + |Z - u|, Z - u + w)] \} - \\ & e(Z - u) \end{aligned}$$

□

The recurrence in Lemma 3.2.1 is the main part of our dynamic programming algorithm. To compute $\alpha(T, k, Z)$, we associate each node v with an array $a_v[0..c(v), 0..k, S]$ where $S \subseteq X_v$ such that $a_v[i, j, Z] = \alpha(T_v^i, j, Z)$.

To compute the array for each node, we process nodes according to their depth in T starting with the deepest leaves. For each $Z \subseteq X_v$, we fill the array $a_v[0..c(v), 0..k, Z]$ row by row.

For a regular tree decomposition T of a graph of n vertices, $\sum_{v \in T} c(v) = O(n)$. For each node X_v , there are 2^{t+1} possible subsets of it. So the total number of entries in array $a_v[0..c(v), 0..k, S]$ for all $S \subseteq X_v$ is $O(kn2^{t+1})$. Each entry requires $O(k)$ time to compute, so the total running time of the dynamic programming algorithm is $O(k^2 2^{t+1} n)$.

To find a maximum k -vertex cover in G , we need to store extra information in the dynamic programming algorithm. For each node X_v , we use two arrays $x_v[0..c(v), 0..k, Z]$ and $\lambda_v[0..c(v), 0..k, Z]$ for each $Z \subseteq X_v$ to store the extra information. As discussed in the proof of Lemma 3.2.1, for a set of j vertices $V' \subseteq T_v^i$ which covers $\alpha_v(T_v^i, j, Z)$ edges and $V' \cap X_v = Z$, V' will contain x vertices in T_v^{i-1} and $j - x + |Z - u|$ vertices in T_{v_i} . The value of x can be found during the computation of $\alpha(T_v^i, j, Z)$ and we store it in $x_v[i, j, Z]$.

Furthermore, to know the $j - x + |Z - u|$ vertices in T_{v_i} , we need to know whether the V' contain vertex w , the only vertex in $X_{v_i} - X_v$. In computing $\alpha(T_v^i, j, Z)$, we put the vertex w or \emptyset in $\lambda_v[i, j, Z]$ depending on whether V' contains w .

Therefore, for each $\alpha_v[i, j, Z] \neq -\infty$, we maintain the values $x_v[i, j, Z]$ and $\lambda_v[i, j, Z]$. It is obvious that the computations of $x_v[i, j, Z]$ and $\lambda_v[i, j, Z]$ take $O(1)$ extra work. Let $C(T_v^i, j, Z)$, with $\alpha_v(i, j, Z) \neq -\infty$, be a set of j vertices from $X(T_v^i)$, such that it covers $\alpha_v(i, j, Z)$ edges in G and $C(T_v^i, j, Z) \cap X_v = Z$. Then we have the following recurrence:

$$C(T_v^i, j, Z) = \begin{cases} Z & \text{if } i = 0 \\ C(T_v^{i-1}, x_v[i, j, Z], Z) \cup & \text{otherwise} \\ C(T_{v_i}, j - x_v[i, j, Z] + |Z - u|, Z \cup \lambda_v[i, j, Z]) \end{cases}$$

(In the recurrence, u represents the only vertex in $X_v - X_{v_i}$.)

According to Lemma 3.2.1, when $\alpha_v(i, j, Z) \neq -\infty$ for $i > 0$, then there exists two values $\alpha(T_v^{i-1}, x, Z) \neq -\infty$ and $\alpha(T_{v_i}, j - x + |Z - u|, Z') \neq -\infty$ for some $0 \leq x \leq j$ and $Z' \subseteq X_{v_i}$. So when $\alpha_v(i, j, Z) \neq -\infty$, $C(T_v^i, j, Z)$ is defined and so as the two subsets constructing it.

For all $Z \subseteq X_r$, the set $C(T_r, k, Z)$ that covers the maximum number of edges will be the maximum k -vertex cover V^* of G . We can use the above recurrence to compute all 2^{t+1} k -covers and take the largest as V^* .

In the above recurrence, the two subsets from T_v^{i-1} and T_{v_i} may not be disjoint. As the size of these two subsets are at most k , we can construct $C(T_v^i, j, Z)$ in $O(k)$ time. And there are up to $O(kn)$ recursive calls of the above recurrence. Therefore the total time to find V^* is $O(k^2 2^{t+1} n)$ time.

Theorem 3.2.2 *A maximum k -vertex cover on a partial t -tree can be found in $O(k^2 2^{t+1} n)$ time.*

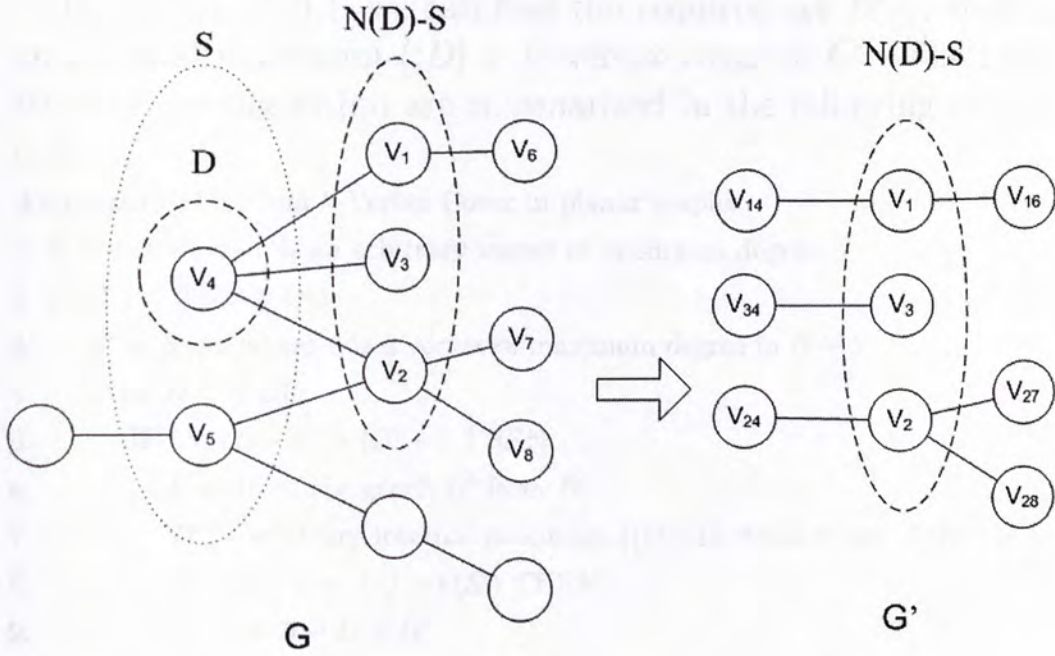
3.3 Planar graphs

In this section, we present the main result of this chapter – a uniformly polynomial-time algorithm for finding a maximum k -vertex cover in a planar graph. The main idea of our algorithm is to use the Extension Lemma in Chapter 2 to reduce the problem on a planar graph to that on a partial t -tree, and then use the algorithm in the previous section to solve the problem.

To find a maximum k -vertex cover in a planar graph, we start with a maximum 1-vertex cover by taking an arbitrary vertex of maximum degree. Then for each $1 \leq i \leq k - 1$, we use the Extension Lemma to update a maximum i -vertex cover S to a maximum $(i+1)$ -vertex cover S' by either adding a vertex to S or replacing a subset $D \subseteq S$ by a set D' of $|D| + 1$ vertices from $N(D) - S$.

If S is extendible to a maximum $(i + 1)$ -vertex cover $S'_v = S \cup \{v\}$, then v is a vertex of maximum degree in $G - S$, which can be found in linear time. Otherwise, for every $D \subseteq S$, we find a replacement D' of D in $N(D) - S$ and hence a candidate $(i + 1)$ -vertex cover $S'_D = S - D + D'$. Amongst all $(i + 1)$ -vertex covers in $\{S'_v, S'_D: D \subseteq S\}$, we take the one that covers the maximum number of edges as the maximum $(i + 1)$ -vertex cover S' .

To find the set D' for a given D , we construct a graph G' from $G[N(D) - S]$, the subgraph of G induced by $N(D) - S$, by the following operations. For every edge (u, v) in G with $u \in N(D) - S$, $v \notin N(D) \cup (S - D)$, we add a new vertex w_{uv} and connect it with vertex u in $G[N(D) - S]$. See Figure 3.4 for an example.


 Figure 3.4: An example on constructing G'

A maximum i -vertex cover of G' is called *internal* if it consists of vertices in $N(D) - S$ only. Note that for any set V' of vertices in G' with $w_{uv} \in V'$, the number of edges covered by V' will not be decreased if we replace w_{uv} by u . Therefore G' always has an internal maximum $(|D| + 1)$ -vertex cover.

Lemma 3.3.1 *For any $D \subseteq S$, every internal maximum $(|D| + 1)$ -vertex cover C of G' is a required set D' for D .*

Proof Assume the lemma is wrong, that is, there is a set of $(|D| + 1)$ vertices in $N(D) - S$, say V' , covers more edges than C in $G - (S - D)$. Note that the construction of G' guarantees that any subset of $N(D) - S$ covers the same number of edges in both $G - (S - D)$ and G' . So V' covers more edges than C in G' . This contradicts to that C is a maximum $(|D| + 1)$ -vertex cover of G' . \square

By Lemma 3.3.1, we can find the required set D' by finding an internal maximum $(|D| + 1)$ -vertex cover in G' . The main steps of our algorithm are summarized in the following pseudo code.

Algorithm Maximum k -Vertex Cover in planar graphs

1. $S = \{v\}$ where v is an arbitrary vertex of maximum degree.
2. FOR $i = 2$ TO k DO
3. $S' = S \cup v$ where v is a vertex of maximum degree in $G - S$
4. FOR $D \subseteq S$ DO
5. IF $|N(D) - S| \geq |D| + 1$ THEN
6. Construct the graph G' from D
7. $D' =$ arbitrary internal maximum $(|D|+1)$ -vertex cover of G'
8. IF $e(S - D + D') > e(S')$ THEN
9. $S' = S - D + D'$
10. $S = S'$
11. output S

We claim that the above algorithm runs in uniformly polynomial time. The key to this claim is that the graph G' has bounded tree width and thus we can use the algorithm for partial t -trees in the previous section to find D' . First we note that $G[N[D]]$, which is related to G' as we will see shortly, is a planar graph with domination number $\leq |D|$, since D forms a dominating set. It follows from the following lemma that $G[N[D]]$ has treewidth at most $6\sqrt{34}\sqrt{|D|}$.

Lemma 3.3.2 (Alber et.al. [3]) *A planar graph with domination number d has treewidth at most $6\sqrt{34}\sqrt{d}$ and such a tree decomposition can be found in $O(\sqrt{d}n)$ time.*

For graph G' , we recall that it is constructed from $G[N(D) - S]$ by attaching degree-one vertices. It is easy to see, by the definition of partial t -trees, that the treewidth of G' is the same as $G[N(D) - S]$. Since $G[N(D) - S]$ is an induced subgraph

of $G[N[D]]$, it also has treewidth at most $6\sqrt{34}\sqrt{|D|}$. Therefore G' has treewidth at most $6\sqrt{34}\sqrt{|D|}$ and thus is a partial $(6\sqrt{34}\sqrt{|D|})$ -tree. By Theorem 3.2.2, a maximum $(|D| + 1)$ -vertex cover in G' can be found in $O(k^2 2^{6\sqrt{34}\sqrt{k}} n)$ time.

Theorem 3.3.3 Algorithm **Maximum k -Vertex Cover in planar graphs** can solve **MAXIMUM k -VERTEX COVER** on planar graphs in $O(k^3 2^{k+6\sqrt{34}\sqrt{k}} n)$ time.

Proof In the algorithm, line 5-9 will be executed for $O(k^2)$ times. By Lemma 3.3.2, line 6 can be done in $O(\sqrt{k}n)$ time. Line 7 requires $O(k^2 2^{6\sqrt{34}\sqrt{k}} n)$ operations. So the total running time for finding a maximum k -vertex cover on a planar graph is $O(k^3 2^{k+6\sqrt{34}\sqrt{k}} n)$. \square

Chapter 4

Perfect Graphs

A graph G is called a perfect graph if for every induced subgraph H of G , its clique number equals chromatic number. The notion of perfect graph was introduced by Berge in early 1960s, and since then, many classes of graph are shown to be perfect. In this chapter, we study MAXIMUM k -VERTEX COVER and some related problems for various subclasses of perfect graphs. In Section 4.1, we will show how to solve MAXIMUM k -VERTEX COVER in a cograph in $O(k^2n)$ time. In Section 4.2, we give an $O(kn^2)$ algorithm to find a maximum dominating k -set in an interval graph, and, in Section 4.3, we present a uniformly polynomial time algorithm to find a maximum k -vertex subgraph in a chordal graph.

4.1 Maximum k -vertex cover in cographs

In this section, we give an $O(k^2n)$ algorithm to find a maximum k -vertex cover in a cograph. For the definition of cographs please refer to Section 2.1.3.

To derive a dynamic programming algorithm for finding a maximum k -vertex cover in a cograph G , we first give some definitions. We assume that G has at least $k + 1$ vertices, and

its cotree T is a rooted ordered tree.

Definition 4.1.1 Let v be a node of a cotree T . We use $c(v)$ represent the number of children of v , and v_i ($1 \leq i \leq c(v)$) to represent the i -th child of v .

Definition 4.1.2 For each node v of a cotree T , T_v denotes the subtree rooted at v , and $T_v^i, 0 \leq i \leq c(v)$, denotes the rooted tree $T_v - \bigcup_{j=i+1}^{c(v)} T_{v_j}$, i.e., rooted tree obtained from T_v by deleting subtrees $T_{v_{i+1}}, T_{v_{i+2}}, \dots, T_{v_{c(v)}}$.

Note that T_v^0 is the tree containing the single node v , and $T_v = T_v^{c(v)}$.

Definition 4.1.3 Let $G(T_v)$ represent the graph induced by the vertices of G corresponding to leaves of T_v . We use $|G(T_v)|$ to denote the number of vertices in $G(T_v)$, i.e., the number of leaves of T_v .

Definition 4.1.4 For each node v of a cotree T , $f(T_v^i, j)$ equals the maximum number of edges in G that can be covered by j vertices corresponding to leaves of T_v^i . If $j >$ number of leaves in T_v^i , $f(T_v^i, j)$ is defined as $-\infty$ to represent it is impossible.

By definition, the maximum number of edges can be covered by k vertices in a cograph is equal to $f(T_r, k)$.

Lemma 4.1.5 *For every $0 \leq i \leq c(v)$, $0 \leq j \leq k$, $f(T_v^i, j)$ satisfies the following recursive formula.*

If T_v^0 is a leaf node, $f(T_v^0, 1) = \text{degree of the corresponding vertex}$.

For $j = 0$, $f(T_v^i, j) = 0$

For $i > 0$ and $j > |G(T_v^i)|$, $f(T_v^i, j) = -\infty$.

For the other cases,

$$f(T_v^i, j) = \begin{cases} \max_{0 \leq l \leq j} [f(T_v^{i-1}, j-l) + f(T_{v_i}, l)] & \text{if } v \text{ is 0-node} \\ \max_{0 \leq l \leq j} [f(T_v^{i-1}, j-l) + f(T_{v_i}, l) - l(j-l)] & \text{if } v \text{ is 1-node} \end{cases}$$

Proof The first 2 lines of the lemma are self-explaning.

When $i > 0$ and $j > |G(T_v^i)|$, it is impossible to choose such amount of vertices from $G(T_v^i)$, so we define $f(T_v^i, j)$ as $-\infty$ to ignore such case.

For the remaining cases, when v is a 0-node, consider u , a leaf node of T_{v_i} and w , a leaf node of $T_{v_{i'}}$ where $i \neq i'$. The corresponding vertices of u and w are not adjacent in G , because the path from u to r meets the path from w to r at a 0-node, v . So, for a set V' of j leave vertices of T_v^i which covers $f(T_v^i, j)$ edges in G , if l vertices is from T_{v_i} , these l vertices cover $f(T_{v_i}, l)$ edges in G .

For v is a 1-node, consider u , a leaf node of T_{v_i} and w , a leaf node of $T_{v_{i'}}$ where $i \neq i'$. The corresponding vertices of u and w are adjacent in G , because the path from u to r meets the path from w to r at a 1-node, v . So, when we combine l leave vertices from T_{v_i} with $j-l$ leave vertices of T_v^{i-1} , $l(j-l)$ edges are covered by both sets. So in the formula we minus the term

$l(j - l)$ to cancel the double counting.

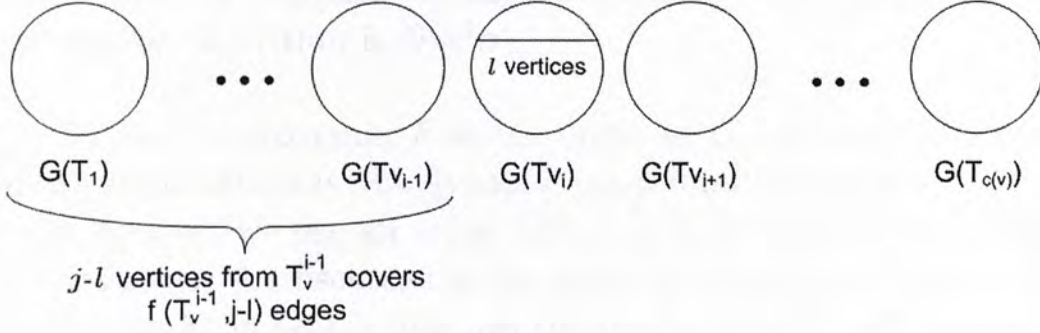


Figure 4.1: When v is a 0-node, the subgraphs induced by all its subtree are not connected.

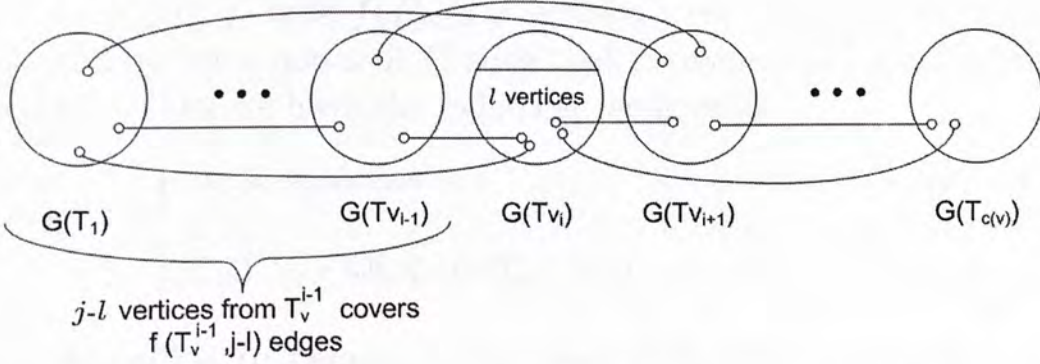


Figure 4.2: When v is a 1-node, vertices in different subtrees are adjacent to each other.

□

To compute $f(T_v^i, j)$, we associate each node with an array $a_v[0..c(v), 0..k]$ such that $a_v[i, j] = f(T_v^i, j)$. To compute the array for each node, we process nodes according to their depth in T starting with deepest leaves. For each node, we fill the array $a_v[i, j]$ row by row.

As every internal node of a cotree has at least 2 children node, $\sum_{v \in T} c(v) = O(n)$. So for the whole cotree, there are totally

$O(kn)$ entries in array $a_v[i, j]$ to be filled. Each entry requires $O(k)$ time to compute, so the total time of the dynamic programming algorithm is $O(k^2n)$.

To find a maximum k -vertex cover in G , we need to store extra information in the dynamic programming algorithm. For each node v , we use an array $l_v[0..c(v), 0..k]$ to store the extra information. As discussed in the proof of Lemma 4.1.5, a set of j vertices V' in $G(T_v)$ that has the nodes from T_v^i , will contain l nodes from T_{v_i} for some l . The value of l can be found in computation of $f(T_v^i, j)$ and we store it in $l_v[i, j]$.

Let $C(T_v^i, j)$ with $f(T_v^i, j) \neq -\infty$ be a set of j vertices represented by leave nodes of T_v^i such that it covers $f(T_v^i, j)$ edges in $G(T_v)$. Then we have the following recurrence:

$$C(T_v^i, j) = \begin{cases} \text{vertex represented by } v & \text{if } v \text{ is a leave node and } j = 1 \\ \phi & \text{if } v \text{ is a leave node and } j = 0 \\ C(T_v^{i-1}, j - l_v[i, j]) \cup C(T_{v_i}, l_v[i, j]) & \text{otherwise} \end{cases}$$

According to Lemma 4.1.5, when $f(T_v^i, j) \neq -\infty$ with v is an internal node, then there exists $f(T_v^{i-1}, j - l) \neq -\infty$ and $f(T_{v_i}^{c(v_i)}, l) \neq -\infty$ for some $0 \leq l \leq j$. So when $f(T_v^i, j) \neq -\infty$, $C(T_v^i, j)$ is defined and so as the 2 subsets constructing it.

In the above recurrence, there are up to $O(kn)$ recursive calls of the recurrence, and each call requires $O(1)$ operation. So the above recurrence can form $C(T_r^{c(r)}, k)$ in $O(kn)$. As we need $O(k^2n)$ to build the $l_v[i, j]$ table, the total running time to find a maximum k -vertex cover in a cograph is $O(k^2n)$.

Theorem 4.1.6 *A maximum k -vertex cover in a cograph can be found in $O(k^2n)$ time.*

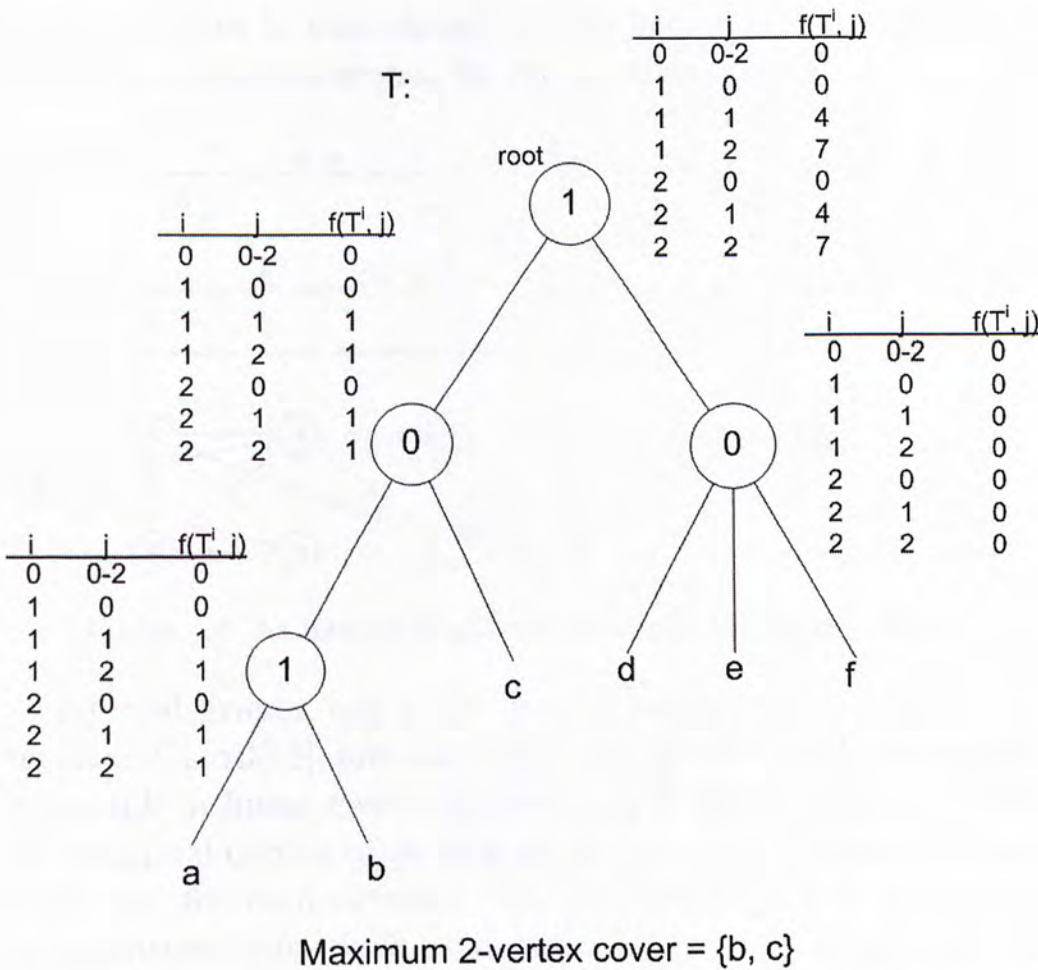


Figure 4.3: Finding a maximum 2-vertex cover of the cograph in Figure 2.3

4.2 Maximum dominating *k*-set in interval graphs

In this section, we present an $O(n^2k + n^w)$ algorithm to find a maximum dominating k -set in an interval graph, where n^w , current $w \simeq 2.376$, is the running time of multiplying two $n \times n$ matrices.

A graph $G = (V, E)$ is an interval graph if every vertex can be associated with an interval on the real line, such that two vertices are adjacent if their associated intervals intersect. A set

S of k vertices is a maximum dominating k -set in a graph G if $|N[S]|$ is maximum among all the set of k vertices.

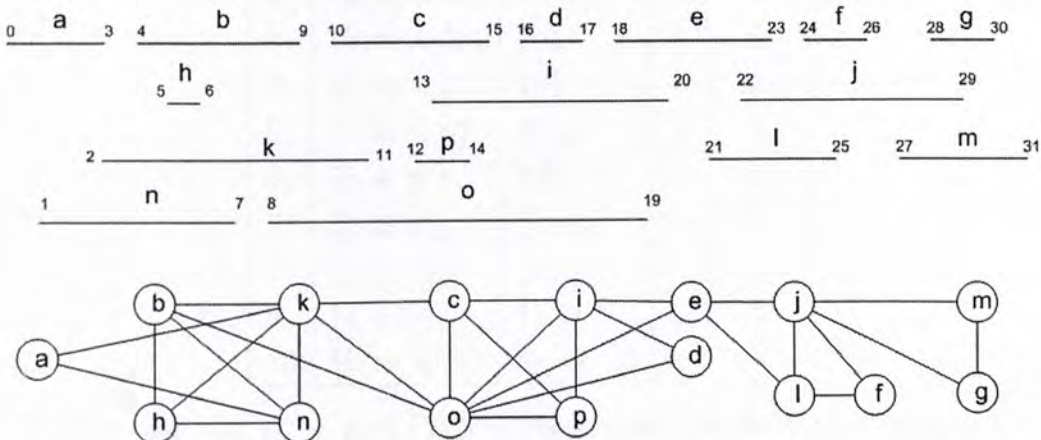


Figure 4.4: An interval graph and its corresponding interval set.

Interval graphs are a subclass of chordal graphs. Using a result of Gavril[15], one can find all maximal cliques of an interval graph in linear time. Gilmore and Hoffman [4] proved that the maximal cliques of an interval graph can be linearly ordered such that for each vertex v , the maximal cliques containing v occur successively. Such an ordering of maximal cliques can be found in linear time [5].

Definition 4.2.1 We call the linearly ordered cliques as C_1, C_2, \dots, C_p where p is the number of maximal cliques in G .

Definition 4.2.2 For each C_i , we define L_i as follow. $L_p = C_p$, $L_i = C_i - \cup_{j=i+1}^p C_j$ for $i < p$. That is, if $v \in L_i$, C_i contains the last appearance of v in the ordered cliques. Note that $L_1 \cup L_2 \cup \dots \cup L_p = V$.

Definition 4.2.3 The index $F(v)$ of v is the smallest number i such that $v \in C_i$. Let v_i be a vertex in L_i with the smallest index.

| i | C_i | L_i | v_i | $F(v_i)$ |
|-----|--------------|-----------|-------|----------|
| 1 | {k, n, a } | {a} | a | 1 |
| 2 | {k, n, b, h} | {n, h} | n | 1 |
| 3 | {k, o, b } | {b} | b | 2 |
| 4 | {k, o, c } | {k} | k | 1 |
| 5 | {i, o, c, p} | {c, p} | c | 4 |
| 6 | {i, o, d } | {d} | d | 6 |
| 7 | {i, o, e } | {i, o} | o | 3 |
| 8 | {j, l, e } | {e} | e | 7 |
| 9 | {j, l, f } | {l, f} | l | 8 |
| 10 | {j, m, g } | {j, m, g} | j | 8 |

Table 4.1: C_i , L_i , v_i and $F(v_i)$ of the interval graph in the Figure 4.4

Lemma 4.2.4 *A maximum dominating k -set of G can be obtained from k vertices in $\{v_1, v_2, \dots, v_p\}$.*

Proof For a vertex u in L_i , it appears in $C_{F(u)}C_{F(u)+1} \dots C_i$. Since v_i appears in $C_{F(v_i)}C_{F(v_i)+1} \dots C_i$ where $F(v_i) \leq F(u)$, so $N[u] \subseteq N[v_i]$ \square

Before deriving the algorithm to find a maximum dominating k -set, we first introduce a labelling scheme that will help us to derive the algorithm. Ramalingam and Rangan [6] showed that for an interval graph $G = (V, E)$, its vertices can be labelled from 1 to n , represented by $label(v)$, such that for $i < j < k$, if $(i, k) \in E$, then $(j, k) \in E$. They also gave an algorithm for finding such labelling. We slightly modify their algorithm so that v_i will get the smallest label in L_i . This property will be used in the next part to derive the final algorithm.

Algorithm Labelling

1. $l = n$
2. FOR $i = p$ downto 1 DO
3. FOR each vertex v in L_i DO

4. label v as l
5. $l = l - 1$;
6. $v = v_i$ (the vertex with smallest $F(v)$ in L_i).
7. $u =$ the vertex with smallest label in L_i .
8. IF $v \neq u$, THEN swap their label.

Lemma 4.2.5 *The labelling produced by algorithm **Labelling** will maintain the following 2 properties:*

- For $i < j < k$, if $(i, k) \in E$, then $(j, k) \in E$.
- v_i will get the smallest label among all vertices in L_i

Proof To prove the first property, we will use $first(v)(last(v))$ to represent the minimum(maximum) index i such that $v \in C_i$. In the algorithm, vertices in L_{i+1} will be labelled before that of L_i . So for $i < j < k$, $last(i) \leq last(j) \leq last(k)$. If $(i, k) \in E$, i, k will both exist in some clique C_x where $x \leq last(i)$, and so $first(k) \leq last(i)$. From this inequality, we see that $first(k) \leq last(i) \leq last(j) \leq last(k)$, that means j, k will both exist in a certain maximal clique. So we know $(j, k) \in E$.

The second property is obviously maintained by lines 6-8 of the algorithm. \square

Definition 4.2.6 *Let $DS(i, k')$ be the maximum number of vertices dominated by v_i and $k' - 1$ vertices from $\{v_{i+1}, v_{i+2}, \dots, v_p\}$ where v_i has the smallest $F(v)$ among the k' vertices.*

Note that for certain i, k' , there may be less than $k' - 1$ vertices in $\{v_{i+1}, v_{i+2}, \dots, v_p\}$ with $F(v) \geq F(v_i)$, in such case we define $DS(i, k') = -\infty$.

Lemma 4.2.7 *For $1 \leq i \leq p$, $1 \leq k' \leq k$, $DS(i, k')$ satisfies the following recurrence relation:*

For $k' = 1$, $DS(i, k') = |N[v_i]|$

For the other cases,

$$DS(i, k') = \begin{cases} -\infty & F(v_i) > F(v_x) \ \forall x > i \\ \max_{i < x \leq p, F(v_i) \leq F(v_x)} (DS(x, k' - 1) + |N[v_i]| - |N[v_x] \cap N[v_i]|) & \text{otherwise} \end{cases}$$

Proof First line of the lemma is trivial.

To prove the correctness of second line, let S be the set of vertices corresponding to $DS(x, k' - 1)$ as S . If $F(v_i) > F(v_x)$ for all $x > i$, then we have no need to consider taking v_i because $N[v_i] \subset N[v_x]$, and this should be considered as impossible and defined with the value $-\infty$.

Now we assume $F(v_i) \leq F(v_x)$ for some $x > i$, we will show that every vertex u dominated by both v_i and S is dominated by v_x . First, $label(v_i) < label(v_x)$. Because our labelling algorithm will first label vertex in L_x before labelling L_i for $i < x$. $label(u)$ will fall in either one of the following cases:

- $label(u) > label(v_x)$
Because $label(v_i) < label(v_x) < label(u)$, $(v_i, u) \in E$ implies $(v_x, u) \in E$.
- $label(u) < label(v_x)$
As $label(u) < label(v_x)$, $u \in L_{x'}$ for some $x' < x$.
Assume u is dominated by some $v_y (y > x)$ in S but not v_x .
Then the edge (u, v_y) will not exist in $C_{F(v_x)}, C_{F(v_x)+1}, \dots, C_x$, otherwise the edge (u, v_x) exists. If (u, v_y) exists in $C_{F(v_x)-\delta}$ for $\delta > 0$, then $F(v_y) < F(v_x)$, violating v_x has the smallest $F(v)$ in S . If (u, v_y) exists in $C_{x+\delta}$ for $\delta > 0$, then it conflicts with $u \in L_{x'}$ for some $x' < x$.

As every vertex u dominated by both v_i and S is dominated by v_x , we know that there are $|N[v_i] \cap N[v_x]|$ vertices dominated by

both S and v_i . So we get we formula $DS(x, k' - 1) + |N[v_i]| - |N[v_i] \cap N[v_x]|$. \square

By definition, the maximum number of vertices dominated by k vertices is equal to maximum $DS(i, k)$ for $1 \leq i \leq p$. However, $DS(i, k)$ may be equal to $-\infty$ for all $1 \leq x \leq p$ if the interval graph can be dominated by less than k vertices, so we will compute all $DS(i, k')$ for $1 \leq i \leq p$, $1 \leq k' \leq k$, and output the entry with maximum value as the solution. As finding the consecutive clique ordering C_1, C_2, \dots, C_p can be done in linear time, we assume it is part of the input.

Algorithm MaxDS(k)

1. Find v_i for $1 \leq i \leq p$.
2. Compute $|N[v] \cap N[u]|$ for all pair of v, u by matrix multiplication
3. $DS(p, 1) = \deg(v_p) + 1$
4. FOR $i = p - 1$ downto 1 DO
5. FOR $k' = 1$ to k DO
6. Compute $DS(i, k')$ based on the Lemma 4.2.7.
7. IF $k' = 1$ THEN
8. $a[i, k'] = \emptyset$
9. ELSE IF $DS(i, k') \neq -\infty$ THEN
10. $a[i, k'] = v_i$.
11. Find i, k' where $DS(i, k')$ is maximum.
12. $S = \phi$
13. WHILE $k' > 0$
14. $S = S \cup v_i$
15. $i = a[i, k']$
16. $k' = k' - 1$
17. OUTPUT S

Theorem 4.2.8 *The algorithm **MaxDS**(k) can find a maximum dominating k -set in an interval graph in $O(n^2k + n^w)$ time.*

Proof The algorithm uses Lemma 4.2.7 to compute $DS(i, k')$ for all possible i, k' pairs. From Lemma 4.2.7, we know that the maximum value of $DS(i, k')$ must be positive, since $DS(i, 1) > 0$ for all i . For each $DS(i, k') \neq -\infty$, $a[i, k']$ is defined and store the next vertex to be included in the set corresponding to $DS(i, k')$. So in line 11 to line 17, our algorithm will find out the maximum dominating k -set correctly.

The complexity analysis of the algorithm is as follows. Line 1 needs $O(n)$ operations. Line 6 to 10 will be run for $O(pk) = O(nk)$ times, where each time needs $O(n)$ operations. So line 6 to 10 needs $O(n^2k)$ operations. Line 11 to 17 can be done in $O(nk)$ time. So the total complexity of our algorithm is $O(n^2k + n^w)$ where n^w is running of multiplying two $n \times n$ matrices. (The best matrix multiplication algorithm currently known was presented by Don Coppersmith and S. Winograd in 1990 [16], has an asymptotic complexity of $O(n^{2.376})$). \square

| $i \backslash k'$ | 1 | 2 | 3 |
|-------------------|---|-----------|-----------|
| 10 | 6 | $-\infty$ | $-\infty$ |
| 9 | 4 | 6 | $-\infty$ |
| 8 | 5 | 5 | 8 |
| 7 | 7 | 7 | 13 |
| 6 | 3 | 3 | 9 |
| 5 | 5 | 5 | 12 |
| 4 | 7 | 7 | 16 |
| 3 | 5 | 5 | 15 |
| 2 | 5 | 5 | 16 |
| 1 | 3 | 3 | 15 |

Table 4.2: $DS(i, k')$ of interval graph in the Figure 4.4

For a subclass of interval graphs, called proper interval graphs,

there is an ordering such that for all vertex v , $N[v]$ comes in a consecutive sequence. And this ordering can be found in linear time[17]. With this ordering, we can find $|N[v] \cap N[u]|$ in $O(1)$. So we can modify the line 2 of our algorithm to find a maximum dominating k -set in proper interval graphs in $O(n^2k)$ time.

Theorem 4.2.9 *A maximum dominating k -set can be found in a proper interval graph in $O(n^2k)$ time.*

4.3 Maximum k -vertex subgraph in chordal graphs

A set S of k vertices is a maximum k -vertex subgraph of graph G if the subgraph induced by S , $G(S)$, has the maximum number of edges among all the subgraphs induced by k vertices of G . In this section we first derive a uniformly polynomial time algorithm for finding a maximum k -vertex subgraph in partial t -trees. And then extend the algorithm to an $O(k^2 2^k n)$ algorithm for finding a maximum k -vertex subgraph in chordal graphs.

4.3.1 Maximum k -vertex subgraph in partial t -trees

In this part we will find a maximum k -vertex subgraph in a partial t -tree $G = (V, E)$ by using its “regular” tree-decomposition $(\{X_v | v \in V_T\}, T = (V_T, E_T))$.

We also use the same notation $c(v)$, v_i , T_v and $X(T')$ as section 3.2.

Definition 4.3.1 *Let $\alpha(T_v^i, j, Z)$ be the size of a maximum subgraph induced by a set, S , of j vertices from $X(T_v^i)$ with $S \cap X_v = Z$.*

Recall that in a “regular” tree-decomposition, for all $(i, j) \in E_T$, $|X_i - X_j| = 1$, and we have the following definition.

Definition 4.3.2 We use u to represent the only vertex in $X_v - X_{v_i}$ and w to represent the only vertex in $X_{v_i} - X_v$.

Lemma 4.3.3 For $0 \leq j \leq c(v)$, $0 \leq j \leq k$, $\alpha(T_v^i, j, Z)$ satisfies the following recursive formulas.

For all node v (leave node and internal node):

$$\alpha(T_v^0, j, Z) = \begin{cases} -\infty & \text{if } |Z| \neq j \\ \text{size of the subgraph induced by } Z & \text{otherwise} \end{cases}$$

For each internal node v and $1 \leq i \leq c(v)$ where $X_v - u = X_{v_i} - w$ for some vertices u, w :

$$\alpha(T_v^i, j, Z) = \begin{cases} -\infty & \text{if } |Z| > j \\ \max_{0 \leq x \leq j} \{ \alpha(T_v^{i-1}, x, Z) + \\ \quad \max[\alpha(T_{v_i}, j - x + |Z - u|, Z - u), \\ \quad \alpha(T_{v_i}, j - x + |Z - u|, Z - u + w)] \} - \\ \quad \text{size of the subgraph induced by } Z & \text{otherwise} \end{cases}$$

The proof of this lemma and the algorithm for finding a maximum k -vertex subgraph is similar to that in Section 3.2, we omit it here.

4.3.2 Maximum k -vertex subgraph in chordal graphs

A graph G is called a chordal graph if every cycle of length greater than 3 has a chord, i.e., an edge joining 2 non-consecutive vertices in the cycle. Split graphs and interval graphs are subclasses of chordal graphs. For chordal graphs, there exists linear time algorithm to find its maximum clique[15] and its tree-decomposition [20].

For any graph G , the size of a maximum k -vertex subgraph is between 0(the k vertices form an independent set) and $k(k -$

$1)/2$ (the k vertices form a clique). So we can find a maximum k -vertex subgraph in a chordal graph G in this way. First, find a maximum clique C in G . If C contains more or equal to k vertices, output C as solution. Otherwise the treewidth of $G \leq k$ (treewidth of a chordal graph is bounded by the size of its maximum clique), then we can find a maximum k -vertex subgraph by the uniformly polynomial time algorithm in section 4.3.1.

The running time of the above method is dominated by the algorithm of finding a maximum k -vertex subgraph in a partial t -tree, which needs $O(k^2 2^{t+1} n)$ time. If the treewidth is bounded by k , the running time becomes $O(k^2 2^k n)$. So we come to the following result.

Theorem 4.3.4 *A maximum k -vertex subgraph can be found in a chordal graph in $O(k^2 2^k n)$ time.*

Chapter 5

Concluding Remarks

5.1 Summary of results

In this thesis, we have mainly studied the problem of finding maximum k -vertex covers in various graph classes.

First we have obtained the Extension Lemma which shows a useful relation between maximum $(k - 1)$ -vertex covers and maximum k -vertex covers in general graphs and plays a key role in our uniformly polynomial time algorithm for planar graphs.

For trees, we have given an $O(k^2n)$ algorithm to find maximum k -vertex covers. We have also devised an $O(k^22^{t+1}n)$ algorithm to find maximum k -vertex covers in partial t -trees. Combining the Extension Lemma and the algorithm for partial t -trees, we have obtained a uniformly polynomial algorithm for finding maximum k -vertex covers in planar graphs. We have also obtained an $O(k^2n)$ algorithm for finding maximum k -vertex covers in cographs.

Furthermore we have studied two related problems for some subclasses of perfect graphs. We have devised an $O(kn^2)$ algorithm to find maximum dominating k -sets in interval graphs, and a uniformly polynomial time algorithm for finding maxi-

mum induced k -subgraphs in chordal graphs.

5.2 Open problems

In Chapter 3, we develop an $O(k^2n)$ algorithm to find a maximum k -vertex cover in a tree. It is interesting to investigate the structural properties of tree, to help us solving the problem more efficiently. If we can find a faster algorithm, it may give us direction to improve the algorithm of partial t -trees and planar graphs.

Extension Lemma is the key idea of Section 3.3. We can consider if this Lemma can be applied in other graph classes to solve MAXIMUM k -VERTEX COVER efficiently. Using similar logic, we can easily derive the "Extension Lemma" for MINIMUM k -VERTEX COVER :

Lemma 5.2.1 *For every minimum k -vertex cover S , $1 \leq k < n$, if S is not extendible, then for any maximum $(k+1)$ -vertex cover S' , every vertex in $S' - S$ is NOT adjacent to any vertex in $S - S'$.*

It is interesting to study if this lemma can help us to solve MINIMUM k -VERTEX COVER or MINIMUM $(n-k)$ -VERTEX COVER.

In section 2.3, we proposed the Extension method to solve MAXIMUM k -VERTEX COVER on bounded degree graphs. Such methods can be applied to any problem once we find some relationship between solution of size i and solution of size $(i+1)$. So we are interested to find such relationship in other NP-Complete problems.

VERTEX COVER becomes polynomial time solvable when restricted to various graph classes, for example, bipartite graphs,

interval graphs, chordal graphs, etc. However, in these graph classes, it is still unknown how fast we can solve MAXIMUM k -VERTEX COVER.

Bibliography

- [1] Hans L. Bodlaender, A Linear Time Algorithm for Finding Tree-decompositions of Small Treewidth, *SIAM Journal on Computing*, Vol. 25, 1305-1317, 1996.
- [2] Hans Bodlaender, A treewidth guide through complexity theory, *Combinatorica* 11, 1-21, 1993.
- [3] Jochem Alber and Hans L. Bodlaender and Henning Fernmel and Rolf Niedermeier, First Parameter: The search for PLANAR DOMINATING SET and Related Problems, Scandinavian Workshop on Algorithm Theory, 404-419, 1999.
- [4] P.C. Gilmore and A.J. Hoffman, A characterization of comparability graphs and of interval graphs, *Canadian Journal of Math.* Vol. 16, 539-548, 1964.
- [5] R.S. Booth, G.S. Lasker, Tables for the recognition of the property interval graphs and comparability graphs, *Linear algorithms, J. Comput. Syst. Sci.*, Vol. 1, 375-377, 1970.
- [6] U. Brandenburg, and C. Pander, Maximal 3-uniform hypergraphs to domination problems in interval graphs, *Information Processing Letters*, Vol. 27, 271-274, 1987.
- [7] P.G. Cornuéjols, H. Lewis, L. Schaefer, and S. Voss, The unit interval Reducible Graphs, *Discrete Math.*, Vol. 94, 163-174, 1991.

Bibliography

- [1] Hans L. Bodlaender. A Linear Time Algorithm for Finding Tree-decompositions of Small Treewidth, *SIAM Journal on Computing*, Vol. 25, 1305-1317, 1996.
- [2] Hans Bodlaender. A tourist guide through treewidth, *Acta Cybernetica* 11, 1-21, 1993.
- [3] Jochen Alber and Hans L. Bodlaender and Henning Fernau and Rolf Niedermeier. Fixed Parameter Algorithms for PLANAR DOMINATING SET and Related Problems, *Scandinavian Workshop on Algorithm Theory*, 97-110, 2000
- [4] P.C. Cilmore and A.J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian J. Math.*, Vol. 16, 539-548, 1964.
- [5] K.S. Booth, G.S Leuker. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, Vol. 13, 335-379, 1976.
- [6] G. Ramalingam, and C. Pandu Rangan. A unified approach to domination problems in interval graphs. *Information Processing Letters*, Vol. 27, 271-274 1988.
- [7] D.G.Corneil, H.Lerchs, L.Stewart Burlingham. Complement Reducible Graphs. *Discrete Applied Mathematics*, Vol. 3, 163-174, 1981.

- [8] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques and maximum independent set of a chordal graph. *SIAM J. Comput.* 1, 180-187, 1972.
- [9] D.B. West, *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River 1996.
- [10] Karp, R.M., "Reducibility among combinatorial problems", in *Complexity of Computer Communications*, R.E. Miller and J.W. Thatcher (editors) , pp.85-103, Plenum Press, New York 197
- [11] I. Dinur and S. Safra. The importance of being biased. In *Proceedings of the 33th Annual ACM Symposium on Theory of Computing*, Montr eal, Canada, pages 33-42, 2002
- [12] P. Scheffler. The graphs of tree-width k are exactly the partial k -trees, manuscript 1986.
- [13] Martin Grohe and Julian Mari. Definability and Descriptive Complexity on Databases of Bounded Tree-Width, *LNCS* Vol. 1540, 70-82, 1998.
- [14] N. Robertson, PD Seymour. Graph minors II. Algorithmic aspects of tree-width, *Journal of Algorithms*, Vol.7, 309-322, 1986.
- [15] Fanica Gavril. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph, *SIAM J. Comp.* Vol.1, 180-187, 1972.
- [16] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions, *J. Symbolic Computation* Vol. 9, 251-280, 1990.

- [17] D.R. Fulkerson, O.A. Gross. Incidence matrices and interval graphs, *Pacific J. Math.*, Vol.15 (1965), 835-855.
- [18] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems on graphs embedded in k-trees. *Discrete Applied Mathematics*, 23:11-24, 1989.
- [19] A Hans L. Bodlaender. Dynamic Programming on Graphs with Bounded Treewidth, in *Proc. 15th International Colloquium on Automata, Languages and Programming, LNCS 317(1988)*, 105-118.
- [20] Golumbic, Martin Charles., *Algorithmic graph theory and perfect graphs*, published by New York : Academic Press, 1980.
- [21] N. Robertson and P. D. Seymour, Graph Minors I. Excluding a forest, *J. Combin. Theory Ser. B* 35 (1983), 39-61. 8.
- [22] J. F. Buss and J. Goldsmith, Nondeterminism within P, *SIAM J. Comput.* 22 (1993), 560-572.
- [23] R. G. Downey and M. R. Fellows, Parameterized computational feasibility, in *Feasible Mathematics II* (P. Clote and J. Remmel, Eds.), pp. 219-244, Boston, Birkhauser, 1995.
- [24] R. G. Downey, M. R. Fellows, and U. Stege, Parameterized complexity: A framework for systematically confronting computational intractability, in *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future* (F. Roberts, J. Kratochvil, and J. Nešetřil, Eds.), *AMS-DIMACS Proceedings Series*, Vol. 49, pp. 49-99, Am. Math. Soc., Providence, 1999.
- [25] R. Niedermeier and P. Rossmanith, Upper Bounds for Vertex Cover Further Improved, *Lecture Notes in Computer*

- Science, Vol. 1563, STACS99, pp. 561-570, Springer-Verlag, New York, 1999.
- [26] R. Niedermeier and P. Rossmanith, A general method to speed up fixed-parameter tractable algorithms, *Inform. Process. Lett.* 73 (2000), 125-129.
- [27] R. Balasubramanian, M.R. Fellows, V. Raman, An improved fixed parameter algorithm for vertex cover, *Inform. Process. Lett.* 65 (3) (1998) 163-168.
- [28] J. Chen, I.A. Kanj, W. Jia, Vertex cover: Further observations and further improvements, *J. Algorithms* 41 (2001) 280-301. A preliminary version appeared in: WG99, in: *Lecture Notes in Comput. Sci.*, Vol. 1665, Springer-Verlag, 1999, pp. 313-324.
- [29] M. R. Garey, D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, published by W. H. Freeman, New York, 1979.
- [30] Uriel Feige, Michael Langberg, Approximation Algorithms for Maximization Problems Arising in Graph Partitioning, *Journal of Algorithms* Volume 41, Issue 2, Pages 174-211 (November 2001)
- [31] Qiaoming Han, Yinyu Ye, Hantao Zhang and Jiawei Zhang. On Approximation of Max-Vertex-Cover, *European Journal of Operational Research* , Vol. 143, No. 2 (2002), 207-220.
- [32] R. G. Downey, M. R. Fellows, *Parameterized Complexity*, published by Springer, 1997.
- [33] T. Kikuno, N. Yoshida, Y. Kakuda, A linear time algorithm for the domination number of a series-parallel graph *Discrete Appl. Math.* 5 1983, 299-311.

- [34] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, Volume 41(1):153-180.
- [35] Kellogg Booth, J. Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11:191-199, 1982.
- [36] M. Hallett, G. Gonnet, and U. Stege, "Vertex Cover Revisited: A Hybrid Algorithm of Theory and Heuristic," manuscript 1998.
- [37] Judit Bar-Ilan and Guy Kortsarz and David Peleg, How to allocate network centers, *J. Algorithms*, Vol. 15, number 3, pages 385-415, 1993.
- [38] S. E. Dreyfus and RA Wagner. The Steiner problem in graphs. *Networks*, 1:195-207, 1971.
- [39] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal and interval graphs: Minimum fillin and physical mapping. In *Proceedings of the 35th Symposium on Foundations of Computer Science*, pages 780-791. IEEE Computer Science Press, Los Alamitos, California, 1994.
- [40] Jochen Alber , Henning Fernau , Rolf Niedermeier, Parameterized complexity: exponential speed-up for planar graph problems, *Journal of Algorithms*, v.52 n.1, p.26-56, July 2004.
- [41] Rajiv Gandhi and Samir Khuller and Aravind Srinivasan, Approximation algorithms for partial covering problems, *J. Algorithms*, Vol. 53, page 55-84, 2004.
- [42] Michael R. Fellows, Michael A. Langston, On search, decision, and the efficiency of polynomial-time algorithms

Source Journal of Computer and System Sciences archive
Volume 49 , Issue 3 Pages: 769-779, 1994

- [43] RG Downey and MR Fellows, "Parameterized Computational Feasibility," Proc. Second Cornell Workshop on Feasible Mathematics (Birkhauser,Boston), page 219-244, 1995.
- [44] Leizhen Cai, The complexity of finding fixed-size optimal solutions, manuscript, 2005.
- [45] J. Alber, H.L. Bodlaender, H. Fernau, R. Niedermeier, Fixed parameter algorithm for planar dominating set and related problems. SWAT 2000, LNCS 1851, pp. 97-110.

CUHK Libraries



004280653