ODD CYCLES, BIPARTITE SUBGRAPHS AND APPROXIMATE GRAPH COLORING

Susan S. Wang Yeh
(Thesis)

CS-TR-208-89

June 1989

Odd Cycles, Bipartite Subgraphs, and Approximate Graph Coloring

Susan S. Wang Yeh

A DISSERTATION
PRESENTED TO THE
FACULTY OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

JUNE 1989

Abstract

The *graph coloring problem* is to color the vertices of a graph so that no two adjacent vertices have the same color. This problem is not only NP-complete but also seems hard to approximate. In this thesis, we investigate the interplay of three different topics: odd cycles, bipartite subgraphs, and approximate graph coloring. Our primary goal is to design efficient approximate graph coloring algorithms with good performance. Our focus is on odd cycles and our central approach is to find bipartite subgraphs of graphs.

We examine the role played by odd cycles of graphs in connection with graph coloring. We show that the presence or absence of certain size odd cycles gives graphs more structure and hence simplifies graph coloring. More specifically, we show that the absence of small odd cycles enables us to find large bipartite subgraphs. On the other hand, the absence of large odd cycles in a biconnected graph implies the absence of large even cycles, and these conditions imply that the graph contains special structures. We present efficient polynomial-time graph coloring algorithms which improve the performance guarantee on certain graphs, sometimes by a large margin, over the existing approximate graph coloring algorithms. For example, we can color triangle-free graphs with $O(\sqrt{n})$ colors, and in most cases, optimally color graphs with only 3-cycles and 5-cycles in odd cycles.

We also present efficient polynomial-time algorithms for finding a maximum bipartite subgraph for special classes of graphs which include proper circular-arc, circular-arc, permutation, and split graphs. On numerous occasions, we use one technique, namely dynamic programming, for the development of the algorithms. Furthermore, we show that we can modify the algorithms in a slight way to find a maximum bipartite subgraph that includes a nontrivial subset of vertices.

Acknowledgements

My sincere thanks go to my advisor Andrea LaPaugh. She has provided guidance, advice, motivation, and support throughout my study at Princeton. I have benefited enormously from our weekly discussions, from her constructive criticism, and from her careful reading of this dissertation. I am grateful and privileged to have her as an advisor; one could not ask for a better one.

I thank my readers Joel Friedman and Bob Tarjan for reading this dissertation and for their valuable suggestions.

I thank Fan Chung, Paul Erdos, Laszlo Lovasz, Rachel Manber, Ken Supowit, and Mihalis Yannakakis for helpful discussions and for their willingness to share ideas.

I have many fond memories of my stay at Princeton, thanks to the people here, including the faculty, the students, the staff, and my friends. Special thanks go to Toshio Nakatani and Deborah Silver for their fellowship, and to Sharon Rogers for being so helpful.

I am forever grateful to my parents for their teaching and their unconditional support.

This research was supported in part by ARO fellowship grant DAAG29-83-G-0110, an IBM fellowship, and NSF grant MIP8619335.

My deepest thanks go to Michael Yeh for his patience and love.

To My Parents

Contents

Abstract i

Chapter 1: Introduction 1
1.1 General Introduction and Motivation 1
1.2 Organization of the Dissertation 4

Chapter 2: Background 6
2.1 Basic Definitions 6
2.2 Biconnected Graphs 8
2.3 Breadth-First Search 9
2.4 Finding a Smallest Odd Cycle 10

Chapter 3: Bipartite Neighborhood of a Smallest Odd Cycle 11
3.1 Introduction 11
3.2 Preliminaries 12
3.3 Properties 13
3.4 Association 16
3.5 1-Neighborhood 27
3.6 The Main Theorem 32
3.7 A Special Case 36

Chapter 4: Coloring Graphs with only "Large" Odd Cycles 41
4.1 Introduction 41
4.2 k-Cycle and its 1-Neighborhood 43
4.2.1 $k > \lceil n/2 \rceil$: 3 Colors 44
4.2.2 $k > \lceil n/3 \rceil$: 4 Colors 46
4.2.3 $k > \lceil n/4 \rceil$: 5 Colors 47
4.2.4 $k > \lceil n/i \rceil$: $6[i/3]^{1/2}$ Colors 49
4.3 k-Cycle and its Local Neighborhood 56
4.3.1 A 4-Colorable Neighborhood of the k-Cycle 56
4.3.2 A Strategy 57
4.3.3 An Algorithm that 4-Colors a Subgraph of G 60
4.3.4 An Approximate Graph Coloring Algorithm 62
4.4 Performance Analysis 65

Chapter 5: Coloring Graphs with only "Small" Odd Cycles 69
5.1 Introduction 69
5.2 Odd Cycles, Even Cycles, and the Chromatic Number 70
5.3 Odd Cycles = k-Cycles 73
5.3.1 $k = 3$ 73
5.3.2 $k = 5$ 75
5.3.3 $k > 5$ 83
5.4 Odd Cycles = 3-Cycles + 5-Cycles 85
5.5 Discussion 95

Chapter 6: Algorithms for Finding a Maximum Bipartite Subgraph for Special Classes of Graphs 96
6.1 Introduction 96
6.2 Preliminaries 99
6.3 Interval Graphs 100
6.4 Placing Intervals on Tracks 102
6.5 Proper Circular-Arc Graphs 106
6.6 Circular-Arc Graphs 111
6.7 Permutation Graphs 113
6.8 Split Graphs 117
6.9 Concluding Remarks and Suggestions for Future Research 119

Chapter 7: Conclusions and Future Work 121

References 124

Chapter 1

Introduction

1.1. General Introduction and Motivation

Graph coloring problems have intrigued many researchers in the past century, who have contributed in turn to the various deep and interesting results in graph theory. Graph coloring problems come in many flavors, and the favorite one, in the author's opinion, is *vertex coloring*: color the vertices of a graph G so that no two adjacent vertices have the same color. Vertex coloring is the problem we shall concentrate on in this dissertation. Henceforth when we speak of graph coloring, we shall mean vertex coloring.

The "field" of graph coloring or colorability arrived at its present status much to the credit of the celebrated Four Color Problem, posed by F. Guthrie around 1850 [May, 1965], which asks whether every planar graph is 4-colorable. The Four Color Problem has become the Four Color Theorem, after many unsuccessful attacks by graph theorists, and finally solved, with the aid of computers, by Appel and Haken[1976, 1977, 1986] and by Appel, Haken, and Koch[1977].

Besides map coloring, the problem of graph coloring arises in many applications, including scheduling ([Broder, 1964], [Brown, 1972], [Hall and Acton, 1967], [Neufield and Tartar, 1974], [Peck and Williams, 1966], [Welsh and Powell, 1967], [Wood, 1968], [Wood, 1969]), loading problems ([Eilon and Christofides, 1971]), resource allocation ([Christofides, 1975]), storage problems ([Bondy and Murty, 1976]), routing ([Tucker and Bodin, 1976]), estimation of sparse jacobian matrices ([Coleman and Moré, 1981], [Curtis, Powell, and Reid, 1974]), printed circuit testing ([Garey, Johnson, and So, 1976]), layer assignment ([Akers, 1972], [Ng and Johnsson, 1985]), and register allocation ([Chaitin, Auslander, Chandra, Cocke, Hopkins, and Markstein, 1981], [Chaitin, 1982]).

The graph coloring problem has also attracted interest in the computer science domain, not only because it arises in practice and yet it is a ''hard'' problem to solve exactly, even when the graph under consideration is restricted, but also because it seems to be a ''hard'' problem to approximate. Graph coloring, in its general form, in which one asks whether a graph G is k-colorable, has been shown to be NP-complete by Karp[1972]. It remains NP-complete in the special case when $k = 3$ [Garey, Johnson, and Stockmeyer, 1976], and furthermore, even if the graph is restricted to be 4-regular planar [Dailey, 1980]. Knowing the difficulty of obtaining an exact solution, one may be willing to settle for an approximate one. Graph coloring, however, has resisted heuristic attacks. Johnson[1974] has shown that many graph coloring heuristics perform poorly. More specifically, for each algorithm from a collection of prominent heuristics for coloring, Johnson showed that there exists a sequence of 3-colorable graphs G_m with $O(m)$ vertices such that $A(G_m) \geq m$, where $A(G)$ is the number of colors used by algorithm A on graph G. Garey and Johnson[1976] took a step further and proved that if P≠NP then there exists no polynomial-time approximation algorithm that is guaranteed to use less then two times the optimal number of colors. These are rather negative and discouraging results.

On the more positive side, Johnson[1974] was the first to present a polynomial-time algorithm that has a performance guarantee (the number of colors used by the algorithm over the chromatic number, in the worst case) of $O(n/\log n)$. The approach of Johnson's algorithm is: recursively find an approximate maximum independent set by repeatedly adding a vertex of smallest degree to the set and deleting the vertex and its neighbors from the graph. Wigderson[1983] improved the performance guarantee to $O(n(\log\log n)^2/(\log n)^2)$. Wigderson's algorithm is based on the following interesting observation: in a $(k+1)$-colorable graph, the neighbors of every vertex is k-colorable. His approach for graph coloring is: find a vertex of largest degree (instead of smallest, as in Johnson's algorithm) and color the neighbors of the vertex by a recursive coloring algorithm that uses a 2-coloring algorithm as its basis (since all bipartite graphs can be easily colored). More recently, Berger and Rompel[1988] further improved the performance guarantee to $O(n(\log\log n)^3/(\log n)^3)$ by combining the techniques of Johnson's and Wigderson's but focusing on a group of vertices rather than just a single vertex.

The existence of these polynomial-time algorithms that have a guaranteed performance is encouraging, yet what these algorithms can actually guarantee in performance is far from close to the chromatic number and certainly far from desirable. Narrowing the gap between the guaranteed performance of

polynomial-time algorithms and what is known as NP-hard: a performance guarantee less than 2, however, is a challenging and difficult task.

Motivated by the desire for more efficient approximation algorithms for graph coloring, the principal theme of this dissertation is in the design and development of algorithms for graph coloring with "good" performance. In our attempt to accomplish such a task, we shall examine more global properties and structures of graphs. Graph coloring is solvable in polynomial-time for bipartite graphs (k=2), but NP-complete for non-bipartite graphs ($k\geq 3$). The property that distinguishes non-bipartite from bipartite graphs is the presence of odd cycles. This result has stimulated our interest in the role played by odd cycles of a graph in connection with graph coloring, because it is the presence of odd cycles that makes the problem of graph coloring interesting. Hence we have made odd cycles the focus in our investigation for more global properties and structures of graphs. Our approach for graph coloring is: find subgraphs of a graph which we can color with a small number of colors. In particular, we are interested in finding bipartite subgraphs of a graph, and this is a subject we pursue not only in the design and development of graph coloring algorithms, but also as a problem in its own right for special classes of graphs.

Our approach for graph coloring is significantly different from previous ones in that we focus on odd cycles instead of a single vertex of a graph, and in doing so we are able to extract more global properties and structures of graphs. For example, focusing on a smallest odd cycle of a graph enables us to find bipartite subgraphs in the neighborhood of a smallest odd cycle. Focusing on odd cycles naturally leads us to consider two different classes of graphs: graphs with only "large" odd cycles, and graphs with only "small" odd cycles. For both classes of graphs, we have developed efficient approximate graph coloring algorithms with good performance.

Readers who are interested in general reading and background on graph coloring can find more information in Chapter 15 of Berge[1973], Chapter 5 of Bollobás[1978], Chapter 8 of Bondy and Murty[1976], Chapter 4 of Christofides[1975], and Chapter 12 of Harary[1969].

In this dissertation, we shall be dealing with deterministic algorithms and their worst-case behaviors. A different approach is to analyze the probable performance or the "average-case" behavior of graph coloring heuristics; see Dyer and Frieze[1986], Grimmett and McDiarmid[1975], Kučera[1977], McDiarmid[1979], Shamir and Upfal[1984], Turner[1984, 1988], and Wilf[1984]. A variety of heuristics for graph coloring have been developed, some of which are described in Matula, Marble, and

Isaacson[1972], and Manvel[1985].

For some special classes of graphs, such as series-parallel and perfect (which includes chordal and comparability), the graph coloring problem is polynomial-time solvable; see Johnson[1985] for more details and references. In addition, the k-coloring problem is polynomial-time solvable for dense graphs [Edwards, 1986].

1.2. Organization of the Dissertation

In this dissertation, as the title suggests, we investigate the interplay of three different topics: odd cycles, bipartite subgraphs, and approximate graph coloring. Our primary goal is to design efficient approximate graph coloring algorithms with good performance. Our focus is on odd cycles and our central approach is to find bipartite subgraphs of graphs. We begin the dissertation with a general introduction in this chapter – Chapter 1. Next we give some background material in Chapter 2.

In Chapter 3 we investigate the role played by smallest odd cycles of a graph in connection with graph coloring. We explore the local neighborhood of a smallest odd cycle of a graph and show that we can find bipartite subgraphs within that local neighborhood. We use the results of Chapter 3 in Chapter 4 to color graphs with only "large" odd cycles. We present two simple and efficient approximate graph coloring algorithms. Applying the first coloring algorithm, we show that we can $6(i/3)^{1/2}$-color a graph G when the size of a smallest odd cycle in G is greater than $\lceil n/i \rceil$. Using the second coloring algorithm, we show that we can $4\lg n$†-color a graph G if the size of a smallest odd cycle in G is at least $4(\lg n - \lg\lg n)$. Thus if the size of a smallest odd cycle in G is at least 5, i.e., if G is triangle-free, then we can $O(n^{1/2})$ color G. Hence for triangle-free graphs, the performance of the coloring algorithms presented in this thesis are better than that of all known approximate graph coloring algorithms. To provide evidence that graphs with only large odd cycles are not trivial nor easy to color, we review graph theoretical results that give tight bounds on their chromatic numbers. We then show the existence of such graphs by giving examples, such as Borsuk graphs, Kneser graphs, and Ramanujan graphs, all of which can be constructed.

---

† $\lg n = \log_2 n$

In Chapter 5 we consider coloring graphs with only ''small'' odd cycles. We show that graphs without large odd cycles possess many structures. Knowing the structures of graphs enables us to find bipartite subgraphs which aid graph coloring. We start by focusing on graphs with only 3-cycles in odd cycles, and give a complete characterization of their structures as well as an optimal coloring. Using breadth-first search and more structural analysis, we show that we can 3-color graphs with only 5-cycles in odd cycles. Then we present a simple algorithm that 4-colors graphs with only k-cycles in odd cycles. Lastly we consider graphs with only 3-cycles and 5-cycles in odd cycles, and develop an algorithm that in most cases uses an optimal number of colors. All the algorithms presented in this chapter are $O(e)$ in complexity.

In Chapter 6 we present efficient polynomial-time algorithms for finding a maximum bipartite subgraph for special classes of graphs which include proper circular-arc, circular-arc, permutation, and split graphs. On numerous occasions, we use one technique, namely dynamic programming, for the development of the algorithms. Furthermore, we show that we can modify the algorithms in a slight way to find a maximum bipartite subgraph that includes a nontrivial subset of vertices.

In Chapter 7 we close with a summary of contributions of this work, and give directions for extensions and suggestions for future research.

Chapter 2

Background

2.1. Basic Definitions

In this chapter we introduce some basic definitions, notation, and background material that we will use in this dissertation. More specific terminology and concepts will be presented later as needed at the appropriate times.

In this dissertation $G = (V, E)$ will denote a simple, connected, finite, and undirected graph with n vertices and e edges. A *subgraph* G' of G is a graph having all its vertices and edges in G. If $V' \subset V$ then $G-V'$ denotes the graph resulting from deleting vertices in V' and edges incident to them from G. Note that $G-V'$ is a subgraph of G.

A *path* of a graph G is an alternating sequence of vertices and edges $v_1 e_1 v_2 \cdots v_{n-1} e_n v_n$ such that all vertices and edges are distinct. Because G is a simple graph, we denote a path by its vertices only: $v_1 - v_2 - \cdots - v_n$, where $v_1 - v_2$ represents a path of length 1 from v_1 to v_2. Let $path_1$ be a path in G. Then we shall let $v_1 \overset{path_1}{\rightarrow} v_2$ denote the path from v_1 to v_2 on $path_1$, where v_1 and v_2 are two vertices on $path_1$.

A *cycle* of a graph G is an alternating sequence of vertices and edges $v_1 e_1 v_2 \cdots v_n e_n v_1$, starting and ending at the same vertex such that all the edges are distinct (the vertices on the cycle are not necessarily distinct). We also denote a cycle by its vertices only: $v_1 - v_2 - \cdots - v_n - v_1$. A *simple cycle* is a cycle where all the vertices are distinct.

The *length* of a path or a cycle is the number of edges on it. The *girth* of a graph is the length of a smallest cycle in G; the *odd girth* is the length of a smallest odd cycle in G; the *even girth* is the length of a smallest even cycle in G. The *circumference* of a graph is the length of a largest cycle in G; the *odd*

circumference is the length of a largest odd cycle in G; the *even circumference* is the length of a largest even cycle in G. The *diameter* of a graph is the length of a longest shortest path between two vertices in G.

A *complete graph* K_n is a graph on n vertices with an edge between every pair of vertices. A *clique* of a graph is a maximal complete subgraph.

A graph G is *p-colorable* if it can be legally (or properly) colored with p colors, that is, if there exists a function C defined over the vertices of G such that C assigns each vertex of G one of the values 1, 2, ... , p satisfying the condition that $C(u) \neq C(v)$ if $(u, v) \in E$. The *chromatic number* of a graph G, denoted by $\chi(G)$, is the minimum number p for which G is p-colorable. A p-coloring of G is the actual assignment of colors so that G is p colored.

A *bipartite graph* G is a graph whose vertex set V can be partitioned into two subset V_1 and V_2, $V_1 + V_2 = V$, such that every edge in E connects a vertex in V_1 with a vertex in V_2. It is easy to see that a bipartite graph is 2-colorable and all 2-colorable graphs are bipartite. König[1936] gave a characterization of bipartite graphs.

Theorem 2.1. [König, 1936] A graph is bipartite if and only if it has no odd cycles.

Unfortunately there exists no characterization of p-colorable graphs for $p \geq 3$. It seems to be a difficult open problem. Due to Theorem 2.1, a bipartite graph G can be recognized and colored in time $O(e)$ using breadth-first-search (to be discussed in Section 2.3).

Let G_i denote the graph obtained by iteratively deleting vertices of degree less than i from G. Hence the minimum degree of G_i is at least i.

Lemma 2.1. If we can i-color G_i, then we can i-color G.

Proof. We shall show how to i-color G once G_i is i-colored. We will add the deleted vertices back to G_i, one at a time, in the reverse order that those vertices were deleted from G. Each time we add a vertex v to G_i, the degree of v is less than i (due to the way G_i is created). Thus vertex v can be (legally) colored with one of the i colors. The above condition holds for all vertices added to G_i. Hence we can i-color G.

□

2.2. Biconnected Graphs

A vertex v is an *articulation point* of G if the removal of v disconnects G. Equivalently a vertex v is an articulation point if there exist two distinct vertices a and b, $a, b \neq v$, such that every path between a and b contains the vertex v. A graph G is *biconnected* if it is connected and has no articulation point. Equivalently a graph G is biconnected if every two vertices of G lie on a common simple cycle. Theorem 2.2 describes more equivalent characterizations of biconnected graphs.

Theorem 2.2. [Berge, 1973] The following properties are equivalent in a graph G of size at least 3:

1. G is biconnected.
2. Every two vertices of G lie on a common simple cycle.
3. Every vertex and edge of G lie on a common simple cycle.
4. Every two edges of G lie on a common simple cycle.

The following definition is adopted from Aho, Hopcroft, and Ullman[1974]. Let us define a natural relation on the set of edges of G by saying that two edges e_1 and e_2 are related if $e_1 = e_2$ or there is a cycle containing both e_1 and e_2. It is easy to show that this relation is an equivalence relation that partitions the edges of G into equivalence classes $E_1, E_2, \cdots, E_l$ such that two distinct edges are in the same class if and only if they lie on a common cycle. For $1 \leq i \leq l$, let V_i be the set of vertices of the edges in E_i. Each graph $G_i = (V_i, E_i)$ is called a *biconnected component* (or *block*) of G.

Theorem 2.3 lists some properties concerning the biconnected components of G.

Theorem 2.3. [Aho, Hopcroft, and Ullman, 1974] For $1 \leq i \leq l$, let $G_i = (V_i, E_i)$ be the biconnected components of a connected undirected graph $G = (V, E)$. Then

1. G_i is biconnected for each i, $1 \leq i \leq l$.
2. For all $i \neq j$, $V_i \bigcap V_j$ contains at most one vertex.
3. a is an articulation point of G if and only if $a \in V_i \bigcap V_j$ for some $i \neq j$.

Biconnected components of G can be found in time $O(e)$ (see Aho, Hopcroft, and Ullman[1974], or Tarjan[1972]). From the results of Theorem 2.3, it is easy to see that the chromatic number of G is the largest chromatic number of all the biconnected components of G, as also observed by Roschke and Furtado[1973]. Analogously, in approximate graph coloring, the problem of coloring G can be reduced to

smaller and hence simpler problems of coloring the biconnected components of G. The number of colors used to color G is the maximum of number of colors used to color any biconnected component of G. Henceforth we will assume that the graphs we are dealing with are all biconnected.

2.3. Breadth-First Search

Breadth-First Search is a procedure of visiting the vertices of a graph in a consistent manner. We begin the breadth-first search procedure at a vertex r called the *root* (often the root is chosen arbitrarily). First we search r, that is, visit all neighbors of r, then search all neighbors of r, and then search all of neighbors of neighbors of r, and so on. Each vertex in G is searched exactly once, and the breadth-first search procedure halts after all vertices have been searched. It is clear that this breadth-first search procedure takes time $O(e)$.

Let $BFS(G_r)$ denote the breadth-first search graph of G rooted at r, that is, $BFS(G_r)$ is a leveled graph isomorphic to G. Let $l_r(v)$ denote the level of vertex v in $BFS(G_r)$. The level of the root of $BFS(G_r)$ is 0, and the level of all other vertices is inductively defined as: if vertex v is first visited through the edge (u,v), then $l_r(v) = l_r(u)+1$. Vertex u is a *parent* of vertex v in $BFS(G_r)$ if $(u,v)\in E$ and $l_r(u) = l_r(v)-1$. If u is a parent of v, then v is a *child* of u. Vertex u is an *ancestor* of vertex v if either u is a parent of v, or u is an ancestor of a parent of v. Vertex w is a *common-ancestor* of vertices u and v if w is an ancestor of both u and v. Vertex w is a *nearest-common-ancestor* of vertices u and v, denoted $NCA(u,v)$, if w is a common-ancestor of u and v, and the distance between w and u $(l_r(u)-l_r(w))$ is no greater than that of any other common-ancestor of u and v. Note that u and v may have more than one nearest-common-ancestor, as long as their nearest-common-ancestors are all equal distance away from them.

$BFS(G_r)$ partitions the edges in E into three types: tree edges, lattice edges, and cross edges. An edge (v,w) is a tree edge if vertex w is first visited through edge (v,w). An edge (v,w) is a lattice edge if vertex w has been visited, and $l_r(w) = l_r(v)+1$. An edge (v,w) is a cross edge if $l_r(w) = l_r(v)$.

We now state some facts concerning the breadth-first search graph of G.

Fact 2.1. The shortest distance between the root r and any of its descendant v is $l_r(v)$.

Fact 2.2. The length of any cycle containing the root r and a vertex v is at least $2l_r(v)$.

Fact 2.3. If (u,v) is a cross edge of $BFS(G_r)$, then the length of any cycle containing the root r and the cross edge (u,v) is at least $2l_r(u)+1$.

Fact 2.4. For all $b \neq a-1, a+1$, vertices on level a of $BFS(G_r)$ are disjoint from those on level b of $BFS(G_r)$. Hence all the subgraphs induced by vertices on even (odd) levels of $BFS(G_r)$ are disjoint.

Fact 2.5. Every cross edge of $BFS(G_r)$ determines an odd cycle of G.

From Facts 2.4 and 2.5 it is clear that we can recognize and color a bipartite graph in time $O(e)$, since if a cross edge is present in a $BFS(G_r)$, for any vertex r, then G contains an odd cycle (Fact 2.5); otherwise each level of $BFS(G_r)$ is an independent set of vertices and can be colored with 1 color, hence $BFS(G_r)$ (or G) can be 2-colored (Fact 2.4).

2.4. Finding a Smallest Odd Cycle

Since one of our major goals in this dissertation is to investigate the roles played by odd cycles of a graph in connection with graph coloring, hence it is of interest to find odd cycles and do so efficiently. In particular, we will be focusing on a smallest odd cycle of a graph, and our approach requires us first to find one. Itai and Rodeh[1978] have studied the problem of finding a smallest odd cycle and reduced it within time $O(n^2)$ to that of finding a triangle (a 3-cycle) in an auxiliary graph. They then proposed three different methods for finding a triangle in a graph and obtained algorithms that take 1). $O(e^{3/2})$ time in the worst case, 2). $O(n^{5/3})$ time on the average, and 3). $O(n^{\alpha})$ time in the worst case, where α is the matrix exponent of a fast matrix multiplication algorithm. Presently the most efficient, asymptotically speaking, matrix multiplication algorithm runs in time $O(n^{2.376})$; see Coppersmith and Winograd[1987]. (To be more precise, an *nxn* matrix may be multiplied using $O(n^{2.376})$ arithmetical operations; 2.376.. is the *matrix exponent*. For a more practical matrix multiplication algorithm – $O(n^{2.807})$, see Strassen[1969]). When dealing with sparse graphs, we will use the bound of $O(\min\{e^{3/2}, n^{2.376}\})$ for finding a smallest odd cycle of a graph; otherwise we will use the bound of $O(n^{2.376})$.

Chapter 3

Bipartite Neighborhood of a Smallest Odd Cycle

3.1. Introduction

In this chapter we investigate the role played by smallest odd cycles of a graph in connection with graph coloring. We shall explore the local neighborhood of a smallest odd cycle of a graph G, and show that we can find bipartite subgraphs within that local neighborhood. To be more precise, let k be the size of a smallest odd cycle in G. Let *k-cycle* denote a smallest odd cycle of G. The *1-neighborhood* of a set of vertices S are all those vertices that are distance 1 away from S but are not in S. In general, the *t-neighborhood* of S are all those vertices that are distance t away (the shortest distance) from S but are not in S. The main result of this chapter is Theorem 3.1G, which essentially states that, for all $t<\frac{k-1}{2}$, the subgraph which consists of any $k-(2t+1)$ consecutive vertices on the k-cycle plus the union of their 1-neighborhood through t-neighborhood is bipartite. As Theorem 3.1G indicates, the coverage of the bipartite neighborhood of a smallest odd cycle (the k-cycle) depends on k – the size of a smallest odd cycle. The larger the value k is, the larger (more global) is the bipartite neighborhood of the k-cycle.

As a special case, we show that (Theorem 3.2), if $k>5$, then we can 3-color the subgraph which consists of vertices on the k-cycle and their 1-neighborhood, and this is the best we can do.

To give some motivation and insight into why the stated theorems are true, we offer the following lines of reasoning. First we are focusing on a *smallest* odd cycle (the k-cycle) of a graph. Second we are working with the *local neighborhood* of the k-cycle. The k-cycle and its local neighbors possess some special structures and properties, for example, if the size of a cycle is "small" (less than k) then that cycle must be even. Third when we want to show that a certain subgraph in the local neighborhood of the k-cycle is bipartite, often our strategy is to show that any cycle in the subgraph is composed of

smaller cycles usually consisting of some vertices on the k-cycle (because all vertices under consideration are local to the k-cycle), and each smaller cycle is even (because each cycle is "small"), which implies that the cycle under consideration is even (because even plus even is even).

The remaining portion of this chapter is devoted to proving Theorems 3.1G and 3.2. Section 3.2 contains some preliminaries. Sections 3.3, 3.4, and 3.5 set the foundation for the two theorems. Section 3.6 is the main theorem – 3.1G and its proof. Section 3.7 covers the special case – Theorem 3.2.

3.2. Preliminaries

In this study, the odd cycles in G have sizes $\geq k$, and the even cycles in G may have arbitrary sizes. We assume that the size of a smallest odd cycle in G is greater than 3, i.e., $k \neq 3$.

We will use the following notation throughout the chapter.

Given a graph $G=(V,E)$,

k – size of a smallest odd cycle in G, i.e., *odd girth* of G.

k-cycle – a smallest odd cycle in G, usually a specific one found by any smallest odd cycle algorithm.

$K = \{v \mid v \text{ is a vertex on the } k\text{-cycle}\}$.

$S_t = \{v \mid \text{shortest distance from the } k\text{-cycle to } v \text{ is } t\}$, for $t \geq 1$.

S_t is also known as the t-neighborhood of the k-cycle, for $t \geq 1$.

Let *cycle-a* be a simple cycle in G. We shall pick arbitrarily a direction to traverse cycle-a, and call that direction *clockwise*. In later arguments, clockwise traversal of cycle-a shall mean the same direction. The direction opposite of clockwise is called *counter-clockwise*. We shall define some path notation.

$v_1 \overset{c}{\rightarrow} v_2$, v_1 and $v_2 \in$ cycle-a in G, means the path (0 or more edges) from v_1 to v_2 within cycle-a traversed in clockwise direction.

$v_1 \overset{r}{\rightarrow} v_2$, v_1 and $v_2 \in$ cycle-a in G, means the path (0 or more edges) from v_1 to v_2 within cycle-a traversed in counter-clockwise, or the reverse, direction.

$v_1 \to v_2$, v_1 and $v_2 \in V$, means a path (0 or more edges) from v_1 to v_2.

$v_1 - v_2$, v_1 and $v_2 \in V$, means the path of length 1 from v_1 to v_2.

In what follows, we will assume that we have found a smallest odd cycle in G, and call it *the k-cycle*. We will rename the k vertices on the k-cycle $u_0, u_1, \ldots, u_{k-1}$ in clockwise consecutive order (an arbitrary direction of traversal is first picked to be clockwise), hence $K=\{u_0, u_1, \ldots, u_{k-1}\}$. When we make references to vertices in K, say u_i, we really mean $u_{i \,(mod\, k)}$.

In the following sections, we will number the statements, including Theorems, Lemmas, Corollaries, and Properties, in increasing order. In general, if a statement has a numerical numbering, for example – Theorem 3.1, then that statement applies to vertices on the k-cycle or vertices on the k-cycle plus their 1-neighborhood. If a statement has a numerical numbering followed by G, for example – Theorem 3.1G, then that statement applies to vertices on the k-cycle plus their t-neighborhood, for $t < \frac{k-1}{2}$ (thus Theorem 3.1G generalizes Theorem 3.1).

3.3. Properties

In this section we observe some special structures concerning the k-cycle and its local neighbors, and list them as properties.

Property 3.1. Any subgraph of G with size smaller than k is bipartite.

Proof. Since k is the size of a smallest odd cycle in G, then any subgraph of G of size smaller than k must not contain any odd cycles; hence it is bipartite.

□

Property 3.2. The k-cycle does not have any chords.

Proof. Any chord on the k-cycle divides it into two smaller cycles, one odd and one even. But the k-cycle is a smallest odd cycle in G; hence the k-cycle does not have any chords.

□

Property 3.3. If a vertex s in S_1 is adjacent to 2 vertices u_a and u_b in K, then the shorter distance within the k-cycle between u_a and u_b is 2.

Proof. Suppose that the shorter distance within the k-cycle between u_a and u_b is greater than 2. Note that the shorter distance can not be 1, otherwise we obtain a 3-cycle s–u_a–u_b–s. Then the distance in either direction within the k-cycle between u_a and u_b is less than k–2. One of the cycles, s–$u_a{}^c$→u_b–s, s–$u_a{}^r$→u_b–s, is even, and the other odd. But the length of either cycle is less than k; more specifically, the length of the odd cycle is less than k. Contradiction.

On the other hand, if the shorter distance within the k-cycle between u_a and u_b is 2, we don't find any violations (i.e., odd cycles of length less than k). Hence this is fine.

□

Property 3.3G. Let s be a vertex in S_t; u_a and u_b two vertices in K. If there exist two paths of length t from s to u_a and u_b, then the shorter distance within the k-cycle between u_a and u_b is less than or equal to $2t$.

Proof. Note that this property is nontrivial only if $k > 4t$. We shall prove Property 3.3G by induction on t.

Basis: t=1

This is Property 3.3.

Hypothesis: $t \leq m-1$

Assume that for all $t \leq m-1$, Property 3.3G holds.

Induction: $t=m$

We shall consider two cases.

I. The two paths of length t from s to u_a and u_b are disjoint.

Without loss of generality, let $s^{path\,1}$→u_a and $u_b{}^{path\,2}$→s denote the two disjoint paths of length m from s to u_a and from u_b to s, respectively. One of the cycles, $s^{path\,1}$→$u_a{}^c$→$u_b{}^{path\,2}$→s, $s^{path\,1}$→$u_a{}^r$→$u_b{}^{path\,2}$→s, is odd, and the other even. Without loss of generality, let the first cycle be odd. Hence the size of the first cycle is greater than or equal to k, which means that the length of the path $u_a{}^c$→u_b is greater than or equal to $k-2m$. This implies that the length of the path $u_a{}^r$→u_b is less than or equal to $2m$.

II. The two paths of length t from s to u_a and u_b intersect.

Without loss of generality, let $s^{path1}\rightarrow u_a$ and $s^{path2}\rightarrow u_b$ denote the two paths of length m, and let vertex v be the first place path1 intersects path2 when path1 is traversed from s to u_a. Vertex v belongs in S_i, for some $i<m$. The length of both paths – $v^{path1}\rightarrow u_a$ and $v^{path2}\rightarrow u_b$ must be i, otherwise s does not belong in S_t. By the induction hypothesis, the shorter distance within the k-cycle between u_a and u_b is less than or equal to $2i$ which is less than $2m$.

□

Property 3.4. If a vertex s is in S_1, then s is adjacent to at most 2 vertices in K.

Proof. Suppose that s is adjacent to at least 3 vertices, say u_a, u_b, and u_c, in K. According to Property 3.3, the shorter distances within the k-cycle between u_a and u_b, u_b and u_c, u_c and u_a are all 2. This implies that the k-cycle contains 6 vertices. But the k-cycle is odd. Contradiction.

□

Property 3.5G. Let $t<\frac{k-1}{2}$. Let s_1 and s_2 be two vertices in S_t. If s_1 is adjacent to s_2, then any two paths of length t from s_1 and s_2 to vertices in K must be disjoint.

Proof. Without loss of generality, let *path*1 and *path*2 denote two paths of length t from s_1 to u_a and from s_2 to u_b, respectively, for some $u_a,u_b\in$K. Suppose that path1 and path2 intersect at some vertex v. Now let's consider the cycle ${s_1}^{path1}\rightarrow v^{path2}\rightarrow s_2-s_1$, where ${s_1}^{path1}\rightarrow v$ is the part of path1 that goes from s_1 to v (similar interpretation for $v^{path2}\rightarrow s_2$). The size of the cycle is no more than $2t+1$. Since $k>2t+1$, the cycle must be even. This implies that the parity as well as the length of the path ${s_1}^{path1}\rightarrow v$ are different from those of ${s_2}^{path2}\rightarrow v$. Without loss of generality, let $|{s_1}^{path1}\rightarrow v| > |{s_2}^{path2}\rightarrow v|$. This means that the length of the path ${s_2}^{path2}\rightarrow v^{path1}\rightarrow u_a$ is less than t. But s_2 is in S_t. Contradiction.

□

Property 3.6. Let s_1 and s_2 be two distinct vertices in S_1; u_a and u_b two distinct vertices in K; s_1 adjacent to u_a and s_2 adjacent to u_b. If s_1 is adjacent to s_2, then

a) for $k\geq 5$ the shorter distance within the k-cycle between u_a and u_b is less than or equal to 3,

b) for $k>5$ the shorter distance within the k-cycle between u_a and u_b is 1 or 3.

Proof. Suppose that the shorter distance within the k-cycle between u_a and u_b is more than 3. The distance in either direction within the k-cycle between u_a and u_b must be less than $k-3$. One of the cycles $s_1-u_a{}^c\rightarrow u_b-s_2-s_1$, $s_1-u_a{}^r\rightarrow u_b-s_2-s_1$, is even, and the other odd. But the length of either cycle is less than $k-1$; more specifically, the length of the odd cycle is less than $k-1$. Contradiction.

If $k>5$ then the shorter distance within the k-cycle between u_a and u_b can not be 2, otherwise one of the following cycles: $s_1-u_a{}^c\rightarrow u_b-s_2-s_1$, $s_1-u_a{}^r\rightarrow u_b-s_2-s_1$, is a 5-cycle.

□

Property 3.6G. Let s_1 and s_2 be two vertices in S_t; u_a and u_b two vertices in K. If s_1 is adjacent to s_2 and there exist two paths of length t from s_1 to u_a and from s_2 to u_b, then the shorter distance within the k-cycle between u_a and u_b is less than or equal to $2t+1$.

Proof. Similar to that of 3.3G.

□

Property 3.7. Let $s_1-v_2-\cdots-v_a-s_{a+1}$ be a path of length a; s_1 and s_{a+1} two vertices in S_1; u_c and u_d two distinct vertices in K; s_1 adjacent to u_c and s_{a+1} adjacent to u_d (see Figure 3.1). If the distance, either clockwise or counter-clockwise, within the k-cycle between u_c and u_d is b and $a+b$ is even, then $a \geq b-2$.

Proof. Without loss of generality, let's assume that $c<d$ and the length of the path $u_c{}^c\rightarrow u_d$ is b. Let us consider the cycle (which may not be simple) $s_1-v_2-\cdots-v_a-s_{a+1}-u_d-u_{d+1}-\cdots-u_{c-1}-u_c-s_1$. Since k is odd and $a+b$ is even, the cycle is odd and its length is $a+(k-b+2)$. Since k is the size of a smallest odd cycle, thus $a \geq b-2$.

Note that if $a = 0$, then this is Property 3.3 with $b = 2$.

□

3.4. Association

To relate the vertices in K and their local neighbors in S_t, we introduce the notion of *association.* We show in Lemmas 3.1, 3.1G, and 3.2G that cycles formed by "associated" vertices are even cycles.

Definition 1. A subset R, $R=\{s_1,s_2,...,s_p\}$, of S_1 is *1-associated* with a subset S_0, $S_0=\{u_c,u_{c+1},...,u_d\}$, of K (see Figure 3.2) if the following three conditions are satisfied:

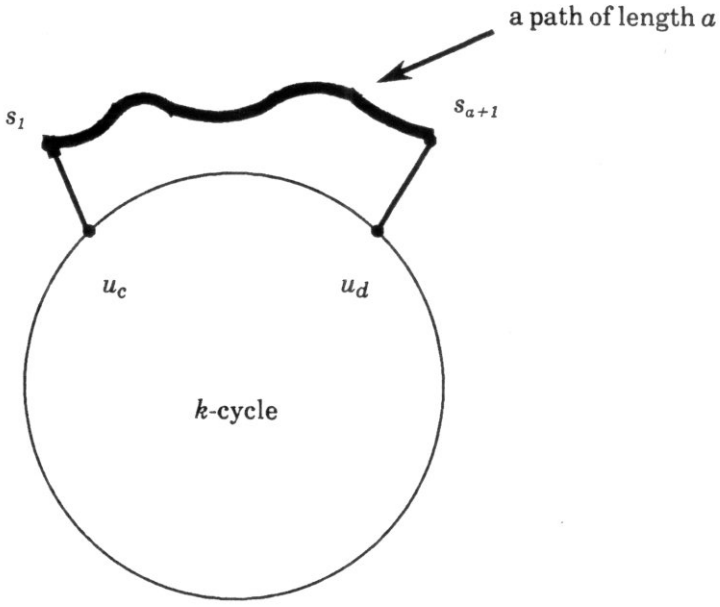

Figure 3.1. A configuration of vertices in Property 3.7.

i) there exists a simple path s_1–s_2– · · · –s_p covering all vertices in R,

ii) s_1 and s_p are adjacent to u_c and u_d, respectively,

Note: If $R=\{s_1\}$ then s_1 is adjacent to u_c and u_d; if $S_0=\{u_c\}$ then s_1 and s_p are adjacent to u_c.

iii) there exist at least 3 consecutive vertices u^1,u^2, and u^3 that are in K but not in S_0 such that every vertex s_i in R is adjacent to at least one vertex in $K-\{u^1,u^2,u^3\}$.

Definition 1G. A subset R, $R=\{s_1,s_2,...,s_p\}$, of S_t is *t-associated* with a subset S_0, $S_0=\{u_c,u_{c+1},...,u_d\}$, of K if the following three conditions are satisfied:

i) there exists a simple path s_1–s_2– · · · –s_p covering all vertices in S_t,

ii) there exist two paths (not necessarily disjoint) of length t from s_1 and s_p to u_c and u_d, respectively,

Note: If $R=\{s_1\}$ then there exist two paths of length t from s_1 to u_c and u_d; if $S_0=\{u_c\}$ then there exist two paths of length t from s_1 and s_p to u_c; if $R=\{s_1\}$ and $S_0=\{u_c\}$ then there exist two distinct

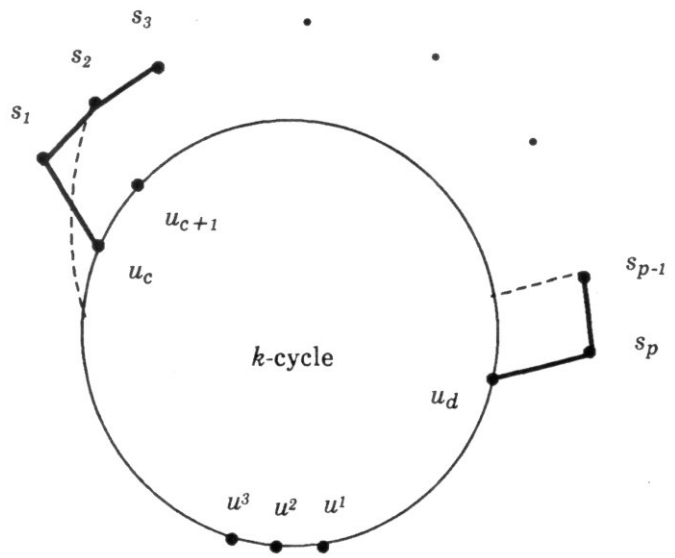

Figure 3.2. An illustration of 1-association.

paths of length t from s_1 to u_c.

iii) there exist at least $2t+1$ consecutive vertices $u^1,u^2,\ldots,u^{2t+1}$ that are in K but not in S_0 such that for every vertex s_i in R there exits a path of length t from s_i to at least one vertex in $K-\{u^1,u^2,...,u^{2t+1}\}$.

Note that Definition 1G makes sense only if $t<\frac{k-1}{2}$. Thus when we speak of t-association, we will assume that $t<\frac{k-1}{2}$.

The following Properties – 3.8 and 3.8G, although not used later, give some insight into the meaning of 1-association and t-association. Property 3.8 shows that if R is 1-associated with S_0, that is, if every vertex s_i in R is adjacent to at least one vertex in $K-\{u^1,u^2,u^3\}$, for some 3 consecutive vertices u^1,u^2, and u^3, then R is not 1-associated with S_0', for some S_0' that includes u^1,u^2, and u^3 (because otherwise some vertex in R would be adjacent to u^1,u^2,u^3 exclusively). For example, if R is 1-associated with S_0, $S_0=\{u_c,u_{c+1},...,u_d\}$, then R is not 1-associated with S_0', $S_0'=\{u_c,u_{c-1},\cdots,u_d\}$. Note that the

parity of the size of S_0 differs from that of S_0'. This property is significant in that (see Lemmas 3.1, 3.1G, and 3.2G) cycles formed by "associated" vertices are even cycles. If R is 1-associated with both S_0 and S_0', as in the previous example, then Lemma 3.1 says that both cycles formed by associated vertices in R and S_0, and in R and S_0' are even. But the parity of the size of S_0 differs from that of S_0'. So one of the cycles must be odd, hence we have reached a contradiction. Property 3.8G generalizes Property 3.8.

Property 3.8. If a subset R, $R=\{s_1,s_2,...,s_p\}$, of S_1 is 1-associated with $S_0=\{u_c,u_{c+1},...,u_d\}$, $S_0 \subset K$, then for any 3 consecutive vertices u_i,u_{i+1},u_{i+2} in S_0, at least one vertex in R is adjacent to vertices in $\{u_i,u_{i+1},u_{i+2}\}$ exclusively among vertices in K.

Proof. Since R is 1-associated with S_0, there exist at least 3 consecutive vertices u_r,u_{r+1},u_{r+2}, between u_d and u_c in clockwise direction such that none of the vertices in R is adjacent to them exclusively among those in K. Thus every vertex in R is adjacent to a vertex in $K-\{u_r,u_{r+1},u_{r+2}\}$. Now suppose that Property 3.8 is false, that is, there exist 3 consecutive vertices in S_0 such that none of the vertices in R is adjacent to them exclusively among those in K. Let u_b be the first vertex starting from u_c going clockwise for which the condition holds. Hence every vertex in R is adjacent to a vertex in $K-\{u_b,u_{b+1},u_{b+2}\}$. Because of the existence of u_r,u_{r+1},u_{r+2}, and of u_b,u_{b+1},u_{b+2}, all of which distinct, K contains at least 6 vertices, i.e., $k > 5$. Since u_b,u_{b+1},u_{b+2} are in S_0, then u_d is either u_{b+2} or it comes after u_{b+2} but before u_r in clockwise direction on the k-cycle.

If $u_b \neq u_c$, then u_{b-1} must be adjacent to some vertex, say s_a, in R (if not, then u_{b-1} would have been a vertex before u_b that satisfies the mentioned condition). Figure 3.3 illustrates a possible configuration of vertices. If $u_b=u_c$, then s_1 is adjacent to u_b. s_1 is also adjacent to a vertex in $K-\{u_b,u_{b+1},u_{b+2}\}$, thus s_1 is adjacent to u_{b-2} by Property 3.3. Note that if $R = \{s_1\}$, then s_1 is adjacent to u_c and $u_d=u_{c+2}$ by Property 3.3. Hence it is not possible that s_1 is also adjacent to $u_{b-2}=u_{c-2}$ because of Property 3.4. So we have reached a contradiction and Property 3.8 holds when $R = \{s_1\}$. For the discussion which follows, we will assume that $|R| > 1$. Without loss of generality, we shall let $s_a = s_1$ when $u_b = u_c$.

Let us now consider vertices in K that are adjacent to s_{a+1}. What we want to show is that if a vertex in K is adjacent to s_{a+1}, then that vertex does not belong in $\{u_{b+2},u_{b+3},\cdots,u_d\}$. Furthermore, we want to show that if a vertex in K is adjacent to s_i, $a \leq i \leq p$, then that vertex does not belong in $\{u_{b+2},u_{b+3},\cdots,u_d\}$. But since s_p is adjacent to u_d, we will then arrive at a contradiction.

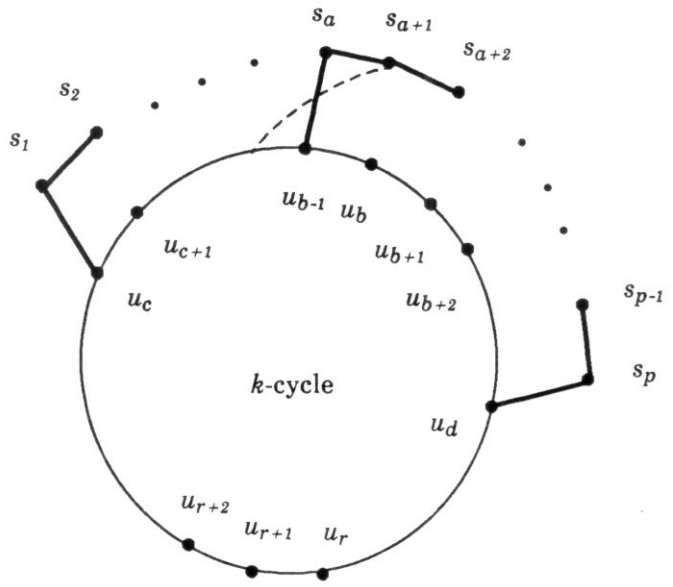

Figure 3.3. A configuration in the proof of Property 3.7.

We shall accomplish the above task by a two step process. The first part of the process involves an induction. To set up the induction, we begin with some definitions.

Let s_i, s_{i+1} be vertices in R, s_i adjacent to s_{i+1}. Let V_1, V_2 be sets of vertices defined as follows:

$V_1 = \{v \mid v \text{ is in } R \text{ and adjacent to a vertex in } \{u_{r+3}, u_{r+4}, \cdots, u_{b+1}\}\}$,

$V_2 = \{v \mid v \text{ is in } R \text{ and adjacent to a vertex in } \{u_{r+1}, u_{r+2}, \cdots, u_{b-1}\}\}$.

The first step of the process is to show that:

A. If s_i belongs in V_1 and V_2, then s_{i+1} belongs in V_1 and V_2.

Proof. Suppose that s_{i+1} does not belong in both V_1 and V_2. There are 3 possibilities.

A.1. s_{i+1} belongs in V_1 but not to V_2.

A.2. s_{i+1} belongs in V_2 but not to V_1.

A.3. s_{i+1} does not belong in either V_1 or V_2.

Case A.1. Since s_{i+1} belongs in V_1 but not V_2, then s_{i+1} must be adjacent to either u_b or u_{b+1}.

Suppose that s_{i+1} is adjacent to u_b. Because s_{i+1} is adjacent to a vertex in $K-\{u_b,u_{b+1},u_{b+2}\}$ and Properties 3.3 and 3.4 (s_{i+1} is adjacent to at most 2 vertices in K and these two vertices must be distance 2 apart), then s_{i+1} must be adjacent u_{b-2}. Note that $u_{b-2}\in\{u_{r+1},u_{r+2},\cdots,u_{b-1}\}$. But s_{i+1} does not belong in V_2. Contradiction.

Suppose that s_{i+1} is adjacent to u_{b+1}. Because s_{i+1} is adjacent to a vertex in $K-\{u_b,u_{b+1},u_{b+2}\}$ and s_{i+1} is not in V_2, then s_{i+1} must be adjacent to u_{b+3} by Properties 3.3 and 3.4. Note that $u_{b+3}\notin\{u_{r+1},u_{r+2},\cdots,u_{b-1}\}$. Because s_i is in V_1, s_{i+1} is adjacent to u_{b+3}, and Property 3.6, s_i must be adjacent to u_b. s_i is also in V_2, hence s_i is adjacent to u_{b-2} by Property 3.3. But this violate Property 3.6 – between $s_i,s_{i+1},u_{b-2},u_{b+3}$ in clockwise direction as well as counterclockwise direction (recall that k is odd and greater than 5). Contradiction. Therefore Case A.1 does not hold.

Case A.2. Cases A.1 and A.2 are symmetrical. Thus Case A.2 does not hold either.

Case A.3. Since s_{i+1} does not belong to either V_1 or V_2, then s_{i+1} must be adjacent to a vertex in $\{u_{b+2},u_{b+3},\cdots,u_r\}$. On the one hand, because s_{i+1} is adjacent to a vertex in $K-\{u_b,u_{b+1},u_{b+2}\}$, s_{i+1} must be adjacent some vertex u_q in $\{u_{b+3},u_{b+4},\cdots,u_r\}$. According to Property 3.6, the possible candidates for the neighbors of s_i in K are $u_{q-3},u_{q-1},u_{q+1},u_{q+3}$. Since s_i is in V_2, s_i must be adjacent to u_{q+1} or u_{q+3} or both. On the other hand, because s_{i+1} is adjacent to a vertex in $K-\{u_r,u_{r+1},u_{r+2}\}$, s_{i+1} must be adjacent some vertex u_l in $\{u_{b+2},u_{b+3},\cdots,u_{r-1}\}$. According to Property 3.6, the possible candidates for the neighbors of s_i in K are $u_{l-3},u_{l-1},u_{l+1},u_{l+3}$. Since s_i is in V_1, s_i must be adjacent to u_{l-1} or u_{l-3} or both. To summarize, both of the following conditions must hold:

1. s_i is adjacent to u_{q+1} or u_{q+3} or both,
2. s_i is adjacent to u_{l-1} or u_{l-3} or both.

Because of the above two conditions, u_q must not be the same as u_l. Hence s_{i+1} is adjacent to two vertices: u_q and u_l. Because of Property 3.3, either $u_q=u_{l+2}$ or $u_q=u_{l-2}$. If $u_q=u_{l+2}$, then Properties 3.3, 3.4, and 3.6 say that s_i is adjacent to either u_{l-1}, or u_{l-1} and u_{l+1}, or u_{l+1}, or u_{l+1} and u_{l+3}, or u_{l+3}. But none of the above possibilities satisfies the two conditions described earlier. Contradiction. If $u_q=u_{l-2}$, then s_i is adjacent to only one vertex $u_{q+1}=u_{l-1}$ in K. Note that $u_{l-1}\in$

$\{u_{b+1},u_{b+2}, \cdots ,u_{r-2}\}$. But s_i is in V_2. Contradiction.

s_a belongs in both V_1 and V_2, even when $s_a = s_1$ ($u_b = u_c$). Therefore, by the induction, we have shown that for all i, $1 \le i \le p$, s_i belongs in both V_1 and V_2.

The second step of the process is to show that:

B. If s_i belongs in V_1 and V_2 and if a vertex v in K is adjacent to s_i, then v does not belong in $\{u_{b+2},u_{b+3}, \cdots ,u_d\}$.

Proof. We shall consider two possibilities.

If s_i does not have a neighbor in $\{u_{r+3},u_{r+4}, \cdots ,u_{b-1}\}$, then the possible neighbors of s_i are: $U_1 = \{u_{r+1},u_{r+2}\}$, $U_2 = \{u_b,u_{b+1}\}$. Since s_i is in both V_1 and V_2, then s_i must be adjacent to two of the possible neighbors listed above, one from each set. The only combinations that satisfy Property 3.3 are:

1. If $u_b = u_{r+3}$ and s_i is adjacent to u_{r+1} and u_b,
2. If $u_b = u_{r+3}$ and s_i is adjacent to u_{r+2} and u_{b+1},
3. If $u_b = u_{r+4}$ and s_i is adjacent to u_{r+2} and u_b.

In all of the above combinations, s_i does not have a neighbor in $\{u_{b+2},u_{b+3}, \cdots ,u_d\}$ by Property 3.4.

If s_i has a neighbor in $\{u_{r+3},u_{r+4}, \cdots ,u_{b-1}\}$, then according to Property 3.3, s_i does not have a neighbor in $\{u_{b+2},u_{b+3}, \cdots ,u_d\}$.

Combining the results of A and B, we conclude that for all i, $1 \le i \le p$, if a vertex v in K is adjacent to s_i, then v does not belong in $\{u_{b+2},u_{b+3}, \cdots ,u_d\}$. But s_p is adjacent to u_d. Contradiction.

□

Property 3.8G. If a subset R, $R=\{s_1,s_2,...,s_p\}$, of S_t is t-associated with $S_0=\{u_c,u_{c+1},...,u_d\}$, $S_0 \subset K$, then for any $2t+1$ consecutive vertices $u_i,u_{i+1},...,u_{i+2t}$ in S_0, there exists at least one vertex in R that has paths of length t to only vertices in $\{u_i,u_{i+1}, \ldots ,u_{i+2t}\}$ among vertices in K.

Proof. This proof is similar to that of Property 3.8.

□

Note that we could have presented Properties 3.8 and 3.8G as corollaries of Lemmas 3.1 and 3.1G.

Next we show in Lemmas 3.1, 3.1G, and 3.2G that cycles formed by "associated" vertices are even cycles.

Lemma 3.1. If a subset R, $R=\{s_1,s_2,...,s_p\}$, of S_1 is 1-associated with a subset S_0, $S_0=\{u_c,u_{c+1},...,u_d\}$, of K, and $|R|+|S_0|>3$, then the cycle $s_1-s_2-\cdots-s_p-u_d-u_{d-1}-\cdots-u_c-s_1$ is an even cycle.

Proof. The basic idea behind the proof is that the cycle under consideration is composed of even cycles, and since the sum of even cycles is even, we then have the desired result.

Since R is 1-associated with S_0, every s_i in R is adjacent to at least one vertex in $K-\{u^1, u^2, u^3\}$, for some 3 consecutive vertices u^1, u^2, and u^3. For each $s_i\in R$, we pick arbitrarily a vertex $u_i'\in$ K$-\{u^1, u^2, u^3\}$ to which s_i is adjacent, and use it consistently in later arguments. Exception: we pick u_c for s_1 and u_d for s_p so $u_1'=u_c$ and $u_p'=u_d$. Now there exist 2 "picked" vertices u_a and u_b such that the path $u_a{}^r\rightarrow u_b$ includes u^1, u^2, and u^3, and no vertex on that path besides u_a and u_b is a "picked" vertex. Let S_{01} denote the set of vertices on the path $u_a{}^c\rightarrow u_b$; $S_{01}=\{u_a,u_{a+1},...,u_b\}$. Note that $S_0\subseteq S_{01}\subset K$.

We now break the path $s_1-s_2-\cdots-s_p$ into p segments of length 1. We shall discuss the case $R=\{s_1\}$ later. For each segment s_i-s_{i+1}, the shorter distance within the k-cycle between u_i' and u_{i+1}' is less than or equal to 3 by Property 3.6, which implies that the shorter path from u_i' to u_{i+1}' must use vertices in S_{01}. Now let's consider *cycle-i* which is $s_i-u_i'^{S_{01}}\rightarrow u_{i+1}'-s_{i+1}-s_i$ (see Figure 3.4), where $u_i'^{S_{01}}\rightarrow u_{i+1}'$ is the path from u_i' to u_{i+1}' on the k-cycle that uses only vertices in S_{01}. The size of the cycle is less than k because the length of the path $u_i'^{S_{01}}\rightarrow u_{i+1}'$ is less than $k-3$ (since the set $K-S_{01}+u_i'+u_{i+1}'$ contains at least 5 vertices). Hence cycle-i is even.

If we add (take the exclusive-or of) cycle-i and cycle-i+1 (cycle-i ⊗ cycle-i+1), we get a simple even cycle $s_i-u_i'^{S_{01}}\rightarrow u_{i+2}'-s_{i+2}-s_{i+1}-s_i$ (see Figure 3.4), where $u_i'^{S_{01}}\rightarrow u_{i+2}'$ is the path from u_i' to u_{i+2}' on the k-cycle that uses only vertices in S_{01}. If we add cycles 1 through p–1, i.e., cycle-1⊗cycle-2⊗ ... ⊗cycle-p–1, we get the cycle $s_1-s_2-\cdots-s_p-u_d-u_{d-1}-\cdots-u_c-s_1$ that must be even, because all cycles 1 through p–1 are even. Note that if $S_0=\{u_c\}$, then $s_1-s_2-\cdots-s_p$ is an even path, since $|R|+|S_0|>3$.

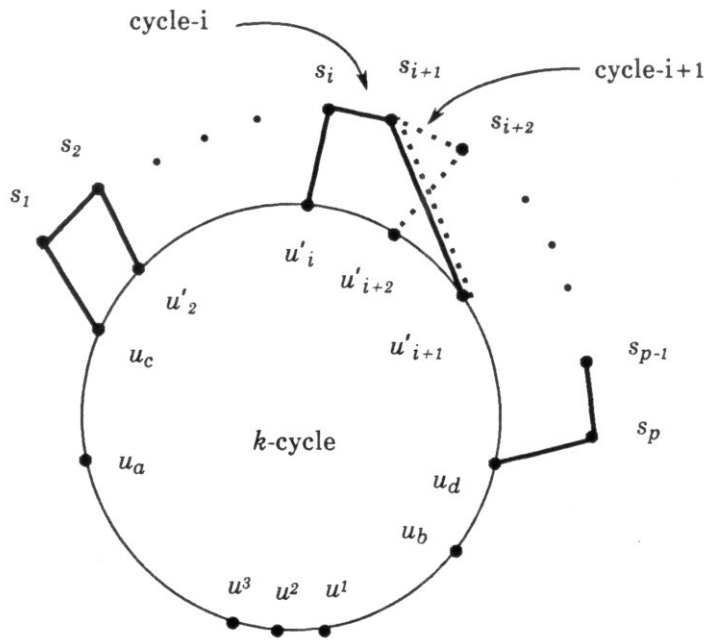

Figure 3.4. An illustration of cycle-i and cycle-i+1.

If $R = \{s_1\}$, then $S_0 = \{u_c, u_{c+1}, u_{c+2}\}$ by Property 3.3. The cycle $s_1-u_c-u_{c+1}-u_{c+2}-s_1$ is even.

□

Corollary 3.1. If a subset R, $R=\{s_1,s_2,...,s_p\}$, of S_1 is 1-associated with a subset S_0, $S_0=\{u_c,u_{c+1},...,u_d\}$, of K, then parity of the path $s_1-s_2-\cdots-s_p$ in R equals parity of the path $u_c-u_{c+1}-\cdots-u_d$ in S_0.

Lemma 3.1G. For all $t<\frac{k-1}{2}$, if a subset R, $R=\{s_1,s_2,...,s_p\}$, of S_t is t-associated with a subset S_0, $S_0=\{u_c,u_{c+1},...,u_d\}$, of K, and path1 and path2 are two paths of length t from s_1 to u_c and from s_p to u_d, respectively, then the cycle $s_1-s_2-\cdots-s_p{}^{path2}\rightarrow u_d-u_{d-1}-\cdots-u_c{}^{path1}\rightarrow s_1$ is an even cycle.

Proof. This proof is similar to that of Lemma 3.1. We show that the cycle under consideration is composed of cycles of even lengths, and since the sum of even cycles is even, we have the desired result. Note that path1 and path2 may intersect, hence the cycle under consideration may not be a simple cycle.

Since R is t-associated with S_0, then for every s_i in R there exists a path of length t from s_i to at least one vertex in $K-\{u^1,u^2,\ldots,u^{2t+1}\}$, for some $2t+1$ consecutive vertices $u^1,u^2,\ldots,u^{2t+1}$. For each $s_i \in R$, we pick arbitrarily a path of length t, call it *path-i*, from s_i to u_i', for some $u_i' \in$ K$-\{u^1,u^2,\ldots,u^{2t+1}\}$, and use that path consistently in later arguments. Exception: we pick path1 for path-1 and path2 for path-p. Now there exist 2 vertices u_a and u_b in K that are on some "picked" paths such that the path $u_a{}^r \rightarrow u_b$ includes $u^1,u^2,\ldots,u^{2t+1}$, and no vertex on that path besides u_a and u_b is on a "picked" path. Let S_{01} denote the set of vertices on the path $u_a{}^c \rightarrow u_b$; $S_{01} = \{u_a,u_{a+1},\ldots,u_b\}$. Note that $S_0 \subseteq S_{01} \subset K$.

We now break the path $s_1-s_2-\cdots-s_p$ into p segments of length 1. We shall discuss the case $R = \{s_1\}$ later. The shorter distance within the k-cycle between u_i' and u_{i+1}' is less than or equal to $2t+1$ by Property 3.6G, which implies that the shorter path from u_i' to u_{i+1}' must use vertices in S_{01}. Now let's consider *cycle-i* which is $s_i{}^{path-i} \rightarrow u_i'{}^{S_{01}} \rightarrow u_{i+1}'{}^{path-i+1} \rightarrow s_{i+1}-s_i$, where $u_i'{}^{S_{01}} \rightarrow u_{i+1}'$ is the path from u_i' to u_{i+1}' on the k-cycle that uses only vertices in S_{01}. The size of cycle-i is less than k because the length of the path $u_i'{}^{S_{01}} \rightarrow u_{i+1}'$ is less than $k-(2t+1)$ (since the set $K-S_{01}+u_i'+u_{i+1}'$ contains at least $2t+3$ vertices). Hence cycle-i is even.

If we *add* cycle-i and cycle-i+1 (here adding cycle-i and cycle-i+1 means taking the exclusive-or of paths on the k-cycle and retaining all other edges on both cycles except those on path-i+1) we get an even cycle (which may not be simple) $s_i{}^{path-i} \rightarrow u_i'{}^{S_{01}} \rightarrow u_{i+2}'{}^{path-i+2} \rightarrow s_{i+2}-s_{i+1}-s_i$, where $u_i'{}^{S_{01}} \rightarrow u_{i+2}'$ is the path from u_i' to u_{i+2}' on the k-cycle that uses only vertices in S_{01}. If we add cycles 1 through p–1, we get the cycle (may not be simple) $s_1-s_2-\cdots-s_p{}^{path2} \rightarrow u_d-u_{d-1}-\cdots-u_c{}^{path1} \rightarrow s_1$ which must be even, because all cycles 1 through p–1 are even. Note that if $S_0 = \{u_c\}$, then $s_1-s_2-\cdots-s_p$ is an even path.

Now we shall consider the case where $R = \{s_1\}$. If $u_c = u_d$, then $s_1{}^{path1} \rightarrow u_c{}^{path2} \rightarrow s_1$ is an even cycle of length $2t$. If $u_c \neq u_d$, then by Property 3.3G the shorter distance within the k-cycle between u_c and u_d is less than or equal to $2t$, which means that the shorter path from u_c to u_d must use vertices in S_{01}. Hence by a similar reasoning as above, we see that the cycle $s_1{}^{path2} \rightarrow u_d{}^{S_{01}} \rightarrow u_c{}^{path1} \rightarrow s_1 = s_1{}^{path2} \rightarrow u_d-u_{d-1}-\cdots-u_c{}^{path1} \rightarrow s_1$ is even.

□

Corollary 3.1G. For all $t<\frac{k-1}{2}$, if a subset R, $R=\{s_1,s_2,...,s_p\}$, of S_t is t-associated with a subset S_0, $S_0=\{u_c,u_{c+1},...,u_d\}$, of K, then parity of the path $s_1-s_2-\cdots-s_p$ in R equals parity of the path $u_c-u_{c+1}-\cdots-u_d$ in S_0.

Lemma 3.2G. Each simple cycle of the even cycle $s_1-s_2-\cdots-s_p{}^{path\,2}\rightarrow u_d-u_{d-1}-\cdots-u_c{}^{path\,1}\rightarrow s_1$ in Lemma 3.1G is an even cycle.

Proof. Without loss of generality, let us assume that the even cycle of Lemma 3.1G is not simple. We will use the same notation as that in Lemma 3.1G.

We shall prove Lemma 3.2G by induction on t.

Basis: $t=1$

This is Lemma 3.1.

Hypothesis: $t\leq m-1$

Assume that for all $t\leq m-1$, Lemma 3.2G holds.

Induction: $t=m$

Since the even cycle of Lemma 3.1G is not simple, then path1 and path2 intersect. Let vertex v be the first place path1 intersects path2 when path1 is traversed from s_1 to u_c. Now the even cycle $s_1-s_2-\cdots-s_p{}^{path\,2}\rightarrow u_d-u_{d-1}-\cdots-u_c{}^{path\,1}\rightarrow s_1$ is composed of a simple cycle – *cycle1* which is $s_1-s_2-\cdots-s_p{}^{path\,2}\rightarrow v^{path\,1}\rightarrow s_1$ and and another cycle – *cycle2* which is $v^{path\,1}\rightarrow u_c-u_{c+1}-\cdots-u_d{}^{path\,2}\rightarrow v$. Note that vertex $v\in S_l$, for some $l<m$. The length of path $v^{path\,1}\rightarrow u_c$ must be the same as the length of path $v^{path\,2}\rightarrow u_d$, namely l (otherwise either s_1 or s_p does not belong in S_m). Thus $\{v\}$, a subset of S_l, is l-associated with $S_0=\{u_c,u_{c+1},\cdots,u_d\}$. Therefore cycle2 is an even cycle by Lemma 3.1G; moreover, each simple cycle of cycle2 is even by the induction hypothesis. This implies that cycle1 is a simple even cycle.

□

3.5. 1-Neighborhood

In this section we focus on the 1-neighborhood of the k-cycle. First we prove a special case of Theorem 3.1G, Theorem 3.1, which states that, for all $k > 3$, the subgraph which consists of any $k-3$ consecutive vertices on the k-cycle plus their 1-neighborhood is bipartite. Theorem 3.1 actually sets the foundation for Theorem 3.1G.

Let $N_1(u_a,u_b, \cdots ,u_l)$, $\{u_a,u_b,...,u_l\} \subseteq K$, denote vertices in S_1 that are adjacent to only vertices in $\{u_a,u_b, \cdots ,u_l\}$ among all vertices in K. Note that $N_1(u_a,u_b, \cdots ,u_l)$ are all those vertices in S_1 that are *not* adjacent to any of the vertices in $K-\{u_a,u_b, \cdots ,u_l\}$.

Theorem 3.1. For any 3 consecutive vertices, u_i, u_{i+1}, u_{i+2}, in K, the subgraph induced by vertices $K-\{u_i,u_{i+1},u_{i+2}\}+S_1-N_1(u_i,u_{i+1},u_{i+2})$ is bipartite.

Proof. Without loss of generality, let's reindex the k vertices in K in clockwise consecutive order starting with i=0; hence u_i becomes u_0, u_{i+1} becomes u_1, and so on. Let $W_1=K-\{u_0,u_1,u_2\}+S_1-N_1(u_0,u_1,u_2)$. Let *cycle-c* be a cycle in W_1 (here W_1 means the subgraph induced by vertices in W_1). Note that every vertex in $S_1-N_1(u_0,u_1,u_2)$ is adjacent to at least one vertex in $K-\{u_0,u_1,u_2\}$. We now consider different structures of cycle-c.

I Cycle-c consists of vertices in S_1 only; cycle-c = $s_1-s_2-\cdots-s_p-s_1$.

Let us identify 2 vertices s_a and s_b on cycle-c. Without loss of generality, let us assume that a < b. Let $R_1=\{s_a,s_{a+1},...,s_b\}$, path-1=$s_a-s_{a+1}-\cdots-s_b$, $R_2=\{s_a,s_{a-1},...,s_1,s_p,s_{p-1},...,s_b\}$, and path-2=$s_a-s_{a-1}-\cdots-s_1-s_p-s_{p-1}-\cdots-s_b$. Pick a vertex in $K-\{u_0,u_1,u_2\}$ to which s_a is adjacent, call it u_c; pick a vertex in $K-\{u_0,u_1,u_2\}$ to which s_b is adjacent, call it u_d. If c < d, let S_0 = $\{u_c,u_{c+1},...,u_d\}$, else let S_0 = $\{u_c,u_{c-1},...,u_d\}$. It is clear that u_0, u_1, and u_2 are not in S_0, since c < d in the first case and c ≥ d in the second case. Every vertex in $R_1(R_2)$ is adjacent to at least one vertex in $K-\{u_0,u_1,u_2\}$. Hence both R_1 and R_2 are 1-associated with S_0. By Corollary 3.1 and for c < d (c ≥ d), parity of path-1 equals parity of path $u_c-u_{c+1}-\cdots-u_d$ ($u_c-u_{c-1}-\cdots-u_d$) equals parity of path-2. Since cycle-c is composed of path-1 and path-2, it is therefore an even cycle.

II Cycle-c consists of alternating sequences of vertices in K and S_1.

If cycle-c consists of exactly one vertex in K, that is, cycle-c = $u_a-s_1-s_2-\cdots-s_p-u_a$, then $R=\{s_1,s_2, \cdots ,s_p\}$ is 1-associated with $S_0=\{u_a\}$ and cycle-c is an even cycle by Lemma 3.1.

If cycle-c consists of more than one vertex in K, then let us identify 2 vertices u_a and u_b in K and on cycle-c such that $a<b\leq k-1$, and if u_i is in K and on cycle-c then $u_i\in\{u_a,u_{a+1},\cdots,u_b\}=S_0$. Clearly $u_0,u_1,u_2\notin S_0$. Now cycle-c is composed of 2 paths, path-1 and path-2, each starting at vertex u_a then going through some alternating sequences of vertices in K and S_1 and ending at vertex u_b. The 2 paths are vertex and edge disjoint, except at vertices u_a and u_b. Let us analyze path-1; the same results apply to path-2.

Basically we want to show that parity of path-1 is the same as parity of path $u_a-u_{a+1}-\cdots-u_b$. We do so by showing whenever path-1 leaves K, say at u_{li}, (i.e., path-1 goes from u_{li} in K to some vertex in S_1), parity of path-1 from u_a to u_{li} is the same as parity of path $u_a-u_{a+1}-\cdots-u_{li}$. Similarly, whenever path-1 enters K, say at u_{ei}, (i.e., path-1 goes from some vertex in S_1 to u_{ei} in K), parity of path-1 from u_a to u_{ei} is the same as parity of path $u_a-u_{a+1}-\cdots-u_{ei}$.

Let u_{li} and s_{li} denote the vertices in K and S_1, respectively, such that when path-1 leaves K for the ith time, it goes from u_{li} to s_{li}. Let u_{ei} and s_{ei} denote the vertices in K and S_1, respectively, such that when path-1 enters K for the ith time, it goes from s_{ei} to u_{ei}. Let R_i denote the set of vertices on path-1 from s_{li} to s_{ei}. We shall do induction on i.

Basis: $i=1$

When path-1 leaves K for the first time at u_{l1} (see Figure 3.5), parity of path-1 from u_a to u_{l1} is the same as parity of path $u_a-u_{a+1}-\cdots-u_{l1}$, because the 2 paths are the same. Note that u_{l1} may be u_a.

When path-1 enters K (after leaving it) for the first time at u_{e1} (see Figure 3.5), we have $a\leq l\,1\leq e\,1\leq b$, since path-1 is a simple path. Let S_{01} denote the set of vertices on the path from u_{l1} to u_{e1} in clockwise direction; $S_{01}=\{u_{l1},u_{l1+1},\cdots,u_{e1}\}\subseteq S_0$. Then R_1 is 1-associated with S_{01}, since $u_0,u_1,u_2\notin S_{01}$ and every vertex in R_1 is adjacent to at least one vertex in $K-\{u_0,u_1,u_2\}$. By Corollary 3.1, parity of path-1 from s_{l1} to s_{e1} is the same as parity of path $u_{l1}-u_{l1+1}-\cdots-u_{e1}$. Therefore parity of path-1 from u_a to u_{e1} = parity of path $u_a-u_{a+1}-\cdots-u_{l1}$ + parity of path $u_{l1}-s_{l1}$ + parity of path-1 from s_{l1} to s_{e1} + parity of path $s_{e1}-u_{e1}$ = parity of path $u_a-u_{a+1}-\cdots-u_{l1}$ + parity of path $u_{l1}-u_{l1+1}-\cdots-u_{e1}$ = parity of

path $u_a - u_{a+1} - \cdots - u_{e1}$.

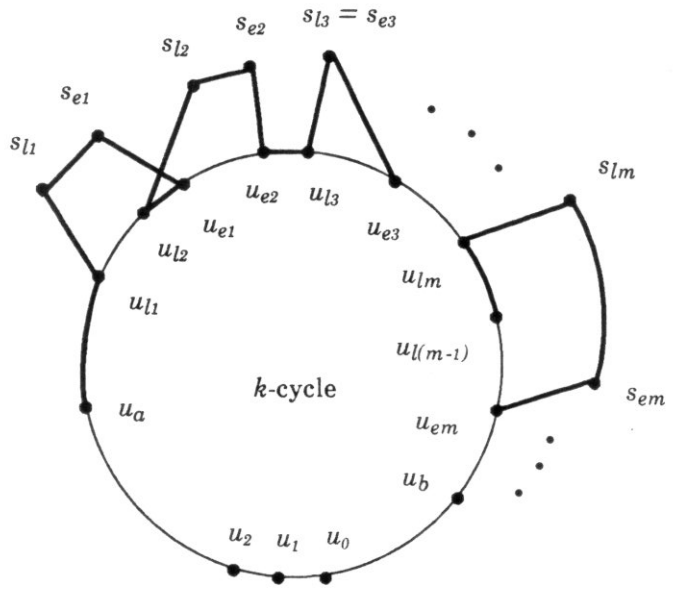

Figure 3.5. A possible configuration of path-1.

Hypothesis: $i=m-1$.

Assume that parity of path-1 from u_a to $u_{l(m-1)}$ is the same as parity of path $u_a - u_{a+1} - \cdots - u_{l(m-1)}$, and parity of path-1 from u_a to $u_{e(m-1)}$ is the same as parity of path $u_a - u_{a+1} - \cdots - u_{e(m-1)}$.

Induction: $i=m$.

When path-1 leaves K for the mth time at u_{lm} (see Figure 3.5), parity of path-1 from u_a to u_{lm} = parity of path-1 from u_a to $u_{e(m-1)}$ + parity of path-1 from $u_{e(m-1)}$ to u_{lm} = parity of path $u_a - u_{a+1} - \cdots - u_{e(m-1)}$ (by induction hypothesis) + parity of path from $u_{e(m-1)}$ to u_{lm} within S_0 = parity of path $u_a - u_{a+1} - \cdots - u_{lm}$.

When path-1 enters K for the mth time at u_{em}, either $lm<em\leq b$ or $em<lm<b$. In the first case (see Figure 3.5), let S_{0m} denote the set of vertices on the path from u_{lm} to u_{em} in clockwise direction; $S_{0m}=\{u_{lm}, u_{lm+1}, \cdots, u_{em}\} \subset S_0$. Then R_m is 1-associated with S_{0m}, since

$u_0,u_1,u_2 \notin S_{0m}$ and every vertex in R_m is adjacent to at least one vertex in $K-\{u_0,u_1,u_2\}$. By Corollary 3.1, parity of path-1 from s_{lm} to s_{em} is the same as parity of path $u_{lm}-u_{lm+1}-\cdots-u_{em}$. Therefore parity of path-1 from u_a to u_{em} = parity of path-1 from u_a to u_{lm} + parity of path $u_{lm}-s_{lm}$ + parity of path-1 from s_{lm} to s_{em} + parity of path $s_{em}-u_{em}$ = parity of path $u_a-u_{a+1}-\cdots-u_{lm}$ + parity of path $u_{lm}-u_{lm+1}-\cdots-u_{em}$ = parity of path $u_a-u_{a+1}-\cdots-u_{em}$.

In the second case, let S_{0m} denote the set of vertices on the path from u_{lm} to u_{em} in counter-clockwise direction; $S_{0m}=\{u_{lm},u_{lm-1},\cdots,u_{em}\}\subset S_0$. Then R_m is 1-associated with S_{0m}, since $u_0,u_1,u_2 \notin S_{0m}$ and every vertex in R_m is adjacent to at least one vertex in $K-\{u_0,u_1,u_2\}$. By Corollary 3.1, parity of path-1 from s_{lm} to s_{em} is the same as parity of path $u_{lm}-u_{lm-1}-\cdots-u_{em}$. Therefore parity of path-1 from u_a to u_{em} = parity of path-1 from u_a to u_{lm} + parity of path $u_{lm}-s_{lm}$ + parity of path-1 from s_{lm} to s_{em} + parity of path $s_{em}-u_{em}$ = parity of path $u_a-u_{a+1}-\cdots-u_{lm}$ + parity of path $u_{lm}-u_{lm-1}-\cdots-u_{em}$ = parity of path $u_a-u_{a+1}-\cdots-u_{em}$.

Hence we have shown that parity of path-1 is the same as parity of path $u_a-u_{a+1}-\cdots-u_b$. The same results apply to path-2, that is, parity of path-2 is the same as parity of path $u_a-u_{a+1}-\cdots-u_b$. Since cycle-c is composed of path-1 and path-2, and the parity of the 2 paths are the same, cycle-c is therefore an even cycle.

Now we conclude that the subgraph induced by vertices $W_1=K-\{u_0,u_1,u_2\}+S_1-N_1(u_0,u_1,u_2)$ is bipartite, because the subgraph does not contain any odd cycles.

□

Corollaries 3.2-3.5 are results that follow from Theorem 3.1 and are used in Lemmas 3.3 and 3.4 of Section 3.7.

Corollary 3.2. Any cycle with vertices in K and S_1 only which does not contain any of the 3 consecutive vertices u_i,u_{i+1},u_{i+2} in K, for some i, and their exclusive neighbors $N_1(u_i,u_{i+1},u_{i+2})$ in S_1, is an even cycle.

Corollary 3.3. Any path from u_a to u_b, denoted by $u_a \rightarrow u_b$, $u_a, u_b \in K$, consisting of alternating sequences of vertices in K and S_1 only and satisfying the following 2 conditions:

for some i,

a) path $u_a \rightarrow u_b$ does not contain any of the 3 consecutive vertices u_i, u_{i+1}, u_{i+2} in K and their exclusive neighbors $N_1(u_i, u_{i+1}, u_{i+2})$ in S_1,

b) $u_i, u_{i+1}, u_{i+2} \notin \{u_a, u_{a+1}, \cdots, u_b\}$ $(u_i, u_{i+1}, u_{i+2} \notin \{u_a, u_{a-1}, \cdots, u_b\})$,

has the same parity as path $u_a - u_{a+1} - \cdots - u_b$ $(u_a - u_{a-1} - \cdots - u_b)$.

Proof. Let us consider the cycle (which may not be simple) $u_a \rightarrow u_b - u_{b-1} - \cdots - u_a$ $(u_a \rightarrow u_b - u_{b+1} - \cdots - u_a)$, and let's call it *cycle-c*. If cycle-c is not a simple cycle, then it consists of simple cycles plus overlapping paths (i.e., 2 of the exact same paths). Each simple cycle of cycle-c is even by Corollary 3.2. Thus cycle-c is an even cycle. This implies that the parity of path $u_a \rightarrow u_b$ is the same as that of path $u_a - u_{a+1} - \cdots - u_b$ $(u_a - u_{a-1} - \cdots - u_b)$.

□

Corollary 3.4. Any path from u_a to s_b, denoted by $u_a \rightarrow s_b$, $u_a \in K$, $s_b \in S_1$, consisting of alternating sequences of vertices in K and S_1 only and satisfying the following 2 conditions:

for some i,

a) path $u_a \rightarrow s_b$ does not contain any of the 3 consecutive vertices u_i, u_{i+1}, u_{i+2} in K and their exclusive neighbors $N_1(u_i, u_{i+1}, u_{i+2})$ in S_1,

b) $u_i, u_{i+1}, u_{i+2} \notin \{u_a, u_{a+1}, \cdots, u_c\}$ $(u_i, u_{i+1}, u_{i+2} \notin \{u_a, u_{a-1}, \cdots, u_c\})$, where u_c is a vertex in K adjacent to s_b,

has different parity than path $u_a - u_{a+1} - \cdots - u_c$ $(u_a - u_{a-1} - \cdots - u_c)$.

Proof. Suppose that u_c is not on path $u_a \rightarrow s_b$. Path $u_a \rightarrow s_b - u_c$ satisfies conditions in Corollary 3.3, hence parity of path $u_a \rightarrow s_b - u_c$ = parity of path $u_a - u_{a+1} - \cdots - u_c$ $(u_a - u_{a-1} - \cdots - u_c)$. Therefore parity of path $u_a \rightarrow s_b \neq$ parity of path $u_a - u_{a+1} - \cdots - u_c$ $(u_a - u_{a-1} - \cdots - u_c)$.

Suppose that u_c is on path $u_a \rightarrow s_b$. If $u_a = u_c$, then the cycle $u_a \rightarrow s_b - u_c = u_a$ is even by Corollary 3.2. Therefore parity of path $u_a \rightarrow s_b$ is odd. The parity of a path consisting of a single vertex is even. If $u_a \neq u_c$, then $u_a \rightarrow s_b = u_a \rightarrow u_c \rightarrow s_b$. Now consider path $u_a \rightarrow s_b - u_c = u_a \rightarrow u_c \rightarrow s_b - u_c$. Cycle $u_c \rightarrow s_b - u_c$ is

even by Corollary 3.2. Path $u_a \rightarrow u_c$ satisfies conditions in Corollary 3.3, hence parity of path $u_a \rightarrow u_c$ = parity of path $u_a - u_{a+1} - \cdots - u_c$ ($u_a - u_{a-1} - \cdots - u_c$). Parity of path $u_a \rightarrow u_c \rightarrow s_b - u_c$ = parity of path $u_a - u_{a+1} - \cdots - u_c$ ($u_a - u_{a-1} - \cdots - u_c$). Therefore parity of path $u_a \rightarrow s_b \neq$ parity of path $u_a - u_{a+1} - \cdots - u_c$ ($u_a - u_{a-1} - \cdots - u_c$).

□

Corollary 3.5. Any path from s_a to s_b, denoted by $s_a \rightarrow s_b$, $s_a, s_b \in S_1$, consisting of alternating sequences of vertices in K and S_1 only and satisfying the following 2 conditions:

for some i,

a) path $s_a \rightarrow s_b$ does not contain any of the 3 consecutive vertices u_i, u_{i+1}, u_{i+2} in K and their exclusive neighbors $N_1(u_i, u_{i+1}, u_{i+2})$ in S_1,

b) $u_i, u_{i+1}, u_{i+2} \notin \{u_c, u_{c+1}, \cdots, u_d\}$ ($u_i, u_{i+1}, u_{i+2} \notin \{u_c, u_{c-1}, \cdots, u_d\}$), where u_c and u_d are vertices in K adjacent to s_a and s_b, respectively,

has the same parity as path $u_c - u_{c+1} - \cdots - u_d$ ($u_c - u_{c-1} - \cdots - u_d$).

Proof. Suppose that u_c is not on path $s_a \rightarrow s_b$. Path $u_c - s_a \rightarrow s_b$ satisfies conditions in Corollary 3.4, hence parity of path $u_c - s_a \rightarrow s_b \neq$ parity of path $u_c - u_{c+1} - \cdots - u_d$ ($u_c - u_{c-1} - \cdots - u_d$). Therefore parity of path $s_a \rightarrow s_b$ = parity of path $u_c - u_{c+1} - \cdots - u_d$ ($u_c - u_{c-1} - \cdots - u_d$).

Suppose that u_c is on path $s_a \rightarrow s_b$. Path $s_a \rightarrow s_b$ = path $s_a \rightarrow u_c \rightarrow s_b$. Now consider path $u_c - s_a \rightarrow u_c \rightarrow s_b$. Cycle $u_c - s_a \rightarrow u_c$ is even by Corollary 3.2. Note that u_d may or may not be on that cycle. Path $u_c \rightarrow s_b$ satisfies conditions in Corollary 3.4, hence parity of path $u_c \rightarrow s_b \neq$ parity of path $u_c - u_{c+1} - \cdots - u_d$ ($u_c - u_{c-1} - \cdots - u_d$). Note that u_d may or may not be on that path. Parity of path $u_c - s_a \rightarrow u_c \rightarrow s_b \neq$ parity of path $u_c - u_{c+1} - \cdots - u_d$ ($u_c - u_{c-1} - \cdots - u_d$). Therefore parity of path $s_a \rightarrow s_b$ = parity of path $u_c - u_{c+1} - \cdots - u_d$ ($u_c - u_{c-1} - \cdots - u_d$).

□

3.6. The Main Theorem

We now focus on a larger neighborhood of the k-cycle – $S_1, S_2, \cdots, S_t$, where $2t+1<k$. We prove the major result of this chapter which says that any subgraph which consists of any $k-(2t+1)$ consecutive vertices on the k-cycle plus their neighbors from the 1-neighborhood up to the t-neighborhood is bipartite.

Let $N_t(u_a,u_b,\cdots,u_l)$, $\{u_a,u_b,...,u_l\}\subseteq K$, denote vertices in S_t that have paths of length t to only vertices in $\{u_a,u_b,\cdots,u_l\}$ among all vertices in K. Note that $N_t(u_a,u_b,\cdots,u_l)$ are all those vertices in S_t that do *not* have paths of length t to any of the vertices in $K-\{u_a,u_b,\cdots,u_l\}$.

Theorem 3.1G. For any $2t+1$ consecutive vertices, u_i, $u_{i+1},...,u_{i+2t}$, in K, the subgraph induced by vertices $K-\{u_i,u_{i+1},\ldots,u_{i+2t}\}+S_1-N_1(u_i,u_{i+1},\ldots,u_{i+2t})+\ ...\ +S_t-N_t(u_i,u_{i+1},\cdots,u_{i+2t})$ is bipartite.

Proof. Without loss of generality, let's reindex the k vertices in K in clockwise consecutive order starting with i=0; hence u_i becomes u_0, u_{i+1} becomes u_1, and so on. Let $W_t=K-\{u_0,u_1,\ldots,u_{2t}\}+S_1-N_1(u_0,u_1,\ldots,u_{2t})+\ ...\ +S_t-N_t(u_0,u_1,\cdots,u_{2t})$. We shall prove Theorem 3.1G by induction on t.

Basis: $t=1$

This is Theorem 3.1.

Hypothesis: $t\leq m-1$

Assume that for all $t\leq m-1$, Theorem 3.1G holds.

Induction: $t=m$

Let *cycle-c* be a cycle in W_m that contains at least one vertex in $S_m-N_m(u_0,u_1,\cdots,u_{2m})$ (here W_m means the subgraph induced by vertices in W_m). Note that for each vertex s_i in $S_m-N_m(u_0,u_1,\cdots,u_{2m})$, there exists a path of length m from s_i to at least one vertex in $K-\{u_0,u_1,\ldots,u_{2m}\}$. We now consider different structures of cycle-c.

I Cycle-c consists of vertices in S_m only; cycle-c = $s_1-s_2-\cdots-s_p-s_1$.

Let us identify 2 vertices s_a and s_b on cycle-c. Without loss of generality, let us assume that a < b. Let $R_1=\{s_a,s_{a+1},...,s_b\}$, path-1=$s_a-s_{a+1}-\cdots-s_b$, $R_2=\{s_a,s_{a-1},...,s_1,s_p,s_{p-1},...,s_b\}$, and path-2=$s_a-s_{a-1}-\cdots-s_1-s_p-s_{p-1}-\cdots-s_b$. Since s_a (s_b) is in $S_m-N_m(u_0,u_1,\cdots,u_{2m})$, there exists a path of length m from s_a (s_b) to a vertex, say u_c (u_d), in $K-\{u_0,u_1,\ldots,u_{2m}\}$. If $c<d$, let S_0 = $\{u_c,u_{c+1},...,u_d\}$, else let S_0 = $\{u_c,u_{c-1},...,u_d\}$. It is clear that $u_0,u_1,\cdots,u_{2m}$ are not in S_0, since $c < d$ in the first case and $c \geq d$ in the second case. For every vertex s_i in $R_1(R_2)$, there exists a path of length m from s_i to at least one vertex in $K-\{u_0,u_1,\ldots,u_{2m}\}$. Hence both R_1 and R_2 are m-associated with S_0. By Corollary 3.1G and for $c < d$ ($c \geq d$), parity of path-1 equals parity

of path u_c–u_{c+1}–$\cdots$–u_d (u_c–u_{c-1}–$\cdots$–u_d) equals parity of path-2. Since cycle-c is composed of path-1 and path-2, it is therefore an even cycle.

II Cycle-c consists of alternating sequences of vertices in S_m and in W_{m-1}.

Basically we want to replace each path of cycle-c in S_m with a path of the same parity in W_{m-1}, and show that the resulting cycle-c (which may not be simple) is an even cycle.

Let us identify one vertex v that is in S_{m-1} and on cycle-c. We shall arbitrarily pick a direction of traversal of cycle-c to be clockwise, and traverse cycle-c from v in clockwise direction. During the traversal, cycle-c enters S_m (i.e., cycle-c goes from some vertex in S_{m-1} to some vertex in S_m) and leaves S_m (i.e., cycle-c goes from some vertex in S_m to some vertex in S_{m-1}), say, a total of p times. Let $s_{(m-1)ei}$ and s_{mei} denote the vertices in S_{m-1} and S_m, respectively, such that when cycle-c traversed in clock-wise direction from v enters S_m for the ith time, it goes from $s_{(m-1)ei}$ to s_{mei}. Let $s_{(m-1)li}$ and s_{mli} denote the vertices in S_{m-1} and S_m, respectively, such that when cycle-c traversed in clock-wise direction from v leaves S_m for the ith time, it goes from s_{mli} to $s_{(m-1)li}$. Let $s_{mei} \overset{cycle-c}{\rightarrow} s_{mli}$ denote the path on cycle-c traversed in clock-wise direction from s_{mei} to s_{mli}. Note that path $s_{mei} \overset{cycle-c}{\rightarrow} s_{mli}$ consists of vertices in S_m only. Let R_i denote the vertices on path $s_{mei} \overset{cycle-c}{\rightarrow} s_{mli}$.

From each vertex $s_{(m-1)ei}$ ($s_{(m-1)li}$), there exists a path of length $m-1$ to at least one vertex in $K-\{u_0,u_1,\cdots,u_{2m}\}$. For each $s_{(m-1)ei}$ ($s_{(m-1)li}$), we pick arbitrarily a path of length $m-1$, call it *path-ei* (*path-li*), from $s_{(m-1)ei}$ ($s_{(m-1)li}$) to u_{ei} (u_{li}), for some u_{ei} (u_{li}) $\in$ $K-\{u_0,u_1,\cdots,u_{2m}\}$, and use that path consistently in later arguments. Now there exist 2 vertices u_a and u_b in K that are on some "picked" paths such that the path $u_a \overset{r}{\rightarrow} u_b$ includes $u_0,u_1,\cdots,u_{2m}$, and no vertex on that path besides u_a and u_b is on a "picked" path. Let S_{00} denote the set of vertices on the path $u_a \overset{c}{\rightarrow} u_b$; $S_{00} = \{u_a,u_{a+1},\cdots,u_b\}$. Let $u_{ei} \overset{S_{00}}{\rightarrow} u_{li}$ denote the path from u_{ei} to u_{li} on the k-cycle that uses only vertices in S_{00}. Let S_{0i} denote the vertices on the path $u_{ei} \overset{S_{00}}{\rightarrow} u_{li}$.

Now we see that R_i is m-associated with S_{0i}, and by Lemma 3.1G $s_{mei} \overset{cycle-c}{\rightarrow} s_{mli}$–$s_{(m-1)li} \overset{path-li}{\rightarrow} u_{li} \overset{S_{00}}{\rightarrow} u_{ei} \overset{path-ei}{\rightarrow} s_{(m-1)ei}$–$s_{mei}$ (see Figure 3.6) is an even cycle, for all $1 \le i \le p$. Hence the parity of the path $s_{(m-1)ei}$–$s_{mei} \overset{cycle-c}{\rightarrow} s_{mli}$–$s_{(m-1)li}$ is the same

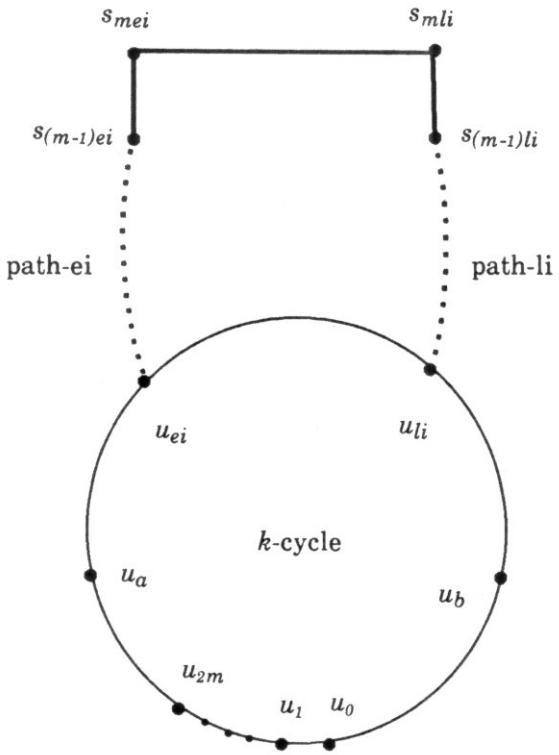

Figure 3.6. A segment of cycle-c and its local neighborhood.

as the parity of the path $s_{(m-1)ei} \xrightarrow{path-ei} u_{ei} \xrightarrow{S_\infty} u_{li} \xrightarrow{path-li} s_{(m-1)li}$, for $1 \le i \le p$

Next we replace each path $s_{(m-1)ei}-s_{mei} \xrightarrow{cycle-c} s_{mli}-s_{(m-1)li}$ of cycle-c with path $s_{(m-1)ei} \xrightarrow{path-ei} u_{ei} \xrightarrow{S_\infty} u_{li} \xrightarrow{path-li} s_{(m-1)li}$ (see Figure 3.6), for $1 \le i \le p$, and obtain a *new* cycle-c. The parity of new cycle-c is the same as that of cycle-c. The new cycle-c consists of vertices in W_{m-1} only (actually new cycle-c consists of vertices in the subgraph induced by $K-\{u_0,u_1,\ldots,u_{2m}\}+S_1-N_1(u_0,u_1,\ldots,u_{2m})+ \cdots +S_{m-1}-N_{m-1}(u_0,u_1,\cdots,u_{2m})$, and this subgraph is a subgraph of W_{m-1}). Hence new cycle-c is an even cycle by the induction hypothesis (because W_{m-1} is bipartite). This implies that cycle-c is an even cycle.

We conclude that the subgraph induced by vertices $W_t=K-\{u_0,u_1,\ldots,u_{2t}\}+S_1-N_1(u_0,u_1,\ldots,u_{2t})+ \ldots +S_t-N_t(u_0,u_1,\cdots,u_{2t})$ is bipartite, because the subgraph does not contain any odd cycles.

□

3.7. A Special Case

In this section we come back to the 1-neighborhood of the k-cycle. First we prove two Lemmas – 3.3 and 3.4. Lemma 3.3 improves Theorem 3.1, and Lemma 3.4 improves Lemma 3.3. Then we prove Theorem 3.2 which uses Lemma 3.4 and shows that, for $k>5$, we can 3-color the k-cycle plus its 1-neighborhood. Last we demonstrate that Theorem 3.2 is tight, i.e., k can not be 5 or less, by showing a graph, the Grötzsch graph, which is 4-chromatic and the entire graph consists of a smallest odd cycle of length 5 plus its 1-neighborhood.

Lemma 3.3. For any 3 consecutive vertices, u_i, u_{i+1}, u_{i+2}, in K, the subgraph induced by vertices $K-\{u_{i+1}\}+S_1-N_1(u_i,u_{i+1},u_{i+2})$ is bipartite.

Proof. Note that Lemma 3.3 differs from Theorem 3.1 in that the subgraph under consideration in Lemma 3.3 is composed of the bipartite subgraph of Theorem 3.1 plus two vertices $-u_i$ and u_{i+2}. Without loss of generality, let's reindex the k vertices in K in clockwise consecutive order starting with i=0, hence u_i becomes u_0, u_{i+1} becomes u_1, and so on.

First we show that the subgraph induced by vertices $K-\{u_1,u_2\}+S_1-N_1(u_0,u_1,u_2)$ is bipartite. Let *cycle-c* be a simple cycle in the above subgraph containing the vertex u_0. We claim that cycle-c is even. To see this, let's consider different types of neighborhoods of u_0.

I Both neighbors of u_0 on cycle-c are in S_1; let's call them s' and s''.

Since both s' and s'' are adjacent to u_0 and are not in $N_1(u_0,u_1,u_2)$, then both of them must be adjacent to u_{k-2} by Property 3.3. The path from s' to s'' of cycle-c not containing u_0 satisfies the conditions of Corollary 3.5. Hence the parity of that path = parity of path $u_{k-2} - u_{k-2}$ = even. This implies that cycle-c is even.

II One neighbor of u_0 on cycle-c is in S_1, the other in K; let's call them s' and u'.

s' must be adjacent to u_{k-2} by the same argument as above. u' must be u_{k-1} because u_1 is not in the subgraph. By Corollary 3.4, the parity of any path from u_{k-1} to s' within the subgraph induced by vertices $K-\{u_0,u_1,u_2\}+S_1-N_1(u_0,u_1,u_2)$ is even. Thus cycle-c is even.

We have established that any cycle in the subgraph induced by $K-\{u_1,u_2\}+S_1-N_1(u_0,u_1,u_2)$ containing the vertex u_0 (or not) is even. Hence the subgraph is bipartite. It is easy to show that Corollaries 3.2 through 3.5 still hold if the cycle or path under consideration includes the vertex u_0 (which is u_i in the Corollaries). Then we can use similar arguments as above to show that the subgraph induced by $K-\{u_1\}+S_1-N_1(u_0,u_1,u_2)$ (by adding u_2 to the above bipartite subgraph) is also bipartite.

□

Lemma 3.4. If $k > 5$, then for any 3 consecutive vertices, u_i, u_{i+1}, u_{i+2}, in K, the subgraph induced by vertices $K-\{u_{i+1}\}+S_1-N_1(u_i,u_{i+2})$ is bipartite.

Proof. Note that Lemma 3.4 differs from Lemma 3.3 in that the subgraph under consideration in Lemma 3.4 is composed of the bipartite subgraph of Lemma 3.3 plus vertices in $N_1(u_{i+1})$, because $N_1(u_i,u_{i+1},u_{i+2}) - N_1(u_i,u_{i+2}) = N_1(u_{i+1})$ (since if a vertex s in S_1 is in $N_1(u_i,u_{i+1},u_{i+2})$ and adjacent to u_{i+1}, then s is not adjacent to either u_i or u_{i+2}). Without loss of generality, let's reindex the k vertices in K in clockwise consecutive order starting with i=0, hence u_i becomes u_0, u_{i+1} becomes u_1, and so on. Let *cycle-c* be a cycle in the subgraph induced by vertices $K-\{u_1\}+S_1-N_1(u_0,u_2)$ containing at least one vertex in $N_1(u_1)$. Let s be a vertex in $N_1(u_1)$ on cycle-c. Note that both neighbors of s on cycle-c are in S_1, because s is not adjacent to any other vertex in K besides u_1; let's call the two neighbors of s on cycle-c s' and s''. Because s' and s'' are not in $N_1(u_0,u_2)$ and $k > 5$, they must be adjacent to u_{k-2} or u_4 by Property 3.6. But it can not be the case that one of s', s'' is adjacent to u_{k-2} and the other adjacent to u_4, since the distance between u_{k-2} and u_4 is 6 and the distance between s' and s'' is only 2 (Property 3.7). Thus both s' and s'' are adjacent to the same vertex, either u_{k-2} or u_4, in K.

Now we divide the two neighbors (which are on cycle-c) of all vertices v such that v is on cycle-c and in $N_1(u_1)$ into 2 sides, side-1 and side-2. Side-1 contains those adjacent to u_4. Side-2 contains those adjacent to u_{k-2}. See Figure 3.7 for illustration. Note that vertices on both sides are distinct, and number of vertices on each side is even because both neighbors of vertices in $N_1(u_1)$ go on the same side.

Next we delete from cycle-c vertices in $N_1(u_1)$ and edges incident to those vertices. Note that no two vertices in $N_1(u_1)$ are adjacent (otherwise we obtain a 3-cycle which consists of u_1 and two vertices in $N_1(u_1)$), hence the number of edges deleted is even. After deletion cycle-c becomes a collection of paths. Each path goes from a vertex on side-i to another vertex on side-j, $i, j \in \{1,2\}$, and contains exactly two vertices among the vertices from the two sides. Note that none of the paths contains u_1 and

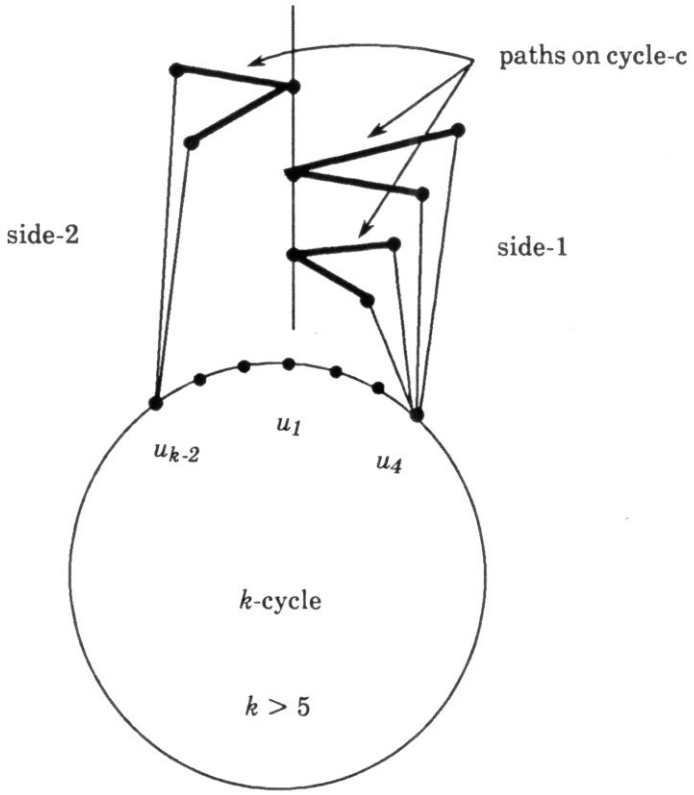

Figure 3.7. An illustration of division of vertices into 2 sides, side-1 and side-2.

vertices in $N_1(u_0,u_1,u_2)$. The number of paths from a vertex on side-1 to a vertex on side-2 is even, because the number of vertices on each side is even.

First let's consider paths that originate and terminate on side-1. The paths are of the form $s_a \rightarrow s_b$, where s_a and s_b are adjacent to u_4. The path $s_a \rightarrow s_b$ satisfies conditions in Corollary 3.5 (modified so that the path under consideration includes vertices u_0 and u_2, or u_i and u_{i+2} in the Corollary), hence parity of path $s_a \rightarrow s_b$ = parity of path $u_4 \rightarrow u_4$ = even. So parity of paths from a vertex on side-1 to another vertex on side-1 is even. The same result applies for side-2, that is, parity of paths from a vertex on side-2 to another vertex on side-2 is even.

Second let's consider paths from a vertex on side-1 to a vertex on side-2. The paths are of the form $s_a \rightarrow s_b$, where s_a is adjacent to u_4 and s_b is adjacent to u_{k-2}. The path $s_a \rightarrow s_b$ satisfies conditions in Corollary 3.5, hence parity of path $s_a \rightarrow s_b$ = parity of path $u_4 - u_5 - \cdots - u_{k-2}$ = odd. So parity of paths

from a vertex on side-1 to a vertex on side-2 is odd.

Cycle-c is composed of an even number of edges (those deleted), an even number of odd paths (side-1 to side-2), and a number of even paths (side-1 to side-1, side-2 to side-2); therefore, cycle-c is an even cycle.

We have established that any cycle in the subgraph induced by $K-\{u_{i+1}\}+S_1-N_1(u_i,u_{i+2})$ containing vertices in $N_1(u_1)$ (or not) is even. Hence the subgraph is bipartite.

□

Theorem 3.2. If $k > 5$, then we can 3-color the subgraph induced by vertices $K+S_1$.

Proof. We know the subgraph induced by vertices $K-\{u_{i+1}\}+S_1-N_1(u_i,u_{i+2})$, for some i, is bipartite from Lemma 3.4. Hence we can use 2 colors to color it.

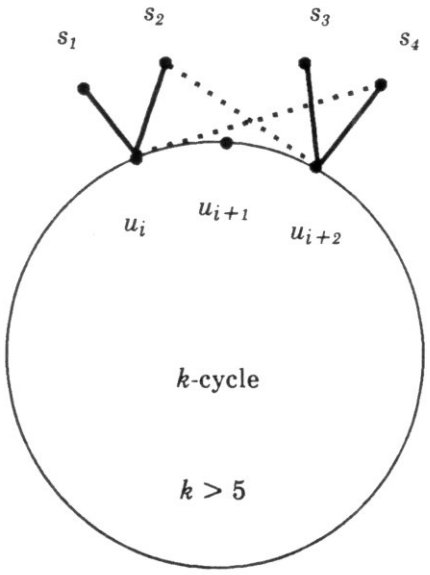

Figure 3.8. A configuration showing the independence of vertices in $N_1(u_i, u_{i+2})$ and u_{i+1}.

Let s_1, s_2 be vertices in $N_1(u_i,u_{i+2})$ adjacent to u_i, and s_3, s_4 be vertices in $N_1(u_i,u_{i+2})$ adjacent to u_{i+2} (see Figure 3.8). s_1 is not adjacent to s_2, otherwise we obtain a 3-cycle: $s_1-s_2-u_i-s_1$. Similarly s_3 is not adjacent s_4. u_{i+1} is not adjacent to s_i, $i \in \{1, 2, 3, 4\}$, otherwise we can easily obtain a 3-cycle.

Since $k > 5$, s_1 is not adjacent to s_3 (in general a vertex in $N_1(u_i,u_{i+2})$ adjacent to u_i is not a neighbor of a vertex in $N_1(u_i,u_{i+2})$ adjacent to u_{i+2}), otherwise we obtain a 5-cycle: $s_1-u_i-u_{i+1}-u_{i+2}-s_3-s_1$. Thus we can use the third color to color the independent set $u_{i+1}+N_1(u_i,u_{i+2})$. Therefore we can 3-color the subgraph induced by the vertices $K+S_1$.

□

Next we show that Theorem 3.2 is tight, that is, when $k = 5$, the subgraph induced by vertices $K+S_1$ is not always 3-colorable. The *Grötzsch* graph [Grötzsch, 1958], shown in Figure 3.9, is one example where a smallest odd cycle size is 5, the chromatic number is 4, and if we pick $v_1-v_2-\cdots-v_5-v_1$ to be the *k-cycle*, then $K+S_1$ is the entire graph.

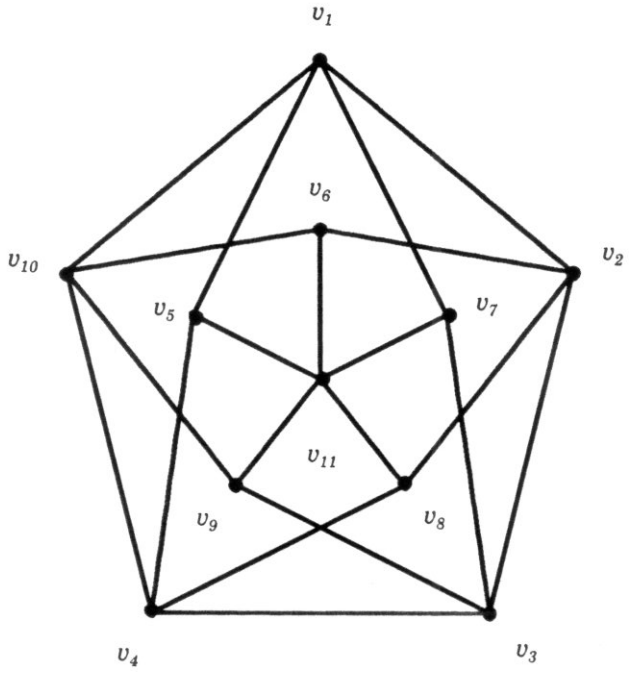

Figure 3.9. The Grötzsch graph.

Chapter 4

Coloring Graphs with only "Large" Odd Cycles

4.1. Introduction

In Chapter 3 we focused on a smallest odd cycle of a graph and its local neighborhood, and we showed that portions of this neighborhood are bipartite. Recall the goal in that chapter is to investigate the role played by smallest odd cycles of a graph in connection with graph coloring. This is an ambitious and challenging goal. Although progress has been made, as we have shown that we can find 2-colorable neighborhood of a smallest odd-cycle, much more remains to be done. We shall continue with this investigation in this chapter.

Our focus in this chapter will still be on a smallest odd cycle of a graph and its local neighborhood; however, our aim is to develop good approximate graph coloring algorithms based on this focus. The "goodness" of a graph coloring algorithm depends not only on its simplicity, efficiency, but also on its performance – in terms of number of colors used.

In this chapter we present two approximate graph coloring algorithms, both of which are simple and efficient. The main idea behind both algorithms is to find a subgraph of G of a certain size that we can color with a small number of colors. The support for this idea is the results of Chapter 3 – the bipartite neighborhood of the k-cycle. The performance of these two algorithms depends greatly on k – the size of a smallest odd cycle in G (actually k relative to n). Each algorithm has a certain range (of k) over which it out performs the other. As we have seen in the last chapter, the larger the value k is, the larger is the bipartite neighborhood of the k-cycle. A direct translation of that result in this chapter is: the larger the value k is (relative to n), the better each algorithm performs (i.e., the fewer colors used by each algorithm) in both absolute and relative (compared with the chromatic number or with other existing

approximate graph coloring algorithms) terms. Hence the title of this chapter.

Now we shall give an outline of the developments in this chapter. We first focus on the k-cycle and its 1-neighborhood ($K+S_1$). We have shown in Theorem 3.2 that we can 3-color $K+S_1$. With the aid of the structures of $K+S_1$ and some counting arguments, we derive a lower bound on the size of $K+S_1$. Once we know how large a graph we can 3-color, then we can recursively find 3-colorable subgraphs until all the vertices in G are exhausted. This gives us the first approximate graph coloring algorithm. We show that if $k\geq 5$ and k is of the form $k>\left\lceil \frac{n}{i} \right\rceil$, for some i, then using this algorithm we can q-color G, where q is upper bounded by $6\left\lceil \frac{i}{3} \right\rceil^{1/2}$ (an explicit formula for q is also given). This result says that we can color all triangle-free graphs ($k\geq 5$) with $O(\sqrt{n})$ colors.

Next we focus on the k-cycle and its local neighborhood – $S_1, S_2, \cdots, S_t$, for some t to be specified later. We introduce the notion of *goals* and *separators*, and a strategy for finding a subgraph of G that we can 4-color. With the help of goals, separators, and Theorem 3.1G, we derive a lower bound on the size of the 4-colorable subgraph that we find in G. Based on this result, we present another recursive algorithm for approximate graph coloring for $k\geq 5$. Using this second algorithm, we show that if $k\geq 4(\lg n - \lg\lg n)$ then we can $4\lg n$-color G.

We evaluate the performance of the two approximate graph coloring algorithms presented by first comparing them against each other, second comparing them against the chromatic numbers of graphs under consideration, and third comparing them against other approximate graph coloring algorithms. As a result of the evaluation, we show that for all triangle-free graphs, the performance of the coloring algorithms presented in this chapter are better than that of all known approximate graph coloring algorithms.

In this chapter, we will use the same notation as that presented in Chapter 3. Again, we assume that the size of a smallest odd cycle in G is greater than 3 ($k\geq 5$). When we say q-color a graph G, we mean $\lceil q \rceil$-color G.

4.2. k-Cycle and its 1-Neighborhood

We have shown in Chapter 3 that the k-cycle and its 1-neighborhood possess many special structures and properties. The most important ones are: 1. a large portion of $K+S_1$ is bipartite – Theorem 3.1, and 2. if $k>5$, then $K+S_1$ can be 3-colored – Theorem 3.2. We will use these results in this section to develop algorithms for graph coloring.

We start by examining graphs with $k>\left\lceil\frac{n}{2}\right\rceil$, $\left\lceil\frac{n}{3}\right\rceil$, and $\left\lceil\frac{n}{4}\right\rceil$, and designing algorithms that 3, 4, and 5-color them, respectively. Then we present a general approach for graph coloring, using the above three classes of graphs as basis. We analyze the performance of the algorithm on k of the form: $k>\left\lceil\frac{n}{i}\right\rceil$, for $i>4$, and find that the number of colors used by the algorithm is at most $6\left\lceil\frac{i}{3}\right\rceil^{1/2}$. Since we assume that $k\geq 5$, then i is $O(n)$, which implies that the algorithm uses $O(\sqrt{n})$ colors.

Before we proceed with the algorithms, we first observe and prove the following fact and lemma.

Fact 4.1. If a smallest odd cycle size of G is k and G' is a subgraph of G, then a smallest odd cycle size of G' is at least k.

This fact is important because in this chapter we deal with recursive algorithms. Using the notation of Chapter 3, knowing the value k of a graph G, we can safely assume that the same k value applies to any subgraph of G (i.e., any subgraph of G has k as its smallest odd cycle size), or we can do better.

Lemma 4.1. If the minimum degree of G is at least q, then for $k>5$ ($k=5$), we can find a bipartite subgraph of size at least $\frac{qk}{2}-q+1$ ($2q$) in $K+S_1$.

Proof. We shall consider the 2 cases $k>5$ and $k=5$ separately.

I. $k > 5$

We shall use Lemma 3.4 to find a bipartite subgraph of the desired size. Since the degree of every vertex in G is at least q, then every vertex in K is adjacent to at least $q-2$ vertices in S_1. Property 3.4 says that every vertex in S_1 is adjacent to at most 2 vertices in K, thus the number of vertices in S_1 is at least $\frac{(q-2)k}{2}$ (i.e., $|S_1|\geq\frac{(q-2)k}{2}$).

Let

$$\min_{0\le i\le k-1} |N_1(u_i,u_{i+2})| = a,$$

then

$$|S_1| \ge \frac{ak}{2}, \qquad a \le \frac{2|S_1|}{k},$$

$$|S_1| - a \ge |S_1| - \frac{2|S_1|}{k} = |S_1|(1-\frac{2}{k}) \ge \frac{(q-2)k}{2} - (q-2).$$

According to Lemma 3.4, we can find a bipartite subgraph of size at least $\frac{(q-2)k}{2}-(q-2)+k-1 = \frac{qk}{2}-q+1$ in $K+S_1$.

II. $k = 5$

Since k=5 and the degree of every vertex is at least q, then any two adjacent vertices in K plus vertices adjacent to them constitute a bipartite subgraph of size $2q$.

□

Corollary 4.1. If the minimum degree of G is at least 4, then we can find an independent set of size at least $k-1$ in $K+S_1$.

Proof. We shall consider 2 cases.

I. $k > 5$.

By Lemma 4.1, we can find a bipartite subgraph of size at least $2k-3$ (since q=4) in $K+S_1$, which means that we can find an independent set of size at least $k-1$ in $K+S_1$.

II. $k = 5$.

By Lemma 4.1, we can find a bipartite subgraph of size at least 8 (since q=4) in $K+S_1$, which means that we can find an independent set of size at least 4=$k-1$ in $K+S_1$.

□

4.2.1. $k > \left\lceil \frac{n}{2} \right\rceil$: 3 Colors

In this section we present an algorithm that 3-colors a graph with $k>\left\lceil \frac{n}{2} \right\rceil$. This algorithm is the

only one in Section 4.2 that does not depend on Theorem 3.2; rather, it exploits special structures of the k-cycle and its 1-neighborhood and takes advantage of the fact that k is large. One such special structure is captured in the following fact.

Fact 4.2. If $k > \left\lceil \frac{n}{2} \right\rceil$, then there exists a vertex u_i in K that does not have the following structure: u_i–u_{i+1}–s–u_{i-1}–u_i, for any $s \in S_1$.

Proof. Suppose that every vertex in K has the described structure. Since every vertex in S_1 is adjacent to at most 2 vertices in K (Property 3.4), then S_1 must contain at least k vertices. But $k > \left\lceil \frac{n}{2} \right\rceil$, and $k+k>n$. Contradiction.

□

The algorithm goes as follows.

Input. A graph $G=(V,E)$, $k > \left\lceil \frac{n}{2} \right\rceil$.

Output. A 3-coloring of G.

1. Find a smallest odd cycle, the *k-cycle*, in G.
   Let K and S_1 be defined relative to the k-cycle in G.
2. Find a vertex u_i in K that does not have the following structure (Fact 4.2): u_i–u_{i+1}–s–u_{i-1}–u_i, for any $s \in S_1$.
3. Color vertex u_i with color C.
4. Color K–u_i (which is bipartite) with colors A and B.
   (The rest of the graph $\{G-K+u_i\}$, i.e., the graph induced by $\{V-K+u_i\}$, is bipartite, since the number of vertices remaining is $n-k-1 \leq \left\lceil \frac{n}{2} \right\rceil$.)
5. Divide {G–K+u_i} into 2 independent sets, I_1 and I_2.
6. WLOG let us assume that $u_i \in I_1$. Color vertices in I_1 with color C.
7. Color vertices s in I_2 as follows:
   8. If s is not adjacent to vertices in K that are colored A or B, then color s with color A (or we could color s with color B).
   9. If s is adjacent to exactly one vertex in K that is colored A(B), then color s with color B(A).
   10. If s is adjacent to two vertices u_a and u_b in K, then the shorter distance within the k-cycle between u_a and u_b is 2 by Property 3.3. Note that u_i is not the vertex that is distance 1 from

both u_a and u_b by the way u_i is picked. Hence u_a and u_b must have been colored the same, say with color A, so we color s with color B.

(If a vertex s is in S_1, then s is adjacent to at most 2 vertices in K by Property 3.4. Therefore we need to consider no further.)

Algorithm 4.1. A 3-coloring algorithm for graphs with $k > \left\lceil \frac{n}{2} \right\rceil$.

Theorem 4.1. If $k > \left\lceil \frac{n}{2} \right\rceil$, then Algorithm 4.1 3-colors G.

Proof. We need to show that Algorithm 4.1 produces a legal 3-coloring of G. All the vertices colored C are in I_1 which is an independent set. All the vertices in I_2 are independent and are colored with colors either A or B. Thus the coloring among vertices in I_2 is legal. The coloring among vertices in I_2 and vertices in K is legal as well, because, as we have shown in Step 10, no vertex in I_2 is adjacent to two vertices, one colored A and the other colored B, in K.

□

Notice that the number of colors used in the above algorithm is tight, since the chromatic number for any non-bipartite graph is at least 3.

The running time of Algorithm 4.1 is dominated by Step 1 – the time needed to find a smallest odd cycle; all other steps can be accomplished in time $O(e)$. As discussed in Chapter 2, the complexity of finding a smallest odd cycle in a graph is the same as that of matrix multiplication when the graph is dense, otherwise it is $O(e^{3/2})$. Hence the complexity of Algorithm 4.1 is $O(\min\{e^{3/2}, n^{2.376}\})$.

4.2.2. $k > \left\lceil \frac{n}{3} \right\rceil$: 4 Colors

Our approach for 4-coloring G with $k > \left\lceil \frac{n}{3} \right\rceil$ is simple. First we find an independent set of size at least $k-1$ in $K+S_1$, and this independent set uses one color. Then we 3-color the rest of the graph using Algorithm 4.1. To find an independent set of size at least $k-1$ in $K+S_1$, we use Corollary 4.1 of Lemma 4.1.

Input. A graph $G=(V,E)$, $k>\left\lceil \frac{n}{3} \right\rceil$.

Output. A 4-coloring of G.

1. Delete (iteratively) all vertices in G with degree less than 4, and call the resulting graph G_4. (Hence after deletion, the minimum degree of G_4 is 4 or more.)
2. Find a smallest odd cycle, the *k-cycle*, in G_4.
   Let K and S_1 be defined relative to the k-cycle in G_4.
3. Find an independent set of size at least $k-1$ in $K+S_1$.
4. Use Algorithm 4.1 to 3-color the remaining graph (G_4 – independent set).
5. Add the vertices deleted in Step 1 to G_4 and 4-color them.

Algorithm 4.2. A 4-coloring algorithm for graphs with $k > \left\lceil \frac{n}{3} \right\rceil$.

Theorem 4.2. If $k>\left\lceil \frac{n}{3} \right\rceil$, then Algorithm 4.2 4-colors G.

Proof. Because of Corollary 4.1 we can carry out Step 3, hence all we need to show is that we are able to perform Step 4. After the independent set of size at least $k-1$ (Step 3) is deleted from G_4, the size of the remaining graph is $n-(k-1) \le n-\left\lceil \frac{n}{3} \right\rceil \le \frac{2n}{3}$ (since $k>\left\lceil \frac{n}{3} \right\rceil$, $k\ge\left\lceil \frac{n}{3} \right\rceil+1$). If we let $m=\frac{2n}{3}$, then $k>\left\lceil \frac{m}{2} \right\rceil$. Hence we can 3-color the remaining graph with Algorithm 4.1 by Theorem 4.1. Next (in Step 5) we can 4-color the vertices that are deleted in Step 1 due to Lemma 2.1.

□

The complexity of Algorithm 4.2 is $O(\min\{e^{3/2}, n^{2.376}\})$ (Steps 2 and 4). All other steps take $O(e)$ time.

4.2.3. $k > \left\lceil \frac{n}{4} \right\rceil$: 5 Colors

We will use another simple approach to 5-color a graph with $k>\left\lceil \frac{n}{4} \right\rceil$. Given a graph G, we first find a bipartite subgraph of size at least $\frac{5k}{2}-4$ in $K+S_1$, and this bipartite subgraph uses two colors. Then we 3-color the rest of the graph using Algorithm 4.1. The algorithm goes as follows:

Input. A graph $G=(V,E)$, $k>\left\lceil \frac{n}{4} \right\rceil$.

Output. A 5-coloring of G.

1. Delete (iteratively) all vertices in G with degree less than 5, and call the resulting graph G_5. (Hence after deletion, the minimum degree of G_5 is 5 or more.)
2. Find a smallest odd cycle, the *k-cycle*, in G_5.
   Let K and S_1 be defined relative to the k-cycle in G_5.
3. Find a bipartite subgraph of size at least $\frac{5k}{2}-4$ in $K+S_1$.
4. Use Algorithm 4.1 to 3-color the remaining graph (G_5 – bipartite subgraph).
5. Add the vertices deleted in Step 1 to G_5 and 5-color them.

Algorithm 4.3. A 5-coloring algorithm for graphs with $k > \left\lceil \frac{n}{4} \right\rceil$.

Theorem 4.3. If $k>\left\lceil \frac{n}{4} \right\rceil$, then Algorithm 4.3 5-colors G.

Proof. Because of Lemma 4.1 we can carry out Step 3 (for $k=5$ and $q=5$, $2q > \frac{5k}{2}-4$), then all we need to show is that we are able to perform Step 4. After the bipartite subgraph of size at least $\frac{5k}{2}-4$ (Step 3) is deleted from G_5, the size of the remaining graph is at most $m = n-(\frac{5k}{2}-4) \le n-\frac{5}{2}(\left\lceil \frac{n}{4} \right\rceil+1)+4 \le \frac{3n}{8}+\frac{3}{2}$ (since $k>\left\lceil \frac{n}{4} \right\rceil$, $k\ge\left\lceil \frac{n}{4} \right\rceil+1$). The number of vertices in G_5 is more than 12 ($n>12$) because $k\ge5$ and the number of vertices in $K+S_1$ is at least $\frac{5k}{2}$. Since $n>12$, $\frac{n}{4} \ge \frac{1}{2}(\frac{3n}{8}+\frac{3}{2})$, thus $k>\left\lceil \frac{m}{2} \right\rceil$. Therefore we can 3-color the remaining graph with Algorithm 4.1 by Theorem 4.1. The rest of the graph G can be 5-colored (Step 5) due to Lemma 2.1.

□

Algorithm 4.3 has the same running time as Algorithm 4.2, that is, $O(\min\{e^{3/2}, n^{2.376}\})$.

4.2.4. $k > \left\lceil \frac{n}{i} \right\rceil : 6\left\lceil \frac{i}{3} \right\rceil^{1/2}$ Colors

We have presented three different approaches for graph coloring specifically tuned for graphs with $k>\left\lceil \frac{n}{i} \right\rceil$, where $i = 2$, 3, and 4. We now present a general approach for approximate graph coloring. This approach does not depend on the form of k. However, the analysis is simplified when k is of the form $k>\left\lceil \frac{n}{i} \right\rceil$, for $i>4$. And this is the form of k we shall use.

The general approach is based on the following ideas:

1. If we are using q colors to color a graph G, then we can just deal with G_q, which has a minimum degree of at least q (Lemma 2.1).
2. We can q-color a graph G_q if we can
   a. p-color a subgraph G_q' of G_q, and
   b. $q-p$-color the remaining graph G_q-G_q' of G_q.

Since the subgraph $K+S_1$ is 3-colorable when $k>5$, it is natural to let p be 3, and 3-color $K+S_1$. To $q-3$-color the remaining graph, we can repeat the same process, that is, recurse until $q-3i\leq 5$, for some i, in which case we can use one of the three algorithms presented earlier in this section. This is actually what Algorithm 4.4 does, and it uses Algorithms 4.1, 4.2, and 4.3 as its basis. Algorithm 4.4 treats $k=5$ as a special case. The correctness of the above approach and of Algorithm 4.4 depends on q, which is yet to be specified.

Note that Algorithms 4.2 and 4.3 are special cases of the general approach, with p taking on the values 1 and 2, respectively. Since we have shown that a large portion of the neighborhood of the k-cycle is bipartite, we can equally let p be 2 in Algorithm 4.4. So why do we choose p to be 3? The first reason is that we are not able to bound the size of the bipartite neighborhood of the k-cycle we obtain. The second reason is that when $p=3$ the analysis of Algorithm 4.4 is simpler, i.e., we obtain a clean (or cleaner) formula for q – the number of colors used. However, with the methods which we have developed for finding 2-colorable and 3-colorable subgraphs, $p=2$ is probably the better approach. That is, it probably takes fewer colors to color G if we recursively find 2-colorable subgraphs in G, using

Lemma 4.1 to bound the size of the 2-colorable subgraphs, than if we recursively find 3-colorable subgraphs in G. Due to the complexity of the analysis when p=2, we justify the preceding statement by the size of 3-colorable subgraphs that we find with each approach, assuming k>5. In the p=3 case, we can find a 3-colorable subgraph – $K+S_1$ of size at least $\frac{k(q-2)}{2}$. In the p=2 case, we can find a bipartite subgraph of size at least $\frac{kq}{2}-q+1$, by Lemma 4.1, in $K+S_1$. In the next iteration, we can find a bipartite subgraph of size at least $\frac{k(q-2)}{2}-(q-2)+1$, again by Lemma 4.1. If we combine the first bipartite subgraph and an independent set of the second bipartite subgraph, we obtain a 3-colorable subgraph of size at least $\frac{3kq}{4}-\frac{k}{2}-\frac{3q}{2}+\frac{5}{2}$, which is larger than the size of the 3-colorable subgraph we find with the first approach (p=3), assuming k>5. Thus in the p=2 case, we may be able to start with a smaller q than the p=3 case. This implies that the number of colors used by the second approach is as few as the first, if not fewer. But for clarity and simplicity, we have decided in favor of p=3. We ask the readers to keep in mind that the number of colors used by Algorithm 4.4 could possibly be improved. The numbers we provide can serve as an upper bound.

Input. A graph $G=(V,E)$, $k>\left\lceil \frac{n}{i} \right\rceil$, $i>4$.

Output. A q-coloring of G.

1. Delete (iteratively) all vertices in G with degree less than q, and call the resulting graph G_q. (Hence after deletion, the minimum degree of G_q is q or more.)
2. Find a smallest odd cycle, the *k-cycle*, in G_q.
   Let K and S_1 be defined relative to the k-cycle in G_q.
3. If k=5, then do
   4. Find a bipartite subgraph of size at least $2q$ in $K+S_1$ and 2-color it.
   5. If there exists a vertex in the remaining graph (G_q –bipartite subgraph) of degree at least $q-2$, then do
      6. Obtain an independent set of size at least $q-2$ in (G_q – bipartite subgraph) and 1-color it. (Hence we have obtained a 3-colorable subgraph of size at least $3q-2$.)
      7. If $q-3$ = 3, 4, or 5, then
         8. Use Algorithm 4.1, 4.2, or 4.3 to $q-3$-color the remaining graph (G_q – 3-colorable subgraph), respectively.

9. Else (q–3> 5)

10. Use Algorithm 4.4 to q–3-color the remaining graph (G_q – 3-colorable subgraph).

11. Else (the maximum degree of (G_q – bipartite subgraph) is at most q–3)

12. q–2 color (G_q – bipartite subgraph).

13. Else (k>5), do

14. 3-color the subgraph $K+S_1$.

15. If q–3 = 3, 4, or 5, then

16. Use Algorithm 4.1, 4.2, or 4.3 to q–3-color the remaining graph ($G_q - K - S_1$), respectively.

17. Else (q–3> 5)

18. Use Algorithm 4.4 to q–3-color the remaining graph ($G_q - K - S_1$).

19. Add the vertices deleted in Step 1 to G_q and q-color them.

Algorithm 4.4. A q-coloring algorithm for graphs with $k > \left\lceil \frac{n}{i} \right\rceil$ and i>4.

Now we shall specify what q is in Algorithm 4.4. If we express i as

$$i = 1 + 6(0+1+2+\cdots+a) + (a+1)b + c,$$

$$b < 6, \qquad c < a+1,$$

that is,

$$i \geq 6\sum_{j=0}^{a} j + 1,$$

but

$$i < 6\sum_{j=0}^{a+1} j + 1, \qquad \text{for some } a,$$

and

b is the quotient of $i - (6\sum_{j=0}^{a} j + 1)$ divided by $a+1$,

c is the remainder of $i - (6\sum_{j=0}^{a} j + 1)$ divided by $a+1$.

Then

$$q = f(a, b, c)$$

where

$$f(a,\ b,\ c) = \begin{cases} 2 + 6a + b, & \text{if } c = 0, \\ 2 + 6a, & \text{if } c = 1,\ b = 0, \\ 3 + 6a + b, & \text{if } c > 0 \text{ and } b \neq 0 \textit{ when } c = 1. \end{cases}$$

Theorem 4.4. If $k > \left\lceil \frac{n}{i} \right\rceil$, $i > 4$, then Algorithm 4.4 q-colors G, with q as specified above.

Proof. Algorithm 4.4 is recursive in nature. Note that the basis of this algorithm is Algorithms 4.1, 4.2, and 4.3. It can be easily verified that the equation for q is valid when $i = 2$, 3, and 4. We shall assume that we are using q colors to color G, thus we can just deal with G_q, which has a minimum degree of at least q.

First let us consider the case $k > 5$. Because of Theorem 3.2, we can 3-color the subgraph $K+S_1$ (Step 14). We now need to prove that we can $q-3$-color the rest of the graph $G_q - K - S_1$. Let $m = |G_q - K - S_1|$. Basically we need to show that $k > \left\lceil \frac{m}{j} \right\rceil$, for some j, and if j can be expressed as $j = 1 + 6(0+1+2+\cdots+a_j) + (a_j+1)b_j + c_j$, for some a_j, b_j, c_j, then $q-3 \geq f(a_j,b_j,c_j)$.

Because of Property 3.4 and the degree of every vertex in G_q is at least q, the number of vertices in S_1 is at least $\frac{k(q-2)}{2}$. Thus the number of vertices in $G_q - K - S_1$ is at most $n - \frac{kq}{2}$. So

$$m \leq n - \frac{kq}{2} < n - \frac{nq}{2i}, \quad \text{since } k > \left\lceil \frac{n}{i} \right\rceil,$$

$$m < n\left(1 - \frac{q}{2i}\right), \quad \frac{n}{i} > \frac{m}{i - \frac{q}{2}} \geq \frac{m}{\left\lceil i - \frac{q}{2} \right\rceil}.$$

Let

$$j = \left\lceil i - \frac{q}{2} \right\rceil,$$

then

$$k > \left\lceil \frac{n}{i} \right\rceil \geq \left\lceil \frac{m}{j} \right\rceil.$$

Knowing the value of j, we will show that $q-3 \geq f(a_j,b_j,c_j)$. We begin with $q-3 = 3, 4, 5$, that is, if $q = 6, 7, 8$, then in order to apply Algorithms 4.1, 4.2, 4.3, we need to show that $j \leq 2, 3, 4$, respectively. We shall consider each case separately.

If $q = 6$ then $a = 0$, hence $c = 0$, so $b = 4$ and $i = 1+0+4+0 = 5$. This means that $j = \left\lceil i-\frac{q}{2} \right\rceil = \left\lceil 5-\frac{6}{2} \right\rceil = 2$. Thus we can use Algorithm 4.1.

If $q = 7$ then $a = 0$, hence $c = 0$, so $b = 5$ and $i = 1+0+5+0 = 6$. This means that $j = \left\lceil i-\frac{q}{2} \right\rceil = \left\lceil 6-\frac{7}{2} \right\rceil = 3$. Thus we can use Algorithm 4.2.

If $q = 8$ then $a = 1$, hence $b = 0$, and either $c = 0$ or $c = 1$. So either $i = 1+6+0+0 = 7$, or $i = 1+6+0+1 = 8$. This means that either $j = \left\lceil i-\frac{q}{2} \right\rceil = \left\lceil 7-\frac{8}{2} \right\rceil = 3$, or $j = \left\lceil i-\frac{q}{2} \right\rceil = \left\lceil 8-\frac{8}{2} \right\rceil = 4$. In both cases we can use Algorithm 4.3.

Having proved the reduction to the basis step, now we move on to prove the general induction step. Again we want to show that $q-3 \geq f(a_j,b_j,c_j)$, where $j = \left\lceil i-\frac{q}{2} \right\rceil = 1 + 6(0+1+2+\cdots+a_j) + (a_j+1)b_j+c_j$. There are 3 cases.

I. $i = 1 + 6(0+1+2+\cdots+a) + (a+1)b, \qquad (c = 0)$

$q = 2 + 6a + b,$

$j = \left\lceil i-\frac{q}{2} \right\rceil = 1 + 6(0+1+2+\cdots+a-1) + a(3+b) + \left\lceil \frac{b}{2} \right\rceil - 1.$

If $b = 0$ and $a=1$, then $f(0, 2, 0) = 2+0+2 = q-4$.

If $b = 0$ and $a>1$, then $f(a-1, 2, a-1) = 3+6(a-1)+2 = q-3$.

If $b = 1$ or 2, then $f(a-1, 3+b, 0) = 2+6(a-1)+3+b = q-3$.

If $b = 3$, then $f(a, 0, 1) = 2+6a = q-3$.

If $b = 4$, then $f(a, 1, 0) = 2+6a+1 = q-3$.

If $b = 5$, then $f(a, 2, 0) = 2+6a+2 = q-3$.

II. $i = 1 + 6(0+1+2+\cdots+a) + 1, \qquad (c = 1 \text{ and } b = 0)$

$q = 2 + 6a, \qquad j = \left\lceil i - \frac{q}{2} \right\rceil = 1 + 6(0+1+2+\cdots+a-1) + 3a,$

$f(a-1, 3, 0) = 2+6(a-1)+3 = q-3.$

III. $i = 1 + 6(0+1+2+\cdots+a) + (a+1)b + c,$

$(0<c<a+1 \text{ and } b\neq 0 \text{ when } c=1)$

$q = 3 + 6a + b,$

$j = \left\lceil i - \frac{q}{2} \right\rceil = 1 + 6(0+1+2+\cdots+a-1) + a(3+b) + c + \left\lceil \frac{b-3}{2} \right\rceil.$

If $b = 0$ and $c > 1$, then $f(a-1, 3, c-1) = 3+6(a-1)+3 = q-3.$

If $b = 1$ and $c = 1$, then $f(a-1, 4, 0) = 2+6(a-1)+4 = q-4.$

If $b = 1$ and $c > 1$, then $f(a-1, 4, c-1) = 3+6(a-1)+4 = q-3.$

If $b = 2$ and $c < a$, then $f(a-1, 5, c) = 3+6(a-1)+5 = q-3.$

If $b = 2$ and $c = a$, then $f(a, 0, 0) = 2+6a = q-3.$

If $b = 3$ and $c = 1$, then $f(a, 0, 1) = 2+6a = q-4.$

If $b = 3$ and $c > 1$, then $f(a, 0, c) = 3+6a = q-3.$

If $b = 4$, then $f(a, 1, c) = 3+6a+1 = q-3.$

If $b = 5$ and $c = 1$, then $f(a, 2, 0) = 2+6a+2 = q-4.$

If $b = 5$ and $c > 1$, then $f(a, 2, c-1) = 3+6a+2 = q-3.$

We have shown that we can $q-3$-color the subgraph G_q-K-S_1. Since we can 3-color $K+S_1$, hence q colors are sufficient to color G_q.

Second let us consider the case $k=5$. Because of Lemma 4.1, we can find a bipartite subgraph of size at least $2q$ in $K+S_1$. If there exists a vertex of degree at least $q-2$ in (G_q – bipartite subgraph), then the neighbors of this vertex form an independent set of size at least $q-2$, because $k=5$. Combining the bipartite subgraph and the independent set, we have a 3-colorable subgraph of size at least $3q-2$. Since $k=5$ and $q>4$ (because $i>4$), then $3q-2 > \frac{kq}{2}$. Hence the size of the 3-colorable subgraph is more than $\frac{kq}{2}$. Using the same analysis as in the case of $k>5$, we can show that the subgraph (G_q – 3-colorable

subgraph) can be $q-3$-colored. Thus q colors are sufficient to color G_q. If there exists no vertex of degree at least $q-2$ in (G_q – bipartite subgraph), which means that the maximum degree of (G_q – bipartite subgraph) is at most $q-3$, then (G_q – bipartite subgraph) can be $q-2$ colored (Step 12), because every vertex is adjacent to at most $q-3$ other vertices. In fact, the graph in Step 12 can be $q-3$-colored, for all $q-3 > 2$, using a theorem of Brooks[1941] (Brooks' theorem says that if the maximum degree of a graph G is $d > 2$, then G is d-colorable unless G contains K_{d+1} as a subgraph). However, this improvement does not reduce the overall number of colors used in Algorithm 4.4.

Next we can q-color the rest of the graph G (Step 19) due to Fact 4.1.

□

Before analyzing the running time of Algorithm 4.4, we shall examine the behavior of q in terms of i. Note that i can also be expressed as

$$i = 1 + 3a(a+1) + (a+1)b + c.$$

Then

$$\frac{i}{3} - \frac{c}{3} = a^2 + (1+\frac{b}{3})a + \frac{1}{3} + \frac{b}{3},$$

$$\frac{i}{3} - \frac{c}{3} - \frac{1}{12} - \frac{b}{6}(1-\frac{b}{6}) = a^2 + (1+\frac{b}{3})a + \frac{1}{4} + \frac{b}{6} + \frac{b^2}{36},$$

$$= (a + \frac{1}{2} + \frac{b}{6})^2$$

$$6\left[\frac{i}{3} - \frac{c}{3} - \frac{1}{12} - \frac{b}{6}(1-\frac{b}{6})\right]^{1/2} = 3 + 6a + b.$$

Therefore

$$6\left[\frac{i}{3}\right]^{1/2} > 3 + 6a + b \geq q.$$

This result is important enough to deserve a theorem of its own.

Theorem 4.5. In a graph G, if $k \geq 5$ and $k > \left\lceil \frac{n}{i} \right\rceil$, then we can q-color G with q upper bounded by $6\left[\frac{i}{3}\right]^{1/2}$.

Theorem 4.5 says that we can color all triangle-free graphs ($k \geq 5$) with $O(\sqrt{n})$ colors.

Now we can analyze the running time of Algorithm 4.4. Each iteration of Algorithm 4.4 takes $O(\min\{e^{3/2}, n^{2.376}\})$ time. The number of times Algorithm 4.4 is called is $O(q)$ or $O(i^{1/2})$. Hence the complexity of Algorithm 4.4 is $O(i^{1/2}\min\{e^{3/2}, n^{2.376}\})$.

4.3. k-Cycle and its Local Neighborhood

In Section 4.2 we focused on the k-cycle and its 1-neighborhood, and essentially found a subgraph of a certain size that we can 3-color. Base on this result we designed a recursive algorithm for graph coloring (which works out well when k is of the form $k>\left\lceil \frac{n}{i} \right\rceil$). In this section we shall focus on a larger neighborhood of the k-cycle – $S_1, S_2, \cdots, S_t$. First we show that for all $k\geq 5$ and $t<\frac{k-1}{2}$, we can q-color the k-cycle plus the union of its 1-neighborhood through t-neighborhood, where $q = 2\left\lceil \frac{k}{k-2t-1} \right\rceil$. If we let $t=\left\lfloor \frac{k-3}{4} \right\rfloor$, then the preceding result says that we can 4-color the k-cycle plus all its first t neighborhoods. We choose the value $t=\left\lfloor \frac{k-3}{4} \right\rfloor$ because it gives us a better asymptotic bound on the number of colors used than other values of t. In this first t neighborhoods of the k-cycle, we show that if we fail to find a "large" 4-colorable subgraph, then we can find a *separator* (a set of vertices which, if deleted, makes the graph disjoint). We then introduce the notion of *goals* which places a bound either on the size of the 4-colorable subgraph or on the size of the separator. This bound enables us to develop a recursive algorithm for finding a 4-colorable subgraph in G of a certain size. Based on this 4-coloring algorithm, we then present an approximate graph coloring algorithm for all $k\geq 5$. Using this algorithm, we show that if $k\geq 4(\lg n - \lg\lg n)$ then we can $4\lg n$-color G.

4.3.1. A 4-Colorable Neighborhood of the k-Cycle

We now present a general theorem on coloring the local neighborhood of the k-cycle. The 4-colorable neighborhood of the k-cycle comes as a special case of this theorem.

Theorem 4.6. For all $k\geq 5$ and $t<\frac{k-1}{2}$, we can q-color the subgraph induced by vertices $K+S_1+S_2+\cdots+S_t$, where $q = 2\left\lceil \frac{k}{k-2t-1} \right\rceil$.

Proof. Because of Theorem 3.1G, we can 2-color the subgraph which consists of any $k-(2t+1)$ consecutive vertices on the k-cycle plus their first t neighborhoods. The number of disjoint sets of $k-(2t+1)$ consecutive vertices in K is $\left\lceil \frac{k}{k-2t-1} \right\rceil$ (where one of the set contains less than $k-(2t+1)$ consecutive vertices). Note that the sum of all $\left\lceil \frac{k}{k-2t-1} \right\rceil$ sets of $k-(2t+1)$ consecutive vertices plus their first t neighborhoods is $K+S_1+S_2+\cdots+S_t$. Hence we can $2\left\lceil \frac{k}{k-2t-1} \right\rceil$-color the subgraph induced by vertices $K+S_1+S_2+\cdots+S_t$.

□

Corollary 4.2. We can 4-color the subgraph induced by vertices $K+S_1+S_2+\cdots+S_t$, where $t = \left\lceil \frac{k-3}{4} \right\rceil$.

Corollary 4.3. We can $k+1$-color the subgraph induced by vertices $K+S_1+S_2+\cdots+S_t$, where $t = \left\lceil \frac{k-3}{2} \right\rceil$.

As Theorem 4.6 indicates, there is a trade off between number of colors used and the number of neighborhoods of the k-cycle colored. The two extremes are Corollaries 4.2 and 4.3, that is, using 4 colors to color the first $\left\lceil \frac{k-3}{4} \right\rceil$ neighborhoods of the k-cycle, and using $k+1$ colors to color the first $\left\lceil \frac{k-3}{2} \right\rceil$ neighborhoods of the k-cycle, respectively. In what follows, we will use the result of Corollary 4.2, because it gives us a better asymptotic bound on the number of colors used.

4.3.2. A Strategy

It is gratifying to know that the k-cycle plus its first $\left\lceil \frac{k-3}{4} \right\rceil$ neighborhoods can be 4-colored. Yet without additional structure and properties of the graph, we do not have a bound on the number of

vertices contained in the first $\left\lceil \frac{k-3}{4} \right\rceil$ neighborhoods. Hence it becomes necessary to devise a strategy to overcome this problem. The strategy we have designed is based on the following ideas:

1. For each neighborhood, say i-neighborhood, of the k-cycle, we establish a goal, called $goal_i$, which is the number of vertices we would like the i-neighborhood to contain.

2. If every neighborhood of the k-cycle achieves its goal, then the total number of vertices contained in the first $\left\lceil \frac{k-3}{4} \right\rceil$ neighborhoods plus those on the k-cycle is at least $k + \sum_{i=1}^{\left\lceil \frac{k-3}{4} \right\rceil} goal_i = Goal$. In this case, we say we have achieved "*Goal*". *Goal* serves as a lower bound on the number of vertices that can be 4-colored.

3. If one of the neighborhood of the k-cycle fail to attain its goal, say j-neighborhood is the first neighborhood starting from the k-cycle that failed, then we know we can 4-color the vertices on the k-cycle plus the vertices in the first $j-1$ neighborhoods. The number of vertices that can be 4-colored is at least $k + \sum_{i=1}^{j-1} goal_i$. In addition, the j-neighborhood, which contains less than $goal_i$ vertices, serves as a *separator*. The separator separates G into two *disjoint* graphs, G_1 and G_2, where $G_1 = K + \sum_{i=1}^{j-1} S_i$, and $G_2 = G - K - \sum_{i=1}^{j} S_i$. G_1 can be 4-colored. Furthermore, we can use the same 4 colors and the same strategy to color G_2 (i.e., we can recurse on G_2 using the same 4 colors).

4. In both cases above, we have found a subgraph that we can 4-color. Either we have achieved "*Goal*", that is, we have found a 4-colorable subgraph of size at least *Goal*, or we have achieved "$n - \sum|\text{separators}|$", that is, we have recursively found separators (never achieving *Goal* in each recursive call), and along the way we have found a 4-colorable subgraph of size at least $n - \sum|\text{separators}|$.

The parameters of the strategy are k and n, actually k relative to n. Based on k and n, we have to decide what values to assign $goal_i$, for i=1 to $\left\lceil \frac{k-3}{4} \right\rceil$. Note that if the values assigned to $goal_i$ are small, then *Goal* is small, which means that the 4-colorable subgraph of the first case is small. Yet if the values

i	$goal_i$	$c=1$	$1<c<2$	$c=2$	$c>2$
k-cycle	k	k	k	k	k
1	k	k	k	k	k
2	ck	k	ck	$2k$	ck
3	c^2k	k	c^2k	2^2k	c^2k
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
$\left\lceil \frac{k-3}{4} \right\rceil$	$c^{\left\lceil \frac{k-3}{4} \right\rceil - 1}k$	k	$c^{\left\lceil \frac{k-3}{4} \right\rceil - 1}k$	$2^{\left\lceil \frac{k-3}{4} \right\rceil - 1}k$	$c^{\left\lceil \frac{k-3}{4} \right\rceil - 1}k$
Goal	$k+\sum_{i=1}^{\left\lceil \frac{k-3}{4} \right\rceil} goal_i$	$k+k\left\lceil \frac{k-3}{4} \right\rceil$	$k+k\left\lceil \frac{c^{\left\lceil \frac{k-3}{4} \right\rceil}-1}{c-1} \right\rceil$	$k2^{\left\lceil \frac{k-3}{4} \right\rceil}$	$k+k\left\lceil \frac{c^{\left\lceil \frac{k-3}{4} \right\rceil}-1}{c-1} \right\rceil$
$\alpha^{\dagger}$ *vs* $\beta^{\dagger\dagger}$		$\lvert\alpha\rvert>\lvert\beta\rvert$	$\lvert\alpha\rvert>\lvert\beta\rvert$	$\lvert\alpha\rvert>\lvert\beta\rvert$	$\lvert\alpha\rvert>\frac{\lvert\beta\rvert}{c-1}$
α *vs* n	$\lvert\alpha\rvert+\lvert\beta\rvert=n$	$\lvert\alpha\rvert>\frac{n}{2}$	$\lvert\alpha\rvert>\frac{n}{2}$	$\lvert\alpha\rvert>\frac{n}{2}$	$\lvert\alpha\rvert>\frac{n}{c}$

† α – vertices in 4-colorable subgraph found by Algorithm 4.5 when all vertices have been placed in either α or β.

†† β – vertices in the separators of Algorithm 4.5 (all vertices not placed in α).

Table 4.1. An assignment of values to $goal_i$.

assigned to $goal_i$ are large, then the size of the separator is large, which means that the 4-colorable subgraph of the second case is small. Without additional structure and properties of the graph, we do not know whether each $goal_i$ can be achieved or not, which means that we do not know which of the two cases will hold. Hence in order to find a "large" 4-colorable subgraph, it is best to assign values to $goal_i$ such that the two cases are more or less balanced. Table 4.1 displays one possible assignment of values to $goal_i$. This assignment is selected because it simplifies the analysis (see Theorem 4.7). In Table 4.1, α

is a set containing vertices in those neighborhoods of the k-cycle that have achieved $goal_i$, and β is a set containing vertices in the separators. A more precise description for α and β is given in the next section.

4.3.3. An Algorithm that 4-Colors a Subgraph of G

Now we are ready to present an algorithm – Algorithm 4.5 which finds a subgraph of G that can be 4-colored based on the strategy described earlier and the assignment in Table 4.1. We declare two global sets, α and β, where α collects vertices that can be 4-colored and β collects vertices in the separators. Algorithm 4.5 halts either when all the neighborhoods (from 1 to $\left\lceil \frac{k-3}{4} \right\rceil$) of the k-cycle achieved their goals, which implies that α contains at least *Goal* (see Table 4.1) number of vertices, or when all vertices in G have been placed in either α or β ($|\alpha|+|\beta|=n$).

Input. A graph $G=(V,E)$, $k\geq 5$.

Output. A 4-coloring of a subgraph of G.

1. Find a smallest odd cycle, the *k-cycle*, in G.
   Let K and $S_1, S_2, \cdots, S_t$ be defined relative to the k-cycle in G, where $t=\left\lceil \frac{k-3}{4} \right\rceil$.
2. Add K to α, where α is a global set.
3. For $i = 1$ to $\left\lceil \frac{k-3}{4} \right\rceil$, do
   4. If $|S_i| \geq c^{i-1}k$, then add S_i to α.
   5. Else do
      6. If $|S_i|=0$, then exit Algorithm 4.5.
      7. Else add S_i to β, where β is a global set.
      8. Call Algorithm 4.5 on all connected components of $G-K-\sum_{j=1}^{i} S_j$.
      9. Exit Algorithm 4.5.
10. Place all the remaining vertices in β.
11. Exit Algorithm 4.5.

Algorithm 4.5. A 4-coloring algorithm of a subgraph of G.

Theorem 4.7. For $c \geq 2$ Algorithm 4.5 finds and 4-colors a subgraph of G of size at least $\min\{k+k\left\lfloor \frac{c^{\left\lfloor \frac{k-3}{4} \right\rfloor}-1}{c-1} \right\rfloor, \frac{n}{c}\}$.

Proof. If Algorithm 4.5 halts at Step 11 some time during a recursive call, which means that every neighborhood of the k-cycle achieved its goal in a particular call of Algorithm 4.5, then due to Corollary 4.2 and for $c \geq 2$, the size of the subgraph that can be 4-colored is at least

$$k+\sum_{i=1}^{\left\lfloor \frac{k-3}{4} \right\rfloor} |S_i| = \alpha \geq k+\sum_{i=1}^{\left\lfloor \frac{k-3}{4} \right\rfloor} goal_i = k+k\left\lfloor \frac{c^{\left\lfloor \frac{k-3}{4} \right\rfloor}-1}{c-1} \right\rfloor.$$

Otherwise the recursive Algorithm 4.5 halts because all vertices in G have been placed in either α or β. This means that during every call of Algorithm 4.5, a separator is found (except possibly in those calls of Algorithm 4.5 when all the vertices have been exhausted and yet *Goal* has not been achieved). Let Sp_j denote the separator found at the j^{th} call of Algorithm 4.5. Sp_j corresponds to some j_l-neighborhood, j_l between 1 and $\left\lfloor \frac{k-3}{4} \right\rfloor$, of the k-cycle of the j^{th} call of Algorithm 4.5. Since Sp_j is a separator, then $|Sp_j| < goal_{j_l}$. The size of the 4-colorable subgraph found at the j^{th} call of Algorithm 4.5 is at least

$$k+\sum_{i=1}^{j_l-1} goal_i \geq k+\sum_{i=1}^{j_l-1} c^{i-1}k = k+k\left\lfloor \frac{c^{j_l-1}-1}{c-1} \right\rfloor \geq \frac{c^{j_l-1}k}{c-1} = \frac{goal_{j_l}}{c-1} > \frac{|Sp_j|}{c-1}, \text{ for } c \geq 2.$$

The above inequality holds for all calls of Algorithm 4.5, thus

$$|\text{4-colorable subgraph found}| > \sum_j \frac{|Sp_j|}{c-1}, \text{ or } |\alpha| > \frac{|\beta|}{c-1}, \qquad \text{for } c \geq 2.$$

Since $|\alpha| + |\beta| = n$, therefore $|\alpha| > \frac{n}{c}$ for $c \geq 2$.

□

Corollary 4.4. If $c=2$ then Algorithm 4.5 finds and 4-colors a subgraph of G of size at least $\min\{k2^{\left\lfloor \frac{k-3}{4} \right\rfloor}, \frac{n}{2}\}$.

Note that for $c=1$, we can also use Algorithm 4.5 to 4-color a subgraph of G. However, the performance of Algorithm 4.5, in terms of the size of the 4-colorable subgraph, when $c=1$ is not as good as that

when c=2. To be more specific, when c=1, the size of *Goal* is smaller – $k+k\left\lfloor\frac{k-3}{4}\right\rfloor$. The size of the separator is also smaller, yet the best we can say about the size of β relative to that of α (the size of the separator relative to that of the 4-colorable subgraph) is that $|\beta|<|\alpha|$ (recall that we derived this same inequality when c=2). So in terms of analysis, when c=1 the size of the 4-colorable subgraph – $\min\{ k+k\left\lfloor\frac{k-3}{4}\right\rfloor, \frac{n}{2} \}$ found by Algorithm 4.5 is smaller than that when c=2. For values of c<1, we find that the size of *Goal* is too small to be of use.

Each call of Algorithm 4.5 takes time $O(\min\{e^{3/2}, n^{2.376}\})$. Note that during the course of Algorithm 4.5, the value of n decreases. The original value of n serves as an upper bound. The total running time of Algorithm 4.5 depends on the number of recursive calls Algorithm 4.5 makes. Clearly the parameters n, k, and c play an important role in the running time. We shall analyze the time complexity of Algorithm 4.5 later on specific cases.

4.3.4. An Approximate Graph Coloring Algorithm

Knowing how to find 4-colorable subgraphs of a graph leads us directly to a recursive algorithm for approximate graph coloring – Algorithm 4.6.

Input. A graph $G=(V,E)$, $k\geq 5$.

Output. A coloring of G.

1. Call Algorithm 4.5 on G.
   Say Algorithm 4.5 produces a 4-coloring of G', where G' is a subgraph of G.
   Note that G' and $G-G'$ contain vertices placed in α and β, respectively, that is, $G'=\alpha$, and $G-G'=\beta$ in Algorithm 4.5.
2. If $G-G'$ is empty, then exit Algorithm 4.6.
3. Else call Algorithm 4.6 on all connected components of $G-G'$.

Algorithm 4.6. An approximate graph coloring algorithm.

In Algorithm 4.5 if we let c=2 and if *Goal* = $k2^{\left\lfloor\frac{k-3}{4}\right\rfloor} \geq \frac{n}{2}$, then according to Corollary 4.4, we can find and 4-color a subgraph of G of size at least $\frac{n}{2}$. Thus for all k such that $k2^{\left\lfloor\frac{k-3}{4}\right\rfloor} \geq \frac{n}{2}$, each iteration

of Algorithm 4.6 finds a 4-colorable subgraph that is at least half of the graph (of that iteration). And this is actually the best Algorithm 4.6 can do; due to the *min* term in Corollary 4.4, Algorithm 4.6 can not guarantee that it finds a 4-colorable subgraph that is more than half of the graph in each iteration. However, being able to 4-color at least half of the graph in every iteration means that the number of iterations is at most lg$n$. Hence the number of colors used to color G is at most 4lg$n$. This result is stated formally in the next theorem.

Theorem 4.8. For all k such that $k\geq 5$ and $k2^{\left\lfloor \frac{k-3}{4} \right\rfloor} \geq \frac{n}{2}$, we can 4lg$n$-color G in time $O(\min\{e^{3/2}, n^{2.376}\}\frac{n\log n}{k})$.

Proof. We will use Algorithm 4.6 to color G. Let $c=2$ in Algorithm 4.5. Because of Fact 4.1, we can safely use the same k value all throughout Algorithm 4.6. Since $k2^{\left\lfloor \frac{k-3}{4} \right\rfloor} \geq \frac{n}{2}$, each iteration of Algorithm 4.6 4-colors at least half of the graph (of that iteration). Algorithm 4.6 starts with n vertices, thus the number of recursive calls it makes is at most lg$n$. Therefore the total number of colors Algorithm 4.6 uses is at most 4lg$n$.

Algorithm 4.6 calls Algorithm 4.5 $O(\log n)$ times. Each time Algorithm 4.5 is called by Algorithm 4.6, it makes $O(\frac{n}{k})$ number of recursive calls, because during each iteration of Algorithm 4.5 a 4-colorable subgraph of size at least k (i.e., the k-cycle) is added to α. Since each iteration of Algorithm 4.5 takes time $O(\min\{e^{3/2}, n^{2.376}\})$, the total running time of Algorithm 4.6 in this case is $O(\min\{e^{3/2}, n^{2.376}\}\frac{n\log n}{k})$.

□

After some simple arithmetic, it is easy to see that if $k \geq 4(\lg n - \lg\lg n)$, then $k2^{\left\lfloor \frac{k-3}{4} \right\rfloor} \geq \frac{n}{2}$, for $n\geq 16$. Thus we arrive at the following corollary.

Corollary 4.5. For all $k\geq 5$ and $k \geq 4(\lg n - \lg\lg n)$, we can 4lg$n$-color G in time $O(\min\{e^{3/2}, n^{2.376}\}\frac{n\log n}{k})$.

Algorithm 4.6 is a general approximate graph coloring algorithm that can be used for all graphs with $k \geq 5$. When k does not satisfy the condition in Theorem 4.8 (roughly when $k < \lg n$) and if we use Algorithm 4.6 to color G with c=2 in Algorithm 4.5, then the strategy of Algorithm 4.5 says that we will achieve *Goal* in every iteration of Algorithm 4.6 until some point when the number of vertices in G falls below 2·*Goal*. The number of times we achieve *Goal* depends on the value of k. Note that we can adjust the parameter c in Algorithm 4.5 to alter the size of *Goal*. In general, the smaller the value k is, the smaller the value *Goal* is, the easier it is to achieve *Goal*, the longer Algorithm 4.6 runs (because *Goal* is achieved more often), hence the larger the number of colors required to color G. This is an expected result because when k is small, the bipartite neighborhood of the k-cycle is small. Although we can adjust (increase) the value of *Goal* ($goal_i$), the trade-off between the size of *Goal* and the size of the separators comes into play. For simplicity, when using Algorithm 4.6 to color graphs with "small" k, we will still choose c=2 in Algorithm 4.5. In doing so, we derive an upper bound on the number of colors needed to color such a graph. Theorem 4.9 presents the performance results of Algorithm 4.6 on graphs with "small" k.

Theorem 4.9. For all k such that $k \geq 5$ and $k2^{\left\lfloor \frac{k-3}{4} \right\rfloor} < \frac{n}{2}$, we can $2\lg Goal \cdot \frac{n}{Goal}$-color G in time $O(\min\{e^{3/2}, n^{2.376}\}\frac{n^2 \log Goal}{kGoal})$, where $Goal = k2^{\left\lfloor \frac{k-3}{4} \right\rfloor}$.

Proof. We will use Algorithm 4.6 to color G. Let c=2 in Algorithm 4.5. Because of Fact 4.1, we can safely use the same k value all throughout Algorithm 4.6. Since $Goal = k2^{\left\lfloor \frac{k-3}{4} \right\rfloor} < \frac{n}{2}$, then according to Corollary 4.4, Algorithm 4.6 4-colors at least *Goal* number of vertices in each iteration until the first time when the size of every connected component falls below 2·*Goal*. From that point on, the condition in Theorem 4.8 holds for each connected component, hence the number of colors Algorithm 4.6 uses is at most $4\lg Goal$ (for each connected component). In the worst case, a graph gets decomposed into connected components of size less than 2·*Goal* at the beginning, thus the number of colors used by Algorithm 4.6 is at most $2\lg Goal \cdot \frac{n}{Goal}$.

Algorithm 4.6 calls Algorithm 4.5 $O(\log Goal \cdot \frac{n}{Goal})$ times. Each call of Algorithm 4.5 by Algo-

rithm 4.6 takes time $O(\min\{e^{3/2}, m^{2.376}\}\frac{m}{k})$, where m is the number of vertices of the graph that is passed from Algorithm 4.6 to Algorithm 4.5. To simplify the running time expression, we will use the original n value (which bounds all subsequent n values from above). Hence the total running time of Algorithm 4.6 in this case is $O(\min\{e^{3/2}, n^{2.376}\}\frac{n^2 \log Goal}{kGoal})$.

□

4.4. Performance Analysis

In this chapter we have presented two approximate graph coloring algorithms – Algorithm 4.4 (including Algorithms 4.1, 4.2, 4.3) and Algorithm 4.6. As we have seen, the performance, in terms of number of colors used, of these two algorithms depends on k – the size of a smallest odd cycle of G. Algorithm 4.4 performs well when k is large relative to n, that is, Algorithm 4.4 $6(i/2)^{1/2}$-colors G when $k > \left\lceil \frac{n}{i} \right\rceil$. Algorithm 4.6, on the other hand, performs its best when k is about $4\lg n$, in which case it uses $4\lg n$ colors. Note that Algorithm 4.6 uses at least $4\lg n$ colors, regardless how large k is. Chart 4.1 displays the performance of each algorithm over various ranges of k. Observe that Algorithm 4.4 out performs Algorithm 4.6 when $k \geq \left\lceil \frac{n}{\frac{4}{3}\lg^2 n} \right\rceil$ and when $k \leq 2\lg n$, while Algorithm 4.6 takes over in performance when $\left\lceil \frac{n}{\frac{4}{3}\lg^2 n} \right\rceil > k > 2\lg n$. Because of the difference in performance strength of these two algorithms over various ranges of k, combining the two algorithms to produce a hybrid algorithm, which is tuned depending on the relative values of k and n, is certainly a good idea and should be pursued in practice. Furthermore, both of these algorithms are recursive, hence the size of k relative to n increases as the algorithms progress. Thus adjusting parameters, such as c in Algorithm 4.5, may generate a large gain in performance and is worthwhile investigating further.

We have analyzed the performance of Algorithms 4.4 and 4.6 in absolute terms, i.e., number of colors used. It is desirable to compare the performance of these algorithms with the chromatic numbers of the graphs under consideration. In Chapters 3 and 4, we have been dealing with graphs with only "large" odd cycles, that is, graphs with large *odd girth*, where *girth* is the size of a smallest cycle in a

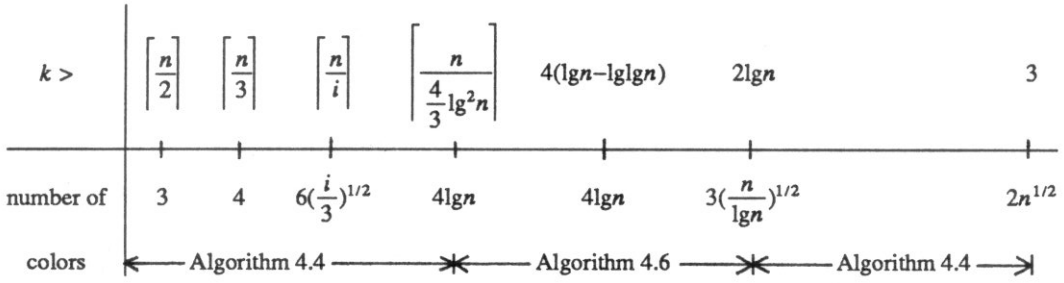

Chart 4.1. Performance of Algorithms 4.4 and 4.6 over various ranges of k.

graph. Notice that sometimes "large" just means that $k \geq 5$. One may wonder whether there is a relationship between the odd girth or girth of a graph and its chromatic number. This is an interesting question and has been considered by many graph theorists. We shall go into some history and background on this problem.

Descartes[1947,1948, 1954] first showed that for all p, there exist p-chromatic triangle-free graphs. Later Mycielski[1955], Zykov[1949], and Kelly and Kelly[1954] also proved the same theorem. The constructions of Mycielski, Zykov, and also Schäuble[1967] all contain 4-cycles. The graphs constructed by Descartes and Kelly and Kelly, on the other hand, have girth of size 6. By a more complicated procedure, Nesetril[1966] constructed p-chromatic graphs of girth at least 8. The generalized form of the theorem is proved by Erdös[1959] (see also Erdös[1961]), who showed, using probabilistic nonconstructive methods, that for all g and p, there exist p-chromatic graphs of girth at least g. A constructive proof was given by Lovász[1968] and simplified by Nešetřil and Rödl[1979]. Indeed, Bollobás and Sauer[1976] and Müller[1979] showed that for all g and p, there exists a uniquely p-colorable graph whose girth is at least g. More recently, Lubotzky, Phillips, and Sarnak[1986, 1988] gave an explicit construction of a family of d-regular graphs, called *Ramanujan* graphs, which have girth at least $4/3\log_{d-1} n$ and chromatic number at least $n^{1/3h}$, where $h \leq$ girth. These bounds actually improve those derived by Erdös using probabilistic methods.

For results on odd girth, Gallai[1973] constructed a 4-chromatic graph that has odd girth at least $n^{1/2}$. Erdös and Gallai (see Erdös[1962]) then conjectured that for every p there is a $(p+2)$-chromatic

graph with odd girth of size at least $n^{1/p}$. Their conjecture is proved by Lovasz[1978]. *Kneser* graphs [Kneser, 1955] are examples of explicit constructions of such graphs (this was shown by Lovasz[1978] and Barany[1978]). As a matter of fact, for each $p \geq 2$, there exist infinitely many $(p+2)$-chromatic graphs whose odd girth is at least $c_p n^{1/p}$, fo some constant c_p. These bounds on the chromatic number and odd girth are actually the best possible, as proved by Kierstead, Szemerédi, and Trotter[1984] (unfortunately, their proof is non-constructive). They showed that if G is a graph on n vertices with odd girth of size at least $4pn^{1/p}$, then the chromatic number of G is at most $p+1$. Although the odd girth of a Kneser graph is large, the girth (or even girth) is small. Another family of graphs that can be simply constructed and contains large odd girth is *Borsuk* graphs. Lovasz[1983] proved that for each $p \geq 2$, the chromatic number of a Borsuk graph is $p+2$ and its odd girth is at least $cn^{1/p}$, for some constant c that depends on p. These bounds are similar to those of Kneser graphs. Borsuk graphs also contain small even cycles.

The results described above show that graphs with large girth, as well as large odd girth, exist and can be explicitly constructed. In addition, graphs with large odd girth are not easy to color, since their chromatic number may be large.

Now we shall review some graph-theoretic bounds on triangle-free graphs. Let $n(p)$ denote the smallest possible number of vertices in a p-chromatic triangle-free graph. Erdös[1961], using probabilistic arguments, showed that for all p, $n(p) \leq c_1 \cdot p^2 \cdot (\log p)^2$, for some constant c_1. For a lower bound on $n(p)$, Erdös and Hajnal[1985] showed that $c_2 p^2 \log p \leq n(p)$, for some constant c_2. Thus we have $c_2 p^2 \log p \leq n(p) \leq c_1 \cdot p^2 \cdot (\log p)^2$, and the bounds are almost tight.

Knowing the tight bounds on the chromatic number of a graph in relationship with the size of the graph and its odd girth, we shall proceed to evaluate the performance of Algorithms 4.4 and 4.6. We begin with Algorithm 4.4. If G is triangle-free, then we can $O(\sqrt{n})$-color G using Algorithm 4.4. From the result of Erdös[1961], we know that there exist triangle-free graphs with chromatic number that is just slightly less than $O(\sqrt{n})$. This shows that Algorithm 4.4 performs well. In fact, because of the existence of graphs satisfying the bound derived by Erdös, Algorithm 4.4 can not perform much better. Next we move on to Algorithm 4.6. Suppose that k is of the form $cn^{1/p}$, k = odd girth, for some constant c. If $c \geq 4p$, then a result of Kierstead, Szemerédi and Trotter[1984] indicates that G is $p+1$-colorable. Theorem 4.8 says that we can $4 \lg n$-color G if $k 2^{\left\lfloor \frac{k-3}{4} \right\rfloor} \geq \frac{n}{2}$. For all sufficiently large k, the condition in Theorem

4.8 is satisfied, i.e., $cn^{1/p}2^{\left\lfloor \frac{cn^{1/p}-3}{4} \right\rfloor} \geq \frac{n}{2}$, hence we can 4lg$n$-color G. However, there exist smaller values of k for which the condition in Theorem 4.8 is not satisfied. So the above analysis is inconclusive. Basically if k is of the form $cn^{1/p}$, then there exist values of k which satisfy Theorem 4.8 as well as values of k which do not satisfy Theorem 4.8. Hence the graph-theoretic bounds that we have do not help in evaluating the performance of Algorithm 4.6, that is, the graph-theoretic bounds that we have do not characterize completely the relationship between the odd girth, the chromatic number, and the size of the graph. More study in this area is needed.

As we have mentioned in Chapter 1, Berger and Rompel have designed an approximate graph coloring algorithm with a performance guarantee of $O(n(\log\log n)^3/(\log n)^3)$, which is the best bound thus far. We shall now compare the performance of Berger and Rompel's algorithm with that of Algorithms 4.4 and 4.6. For triangle-free graphs, Algorithm 4.4 uses $O(\sqrt{n})$ colors. Thus as long as $k \geq 5$, the performance guarantee of Algorithm 4.4 is $O(\sqrt{n})$, regardless of the size of k and the chromatic number. When $k > 4(\lg n - \lg\lg n)$, Algorithm 4.6 can 4lg$n$-color G, so in this case Algorithm 4.6 achieves a performance guarantee of $O(\log n)$, regardless of the chromatic number. Both $O(\log n)$ and $O(\sqrt{n})$ are significant improvements over $O(n(\log\log n)^3/(\log n)^3)$. For 3-chromatic graphs, Berger and Rompel's algorithm achieves a performance guarantee of $O((n/\log n)^{1/2})$. We know from the work of Kierstead, Szemerédi and Trotter that for all $k > 8n^{1/2}$, G is 3-colorable. Thus for all $n > 3$ and $k > 8n^{1/2}$ (which implies that $k \geq 5$), the performance guarantee of Algorithm 4.4 is $\frac{6}{3}(\frac{i}{3})^{1/2} = \frac{n^{1/4}}{\sqrt{6}} = O(n^{1/4})$. If $k > 8n^{1/2}$, then $k > 4(\lg n - \lg\lg n)$ for all sufficiently large k. Hence we can apply Algorithm 4.6 on those k that are sufficiently large and obtain a performance guarantee of $O(\log n)$. Once again, Algorithms 4.4 and 4.6 out perform Berger and Rompel's algorithm. We conclude that for all triangle-free graphs, as well as for those graphs with "large" odd girth, Algorithms 4.4 and 4.6 give the best performance guarantee among all approximate graph coloring algorithms.

Chapter 5

Coloring Graphs with only "Small" Odd Cycles

5.1. Introduction

The role odd cycles plays in connection with the chromatic number of a graph is interesting. First of all we know that the presence of odd cycles in a graph characterizes the bipartiteness of a graph (or lack thereof). It is unfortunate that we know of no similar characterizations for p-chromatic graphs, for $p \geq 3$. Nonetheless, we wonder if the presence or absence of different sizes of odd cycles affects the chromatic number of a graph in any way. In Chapter 4, we have shown some graph-theoretic bounds on the size of a smallest odd cycle and the chromatic number of a graph. Algorithmically, it is the absence of small odd cycles that enables us to find large bipartite subgraphs which in turn give rise to better results for graph coloring. In this chapter, we shall concentrate on the other end of the spectrum – graphs without large odd cycles. We shall show that graphs without large odd cycles possess many special structures. And it is these special structures that enable us to develop efficient and optimal (or near-optimal) graph coloring algorithms.

A theme that is present throughout this chapter is *structural analysis*. For certain classes of graphs, we show that we are able to characterize the structures of graphs completely and color the graphs optimally. For others, we use breadth-first-search to decompose graphs and find bipartite subgraphs. As we will demonstrate, breadth-first-search is an important and powerful tool for exploiting and analyzing the structures of graphs. All the algorithms presented in this chapter are either optimal or near-optimal in terms of the number of colors used, and efficient – $O(e)$ in complexity. Table 5.1 summarizes the graph coloring results of this chapter.

G is biconnected, non-bipartite, and contains:	$\chi(G)$	Number of colors used
No odd cycles other than 3-cycles, and		
No K_4	3	3
K_4	4	4
No odd cycles other than 5-cycles	3	3
No odd cycles other than k-cycles, $k>5$	3 or 4	4
No odd cycles other than 3-cycles & 5-cycles, and		
No K_4, K_5, K_6	3 or 4	4
K_4 but no K_5, K_6	4	4
K_5 but no K_6	5	5
K_6	6	6

Table 5.1. Graphs with various special structures, their chromatic numbers, and how well we can color them using algorithms developed in this chapter.

The rest of this chapter is organized as follows. In Section 5.2, we provide some background and graph-theoretic bounds on cycle sizes and the chromatic number for graphs with only small odd cycles. In Section 5.3, we focus on graphs with only one odd cycle size – k, for $k = 3$, $k = 5$, and $k > 5$. In Section 5.4, we move on to graphs with only 3-cycles and 5-cycles in odd cycles. In Section 5.5, we pose some open problems.

5.2. Odd Cycles, Even Cycles, and the Chromatic Number

In studying graphs with only small odd cycles, we are interested in knowing whether the size of a largest odd cycle determines or bounds the size of the chromatic number. This is a curious question. On the one hand, a $2p$-chromatic graph, as exemplified by K_{2p} – a complete graph on $2p$ vertices, need not contain odd cycles of size $2p+1$ or longer. On the other hand, Erdös and Hajnal[1966] proved that every $(2p+1)$-chromatic graph must contain an odd cycle of size at least $2p+1$. This is an important result and we state it as a theorem.

Theorem 5.1. [Erdös and Hajnal, 1966] If G does not contain odd cycles of size more than $2p-1$ ($p \geq 1$), then G is $2p$-colorable, that is, $\chi(G) \leq 2p$.

Hence the size of the largest odd cycle of a graph provides an upper bound for the chromatic number of the graph. Erdös and Hajnal's result is tight. Unfortunately their proof is non-constructive.

Throughout this study, we have focused on odd cycles in graphs. More specifically, we have restricted attention to special classes of graphs – graphs with only "large" odd cycles and graphs with only "small" odd cycles. One may wonder about the sizes of even cycles in these special classes of graphs. Does limiting the sizes of odd cycles affect the sizes of even cycles of a graph? In the first class of graphs, the answer is negative; a graph with only large odd cycles can have arbitrary small even cycles, as we have seen in Chapter 4. In the second class, however, the answer is affirmative. It has been observed that if G is biconnected, non-bipartite ($\chi(G)>3$), and does not contain large odd cycles, then G does not contain large even cycles. Lemma 5.1 describes this result more precisely. Because Lemma 5.1 is used later in the chapter, we have reproduced the proof.

Lemma 5.1. In a biconnected, non-bipartite graph G, if there exists an even cycle of size $2m$ in G, then there exists an odd cycle of size at least $m+1$ in G.

Proof. Let $C_{2m} = v_1 - v_2 - \cdots - v_{2m} - v_1$ be a simple even cycle of size $2m$ in G. Since G is non-bipartite, then there exists a simple odd cycle C_1 in G. We shall assume that the size of C_1 is less than $m+1$.

If C_1 and C_{2m} edge-overlap, then $C_1 \otimes C_{2m}$ (the exclusive-or of C_1 and C_{2m}) is either a simple odd cycle or a collection of simple cycles. If $C_1 \otimes C_{2m}$ is a simple odd cycle, then the size of $C_1 \otimes C_{2m}$ is at least $m+1$, because the size of C_{2m} is $2m$ and we assumed that the size of C_1 is less than $m+1$. If $C_1 \otimes C_{2m}$ is not a simple cycle, then it contains at least one simple odd cycle. Without loss of generality, let C_2 be a simple odd cycle in $C_1 \otimes C_{2m}$. Let D be a set containing edges that are on C_1 but not on C_{2m}. Since C_1 and C_{2m} edge-overlap, then $|D| < |C_1|$. Because $C_1 \otimes C_{2m}$ contains two or more simple cycles, and for each simple cycle C in $C_1 \otimes C_{2m}$, $C \cap D$ is non-empty, hence the size of $C_2 \cap D$ is less than the size of D. Note that C_2 edge-overlaps C_{2m}, because $C_2 \subset C_1 \otimes C_{2m}$ and both C_1 and C_{2m} are simple cycles. Now let us consider $C_2 \otimes C_{2m}$. Again $C_2 \otimes C_{2m}$ is either a simple odd cycle or a collection of simple cycles. If $C_2 \otimes C_{2m}$ is a simple odd cycle, then using the same argument as above, either C_2 or $C_2 \otimes C_{2m}$ or both is a simple odd cycle of size at least $m+1$. If $C_2 \otimes C_{2m}$ is not a simple cycle,

then let C_3 be a simple odd cycle in $C_2 \otimes C_{2m}$. Because $C_2 \otimes C_{2m}$ contains two or more simple cycles, the size of $C_3 \cap D$ is less than the size of $C_2 \cap D$. Note that C_3 edge-overlaps C_{2m}, because $C_3 \subset C_2 \otimes C_{2m}$ and both C_2 and C_{2m} are simple cycles. Next we consider $C_3 \otimes C_{2m}$. Basically we can repeat the same process until we find a cycle C_i, where C_i is a simple odd cycle in $C_{i-1} \otimes C_{2m}$ such that C_i and $C_i \otimes C_{2m}$ are both simple odd cycles. Note that C_i exists, because the size of $C_j \cap D$ is less than the size of $C_{j-1} \cap D$, for all $j \geq 1$. At least one of C_i, $C_i \otimes C_{2m}$ is a simple odd cycle of size at least $m+1$. Furthermore, since C_i edge-overlaps C_{2m}, and both C_i and $C_i \otimes C_{2m}$ are simple odd cycles, then every edge of C_{2m} is on some simple odd cycle (either on C_i or on $C_i \otimes C_{2m}$).

If C_1 and C_{2m} are edge-disjoint, then according to Theorem 2.2, there exists a simple cycle C_a that contains an edge e_1 of C_1 and an edge e_2 of C_{2m}. If C_a is odd, then using the preceding line of reasoning, there exists a simple odd cycle of size at least $m+1$, because C_a edge-overlaps C_{2m}. If C_a is even, then every edge of C_a is on some simple odd cycle, because C_1 edge-overlaps C_a, using the observation in the above paragraph with C_a replacing C_{2m}. More specifically, edge e_2 is on a simple odd cycle, say C_b. C_b edge-overlaps C_{2m}, thus again using the preceding line of reasoning, there exists a simple odd cycle of size at least $m+1$.

□

In the proof of Lemma 5.1, we have actually proved another useful result, and it is stated as Lemma 5.2.

Lemma 5.2. In a biconnected, non-bipartite graph G, every edge of G lies on a simple odd cycle.

Proof. Since G is biconnected, every edge in G lies on a cycle. Let e be an edge under consideration. Suppose that e lies on a simple even cycle C_e. Since G is non-bipartite, there exists a simple odd cycle C_o in G. We see that edge e of C_e lies on some simple odd cycle by substituting C_o for C_1, C_e for C_{2m}, and substituting edge e for e_2, if necessary, in the proof of Lemma 5.1.

□

A direct consequence of Lemma 5.1 on the structure of a breadth-first search graph of a biconnected, non-bipartite graph G is captured in the following lemma.

Lemma 5.3. In a biconnected, non-bipartite graph G, if G contains no odd cycles of size greater than m, then the maximum level of any breadth-first search graph of G is at most $m-1$. Moreover, there exist no

cross edges at level $m-1$.

Remark. When counting the number of levels of a breadth-first search graph, we do not count the root as a level. Hence a graph consisting of a single vertex has 0 level.

Proof. Since G is biconnected, non-bipartite, and contains no odd cycles of size greater than m, then according to Lemma 5.1 the maximum cycle size in G is at most $2m-2$. Suppose that a breadth-first search graph of G, say $BFS(G_r)$, contains more than $m-1$ levels. From Theorem 2.2 we know that any vertex on level m of $BFS(G_r)$ and the root r lie on a common simple cycle. The size of this common simple cycle is at least $2m$ (Fact 2.2). Contradiction.

Similarly, if there exists a cross edge e at level $m-1$ of some breadth-first search graph of G, say $BFS(G_r)$, then according to Lemma 5.1, cross edge e and the root r lie on a common simple cycle. The size of this common simple cycle is at least $2m-1$ (Fact 2.3). Contradiction.

□

5.3. Odd Cycles = k-Cycles

In studying graphs with only small odd cycles, we shall first focus on graphs with only one odd cycle size – k, i.e., graphs with no odd cycles other than k-cycles. We show that the restriction of having only one odd cycle size plus the property that graphs are biconnected give rise to tightly structured graphs. We start with $k = 3$ and show that we can completely characterize and optimally color such graphs. We then move on to $k = 5$ and use breadth-first-search to decompose the graph into a bipartite portion and an independent set portion. Thus an optimal 3-coloring of the graph follows immediately. Lastly we give a simple approach for 4-coloring graphs with no cycles other than k-cycles, for $k>5$.

5.3.1. $k = 3$

As expected, $k = 3$ is the simplest case among all the values of k. Let G be a biconnected, non-bipartite graph containing no odd cycles other than 3-cycles. First we prove a lemma that illustrates the interactions among 3-cycles in G.

Lemma 5.4. In a biconnected, non-bipartite graph G, if G contains no odd cycles other than 3-cycles, then G does not contain 2 edge-disjoint 3-cycles.

Proof. Suppose that G contains 2 edge-disjoint 3-cycles – C_1: v_1–v_2–v_3–v_1 and C_2: v_a–v_b–v_c–v_a. According to Theorem 2.2, edges (v_1,v_2) and (v_a,v_b) lie on a common simple cycle, say C_3. The size of C_3 is either 3 or 4 by Lemma 5.1. If the size of C_3 is 3, then cycles C_1 and C_2 have a common vertex. Without loss of generality, we assume that $v_2 = v_b$. Thus the edge on C_3 that connects C_1 and C_2 must be (v_1,v_a). But the cycle: v_1–v_a–v_c–v_b–v_3–v_1 is a 5-cycle. This means that the size of C_3 must be 4 and cycles C_1 and C_2 have no common vertices. But the cycle $C_1 \otimes C_3$ is a 5-cycle. Contradiction.

□

Lemma 5.4 indicates that graphs with only 3-cycles in odd cycles are tightly structured. In fact, we will show in the proof of Theorem 5.2 that only 2 families of graphs are possible. Being able to characterize the structures of graphs enables us to color the graphs optimally.

Theorem 5.2. If G is biconnected, non-bipartite, and contains no odd cycles other than 3-cycles, then we can optimally color G in $O(e)$ time.

Proof. First we show that if G contains a 4 clique, then G is exactly K_4. Let K_4 (v_1,v_2,v_3,v_4) be a 4 clique in G. Let v_a be a vertex in G but not in K_4, and let (v_a,v_b) be an edge incident to v_a, for some vertex v_b in G. From Lemma 5.2, we know that every edge in G lies on a 3-cycle. Hence (v_a,v_b) lies on some 3-cycle $C = v_a$–v_b–v_c–v_a, for some vertex v_c in G. According to Lemma 5.4, C shares an edge with every 3-cycle in K_4. But this is not possible for any combination of v_b, v_c. Thus G contains exactly 4 vertices and $G = K_4$.

Next we show that if G is not a 4 clique, then G must belong to the family of graphs illustrated in Figure 5.1. First of all, we know that G contains a 3-cycle, because G is non-bipartite. Second, every edge of G lies on a 3-cycle (Lemma 5.2). Third, since G does not contain a 4 clique, then due to Lemma 5.4 all 3-cycles in G must share the same edge. Thus every vertex not on the shared edge is on some 3-cycle containing the shared edge. Fourth, if we delete the 2 vertices associated with the shared edge from the graph, then all the vertices remaining in the graph are independent. For if an edge exists in the remaining graph, then that edge must lie on a 3-cycle, and the 3-cycle must contain the shared edge, which is not possible (since there are 4 distinct vertices).

Now we have established that G is either K_4 or a member of the family of graphs shown in Figure 5.1. In the first case, the chromatic number of G is 4; in the second case, it is 3. In either case, it is easy

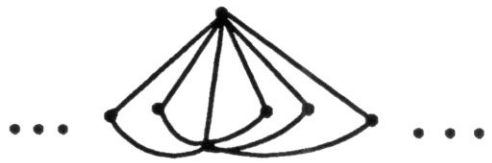

Figure 5.1. A family of graphs containing only 3-cycles in odd cycles and no K_4.

to see that G can be optimally colored in $O(e)$ time.

□

Due to the restricted structures of graphs satisfying the properties of Theorem 5.2, we can recognize such graphs in $O(e)$ time (using breadth-first-search, for example). Observe that all vertices except the two on the shared edge in Figure 5.1 have degree 2. Hence a biconnected, non-bipartite graph of degree at least 3 and containing no odd cycles other than 3-cycles is exactly K_4. Thus one way of optimally coloring a graph satisfying the properties of Theorem 5.2 is: First, iteratively delete vertices of degree less than 3 from G. After deletion, if the graph is non-empty, then it must be K_4, and we can 4-color it. By Lemma 2.1, all the vertices deleted can be 3-colored, and this can be accomplished in time $O(e)$.

5.3.2. $k = 5$

The structures of graphs become more complicated as we move from $k = 3$ to $k = 5$. When k increases, the number of levels in a breadth-first search graph of G increases. The interactions between vertices on adjacent levels of a breadth-first search graph are more involved. And we have more cases to consider. We shall use breadth-first search to strip away complicated structures, as demonstrated in Lemma 5.5.

Lemma 5.5. In a biconnected, non-bipartite graph G, if G contains no odd cycles other than k-cycles, $k>3$, then every level of a breadth-first search graph of G is 2-colorable (bipartite or independent).

Proof. Let $BFS(G_r)$ be some breadth-first search graph of G. Suppose that level p of $BFS(G_r)$ contains an odd cycle, which must be a k-cycle (since k is the only odd size cycle in G). Without loss of

generality, we shall label the vertices on the k-cycle $v_0, v_1, \cdots, v_{k-1}$. Let us consider two vertices v_a and v_{a+2} (we really mean $v_{a+2(mod\ k)}$) on the k-cycle, where $a \in \{0,1,\cdots,k-1\}$. v_a and v_{a+2} must have exactly one and the same parent in $BFS(G_r)$, otherwise there exists a common-ancestor w of v_a and v_{a+2} distance 2 or more away such that the simple cycle that consists of {path from v_a to w + path from w to v_{a+2} + $v_{a+2}-v_{a+3}-\cdots-v_a$} is odd and has size greater than k. The above statement applies to all v_a and v_{a+2}, $a \in \{0,1,\cdots,k-1\}$. Therefore all the vertices on the k-cycle must have the exact same parent, which means that there exist 3-cycles in G. But $k>3$. Contradiction.

□

Notice that Lemma 5.5 does not hold when $k = 3$, since if $G = K_4$, then G contains a 3-cycle on level 1 of a breadth-first-search graph of G.

Our approach for 3-coloring a graph G with only 5-cycles in odd cycles is to analyze the structures of G using breadth-first-search and the results of Lemmas 5.3 and 5.5. To simplify structural analysis, we shall deal with graphs of degree at least 3 (i.e., we first iteratively delete vertices of degree less than 3 and use Lemma 2.1). After considering various structures of $BFS(G_r)$, we show that $BFS(G_r)$ consists of a bipartite portion and an independent set portion. Hence a 3-coloring of G follows immediately.

Theorem 5.3. If G is a biconnected, non-bipartite graph of degree at least 3 and contains no odd cycles other than 5-cycles, then we can 3-colors G in $O(e)$ time.

Proof. According to Lemma 5.3, $BFS(G_r)$ has at most 4 levels, and all the vertices on level 4 are independent (since there are no cross edges). All the vertices on level 1 of $BFS(G_r)$ are independent because there are no 3-cycles in G. Furthermore, vertices on levels 1 and 4 are independent from each other (Fact 2.4).

Next we show that vertices on levels 2 and 3 of $BFS(G_r)$ induce a bipartite subgraph. Hence to 3-color G, we use 2 colors for vertices on levels 2 and 3 of $BFS(G_r)$ plus the root, and a third color for vertices on levels 1 and 4. According to Lemma 5.5, each level of $BFS(G_r)$ is 2-colorable. Thus if an odd cycle exists on levels 2 and 3 of $BFS(G_r)$, then it must be a 5-cycle with vertices on both levels 2 and 3. There are 6 possible structures for a 5-cycle on levels 2 and 3, as illustrated in Figure 5.2. We shall consider each structure individually, and show that none exists.

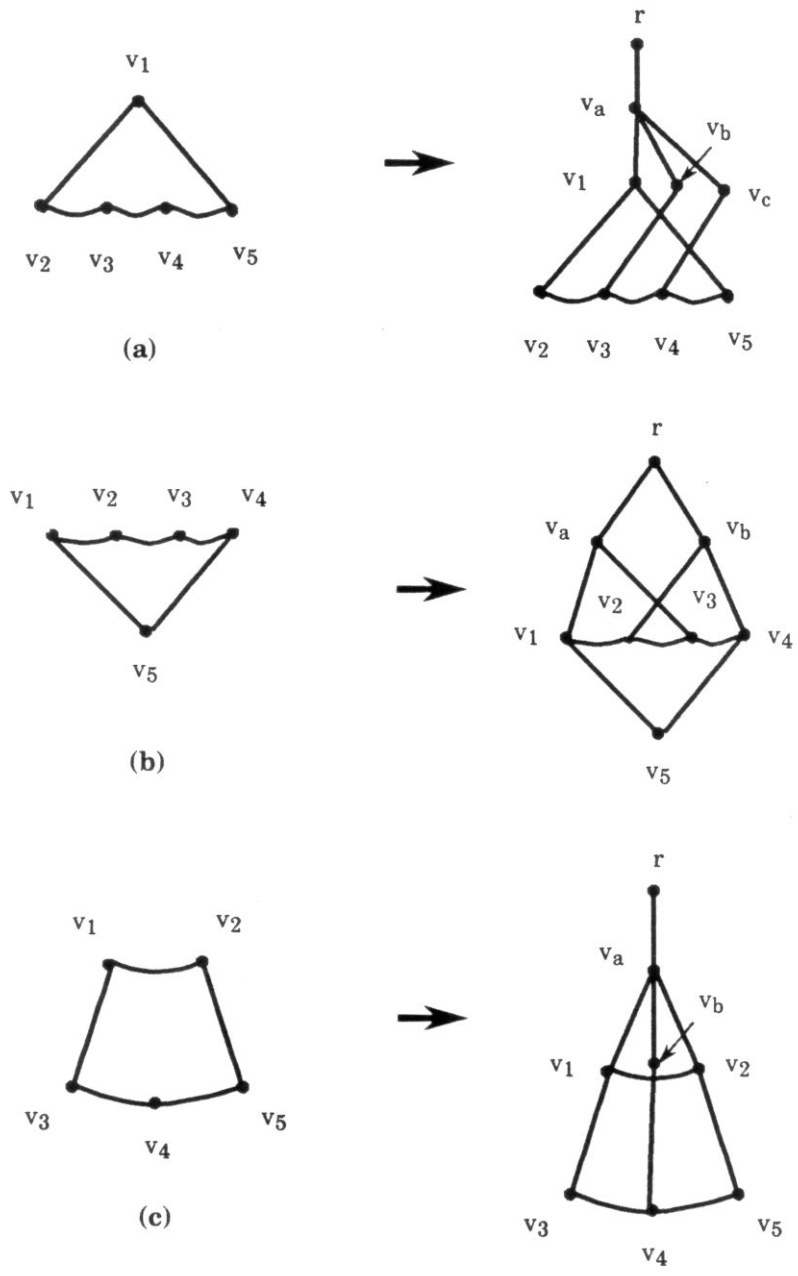

Figure 5.2. Possible structures for a 5-cycle on levels 2 and 3 of a $BFS(G_r)$ in Theorem 5.3.

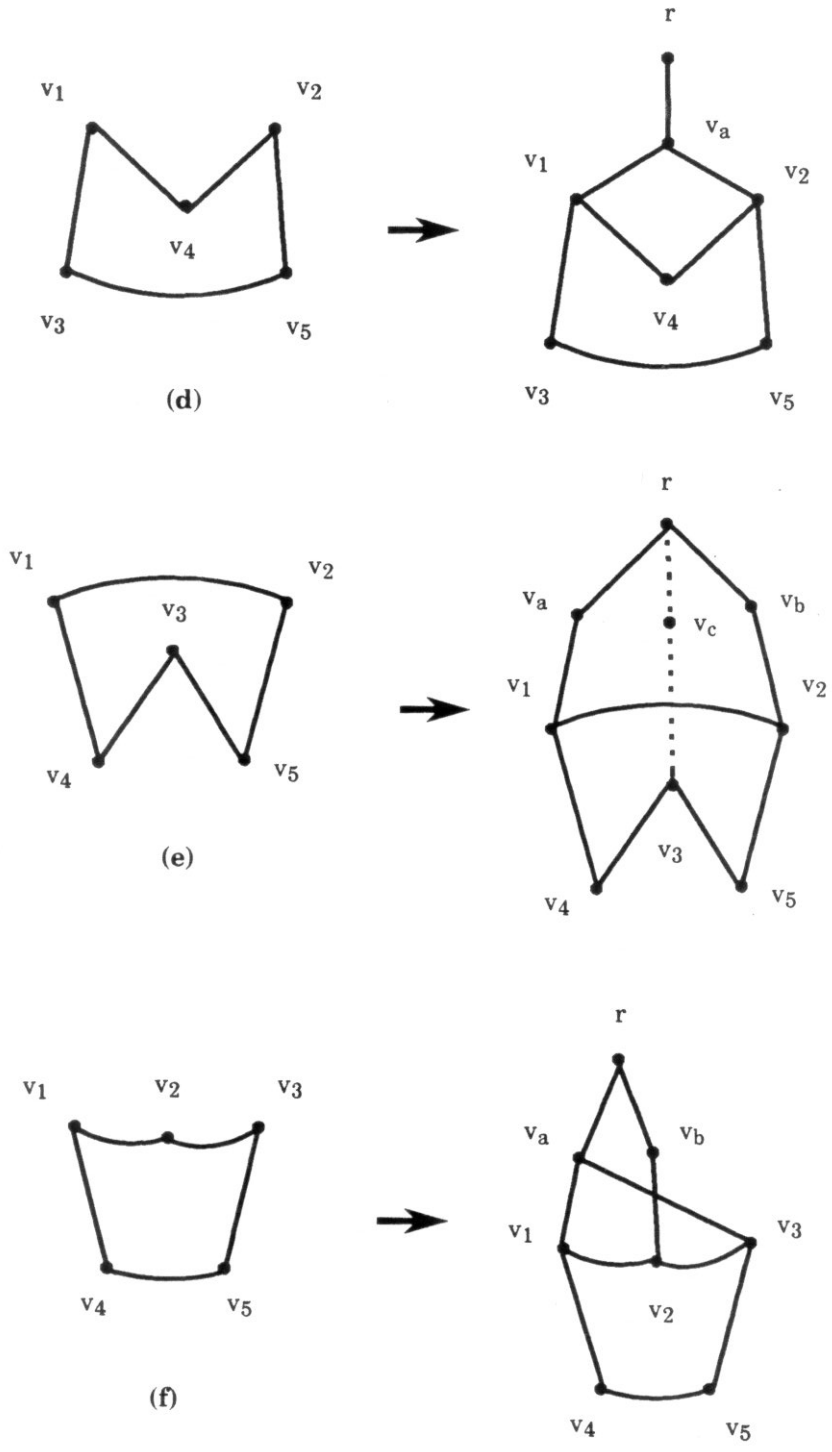

Figure 5.2. Possible structures for a 5-cycle on levels 2 and 3 of a $BFS(G_r)$ in Theorem 5.3.

Structure (a). Since G does not contain 3-cycles, the parents of v_3 and v_4 must be distinct and not equal to v_1. Without loss of generality, let v_b, v_c be the parent of v_3, v_4, respectively, and let v_a be a parent of v_1. The parent of v_a is the root r. v_a must be the unique ancestor of v_2 and v_3 on level 1, otherwise we can easily construct a 7-cycle through the root. Likewise, v_a is the unique ancestor of v_3 and v_4, and of v_4 and v_5 on level 1. Hence v_a is the only grand-parent of v_2, v_3, v_4, v_5. Note that v_1 is the only parent of v_2 and v_5, otherwise we obtain an odd cycle of size at least 7 from the following construction. Let v_p be a parent of v_2 (note that structure (a) is symmetrical with respect to v_2 and v_5, hence results for v_2 also hold for v_5). Then the path from v_p to v_1 ($v_p \rightarrow v_1$) via their nearest-common-ancestor is even of length at least 2. But the cycle: $v_p \rightarrow v_1 - v_5 - v_4 - v_3 - v_2 - v_p$ is odd of size at least 7.

Since G is biconnected, then according to Theorem 2.2 there exists a simple cycle C_1 that contains the root r and the edge (v_2,v_3). The size of C_1 is at least 7 (Fact 2.2), hence C_1 is an 8-cycle (Lemma 5.1). This implies that C_1 contains exactly 2 cross edges. C_1 does not contain cross edge (v_2,v_d), $v_d \neq v_3$, otherwise we obtain a 7-cycle, since the nearest-common-ancestor of v_2 and v_d must be the root. Similarly, C_1 does not contain cross edge (v_3,v_e), $v_e \neq v_2$. Thus C_1 does not contain 2 cross edges on level 3, hence it must contain a cross edge e on level 2 because level 1 does not contain cross edges. This implies that C_1 must contain edge (v_1,v_2) (since v_2 has only one parent – v_1) and another tree or lattice edge incident to v_3, which means that cross edge e is incident to either v_1 or a parent of v_3. If e is incident to v_1, that is, $e = (v_1,v_f)$, for some vertex v_f, then the nearest-common-ancestor of v_1 and v_f must be the root. But we obtain a 9-cycle: $r \rightarrow v_f - v_1 - v_2 - v_3 - v_4 - v_c - v_a - r$. Contradiction. If e is incident to a parent of v_3, say v_g, that is, $e = (v_g,v_h)$, for some vertex v_h, then the nearest-common-ancestor of v_g and v_h must be the root. But we obtain another 9-cycle: $r \rightarrow v_h - v_g - v_3 - v_4 - v_5 - v_1 - v_a - r$. Contradiction. Thus structure (a) does not exist in $BFS(G_r)$.

Structure (b). Let v_a (v_b) be a parent of v_1 (v_2) in $BFS(G_r)$. Note that v_a and v_b are distinct, otherwise we obtain a 3-cycle: $v_1 - v_2 - v_a - v_1$. The root r is the parent of v_a and v_b. v_a must be the only parent of v_3, otherwise there exist 2 disjoint paths from v_3 and v_1 to r and the cycle: $v_1 - v_5 - v_4 - v_3 \rightarrow r \rightarrow v_1$ is a 7-cycle. Similarly, v_b must be the only parent of v_2 and v_4. But the cycle: $r - v_a - v_1 - v_2 - v_3 - v_4 - v_b - r$ is a 7-cycle. Contradiction. Thus structure (b) does not exist in $BFS(G_r)$.

Structure (c). Let v_a, v_b be a parent of v_1, v_4, respectively, in $BFS(G_r)$. Note that $v_b \neq v_1, v_2$ because otherwise we obtain 3-cycles. v_a must be the unique grandparent of v_3 and v_4, otherwise if v_3 and v_4

have distinct ancestors on level 1, then we obtain a 7-cycle through the root. Thus v_a is the only parent of v_b. Likewise, v_a must be the unique grandparent of v_4 and v_5, and v_a is the only parent of v_2. But the cycle: v_a–v_1–v_2–v_a is a 3-cycle. Contradiction. Thus structure (c) does not exist in $BFS(G_r)$.

Structure (d). Let v_a be a parent of v_1 in $BFS(G_r)$. The root r is the parent of v_a. v_a must be the unique grandparent of v_3 and v_5 (which implies that v_a is the only parent of v_1 and v_2), otherwise there exist 2 disjoint paths of length 3 from v_3 and v_5 to r and the cycle: v_3→r→v_5–v_3 is a 7-cycle. Furthermore, v_1 must be the only parent of v_3, for if v_p is another parent of v_3, then the cycle: v_3–v_p–v_a–v_1–v_4–v_2–v_5–v_3 is a 7-cycle. Likewise, v_2 must be the only parent of v_5. Note that v_1 and v_2 are not incident to cross edges, for if (v_1,v_q) is a cross edge, then the cycle v_1–v_q→r–v_a–v_2–v_4–v_1 is a 7-cycle (a similar construction holds if v_2 is incident to a cross edge).

Since G is biconnected, then according to Theorem 2.2 there exists a simple cycle C_1 that contains the root r and the edge (v_3,v_5). The size of C_1 is at least 7 (Fact 2.2), hence C_1 is an 8-cycle (Lemma 5.1). This implies that C_1 contains exactly 2 cross edges. The 2 cross edges on C_1 must be on level 3 of $BFS(G_r)$, because there are no cross edges on level 1 and the only parents of v_3 and v_5 that are on level 2 are not incident to cross edges. Without loss of generality, we shall assume that C_1 contains cross edge (v_3,v_b). Note that the path from r to v_b not going through v_3 on C_1 must be disjoint from the path r–v_a–v_2–v_5, because C_1 is a simple cycle. Thus C_1 consists of r–v_a–v_2–v_5–v_3–v_b–v_c–v_d–r, where v_c is a parent of v_b, and v_d is a parent of v_c (v_c≠v_2, v_d≠v_a, because C_1 is simple). Note that v_c≠v_1 because otherwise we obtain a 3-cycle: v_b–v_1–v_3–v_b. But the cycle: r–v_a–v_1–v_3–v_b–v_c–v_d–r is a 7-cycle. Contradiction. Thus structure (d) does not exist in $BFS(G_r)$.

Structure (e). Let v_a (v_b) be a parent of v_1 (v_2) in $BFS(G_r)$. Note that v_a and v_b are distinct, otherwise we obtain a 3-cycle: v_1–v_2–v_a–v_1. The root r is the parent of v_a and v_b. Let v_c be a parent of v_3. But one of the following cycle: r–v_c–v_3–v_4–v_1–v_2–v_b–r (even if $v_c = v_a$), or r–v_c–v_3–v_5–v_2–v_1–v_a–r (if $v_c = v_b$) is a 7-cycle. Contradiction. Thus structure (e) does not exist in $BFS(G_r)$.

Structure (f). Let v_a (v_b) be a parent of v_1 (v_2) in $BFS(G_r)$. Note that v_a and v_b are distinct, otherwise we obtain a 3-cycle: v_a–v_1–v_2–v_a. The parent of v_a and v_b is the root r. v_a must be the unique grandparent of v_4 and v_5 (which implies that v_a is the only parent of v_1 and v_3), otherwise there exist 2 disjoint paths of length 3 from v_4 and v_5 to r and the cycle: v_4→r→v_5–v_4 is a 7-cycle. Notice the symmetry of structure (f). We now observe some structural properties concerning the neighborhood of vertex

v_4. Due to symmetry, all the structural properties concerning the neighborhood of v_4 apply to v_5 as well.

1. v_4 has only 1 parent – v_1.

   Suppose that v_p is another parent of v_4. $v_p \neq v_2$, v_3, otherwise we obtain 3-cycles. But the cycle that consists of $\{v_4$–v_p + path from v_p to $NCA(v_p,v_1)$ + path from $NCA(v_p,v_1)$ to v_1 + v_1–v_2–v_3–v_5–$v_4\}$ is odd of size greater than 5.

2. v_4 does not have any children.

   Suppose that v_c is a child of v_4. According to Theorem 2.2, there exists a simple cycle in G containing v_c and the root r. Since v_c is not incident to a cross edge (Lemma 5.3) and v_c lies on a common simple cycle with the root r (Theorem 2.2), then v_c must have another parent – v_d ($v_d \neq v_5$). We now consider possible parents of v_d and show that none exists.

   a). v_1 is a parent of v_d.

   But the cycle v_c–v_d–v_1–v_a–v_3–v_5–v_4–v_c is a 7-cycle.

   b). v_2 is a parent of v_d.

   But the cycle v_c–v_d–v_2–v_3–v_a–v_1–v_4–v_c is a 7-cycle.

   c). v_3 is a parent of v_d.

   But the cycle v_c–v_d–v_3–v_a–r–v_b–v_2–v_1–v_4–v_c is a 9-cycle.

   d). v_e is a parent of v_d, $e \neq 1,2,3$.

   But the cycle that consists of $\{v_c$–v_d–v_e + path from v_e to $NCA(v_e,v_3)$ + path from $NCA(v_e,v_3)$ to v_3 + v_3–v_5–v_4–$v_c\}$ is odd of size greater than 5.

3. If v_4 is incident to a cross edge (v_4,v_c), then the parent of v_c must be v_3.

   v_1 is not a parent of v_c, otherwise we obtain a 3-cycle: v_c–v_1–v_4–v_c. v_2 is not a parent of v_c, otherwise we obtain a 7-cycle: v_c–v_2–v_b–r–v_a–v_1–v_4–v_c. If the parent of v_c is not v_3, then the cycle that consists of $\{v_4$–v_c + path from v_c to $NCA(v_c,v_3)$ + path from $NCA(v_c,v_3)$ to v_3 + v_3–v_2–v_1–$v_4\}$ is odd of size greater than 5.

4. If v_4 is incident to a cross edge (v_4,v_c), $v_c \neq v_5$, then v_5 is not incident to any other cross edge besides (v_4,v_5).

   If v_5 is incident to a cross edge (v_5,v_d), then the parent of v_d must be v_1, as noted above. But the cycle v_c–v_4–v_5–v_d–v_1–v_2–v_3–v_c is a 7-cycle.

From observations 1, 2, and 4 of structure (f), we conclude that at least one of v_4, v_5 is a vertex of degree 2. Since G is a graph of degree at least 3, then structure (f) does not exist in $BFS(G_r)$.

We have shown that there exist no 5-cycles on levels 2 and 3 of $BFS(G_r)$, hence the vertices on levels 2 and 3 are 2-colorable. The complexity for 3-coloring G is the same as that of breadth-first-search, which is $O(e)$. Notice that in this case, both the number of colors used and the running time are optimal.

□

Combining Theorem 5.3 and Lemma 2.1, we arrive at the following corollary.

Corollary 5.1. If G is biconnected, non-bipartite, and contains no odd cycles other than 5-cycles, then we can 3-colors G in $O(e)$ time.

The class of graphs that can be 3-colored by the algorithm of Corollary 5.1 (a simple algorithm can be derived by combining the algorithms of Lemma 2.1 and Theorem 5.3) properly contains those that satisfy the properties stated in Corollary 5.1. It seems that recognizing graphs that contain only 5-cycles in odd cycles is harder than coloring them. But from the point of view of graph coloring, a graph need not be recognized before it is colored. Hence for a graph that is suspected of satisfying the properties of Corollary 5.1, we can try to color it using the algorithm of Corollary 5.1. If the graph is properly colored, then our task is done. If not, then we can conclude that the graph does not belong to the class (i.e., biconnected, non-bipartite, and containing only 5-cycles in odd cycles). The whole process takes $O(e)$ time.

With the aid of breadth-first-search, when $k = 5$, we have been able to decompose a graph G satisfying the properties of Theorem 5.3 into components each of which can be colored with a small number of colors. However, unlike the case when $k = 3$, we do not have a complete characterization of the structures of the graph (see Conjecture 5.1). As Corollary 5.1 indicates, additional vertices of degree less than 3 can be added to G iteratively as long as the properties of the graph are preserved in the resultant graph (note that the graph remains 3-colorable even if some of the properties are violated while vertices are added in). To shed some light on the structures of graphs with only 5-cycles in odd cycles, we prove the following lemma, which has the same flavor as Lemma 5.4. The proof for Lemma 5.6 is more involved than that of Lemma 5.4.

Lemma 5.6. In a biconnected, non-bipartite graph G, if G contains no odd cycles other than 5-cycles, then every pair of 5-cycles in G must share at least 2 vertices.

Proof. Suppose that G contains 2 5-cycles – C_1 and C_2 that share at most 1 vertex. Let e_1 and e_2 be two edges on C_1 and C_2, respectively. According to Theorem 2.2, e_1 and e_2 lie on a common simple cycle, say C_3. We shall assume that C_3 is not an odd cycle of size greater than 5.

Because C_3 contains edges e_1 and e_2, there exist 2 vertex-disjoint paths – $path_1$ and $path_2$ on C_3 such that both paths go from a vertex on C_1 to a vertex on C_2, and both paths are disjoint from C_1 and C_2 except at their endpoints. Because C_1 and C_2 share at most 1 vertex, at least 1 of $path_1$, $path_2$ has length greater than or equal to 1. Observe that there are two edge-disjoint paths on C_1 (and on C_2) going from an endpoint of $path_1$ to an endpoint of $path_2$. The lengths of the two paths on C_1 (also on C_2) are either 1 and 4 or 2 and 3. Now it is not difficult to choose a combination of sizes so that a path on C_1 plus $path_1$ plus a path on C_2 plus $path_2$ form an odd cycle of size greater than 5. Contradiction.

□

The combination of Lemmas 5.2 and 5.6 suggests that in a graph that contains only 5-cycles in odd cycles, not only does every edge lie on a 5-cycle, but every 5-cycle shares at least 2 vertices with every other 5-cycle. Hence the interactions among odd cycles are tight. The tight interactions give rise to more structure which may lead to a proof of Conjecture 5.1.

5.3.3. $k > 5$

As we have seen, the approach we used in the previous section for 3-coloring a graph with only 5-cycles in odd cycles involved some structural analysis. As k increases, the complexity of the structures of the graph also increases. We don't see an easy way to extend the method used in the case $k = 5$ for 3-coloring a graph to the case $k > 5$. However, we do have a simple algorithm that 4-colors a graph with only k-cycles in odd cycles for $k > 5$. The algorithm is based on Lemma 5.5 and goes as follows.

Input. A graph G that is biconnected, non-bipartite, and contains no odd cycles other than k-cycles, $k > 5$.

Output. A 4-coloring of G.

1. Arbitrarily select a vertex r in G to be the root.

2. Build $BFS(G_r)$.
3. If vertices on even levels (including level 0) of $BFS(G_r)$ are independent, then color them with color A; else color them with colors A and B.
4. If vertices on odd levels of $BFS(G_r)$ are independent, then color them with color C; else color them with color C and D.

Algorithm 5.1. A 4-coloring algorithm for graphs that contain no odd cycles other than k-cycles, $k>5$.

Theorem 5.4. If G is biconnected, non-bipartite, and contains no odd cycles other than k-cycles, $k>5$, then Algorithm 5.1 4-colors G in time $O(e)$.

Proof. According to Lemma 5.5, all levels of $BFS(G_r)$ are 2-colorable. Since all even (odd) levels of $BFS(G_r)$ are independent from each other (Fact 2.4), hence we can 2-color all the even (odd) levels. Therefore 4 colors are sufficient to color G. It is clear that Algorithm 5.1 runs in time $O(e)$.

□

Observe that in Algorithm 5.1, if all even (odd) levels of $BFS(G_r)$ form an independent set, then only 3 colors are needed to color G. Once again, it is not necessary to recognize a graph before coloring it. Hence we can run Algorithm 5.1 on all *potential* graphs and if any level of a breadth-first-search graph of G fails to be bipartite, then we know that the graph does not satisfy all the properties of Theorem 5.4.

It is unfortunately that we are unable to characterize the structures of graphs with only k-cycles in odd cycles, for $k \geq 5$. We have shown that for $k = 5$, $k \geq 5$, all such graphs are 3-colorable, 4-colorable, respectively. But are these graphs all 3-colorable? This is an interesting question. We have the following conjecture.

Conjecture 5.1. There exist no biconnected, non-bipartite graphs of degree at least 3 and containing no odd cycles other than k-cycles, for all $k \geq 5$.

Note that if Conjecture 5.1 is true, then all graphs that contain only k-cycles in odd cycles, for $k \geq 5$, can be optimally colored in time $O(e)$ due to Lemma 2.1. This will also provide a uniform approach for optimally coloring all graphs with only k-cycles in odd cycles, for all values of k.

5.4. Odd Cycles = 3-Cycles + 5-Cycles

We now move on to graphs with two odd cycle sizes, more specifically, graphs with only 3-cycles and 5-cycles in odd cycles. These graphs may contain K_6 – a 6 clique as a subgraph, hence they may require at least 6 colors. However, as Theorem 5.1 indicates, they are 6-colorable. Once again, our approach for coloring a graph G with only 3-cycles and 5-cycles in odd cycles is to analyze the structures of G. To simplify the analysis, we shall first deal with graphs of degree at least 3, and then use Lemma 2.1 to relax the degree restriction. An outline of the structural analysis is as follows. We show that if G contains either K_5 or K_6, then the size of G is small. If G contains K_4 but no K_5, then we give a complete characterization of the structures of G and an optimal coloring of G. If G contains K_3 but no K_4, then we show that each level of a breadth-first-search graph of G is bipartite. Hence a 4-coloring of G follows immediately. We have designed a simple and efficient algorithm – Algorithm 5.2 based on our analysis. Algorithm 5.2 l-colors G (see Table 5.1 for the value of l), and in many cases l equals the chromatic number. We present our analysis of structures of G in the proof of Theorem 5.5.

Input. A graph G that is biconnected, non-bipartite, of degree at least 3, and contains no odd cycles other than 3-cycles and 5-cycles.

Output. A l-coloring of G (see Table 5.1 for the value of l).

1. If G contains no more than 6 vertices, then color G using $\chi(G)$ colors.
2. Else do (G contains no K_5)
    3. If G contains K_4, then do
        4. Delete (iteratively) all vertices in G with degree 3 (after deletion, G becomes empty).
        5. Add all the vertices deleted in Step 4 back to G and 4-color them.
    6. Else if the size of a smallest odd cycle is 5, then use the algorithm of Theorem 5.3 to 3-color G.
    7. Else do (G contains a 3-cycle but no K_4)
        8. Select a vertex r on a 3-cycle in G to be the root.
        9. Build $BFS(G_r)$. ($BFS(G_r)$ has at most 3 levels and every level of $BFS(G_r)$ is bipartite.)
        10. Color vertices on level 1 of $BFS(G_r)$ with colors C and D.
        11. If vertices on level 2 of $BFS(G_r)$ are independent, then color them with color A.
        12. Else color vertices on level 2 of $BFS(G_r)$ with colors A and B.
        13. Color the root r with color A.

14. Color vertices on level 3 of $BFS(G_r)$ with color D.

Algorithm 5.2. A l-coloring algorithm (see Table 5.1 for the value of l) for graphs that contain no odd cycles other than 3-cycles and 5-cycles.

Theorem 5.5. If G is a biconnected, non-bipartite graph of degree at least 3 and contains no odd cycles other than 3-cycles and 5-cycles, then Algorithm 5.2 l-colors G (see Table 5.1 for the value of l).

Proof. We shall analyze different structures of G, as listed below, and show that in the first three cases we can color G with an optimal number of colors, and in the last case we use at most 1 more than the optimal number of colors.

(1) G contains K_6,

(2) G contains K_5 but no K_6,

(3) G contains K_4 but no K_5 and K_6,

(4) G contains no K_4, K_5, and K_6.

In the following analysis, we will make use of the fact that if G contains no more than 6 vertices, then we can optimally color G by exhaustive search.

Structures (1) and (2). We show that because G contains K_5, the number of vertices in G is at most 6. Suppose that G contains more than 6 vertices. Let us consider the subgraph $G - K_5$. If $G - K_5$ contains an edge, say e_1, then according to Theorem 2.2, there exists a simple cycle C_1 that contains edge e_1 and an edge on K_5. Because K_5 is a complete graph on 5 vertices, it is not hard to construct an odd cycle of size at least 7 from C_1 and K_5. Thus $G - K_5$ must be an independent set. Let v_1 and v_2 be two vertices in $G - K_5$. Because the degrees of v_1 and v_2 are both at least 3, each of v_1, v_2 must have 3 edges to K_5, thus they must share a vertex on K_5. Without loss of generality, let u_1,u_2,u_3,u_4,u_5 be the 5 vertices of K_5, let the shared vertex be u_2, let v_1 be adjacent to u_1, and let v_2 be adjacent to u_3. But we can construct a 7-cycle from the edges (v_1,u_1), (v_1,u_2), (v_2,u_2), (v_2,u_3) and K_5. Contradiction. Hence G contains at most 6 vertices.

For both of these structures we can optimally color G in Step 1. So starting from Step 2 of Algorithm 5.2, we can assume that G does not contain K_5.

Structure (3). Suppose that we have identified a K_4. For simplicity, we shall call it K_4 and label its vertices v_a, v_b, v_c, v_d. We make the following observations.

3.1 All vertices in $G-K_4$ are distance 1 away from K_4.

Suppose not. Without loss of generality, let v_1 be a vertex that is distance (shortest distance) 2 or more away from K_4. According to Theorem 2.2 there exists a simple cycle C_1 that contains v_1 and an edge on K_4. Because v_1 is distance 2 or more away from K_4, the total length of paths on C_1 that connect v_1 with K_4 is at least 4. Thus from C_1, we can easily construct a simple odd cycle of size at least 7, because K_4 is a complete graph on 4 vertices. Contradiction.

3.2 $G-K_4$ does not contain paths of length 2.

Suppose that it does. Without loss of generality, let $v_1-v_2-v_3$ be a path of length 2 in $G-K_4$. We claim that both v_1 and v_3 must be adjacent to only one and the same vertex on K_4. Suppose not. From the preceding observation, we know that both v_1 and v_3 must be adjacent to some vertex (may be different) on K_4. Without loss of generality, we assume that v_1 (v_3) is adjacent to v_a (v_b). Then $v_1-v_2-v_3-v_b-v_c-v_d-v_a-v_1$ is a 7-cycle. Contradiction. This means that both v_1 and v_3 are vertices of degree 2. But G is a graph of degree at least 3. Contradiction.

Figure 5.3. Two families of graphs containing K_4 but no K_5 and only 3-cycles and 5-cycles in odd cycles.

From Observation 3.1 we know that every vertex in $G-K_4$ is adjacent to at least one vertex on K_4. From Observation 3.2 we know that each connected component of $G-K_4$ is either a vertex or a path of length 1. Note that each vertex in $G-K_4$ is adjacent to at most 3 vertices on K_4, because G does not

contain K_5. Since the degree of G is at least 3, an independent vertex in $G-K_4$ is adjacent to exactly 3 vertices on K_4, and a vertex on a path of length 1 in $G-K_4$ is adjacent to either 2 or 3 vertices on K_4. Due to the degree limitations, $G-K_4$ does not contain an independent vertex and a path of length 1, otherwise it is easy to construct a 7-cycle. Thus the connected components of $G-K_4$ are either all independent vertices or all paths of length 1. If $G-K_4$ contains all independent vertices and $|G| \geq 7$, then all the independent vertices are adjacent to the same 3 vertices on K_4 (otherwise it is easy to construct a 7-cycle). If $G-K_4$ contains paths of length 1 and $|G| \geq 7$, then all vertices in $G-K_4$ are adjacent to the same 2 vertices on K_4 (otherwise it is easy to construct a 7-cycle). Thus we have completely characterized the structures of G for $|G| \geq 7$; G belongs to one of the two families of graphs shown in Figure 5.3.

Knowing the structures of G, a 4-coloring of G can be easily accomplished. Note that if we iteratively delete vertices of degree 3 from G, we arrive at an empty graph. Thus 4 colors are sufficient to color G by Lemma 2.1 (Steps 4 and 5).

Structure (4). If G contains no 3-cycles, that is, if G contains only 5-cycles in odd cycles, then we can use the algorithm of Theorem 5.3 to 3-color G (Step 6). In the discussion which follows, we shall assume that G contains a 3-cycle. We make the following observations.

4.1 If G contains Figure 5.4 as a subgraph, then G is Figure 5.4 (i.e., G consists of exactly those 6 vertices).

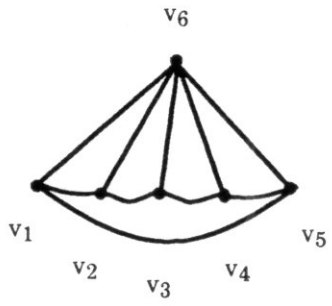

Figure 5.4. A structure containing a 5-cycle.

Suppose not. Then G contains at least 7 vertices. Let u be a vertex in G that is not part of Figure 5.4. We shall label the 6 vertices in G that correspond to v_1 through v_6 in Figure 5.4 in the same manner as they appear in Figure 5.4. According to Theorem 2.2, u and edge (v_1,v_2) lie on a common simple cycle C_1. Due to the symmetry and structure of Figure 5.4, there exist paths of length 2, 3, 4, and 5 between any two vertices in Figure 5.4. Thus it is not hard to construct a 7-cycle from C_1 and the graph of Figure 5.4 (see Figure 5.5 for illustration). Contradiction.

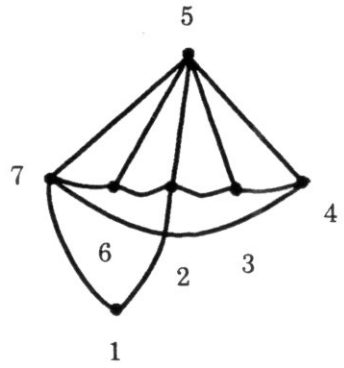

Figure 5.5. An illustration of a 7-cycle.

4.2 If there exists a 5-cycle on some level of a breadth-first search graph of G, then the vertices on the 5-cycle have exactly one parent, that is, the structure of the 5-cycle is the same as Figure 5.4.

Suppose not. Without loss of generality, suppose that v_1 has another parent. Then there exist 2 disjoint paths from v_1 and v_3 to a common ancestor – v_a that is at least 2 levels above them. But the cycle that consists of path v_1–v_5–v_4–v_3 plus the 2 disjoint paths from v_1 and v_3 to v_a is odd of length at least 7. Contradiction.

From Observation 4.1, we see that if G has Figure 5.4 as a subgraph, then G is optimally colored by Algorithm 5.2 at Step 1. Combining Observations 4.1 and 4.2, we can assume that from Step 7 on, a breadth-first-search graph of G does not contain a 5-cycle on any level. Now let r be a vertex on a 3-cycle in G (Step 8). Next we show that every level of $BFS(G_r)$ is bipartite.

4.3 $BFS(G_r)$ has at most 3 levels.

Suppose not. Let (v_1,v_2) be a cross edge on level 1 of $BFS(G_r)$ forming a 3-cycle with r. Let u be a vertex on level 4 of $BFS(G_r)$. According to Theorem 2.2, u and (v_1,v_2) lie on a common simple cycle, say C_1. The size of C_1 is at least 7. Because of Lemma 5.1, G does not contain even cycles of size 10 or more. Thus if C_1 is even, then size of C_1 must be 8. Since C_1 contains the edge (v_1,v_2) and it is of size 8, then C_1 does not contain the root r. Because edge (v_1,v_2) form a 3-cycle with the root, we obtain a 9-cycle by deleting the edge (v_1,v_2) from C_1 and adding edges (v_1,r) and (r,v_2) to C_1. Contradiction.

4.4 There exist no cross edges on level 3 of $BFS(G_r)$.

Suppose that level 3 of $BFS(G_r)$ contains a cross edge. Let (v_1,v_2) be a cross edge on level 1 of $BFS(G_r)$ forming a 3-cycle with r. Let (v_a,v_b) be a cross edge on level 3 of $BFS(G_r)$. According to Theorem 2.2, (v_1,v_2) and (v_a,v_b) lie on a common simple cycle, say C_1. The size of C_1 is at least 6. If C_1 does not contain the root, then we construct an odd cycle of size at least 7 by deleting the edge (v_1,v_2) from C_1 and adding edges (v_1,r) and (r,v_2) to C_1. If C_1 contains the root, then by Lemma 5.1 the size of C_1 is 8 or 9. If C_1 is even, then C_1 contains exactly 2 cross edges: (v_1,v_2) and (v_a,v_b). Thus C_1 must contain either v_1–v_2–r or v_2–v_1–r. Without loss of generality, we assume that C_1 contains v_1–v_2–r. But we obtain a 7-cycle by deleting the edges (v_1,v_2) and (v_2,r) from C_1 and adding edge (v_1,r) to C_1. Contradiction. Thus vertices on level 3 of $BFS(G_r)$ form an independent set.

4.5 If $BFS(G_r)$ contains a 3-cycle on some level, then Figure 5.6 is a part of the structure of $BFS(G_r)$.

Since G does not contain K_4 as a subgraph, $BFS(G_r)$ does not contain a 3-cycle on level 1. Thus the only level that a 3-cycle can appear on is level 2 of $BFS(G_r)$. We shall label vertices on the 3-cycle v_1,v_2, and v_3. These 3 vertices must not have the same parent, because otherwise we obtain a K_4. Without loss of generality, we assume that v_1 and v_2 have different parents, and they are v_4 and v_5, respectively. v_4 is not incident to a cross edge (v_4,v_a), for some vertex $v_a \neq v_5$, since otherwise we obtain a 7-cycle: v_4–v_a–r–v_5–v_2–v_3–v_1–v_4. Similarly, v_5 is not incident to a cross edge (v_5,v_b), for some vertex $v_b \neq v_4$.

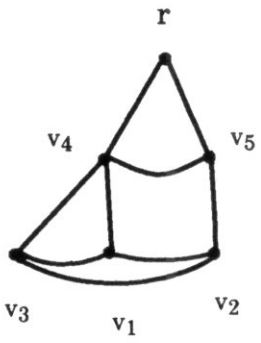

Figure 5.6. A structure containing a 3-cycle.

We know that there exists a cross edge, call it $e_1 = (v_a, v_b)$, on level 1 of $BFS(G_r)$, because r is part of a 3-cycle by construction. We now claim that v_4 is adjacent to v_5 by a cross edge, and it is the only cross edge on level 1. If not, then edge e_1 and edge (v_1,v_2) lie on a common simple cycle C_1 by Theorem 2.2. Note that v_a, $v_b \neq v_4$, v_5 since neither v_4 nor v_5 is incident to a cross edge other than (v_4,v_5). We shall denote cycle v_1–v_2–v_3–v_1 by C_2. Let $path_1$ be the path on C_1 containing edge e_1 with two endpoints v_c and v_d on C_2 such that if we traverse C_1 from edge e_1 in one direction we enter C_2 for the first time at v_c, and if we traverse C_1 from edge e_1 in the other direction we enter C_2 for the first time at v_d (see Figure 5.7). The length of $path_1$ must be either 3 or 4, because if the length of $path_1$ is 5 or more, then we can easily construct an odd cycle of size at least 7 from $path_1$ and C_2. Because $path_1$ contains edge e_1 and its length is only 3 or 4, $path_1$ does not contain the root. Now suppose that the length of $path_1$ is 3. Then v_a and v_b must be adjacent to two distinct vertices on C_2. This means that either v_1 or v_2 or both is on $path_1$. Without loss of generality, we assume that v_1 is on $path_1$ and v_1 is adjacent to v_a. But we obtain a 7-cycle: v_1–v_3–v_2–v_5–r–v_b–v_a–v_1. Contradiction. Suppose then the length of $path_1$ is 4. Since $path_1$ does not contain r, then we can augment $path_1$ by replacing edge e_1 with edges (v_a,r) and (v_b,r) so that $path_1$ becomes a path of length 5. But we can easily construct a 7-cycle from the augmented $path_1$ and C_2. Contradiction.

One more observation we make is that parent(s) of v_3 must be either v_4 or v_5 or both. Because if one of the parent of v_3 is some vertex other than v_4 or v_5, then we obtain a 7-cycle.

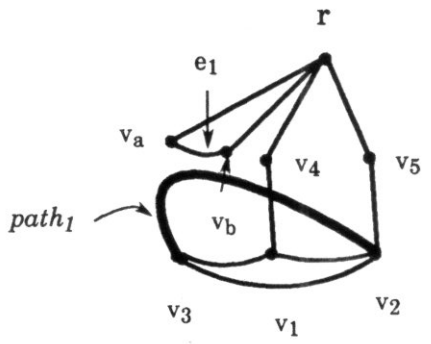

Figure 5.7. An illustration of $path_1$.

Without loss of generality, we assume that one of the parent of v_3 is v_4 due to symmetry. Figure 5.6 displays the structure we have thus far.

4.6 $BFS(G_r)$ does not contain Figure 5.6 as a part of its structure, thus $BFS(G_r)$ does not contain 3-cycles on any level.

We demonstrate the validity of Observation 4.6 by showing that the root r does not have children other than v_4 and v_5, which means that the root is of degree 2. Since the minimum degree of G is at least 3, then the structure in Figure 5.6 does not exist in $BFS(G_r)$, and we have the desired result.

If u is a child of r and $u \neq v_4, v_5$, then edge $e_1 = (u,r)$ and edge (v_1,v_2) lie on a common simple cycle C_1 by Theorem 2.2. We shall denote cycle v_1–v_2–v_3–v_1 by C_2. Let $path_1$ be the path on C_1 containing edge e_1 with two endpoints v_c and v_d on C_2 such that if we traverse C_1 from edge e_1 in one direction we enter C_2 for the first time at v_c, and if we traverse C_1 from edge e_1 in the other direction we enter C_2 for the first time at v_d. Since $path_1$ contains e_1 and two vertices on C_2, its length must be 4 or more. Specifically, the length of $path_1$ is exactly 4, because if the length of $path_1$ is 5 or more, then we can easily construct an odd cycle of size at least 7 from $path_1$ and C_2. Since the length of $path_1$ is 4, then u must be adjacent to a vertex on C_2. Suppose that u is adjacent to v_1. Then we obtain a 7-cycle: u–v_1–v_2–v_3–v_4–v_5–r–u. Contradiction. We can do similar constructions and obtain 7-cycles if u is adjacent to v_2 or v_3. Contradiction.

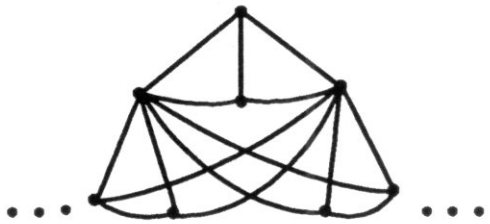

Figure 5.8. A family of graphs containing K_3 but no K_4 and only 3-cycles and 5-cycles in odd cycles.

Lastly we show that we can 4-color G. From Observations 4.3 and 4.4, we know that $BFS(G_r)$ has at most 3 levels and the vertices on level 3, if they exist, form an independent set. Thus 1 color is sufficient to color them (Step 14). From Observations 4.1 and 4.2, we know that $BFS(G_r)$ does not contain 5-cycles on any level. From Observations 4.5 and 4.6, we know that $BFS(G_r)$ does not contain 3-cycles on any level. Thus every level of $BFS(G_r)$ is bipartite and 4 colors are sufficient to color G. Figure 5.8 illustrates a families of graphs having structure (4) and also containing 3-cycles.

□

The number of colors used by Algorithm 5.2 is displayed in Table 5.1. Algorithm 5.2 uses the optimal number of colors in all cases except possibly one – when G does not contain K_4. Note that in this case, Algorithm 5.2 uses at most 4 colors, and the chromatic number of G could be 4, as exemplified by the graph in Figure 5.4. Also note that Algorithm 5.2 uses 3 colors if vertices on level 2 form an independent set (Step 11).

We shall now show that the complexity of Algorithm 5.2 is $O(e)$. First of all, it is not necessary to identify K_5 or K_6 in G, if one exists, in order to color it, because the size of the graph is small (no more than 6 vertices). In addition, to determine whether or not G contains K_4 is simple. Observe that in Figure 5.3, every vertex of degree 3 forms a complete graph – K_4 with its neighbors. Thus to find K_4 in G, we only need to check whether or not G contains a vertex of degree 3, and if it does whether this vertex forms a complete graph with its neighbors. Next we show that when G does not contain K_4, it is not necessary to determine the size of a smallest odd cycle before coloring G. We have decided to present Algorithm 5.2 in an intuitive and simple manner. However, an equivalent execution of Steps 6-14 of

Algorithm 5.2, which is a combination of the algorithm of Theorem 5.3 and Algorithm 5.2, is shown in Figure 5.9.

1. Arbitrarily select a vertex a in G to be the root. (At this point, G contains no K_4 and $|G| \geq 7$).
2. Build $BFS(G_a)$ ($BFS(G_a)$ has at most 4 levels).
3. If there exist no cross edges on level 1 of $BFS(G_a)$ and every level of $BFS(G_a)$ is bipartite, then do
    4. If the vertices on levels 2 and 3 of $BFS(G_a)$ induce a bipartite subgraph, then 3-color G (since both levels 1 and 4, if level 4 exists, are independent sets).
    5. Else (the size of a smallest odd cycle is 3) 3-color or 4-color G (since every level of $BFS(G_a)$ is bipartite).
6. Else do (the size of a smallest odd cycle is 3)
    7. Identify a 3-cycle in G. (*Either* there exists a cross edge on level 1 of $BFS(G_a)$, in which case, the root a and the cross edge forms a 3-cycle, *or* some level of $BFS(G_a)$ is not bipartite, in which case, a 3-cycle can be identified on that level using breadth-first-search. Recall from Observations 4.1 and 4.2 that no level of any breadth-first-search graph of G contains a 5-cycle.)
    8. 3-color or 4-color G (Steps 8-14 of Algorithm 5.2).

Figure 5.9. An equivalent execution of Steps 6-14 of Algorithm 5.2.

All steps in Figure 5.9 take $O(e)$ time. Therefore Algorithm 5.2 runs in $O(e)$ time.

The class of graphs that can be l-colored by Algorithm 5.2 properly contains those which satisfy the properties stated in Theorem 5.5. Once again, it is not necessary to recognize G before coloring it. Thus for a graph G that is suspected of satisfying the properties stated in Theorem 5.5, we can run Algorithm 5.2 on it. If a proper coloring is produced, then we are all set. If not, then we conclude that G does not belong to the class of graphs described in Theorem 5.5.

Using Lemma 2.1, we can relax the degree restriction of Theorem 5.5, and obtain the following corollary.

Corollary 5.2. If G is biconnected, non-bipartite, and contains no odd cycles other than 3-cycles and 5-cycles, then we can l-color G in $O(e)$ time (see Table 5.1 for the value of l).

5.5. Discussion

In this chapter, we have shown that biconnected graphs containing only small odd cycles possess many special structures. We have used breadth-first-search to identify various structures and to show that certain structures do not exist. Knowing the structures of graphs enables us to find bipartite subgraphs which aid graph coloring. We have developed graph coloring algorithms that are simple, efficient, and either optimal or 1 more than optimal in the number of colors used.

Our approach for analyzing graph structures in this chapter has been, more or less, a case study. Although we have had success with it, the approach does not generalize. As k increases, the complexity of the structures of the graph increases, and the number of cases increases. For large k, case analysis becomes infeasible. A more general, uniform method of analyzing and identifying global structures of graphs is certainly more desirable. We pose this as an open problem for further studies. Other open problems include resolving Conjecture 5.1, and improving the performance of algorithms so that optimal coloring is produced for all graphs studied in this chapter.

Chapter 6

Algorithms for Finding a Maximum Bipartite Subgraph for Special Classes of Graphs

6.1. Introduction

In this chapter we are interested in finding a *maximum induced bipartite subgraph* of a graph, that is, 2 independent sets that cover maximum number of vertices. (For the rest of the chapter, when we say subgraph we mean *induced* subgraph.) In general this problem is NP-complete as shown by Lewis and Yannakakis[1980]. Furthermore, even when restricted to planar graphs, the maximum bipartite subgraph problem remains NP-complete [Lewis and Yannakakis, 1980]. This makes one wonder if additional restrictions on graphs affect the complexity of this problem. In particular, it is interesting to identify special classes of graphs for which finding a maximum bipartite subgraph is solvable in polynomial time. The focus of this chapter is in showing efficient polynomial-time algorithms for this problem on proper circular-arc, circular-arc, permutation, and split graphs. More specifically, we present $O(n\delta^2)$ algorithm for finding a maximum bipartite subgraph for circular-arc graphs, when the graphs are given in the form of a family of intervals, where δ is the minimum number of arcs covering any point on the circle. For proper circular-arc graphs, we give an $O(n^2)$ algorithm that solves the same problem. Using dynamic programming, we show how to find a maximum bipartite subgraph for permutation graphs in time $O(n\bar{e}_t)$, where $\bar{e}_t$ is the number of edges in the complement of a permutation graph after transitive reduction. For split graphs, we use matrix multiplication and obtain an $O(n^{2.376})$ algorithm for the maximum bipartite subgraph problem.

In this chapter we will also consider finding a maximum bipartite subgraph that includes a non-trivial subset (size > 0) of vertices. We shall call this the *maximum bipartite subgraph with a chosen*

subset problem, with the understanding that the size of the chosen subset is greater than 0. If we allow the size of the chosen subset to be 0, then the maximum bipartite subgraph problem is a special case of the maximum bipartite subgraph with a chosen subset problem. This means that the second problem is as hard as the first, if not harder. However, if we only consider the cases where the size of the chosen subset is greater than 0, then, as we will show, in most cases the complexity of the algorithms for finding a maximum bipartite subgraph remain the same, and in one case – circular-arc graphs, the complexity of the algorithm is reduced from $O(n\delta^2)$ to $O(n\log n + n\Delta)$, where Δ is the maximum number of arcs covering any point on the circle.

Finding a maximum bipartite subgraph falls in the general frame work of finding a maximum k-colorable subgraph of a graph. Two well-known problems: maximum independent set and the chromatic number, are also special cases of this problem. Interestingly, the maximum independent set problem, though shown to be NP-complete for general graphs and planar graphs, is polynomially solvable for a wide range of other well-known classes of graphs, including perfect (which properly contain chordal, comparability, permutation, and interval), circle, circular-arc, series-parallel, and k-outerplanar graphs; see Johnson[1985] for references. The chromatic number problem, however, is NP-complete for a larger number of classes of graphs, including planar, circular-arc, and circle graphs, but is polynomial for perfect, series-parallel, and k-outerplanar graphs; see Johnson[1985] for references. The existence of polynomial-time algorithms for the maximum independent set and the chromatic number problems on various classes of graphs may lead one to believe the maximum bipartite subgraph problem is solvable in polynomial time for these classes of graphs as well. In fact, the complexity of the maximum k-colorable subgraph problem on special classes of graphs has been investigated by a number of researchers, including Frank[1980] who has presented a polynomial-time algorithm for this problem on comparability and co-comparability graphs, and Yannakakis and Gavril[1987] who have shown for chordal graphs this problem is polynomially solvable when k is fixed, but NP-hard when k is not fixed. One implication of these results is that the maximum bipartite subgraph problem is polynomially solvable for comparability, co-comparability, and chordal graphs.

Graph Class	Running Time		Discoverer	Transformation
	Max. Bipartite	with Subset		Time†
1. Interval	$O(n\log n)$	$O(n\log n)$	Yannakakis and Gavril[1987]	$O(n+e)$
2. Proper Circular-Arc	$O(n^2)$	$O(n\log n + n\Delta)$‡	Yeh and LaPaugh	$O(n^2)$
3. Circular-Arc	$O(n\delta^2)$‡‡	$O(n\log n + n\Delta)$	Yeh and LaPaugh*	$O(n^3)$
4. Permutation	$O(n\overline{e_t})$‡‡‡	$O(n\overline{e_t})$	Yeh and LaPaugh**	$O(de + n^2)$‡‡‡‡
5. Comparability	$O(ne)$	open	Frank[1980]	$O(de)$
6. Co-Comparability	$O(ne)$	open	Frank[1980]	$O(de)$
7. Split	$O(n^{2.376})$	$O(n^{2.376})$	Yeh and LaPaugh	NA
8. Chordal	$O(ne)$	$O(ne)$	Yannakakis and Gavril[1987]	$O(n+e)$
9. Series-Parallel	$O(n+e)$††	open	Takamizawa et al[1982]	NA
10. Planar	NP-complete	NP-complete	Lewis and Yannakakis[1980]	NA
11. Circle	NP-complete	NP-complete	Sarrafzadeh and Lee[1987]	NA
12. General	NP-complete	NP-complete	Lewis and Yannakakis[1980]	NA

† Transformation time corresponds to the time needed to transform a graph $G = (V, E)$ to the proper representation required by the algorithm.

†† Takamizawa, Nishizeki, and Saito[1982] have claimed this result in their paper but a proof was not given.

* An independent study of this problem on circular-arc graphs has been done by Manber and Narasimhan[1987] where they obtained an $O(n^3)$ time algorithm.

** Recently Sarrafzadeh and Lee[1987] have developed an $O(n^2\log n)$ algorithm for this problem on permutation graphs. Their algorithm will be the preferred algorithm when the complement of the permutation graph after transitive reduction is dense.

‡ Δ is the maximum number of arcs covering any point on the circle, $\Delta \le d \le n$.

‡‡ δ is the minimum number of arcs covering any point on the circle, $\delta \le \sqrt{e}$.

‡‡‡ $\overline{e_t}$ is the number of edges in the complement of a permutation graph after transitive reduction.

‡‡‡‡ d is the maximum degree of a vertex.

Table 6.1. Polynomial algorithms and NP-complete results for the maximum bipartite subgraph problem and the maximum bipartite subgraph with a chosen subset problem. Algorithms for graphs 1, 2, 3, 4, 7 are presented in this chapter.

Table 6.1 summarizes the current status of polynomial-time algorithms and NP-complete results for the maximum bipartite subgraph problem and the maximum bipartite subgraph with a chosen subset problem on various classes of graphs. We have analyzed the running time of algorithms presented by Frank for comparability and co-comparability graphs and by Yannakakis and Gavril for chordal graphs [Yannakakis, 1987], and included our findings in the table.

The remaining portion of this chapter is devoted to presentation of the algorithms. We start with some preliminaries in Section 6.2. In Section 6.3 we describe a greedy algorithm for interval graphs; in Section 6.4 we introduce the notion of placing intervals on tracks and present algorithms for track assignment problems. The discussions of Sections 6.3 and 6.4 will be used to develop the algorithms for proper circular-arc and circular-arc graphs. We present algorithms for proper-circular-arc, circular-arc, permutation, and split graphs in Sections 6.5, 6.6, 6.7, and 6.8, respectively. In Section 6.9 we close with some concluding remarks and suggestions for future research.

6.2. Preliminaries

In this chapter when we consider a subgraph G_A ($A \subseteq V$) of G, we mean the *induced subgraph* of G, that is, G_A consists of the set of vertices A, and two vertices are adjacent in G_A if and only if they are adjacent in G. A graph $G = (V, E)$ is called an *intersection* graph for a family F of sets if each set in F is represented by a vertex and two vertices are adjacent if and only if their corresponding sets intersect. The intersection graph of a family of intervals on a real line is called an *interval* graph. The intersection graph of a family of arcs on a circle is called a *circular-arc* graph. A *proper circular-arc* graph is a circular-arc graph such that no arc is contained within another. The intersection representation for interval, proper circular-arc, and circular-arc graphs can be constructed in time $O(n+e)$ [Booth and Lueker, 1976], $O(n^2)$ [Tucker, 1971], and $O(n^3)$ [Tucker, 1980], respectively.

A graph $G = (V, E)$ is called *perfect* if for every subgraph G_A of G ($A \subseteq V$), the chromatic number of G_A equals its maximum clique size. Perfect graphs properly contain comparability, permutation, chordal, split, and interval graphs. *Comparability* graphs, also known as *transitively orientable* and *partially orderable* graphs, are undirected graphs $G = (V, E)$ that can, by appropriate orientation of their edges, be turned into transitive directed graphs (directed graphs such that if (a,b) and (b,c) are arcs, then so is (a,c)). *Co-comparability* graphs are graphs whose complements are comparability graphs. *Permutation* graphs

are comparability graphs whose complements are also comparability graphs (an alternate characterization of permutation graphs is given in Section 6.7). An undirected graph G is called *chordal* if every cycle of length greater than 3 has a chord, i.e., an edge joining two nonconsecutive vertices on the cycle. Chordal graphs are also known as *triangulated*, *perfect elimination*, *rigid-circuit*, and *monotone transitive* graphs. *Split* graphs are chordal graphs whose complements are also chordal graphs (an alternate characterization of split graphs is given in Section 6.8). Comparability and chordal graphs are incomparable, that is, neither is properly contained within another.

6.3. Interval Graphs

Assume an interval graph I is given as a set of intervals, $I=\{f_1, f_2, \cdots, f_n\}$, on the real line. Each interval is in the form of (left_endpoint, right_endpoint), and is represented by a vertex. Without loss of generality we can assume that the $2n$ endpoints of the intervals are distinct.† The algorithm for finding a maximum bipartite subgraph for interval graphs goes as follows:

Input. An interval graph I given as a set of intervals, $I=\{f_1, f_2, \cdots, f_n\}$.

Output. A maximum bipartite subgraph of I.

1. Sort the $2n$ endpoints in ascending order.
2. Number the n intervals (vertices) according to their right_endpoints in ascending order. Let β, $\beta = \{1, 2, ..., n\}$, be the collection of n intervals.
3. While β is nonempty do
    4. Pick an interval with the smallest right_endpoint from β and add it to a set called S (originally empty).
    5. Delete all intervals from β that overlap two intervals already in S.

    end while
6. Return S.

Algorithm 6.1. Maximum bipartite subgraph algorithm for interval graphs.

---

† If the endpoints are not distinct, then we can make them distinct by placing one endpoint just to the left or right of the other, depending on whether we consider intervals to be overlapping if they only overlap at one endpoint. It is important that we do not change the "overlap relationship" among the intervals during this process.

This is a greedy algorithm; at each step we are picking an interval that is as good as, if not better than, any other interval that overlaps it. We will need the following definition for the proof of the correctness of Algorithm 6.1:

> Let $G_a=(V_a, E_a) \subseteq G$ and $G_b=(V_b, E_b) \subseteq G$. G_a *disagrees* with G_b at vertex v_a if v_a is the smallest numbered vertex in V_a but not in V_b. Similarly if an interval graph I is given in the form of a family of intervals, and $I_a \subseteq I$, $I_b \subseteq I$, then we say I_a *disagrees* with I_b at interval f_a if f_a is the smallest numbered interval in I_a but not in I_b.

Theorem 6.1 [Yannakakis and Gavril, 1987]. Algorithm 6.1 finds a maximum bipartite subgraph of an interval graph I given in the form of a family of intervals.

Proof. Assume S is not a maximum bipartite subgraph of I.

Step 5 of the algorithm ensures that S is an bipartite subgraph of I. Let I_B be a maximum bipartite subgraph of I with which S disagrees the latest, i.e., S disagrees with I_B at the highest numbered interval among all maximum bipartite subgraphs of I. Without loss of generality let's say S disagrees with I_B at interval f_s, and I_B disagrees with S at interval f_b. By Step 4 of the algorithm, we know the right_endpoint of interval f_s is less than that of interval f_b. So $f_s < f_b$ by the way intervals are numbered (Step 2). Knowing I_B is bipartite, the number of intervals in I_B numbered larger than f_b and overlapping f_b is at most 1, and let's call it f_c, if it exists. Thus f_b and f_c are the only two possible intervals numbered larger than f_s in I_B that can overlap f_s. Now we have two cases to consider: 1. $f_c \in S$; 2. $f_c \notin S$. In the first case, we can substitute f_s for f_b in I_B without violating the bipartiteness of I_B, because all intervals that overlap f_s in I_B are also in S. In the second case, we observe that at most one of f_b, f_c overlaps intervals numbered less than f_s in I_B. Hence we can substitute f_s for f_b or f_c whichever has a smaller left_endpoint in I_B without violating the bipartiteness of I_B. In either case, S disagrees with this new maximum bipartite subgraph at an interval numbered larger than f_s. Contradiction.

□

The most time consuming part of Algorithm 6.1 is sorting (Step 1), which takes $O(n\log n)$ time. We determine which intervals to delete (Step 5) by keeping track of the interval with the second largest right_endpoint that overlaps another interval in S, and deleting all intervals in β that overlap this right_endpoint. The total amount of time spent in Step 5 is $O(n)$. Hence the whole algorithm runs in time $O(n\log n)$.

We will discuss the maximum bipartite subgraph with a chosen subset problem for interval graphs in Section 6.4.

6.4. Placing Intervals on Tracks

The maximum bipartite subgraph problem is also known as the maximum 2-colorable subgraph problem. The vertices in each color induce an independent set. In the case of interval graphs, the vertices in each color are non-overlapping intervals. If we place these non-overlapping intervals on a "track", then the maximum bipartite subgraph problem for interval graphs can be viewed as a *2-track assignment* problem:†

> Given 2 tracks numbered from 0 through $2n-1$, and n interval of the form (l_i, r_i); l_i, r_i take on values between 0 and $2n-1$ inclusive; all $2n$ values of l_i and r_i are distinct.
>
> Pack as many intervals on the 2 tracks as possible without overlapping.

The algorithm that solves the 2-track assignment problem is the same as Algorithm 6.1, except in Step 4 we actually place intervals on the tracks: Pick an interval with the smallest right_endpoint from β and place it on the "tighter" fitting track, i.e., the track having an interval with the largest right_endpoint, if it fits on both. Ties are broken arbitrarily.

Now let's consider a variation of the 2-track assignment problem - *left-end-limited 2-track assignment* problem:

> Given 2 tracks - track 1 (2) is numbered from p_1 (p_2) through $2n-1$, and n interval of the form (l_i, r_i); p_1, p_2, l_i, r_i take on values between 0 and $2n-1$ inclusive; all $2n$ values of l_i and r_i are distinct.
>
> Pack as many intervals on the 2 tracks as possible without overlapping.

We solve the left-end-limited 2-track assignment problem by first deleting all those intervals that can not be placed on the limited tracks, i.e., deleting all intervals that either contain the point $\min\{p_1,p_2\}-1$ or end before $\min\{p_1,p_2\}$, and then calling the 2-track assignment algorithm on the remaining intervals with one modification: after Step 2 and before Step 3, place dummy intervals $(0, p_1-1)$ and $(0, p_2-1)$ on track1 and track2, respectively. It is clear the complexity of the algorithm for the

† In this chapter we shall always sort the $2n$ endpoints; hence without loss of generality we can assume that the left and right endpoints of intervals take on values between 0 and $2n-1$.

left-end-limited 2-track assignment problem is also $O(n\log n)$.

We can define the *right-end-limited 2-track assignment* problem accordingly. The right-end-limited 2-track assignment problem is actually the left-end-limited 2-track assignment problem in reverse. Hence the right-end-limited 2-track assignment problem can also be solved in time $O(n\log n)$.

Another variation of the 2-track assignment problem is the *two-end-limited 2-track assignment* problem:

> Given 2 tracks - track 1 (2) is numbered from p_1 (p_2) through q_1 (q_2), and n interval of the form (l_i, r_i); p_1, p_2, q_1, q_2, l_i, r_i take on values between 0 and $2n-1$ inclusive; all $2n$ values of l_i and r_i are distinct.
>
> Pack as many intervals on the 2 tracks as possible without overlapping.

The approach we take in solving the two-end-limited 2-track assignment problem is similar to that of Algorithm 6.1 with one difference: in Algorithm 6.2 we look for opportunities to reduce this problem to a left-end-limited 2-track assignment problem plus a right-end-limited 2-track assignment problem. The algorithm goes as follows:

Input. 2 tracks numbered from p_1 (p_2) through q_1 (q_2); an interval graph I given as a set of intervals, $I = \{f_1, f_2, \cdots, f_n\}$.

Output. 2-track assignment – a placement of the maximum number of intervals on the 2 tracks without overlapping.

1. Sort the $2n$ endpoints in ascending order.
2. Number the n intervals according to their right_endpoints in ascending order.
   Let β, $\beta = \{1, 2, ..., n\}$, be the collection of n intervals.
3. Delete all intervals from β that either contain the point $\min\{p_1,p_2\}-1$ or end before $\min\{p_1,p_2\}$.
4. Delete all intervals from β that either contain the point $\max\{q_1,q_2\}+1$ or begin after $\max\{q_1,q_2\}$.
5. Place dummy intervals $(0, p_1-1)$ and $(0, p_2-1)$ on track1 and track2, respectively.
6. While β is nonempty do
   7. Determine f – the interval from β with the smallest right_endpoint.
   8. If the right_endpoint of f is greater than $\min\{q_1,q_2\}$ (the track with the smaller right end is filled), then place a dummy interval $(\min\{q_1,q_2\}+1, 2n-1)$ on the track with the smaller right end, if one is not already placed there.
   9. If there exists an interval f_i, on track1 or track2, such that f_i is the left-most interval that overlaps f, f_i is not a dummy interval, and f_i does not overlap any other interval already placed on the

tracks,

then do
(we have found a cut at a point just to the left of f or f_i; see Figure 6.1 for illustration.)

10. Add f, f_i, and all intervals on the same track as f_i and to the right of f_i (if they exist) back to list β.
11. Solve the right-end-limited 2-track assignment problem on track1 and track2 with intervals remaining in β.
12. Exit Algorithm 6.2.

Now we know no interval satisfies the condition described in Step 9.

13. Place f on the tighter track, if it fits on both.
14. Remove all intervals that overlap two intervals already placed on track1 and track2 from β.

end while

Algorithm 6.2. Two-end-limited 2-track assignment algorithm.

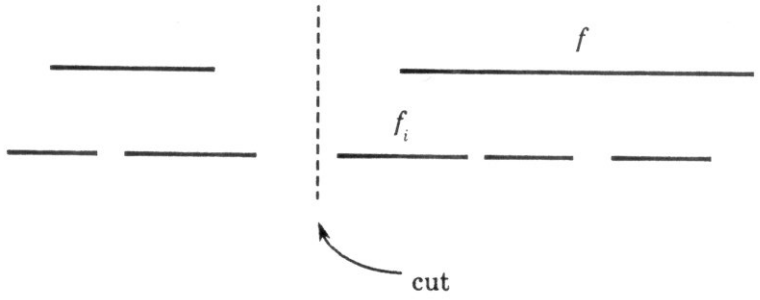

Figure 6.1. An example where a cut is found just to the left of f_i.

Theorem 6.2. Algorithm 6.2 correctly solves the two-end-limited 2-track assignment problem.

Proof. Algorithm 6.2, like Algorithm 6.1, is basically a greedy algorithm. If the condition described in Step 9 is never true, then we can use similar arguments as those in the proof of Algorithm 6.1 to show that there must exist a maximum two-end-limited 2-track assignment T such that T agrees with intervals picked by Algorithm 6.2.

On the other hand, if the condition described in Step 9 is true at some point, then using similar arguments as above, we know there exists a maximum two-end-limited 2-track assignment T_1 such that T_1 agrees with intervals picked by Algorithm 6.2 up to that point of the algorithm. This means that intervals f and f_i of Step 9 are in T_1. Since f and f_i do not overlap intervals that begin to their left in T_1, there must exist an unit interval gap (a_i,a_{i+1}) just to the left of f or f_i that is not covered by any interval in T_1. Knowing T_1 is a maximum two-end-limited 2-track assignment, we can cut the 2 tracks at the unit interval (a_i,a_{i+1}) and reduce the two-end-limited 2-track assignment problem to one left-end-limited 2-track assignment problem plus one right-end-limited 2-track assignment problem.

□

Algorithm 6.2 not only selects but also places the intervals on tracks subject to the constraints of the limited tracks. Note that in the above proof, the reason the arguments in the proof of Theorem 6.1 apply when the condition described in Step 9 is never true is because, in this case, the bipartite subgraph corresponding to those intervals picked by Algorithm 6.2 that fit between the overlapping part of the two tracks (between $\max\{p_1,p_2\}$ and $\min\{q_1,q_2\}$) is connected; hence there is exactly one way to place the intervals on tracks – basically placing an interval on the tighter track when it fits on both is the correct strategy. However when the condition described in Step 9 is true at some point, due to the limited right-end of the tracks, the proper choice for placing intervals f and f_i of Step 9 on the tracks to the right of the gap (a_i,a_{i+1}) is not apparent. (Note that the strategy for selecting the intervals is still correct.) In this case, cutting the tracks at (a_i,a_{i+1}) and approaching the problem from the right-end will give us the proper placement of intervals on tracks.

The complexity of Algorithm 6.2 is still dominated by sorting (Step 1), which is $O(n\log n)$. All other steps take linear time. Hence the total running time of Algorithm 6.2 is $O(n\log n)$.

Now we will consider the maximum bipartite subgraph with a chosen subset problem for interval graphs. Without loss of generality we can assume that the intervals in the chosen subset are bipartite. It is easier to view the maximum bipartite subgraph problem on interval graphs as a 2-track assignment problem when a subset is included. So in what follows, we shall discuss only the 2-track assignment problem. If two of the intervals in the chosen subset overlap, then we can reduce the original problem to two subproblems by cutting the tracks at a point of overlap. Hence without loss of generality, we can assume that all the intervals in the chosen subset are non-overlapping. There are 3 varieties of the 2-track

assignment problem with a chosen subset: 1). un-limited, 2). left-end-limited (or right-end-limited), and 3). two-end-limited.

At the first sight, adding a chosen subset to the 2-track assignment seems to increase the complexity of this problem. However the greedy approach of Algorithm 6.1 and Algorithm 6.2 can still be employed to solve the 2-track assignment problem with a chosen subset; although some care must be taken.

We will use Algorithm 6.1 (or its variation) to solve the first 2 varieties of the 2-track assignment problem with a chosen subset, and Algorithm 6.2 to solve the third variety, and in both cases we will apply the following modifications:

Delete all the intervals in the chosen subset from β.

Place the intervals in the chosen subset, one at a time, on the tighter fitting track, in increasing order of their right_endpoint at the appropriate times.

By the appropriate times, we mean: 1. an interval f_i in the chosen subset is placed on the track right after the first interval that overlaps and ends before f_i is placed on the track, if such an interval exists, and 2. all intervals that end after f_i, for some f_i in the chosen subset, are placed on the tracks after f_i. These modifications can be done with some simple book keeping. Note that Step 5 of Algorithm 6.1, and similarly Step 14 of Algorithm 6.2, is carried out even when an interval in the chosen subset is placed on the track. This is done to ensure that the 2-track assignment is always valid, i.e., the corresponding subgraph is always bipartite.

The correctness of the above approach is validated by proofs of Theorem 6.1 and 6.2, because any maximum 2-track assignment must include all the intervals in the chosen subset; hence substitution, the method used in showing the correctness of Algorithm 6.1, is applied only to those intervals not in the chosen subset, and remains valid.

6.5. Proper Circular-Arc Graphs

In the following we will assume that a proper circular-arc graph C is given in the form of a family of arcs, $C= \{c_1, c_2, \cdots, c_n\}$, on a circle, where each arc is denoted by two endpoints – left and right, in clockwise direction. Again without loss of generality we can assume that the $2n$ endpoints of the arcs are distinct. As pointed out in the preliminaries, if a graph G is given in the form of vertices and edges, then

we can use Tucker's algorithm to construct a circular-arc representation of G in $O(n^2)$ time.

We will consider the following two possibilities when finding a maximum bipartite subgraph for a proper circular-arc graph C:

1. C has a maximum bipartite subgraph that is also an interval graph.
2. Not case 1, that is, the union of arcs of every maximum bipartite subgraphs of C covers the circle.

If case 1 is true, then we can reduce this problem on a proper circular-arc graph to $2n$ problems on interval graphs by cutting the circle at each of the $2n$ endpoints, apply Algorithm 6.1, and take the maximum bipartite subgraph we find. Recall in a proper circular-arc graph no arc is properly contained within another. Thus any arc in a bipartite subgraph overlaps at most two other arcs in the subgraph. Now if case 2 is true, then any maximum bipartite subgraph of C must be an even cycle (in the degenerate case, it consists of 2 arcs or vertices), since each arc in the maximum bipartite subgraph must overlap exactly one arc to the right and one arc to the left of it. The maximum independent set of an even cycle consists of half of the arcs in the cycle. Note that the other half is also a maximum independent set of the even cycle. As we shall show in the proof of Theorem 6.3, if all maximum bipartite subgraphs of C are even cycles, then there exists a maximum bipartite subgraph S of C such that S consists of the union of maximum independent set of interval graphs C_a and C_b, for some a, b, $1 \le a,b \le 2n$, where C_i is the set of arcs in C except all those that contain interval (a_i, a_{i+1}) as defined in Step 2 of Algorithm 6.3.

The following algorithm describes the details of the above approach.

Input. A proper circular-arc graph C given as a set of arcs, $C = \{c_1, c_2, \cdots, c_n\}$.

Output. A maximum bipartite subgraph of C.

1. Sort the $2n$ endpoints in clockwise direction and output them in a list α, $\alpha = \{a_1, a_2, \cdots, a_{2n}\}$, where $a_{2n+1} = a_1$.
2. For i = 1 to $2n$, determine C_i.
   C_i is defined as the set of arcs in C except all those that contain interval (a_i, a_{i+1}). Note that C_i is an interval graph.
3. For i = 1 to $2n$ obtain a maximum bipartite subgraph of C_i by running Algorithm 6.1.
4. For i = 1 to $2n$ obtain a maximum independent set M_i of C_i by the following steps:†

† This maximum independent set algorithm is presented by Gupta, Lee, and Leung[1982].

5. Reindex the endpoints in C_i so a_{i+1} becomes a_1, a_{i+2} becomes a_2, ... , a_i becomes a_{2n}.
6. Number the n_i intervals in C_i according to their right_endpoints in ascending index order. Let β_i, $\beta_i = \{1, 2, ..., n_i\}$, be the collection of n_i intervals.
7. While β_i is nonempty do
    8. Pick an interval with the smallest right_endpoint from β_i and add it to a set called M_i (originally empty).
    9. Delete all intervals from β_i that overlap any interval already in M_i.

    end while
10. Let $first_i$ ($last_i$) denote the first (last) arc placed in set M_i.

11. For each set M_i, find a set M_j, if one exists, such that $|M_i| = |M_j|$ and $M_i + M_j$ is an even cycle. Let M_{ij} denote the set containing such an even cycle.
12. Return the maximum bipartite subgraph produced in Steps 3 and 11.

Algorithm 6.3. Maximum bipartite subgraph algorithm for proper circular-arc graphs.

Theorem 6.3. Algorithm 6.3 finds a maximum bipartite subgraph of a proper circular-arc graph C.

Proof. If C has a maximum bipartite subgraph that is also an interval graph, then Algorithm 6.3 finds it at Step 3.

Now assume that all maximum bipartite subgraphs of C are even cycles. Let B be one of them. B consists of two independent sets, say X and Y, of equal size. Without loss of generality let (a_i, a_{i+1}) be an unit interval on the circle such that exactly one arc in B covers it. Without loss of generality let Y be the set that contains the arc, say y_1, covering interval (a_i, a_{i+1}). We shall label the arcs in Y, $Y = \{y_1, y_2, \cdots, y_r\}$, according to their right_endpoints in clockwise direction. Similarly we shall label the arcs in X, $X = \{x_1, x_2, \cdots, x_r\}$, according to their right_endpoints in clockwise direction such that x_1 overlaps y_1 and y_2. In general, x_i overlaps y_i and y_{i+1}. Note that X is an independent set of interval graph C_i, and M_i (obtained through Steps 4-10) is a maximum independent set of C_i. Let's label the arcs in M_i, $M_i = \{m_{i1}, m_{i2}, \cdots, m_{il}\}$, according to the order the arcs are placed in M_i. Observe that m_{ij} never ends after x_j ends (i.e., the right_endpoint of $m_{ij} \leq$ the right_endpoint of x_j) due to the way m_{ij} is picked (Step 8). Since C is a proper circular-arc graph, m_{ij} never begins after x_j begins (i.e., the left_endpoint of $m_{ij} \leq$ the left_endpoint of x_j). So m_{ij} either equals or proceeds x_j in clockwise direction. Figure 6.2 illustrates the relationship among the arcs in sets X, Y, and M_i.

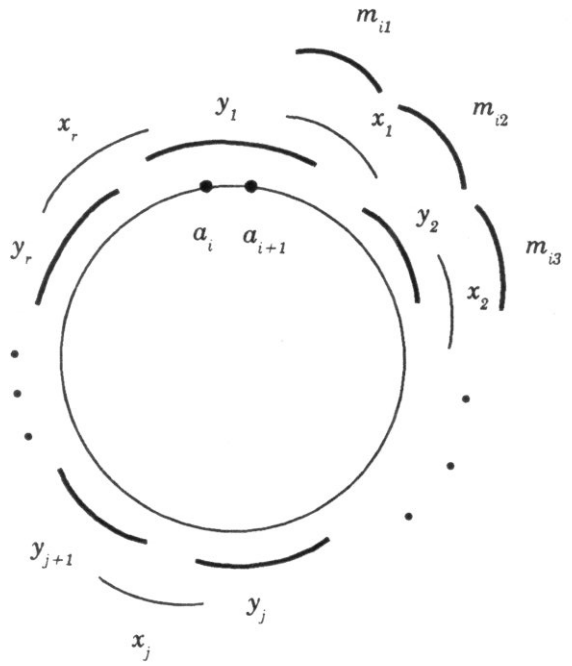

Figure 6.2. Arcs in sets X, Y, and M_i.

Now we claim that the union of two independent sets M_i and Y is also a maximum bipartite subgraph of C. First, we note that the union of M_i and Y is a bipartite subgraph of C. Second, we observe that the size of M_i is no less than size of X. Thus the only way $M_i + Y$ is not a maximum bipartite subgraph of C is when the two sets have arc(s) in common. Without loss of generality let m_{ij} be the first arc in M_i that is in Y, i.e., $m_{ij} = y_k$, for some k. In this case we know m_{ij} proceeds x_j in clockwise direction. Since x_j overlaps y_j and y_{j+1} and $m_{ij} = y_k$, the index j must be equal to or greater than k, i.e., $j \geq k$. Now let's consider the set $Z = \{m_{i1}, m_{i2}, \cdots, m_{ik-1}, x_k, x_{k+1}, \cdots, x_r\}$. Z is an independent set; $|Z| = |X|$; Z is disjoint from Y. Thus $Z + Y$ is a maximum bipartite subgraph of C. Note that m_{ik-1} does not overlap y_k, because $m_{ij} = y_k$, and $j \geq k$. Then $Z + Y$ is an interval graph, since the gap between y_{k-1} and y_k is not completely covered by any of the arcs in Z. This is a contradiction because we assumed that C does not have any maximum bipartite subgraph that is an interval graph. Therefore $M_i + Y$ must be a maximum bipartite subgraph of C.

Using similar arguments as above we can show that if (a_j, a_{j+1}) is an unit interval on the circle such that none of the arcs in Y covers it (this implies that there exists an arc in M_i that covers it), then $M_i + M_j$ is also a maximum bipartite subgraph of C. Therefore we just proved that if all maximum bipartite subgraph of C are even cycles, then there exists a maximum bipartite subgraph M_{ij} of C such that M_{ij} consists of the union of M_i and M_j, for some i, j, $1 \le i,j \le 2n$. Algorithm 6.3 finds $M_i + M_j = M_{ij}$ at Step 11.

□

Before analyzing the running time of Algorithm 6.3, we need to specify how to perform Step 11. First we prove the following:

Lemma 6.1. Given two independent sets M_i and M_j (as defined in Step 4 of Algorithm 6.3) of the same size, if either

1. $last_i$ overlaps $last_j$, and $first_i$ overlaps $last_j$ and $first_j$, or
2. $last_j$ overlaps $last_i$, and $first_j$ overlaps $last_i$ and $first_i$,

then $M_i + M_j$ forms an even cycle.

Proof. Without loss of generality, let's assume that case 1 is true, i.e., 1. M_i and M_j are two independent sets, 2. $|M_i| = |M_j|$, 3. $last_i$ overlaps $last_j$, and 4. $first_i$ overlaps $last_j$ and $first_j$. To prove $M_i + M_j$ forms an even cycle, it suffices to show that I. M_i is disjoint from M_j, and II. arcs in M_i and M_j cover the whole circle.

I. M_i is disjoint from M_j.

Suppose not. Let c be the first arc that is placed in both M_i and M_j. According to the maximum independent set algorithm (Steps 5-9) all the arcs placed in sets M_i and M_j after c are the same, with the possible exception of the last arc that is placed in M_j. Thus $last_i$ is also placed in M_j. Since $first_i$ overlaps $last_j$, $last_j$ is not the same as $last_i$. The above two statements imply $last_i$ does not overlap $last_j$. Contradiction.

II. Arcs in M_i and M_j cover the whole circle.

Suppose not. Without loss of generality let (a_k, a_{k+1}) be an interval on the circle not covered by arcs in either set. The maximum independent set algorithm (Steps 5-9) would have placed the same arcs in both sets M_i and M_j after the point a_{k+1}, with the possible exception of the last arc that is placed in M_j. Using similar arguments as above, we again reach a contradiction.

□

Now we are justified to restate Step 11 of Algorithm 6.3 as: For each set M_i, find a set M_j, if one exists, such that $|M_i| = |M_j|$, and either $last_i$ overlaps $last_j$, and $first_i$ overlaps $last_j$ and $first_j$, or $last_j$ overlaps $last_i$, and $first_j$ overlaps $last_i$ and $first_i$.

After the initial sort (Step 1), which takes $O(n\log n)$ time, Steps 3, 4-10, and 11 all can be carried out in time $O(n^2)$. Therefore we can find a maximum bipartite subgraph of a proper circular-arc graph in $O(n^2)$ time.

The complexity of the maximum bipartite subgraph with a chosen subset problem for proper circular-arc graphs is the same as that for circular-arc graphs, i.e., $O(n\log n + n\Delta)$, where Δ is the maximum number of arcs covering any point on the circle. We will discuss the algorithm for finding a maximum bipartite subgraph with a chosen subset for circular-arc graphs in Section 6.6.

6.6. Circular-Arc Graphs

Assume a circular-arc graph is given in the form of a family of arcs $C= \{c_1, c_2, \cdots, c_n\}$ on a circle, where each arc is denoted by two endpoints – left and right, in clockwise direction. Again without loss of generality we can assume that the $2n$ endpoints of the arcs are distinct.

The approach we take in finding a maximum bipartite subgraph of a circular-arc graph is as follows: For any unit interval (a_i, a_{i+1}) on the circle, exactly one of the following three cases is true of any maximum bipartite subgraph C_B of C:

1. None of the arcs in C_B covers the interval (a_i, a_{i+1}).
2. Exactly one arc in C_B covers the interval (a_i, a_{i+1}).
3. Exactly two arcs in C_B cover the interval (a_i, a_{i+1}).

In the first case, C_B is an interval graph. To obtain C_B we can cut the circle at the interval (a_i, a_{i+1}), and find a maximum bipartite subgraph on the corresponding interval graph. The second and third cases can be handled by considering all possible combinations of arcs covering interval (a_i, a_{i+1}), cutting the circle at the interval (a_i, a_{i+1}), and solving the corresponding two-end-limited 2-track assignment problem (the exact details are given in Algorithm 6.4). Not knowing which of the three cases holds for any particular interval (a_i, a_{i+1}), we shall solve them all and take the maximum bipartite subgraph we find. An unit

interval (a_i, a_{i+1}) best suited for the above approach is one with the least number of arcs covering it in C. All these steps are described in Algorithm 6.4.

Input. A circular-arc graph C given as a set of arcs, $C = \{c_1, c_2, \cdots, c_n\}$.

Output. A maximum bipartite subgraph of C.

1. Sort the $2n$ endpoints in clockwise direction and output them in a list α, $\alpha=\{a_1, a_2, \cdots, a_{2n}\}$, where $a_{2n+1} = a_1$.
2. For i = 1 to $2n$, determine D_i.
   D_i is defined as the set of arcs covering the interval (a_i, a_{i+1}).
3. Determine k such that $|D_k| = \min\{|D_i|, 1 \le i \le 2n\}$.
4. Obtain a maximum bipartite subgraph of the interval graph $C - D_k$ by running Algorithm 6.1.
5. For each arc $c_i = (a_{l_i}, a_{r_i})$ in D_k do
   (c_i is the only arc covering the interval (a_k, a_{k+1}) in the bipartite subgraph.)
   6. Solve the corresponding two-end-limited 2-track assignment problem of C (Algorithm 6.2) by mapping endpoint a_{k+1} to 0, a_{k+2} to 1, ... , a_k to $2n-1$, and numbering track1, track2 from 0 to $2n-1$, $|r_i-k|$ to $|2n-k+l_i|$, respectively.
   7. Let G_i be the set of arcs placed on track1 and track2 in Step 6 plus arc c_i.
8. For each pair of arcs $c_i = (a_{l_i}, a_{r_i})$ and $c_j = (a_{l_j}, a_{r_j})$ in D_k do
   (both arcs c_i and c_j cover the interval (a_k, a_{k+1}) in the bipartite subgraph.)
   9. Solve the corresponding two-end-limited 2-track assignment problem of C (Algorithm 6.2) by mapping endpoint a_{k+1} to 0, a_{k+2} to 1, ... , a_k to $2n-1$, and numbering track1, track2 from $|r_i-k|$ to $|2n-k+l_i|$, $|r_j-k|$ to $|2n-k+l_j|$, respectively.
   10. Let G_{ij} be the set of arcs placed on track1 and track2 in Step 9 plus arcs c_i and c_j.
11. Return the maximum bipartite subgraph produced in Steps 4, 7, and 10.

Algorithm 6.4. Maximum bipartite subgraph algorithm for circular-arc graphs.

Let's define δ to be the minimum number of arcs covering any particular interval (a_i, a_{i+1}) on the circle; $\delta = |D_k|$ in Algorithm 6.4. We shall parametrize the running time of Algorithm 6.4 in terms of δ. After the $2n$ endpoints are sorted (Step 1), the second *for* loop (Steps 5-7) takes $O(n\delta)$ time, since each call to Algorithm 6.2 takes linear time. The third *for* loop (Steps 8-10) takes $O(n\delta^2)$ time, since there are δ^2 number of pairs of arcs in D_k to consider. Thus the total running time of Algorithm 6.4 is $O(n\delta^2)$. Note that δ is upper bounded by the minimum degree of vertices in C. Clearly δ^2 is upper bounded by e

– the number of edges in circular-arc graph C. δ^2 is also upper bounded by the number of edges in a minimum maximal clique of C.

Circular-arc graphs is one class of graphs where the complexity of finding a maximum bipartite subgraph is reduced, from $O(n\delta^2)$ to $O(n\log n + n\Delta)$, when a chosen subset is included (the size of the chosen subset is greater than 0), where Δ is the maximum number of arcs covering any point on the circle. The reason is that if an arc is included in the bipartite subgraph, then we can narrow the searching space for optimal solutions. First, we observe that if there exist two overlapping arcs in the chosen subset, then we can cut the circle at a point of overlap and reduce the original problem on a circular-arc graph to a two-end-limited 2-track assignment problem with the chosen subset (the chosen subset will not include the two overlapping arcs). The complexity of the algorithm in this case is $O(n\log n)$. Second, we note that given that an arc (a_i, a_j) is included in the bipartite subgraph, then in any maximum bipartite subgraph the unit interval (a_i, a_{i+1}) is covered with exactly one arc or exactly two arcs on the circle. We have dealt with this situation previously in circular-arc graphs, but now we know arc (a_i, a_j) is in the bipartite subgraph. Thus we only have to consider all possible arcs (other than (a_i, a_j)) covering or not covering the unit interval (a_i, a_{i+1}), and solve the corresponding two-end-limited 2-track assignment problem with a chosen subset. The number of arcs we need to consider is at most Δ. Note that $\Delta \le d \le n$, where d is the maximum degree of vertices in C. Hence the complexity of finding a maximum bipartite subgraph with a chosen subset for circular-arc graphs and for proper circular-arc graphs is $O(n\log n + n\Delta)$.

6.7. Permutation Graphs

Suppose that $\Pi = [\pi_1, \pi_2, \cdots, \pi_n]$ is a permutation of n numbers. Let π_i^{-1} denote the position in the sequence where the number i can be found. A *permutation* graph $G[\Pi]$ is an undirected graph on n vertices such that vertex i is adjacent to vertex j if the larger of $\{i, j\}$ appears to the left of the smaller in Π. More formally, the graph $G[\Pi] = (V, E)$ is defined as follows:

$$V = \{1, 2, \ldots, n\}$$

and

$$ij \in E \Leftrightarrow (i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0.$$

An undirected graph G is a permutation graph if there exists a permutation Π such that G is isomorphic to $G[\Pi]$.

As mentioned in the preliminaries, permutation graphs are also characterized as comparability graphs whose complements are also comparability graphs. Permutation graphs are a proper subclass of comparability (or transitively orientable) graphs which are a proper subclass of perfect graphs. To obtain a transitive orientation $\vec{G}[\Pi]$ of a permutation graph $G[\Pi]$, we simply orient each edge toward the larger vertex. Note that after orientation, the resulting directed graph must be acyclic. A *transitively reduced* graph $\vec{G}_t[\Pi]$ of $\vec{G}[\Pi]$ is a directed graph that consists of vertices in $\vec{G}[\Pi]$, and an arc (a,c) is in $\vec{G}_t[\Pi]$ if and only if (a,c) is in $\vec{G}[\Pi]$ and there exists no vertex b such that (a,b) and (b,c) are in $\vec{G}[\Pi]$. Note that $\vec{G}_t[\Pi]$ is a subgraph (not induced) of $\vec{G}[\Pi]$. An interesting property of permutation graphs is that the complement of a permutation graph is another permutation graph. If we let Π^r be the reverse of permutation Π, then it is clear $\overline{G[\Pi]} = G[\Pi^r]$.

Given a permutation graph G, a suitable permutation Π can be constructed in time $O(de + n^2)$ [Golumbic, 1980], where d is the maximum degree of a vertex. The transitively reduced graph $\vec{G}_t[\Pi]$ can be constructed from Π in $O(n^2)$ time. In the following, we will assume that a permutation graph G is given as $G[\Pi]$.

Two elementary properties of permutation graphs $G[\Pi]$ are: 1. the increasing subsequences of Π and the independent sets of $G[\Pi]$ are in one-to-one correspondence, and 2. the decreasing subsequences of Π and the cliques of $G[\Pi]$ are in one-to-one correspondence. Thus a maximum bipartite subgraph of $G[\Pi]$ corresponds to a longest subsequence Π_1 of Π such that Π_1 is composed of two increasing subsequences of Π.

In developing an algorithm for finding a maximum bipartite subgraph of $G[\Pi]$, we will be working with the complement of the permutation graph, i.e., $G[\Pi^r]$. More specifically, we will be dealing with the transitive directed graph $\vec{G}[\Pi^r]$, which is obtained from $G[\Pi^r]$ by orienting each edge toward the larger vertex. Now if $(\pi_a{}^r, \pi_b{}^r)$ is a directed edge in $\vec{G}[\Pi^r]$, then $\pi_b{}^r > \pi_a{}^r$ and $b < a$. Note that the directed paths in $\vec{G}[\Pi^r]$ are in the opposite direction as the subsequences in Π^r, and this is done for simplicity of describing the algorithm. Directed paths in $\vec{G}[\Pi^r]$ and cliques of $G[\Pi^r]$ are in one-to-one correspondence; equivalently directed paths in $\vec{G}[\Pi^r]$ and independent sets of of $G[\Pi]$ are in one-to-one correspondence. Our strategy for finding a maximum bipartite subgraph of $G[\Pi]$ is to look for 2 directed paths in $\vec{G}[\Pi^r]$ containing the maximum number of vertices. Note that the 2 directed paths need not be disjoint.

To find 2 such directed paths in $\vec{G}[\Pi^r]$, we first compute, for each pair of vertices u and v, a *best-pair-of-paths* from u and v in $\vec{G}[\Pi^r]$, $1 \le u,v \le n$. A best-pair-of-paths from vertices u and v in $\vec{G}[\Pi^r]$ is defined as 2 directed paths starting from vertices u and v and containing the maximum number of vertices in $\vec{G}[\Pi^r]$. A best-pair-of-paths that contains the maximum number of vertices among all best-pair-of-paths in $\vec{G}[\Pi^r]$ corresponds to a maximum bipartite subgraph of $G[\Pi]$.

An interesting observation we make is that all best-pair-of-paths in $\vec{G}[\Pi^r]$ use only edges in the transitively reduced graph $\vec{G}_t[\Pi^r]$ of $\vec{G}[\Pi^r]$. For if there exists a directed path from u to v in $\vec{G}[\Pi^r]$, then there exists a directed path from u to v traversing only transitively reduced edges that is as long as any other directed path from u to v in $\vec{G}[\Pi^r]$. Thus we only have to deal with the transitively reduced graph $\vec{G}_t[\Pi^r]$ when computing a maximum bipartite subgraph of $G[\Pi]$.

We now present a simple algorithm, based on dynamic programming, for finding a maximum bipartite subgraph of a permutation graph. It goes as follows:

Input. A permutation graph $G[\Pi]$.

Output. A maximum bipartite subgraph of $G[\Pi]$.

1. Let Π^r, the reverse of Π, be indexed as: $\Pi^r = [\pi_1^r, \pi_2^r, \cdots, \pi_n^r]$.
2. Construct the transitively reduced graph $\vec{G}_t[\Pi^r]$ of $\vec{G}[\Pi^r]$.
   The edges in $\vec{G}_t[\Pi^r]$ are always oriented toward the larger vertex.
3. For i = 1 to n, do
   4. For $j = 1$ to i, compute a *best-pair-of-paths* from vertices π_i^r and π_j^r in $\vec{G}_t[\Pi^r]$.
5. Return a best-pair-of-paths from π_a^r and π_b^r in $\vec{G}_t[\Pi^r]$, for some π_a^r and π_b^r, $1 \le \pi_a^r, \pi_b^r \le n$, that contains the maximum number of vertices among all best-pair-of-paths computed in Step 4.

Algorithm 6.5. Maximum bipartite subgraph algorithm for permutation graphs.

Algorithm 6.5 correctly solves the maximum bipartite subgraph problem because the best-pair-of-paths computed in Step 5 corresponds to a longest subsequence Π_1 of Π such that Π_1 is composed of two increasing subsequences of Π which corresponds to a maximum bipartite subgraph of $G[\Pi]$.

The dynamic programming part of Algorithm 6.5 comes into play in Step 4, and it proceeds as follows:

1. A best-pair-of-paths from π_i^r and π_j^r (j < i) = a best of {best-pair-of-paths from π_k^r and π_j^r, where π_k^r is a vertex pointed to by π_i^r} plus the vertex π_i^r.
2. A best-pair-of-paths from π_i^r and π_i^r, = a best of {best-pair-of-paths from π_k^r and π_i^r, where π_k^r is a vertex pointed to by π_i^r}.

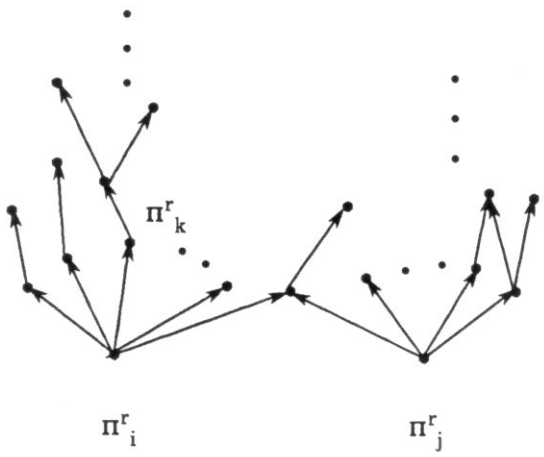

Figure 6.3. A best-pair-of-paths from Π^r_i and Π^r_j through Π^r_k.

Basically a best-pair-of-paths from π_i^r and π_j^r must go through some vertex, say π_k^r, pointed to by π_i^r. See Figure 6.3 for illustration. Note that π_j^r may also be pointed to by π_i^r. Due to dynamic programming, a best-pair-of-paths from π_k^r and π_j^r, k, $j < i$, has been determined earlier, and that information is readily available. If there exist more than one best-pair-of-paths from π_i^r and π_j^r, then we can freely choose one. The amount of work needed to compute a best-pair-of-paths from π_i^r and π_j^r, for some $j < i$, is bounded by the outdegree of vertex π_i^r in the transitively reduced graph $\vec{G}_t[\Pi^r]$. Thus the total running time of Algorithm 6.5 is $O(\sum_v nd_v) = O(n\overline{e_t})$, where d_v is the outdegree of vertex v and $\overline{e_t}$ is the number of edges in $\vec{G}_t[\Pi^r]$.

Due to the nature of dynamic programming, we can easily adapt Algorithm 6.5, with some simple modifications, to find a maximum bipartite subgraph with a chosen subset for permutation graphs. Let S be the chosen set, $S = \{\pi_a^r, \pi_b^r, \cdots, \pi_s^r\}$, $a < b < \cdots < s$. Because all directed paths $\pi_{v_1} \rightarrow \pi_{v_2} \rightarrow \cdots \rightarrow \pi_{v_l}$ in $\vec{G}_t[\Pi^r]$ have the property that $\pi_{v_1} < \pi_{v_2} < \cdots < \pi_{v_l}$ and $v_1 > v_2 > \cdots > v_l$, a best-

pair-of-paths from π_i^r and π_j^r, $j \le i$, must include all those vertices π_h^r of S such that $h \le i$, if they exist. For if a vertex π_h^r of S, $h \le i$, is not included in a best-pair-of-paths from π_i^r and π_j^r, $j \le i$, then all best-pair-of-paths that build on top of best-pair-of-paths from π_i^r and π_j^r will not include the vertex π_h^r either. To find a maximum bipartite subgraph that includes S, we need to modify Step 4 of Algorithm 6.5 as follows: For $j = 1$ to i, compute a *best-pair-of-paths* from vertices π_i^r and π_j^r in $\vec{G}_t[\Pi^r]$ that includes all those vertices π_h^r of S with $h \le i$. This modified Step 4 can be computed by considering only those vertices π_k^r that are pointed to by π_i^r such that a best-pair-of-paths from π_k^r and π_j^r plus the vertex π_i^r include all those vertices π_h^r of S with $h \le i$, if such a best-pair-of-paths exists. Of course the best-pair-of-paths determined in Step 5 of Algorithm 6.5 must include the whole set S. The complexity of this modified algorithm is still $O(n\bar{e_t})$, where $\bar{e_t}$ is the number of edges in $\vec{G}_t[\Pi^r]$.

6.8. Split Graphs

A graph $G = (V , E)$ is called a *split* graph if V can be partitioned into two subsets I and C such that I is an independent set and C induces a complete graph. Split graphs are also characterized as chordal graphs whose complements are also chordal graphs. Split graphs are a proper subclass of chordal graphs. It is clear that the complement of a split graph is also a split graph.

In this section we present a simple algorithm for finding a maximum bipartite subgraph of a split graph. First we make the following observation: If we partition the vertex set V of a split graph into two subsets I and C such that I is an independent set and C induces a complete graph (for example, if we use a maximum independent set algorithm to find I), then any maximum bipartite subgraph of G contains at least one and at most two vertices from C. Thus our task becomes: Find a bipartite subgraph of size $|I| + 2$, if it exists. Otherwise the set I plus any vertex in C is a maximum bipartite subgraph of G. The algorithm for finding a maximum bipartite subgraph of G goes as follows:

Input. A split graph $G = (V, E)$.

Output. A maximum bipartite subgraph of G.

1. Find a maximum independent set of G using Gavril's[1972] algorithm.
   Let I_{max} denote the maximum independent set, and C denote the set of vertices that induces a complete graph, $I_{max} + C = V$.

2. Determine M which is defined to be the adjacency matrix with $(M)_{u,v} = 1$ if and only if $(u,v) \in E$, $u \in I_{\max}$, and $v \in C$.
3. Calculate M^2 which is the boolean multiplication of M with itself.
4. If there exist $v_i, v_j \in C$ such that $(M^2)_{v_i,v_j} = 0$, then
   5. Let $B = I_{\max} + v_i + v_j$.
   Otherwise
   6. Let $B = I_{\max} + v$, for any v in C.
7. Return B.

Algorithm 6.6. Maximum bipartite subgraph algorithm for split graphs.

To see Algorithm 6.6 correctly finds a maximum bipartite subgraph of a split graph G, we note that two vertices v_i, v_j in C form a bipartite subgraph with $I_{\max}$ only if there exists no vertex u in $I_{\max}$ such that u is adjacent to both v_i and v_j, in other words, only if the length of the shortest path between v_i and v_j in the subgraph (actually a bipartite subgraph) determined by the adjacency matrix M (Step 2) is more than two, or $(M^2)_{v_i,v_j} = 0$.

Gavril's maximum independent set algorithm on chordal graphs takes $O(n + e)$ time when it is combined with a perfect vertex elimination scheme of the same time bound (an $O(n + e)$ algorithm of Rose, Tarjan, and Lueker[1976] will do). Thus the running time of Algorithm 6.6 is dominated by matrix multiplication (Step 3), in other words, the complexity of Algorithm 6.6 is equivalent to that of matrix multiplication. Presently the most efficient, asymptotically speaking, matrix multiplication algorithm runs in time $O(n^{2.376})$ [Coppersmith and Winograd, 1987]. (To be more precise, an nxn matrix may be multiplied using $O(n^{2.376})$ arithmetical operations; 2.376.. is the *matrix exponent*. For a more practical matrix multiplication algorithm – $O(n^{2.807})$, see Strassen[1969]). Therefore we can find a maximum bipartite subgraph for a split graph in time $O(n^{2.376})$.

Adding the constraint that a maximum bipartite subgraph must include a chosen subset of vertices may simplify the problem when we are dealing with split graphs. For we know a bipartite subgraph can include at most two vertices from C, and even knowing one vertex in C must be included in the bipartite subgraph will narrow down our choices. The complexity of finding a maximum bipartite subgraph with a chosen subset for split graphs, however, remains the same as the complexity of matrix multiplication, since the chosen subset may contain only vertices that are in I, and in which case we will have to use

Algorithm 6.6.

In Algorithm 6.6 we have reduced the problem: find a bipartite subgraph of size $|I| + 2$, if it exists, to the problem: determine if there exist two vertices in C such that the shortest path between those two vertices is more than 2, and we solved the latter by matrix multiplication. Now we observe that the second problem falls in the framework of a more general question: is the diameter of a bipartite graph larger than 2? The diameter of a graph is defined as the length of any longest shortest path between 2 vertices in the graph. The complexity of this problem is the same as that of matrix multiplication. An open problem is whether we can improve the running time of this algorithm, or whether we can use a more combinatorial approach and achieve a time bound better than $O(ne)$ (i.e., without using matrix multiplication). Interestingly, the complexity of determining whether the diameter of a bipartite graph is greater than k, for any fixed k, is the same as that of matrix multiplication; however, when k is not fixed, the complexity becomes $O(ne)$ using breadth first search or $O(n^3(\log\log n/\log n)^{1/3})$ using an algorithm of Fredman[1976].

6.9. Concluding Remarks and Suggestions for Future Research

In this chapter we have used the dynamic programming technique on numerous occasions to solve the maximum bipartite subgraph problem, including interval graphs, 2-track assignment problems, circular-arc graphs, and permutation graphs (the greedy approach of Algorithms 6.1 and 6.2 can be considered as dynamic programming). In fact, dynamic programming is also employed to solve the same problem for chordal graphs, by Yannakakis and Gavril[1987]. We believe the approaches of Bern, Lawler, and Wong[1987] and Baker[1983], both of which use dynamic programming, can be applied to find a maximum bipartite subgraph for outerplanar and k-outerplanar graphs, respectively; although we have not checked the details. Dynamic programming has proven itself to be a powerful technique. However, it is not powerful enough (at least it has not been shown to be) to solve the maximum bipartite subgraph problem for all classes of graphs when such polynomial-time algorithms exist, as demonstrated by Frank[1980] for comparability graphs. Another interesting observation we make is that the decomposition approach in dynamic programming in each class of graphs is different. Usually we have put the graph in proper representation before the decomposition approach becomes apparent. One such example is permutation graphs; in Algorithm 6.5 the permutation graph is built as a transitively reduced transitive

directed graph to be operated on by dynamic programming. Another example is chordal graphs; in the algorithm of Yannakakis and Gavril[1987], a chordal graph is first represented as an intersection graph of a family of subtrees in a tree before the algorithm proceeds.

Due to the nature of dynamic programming, we can easily adapt the algorithms based on dynamic programming for finding a maximum bipartite subgraph, with some simple modifications, to find a maximum bipartite subgraph with a chosen subset, as we have done for interval graphs, 2-track assignment problems, circular-arc graphs, and permutation graphs. In the same spirit, the algorithm of Yannakakis and Gavril[1987] for chordal graphs can also be modified to find a maximum bipartite subgraph with a chosen subset by considering only those subgraphs (cliques) such that if a vertex is in the chosen subset, then that vertex is colored in the subgraph, i.e., included in the maximum bipartite subgraph.

In this chapter we have presented efficient algorithms for finding a maximum bipartite subgraph for various classes of graphs; however, the complexity of this problem for some important classes of graphs still remain open, including perfect graphs. For perfect graphs, both independent set and chromatic number problems are known to be polynomial-time solvable via the ellipsoid method (see Grötschel, Lovász, and Schrijver[1981]). These results may or may not provide a clue to the complexity of the maximum bipartite subgraph problem for perfect graphs. In any event, determining its complexity can be interesting and challenging, and is left for future research. In addition, the complexity of the maximum bipartite subgraph with a chosen subset problem for comparability graphs and series-parallel graphs are open, and deserve further studies.

In this chapter we have restricted our attention to those classes of graphs that have polynomial-time algorithms for the maximum bipartite subgraph problem. Another possible direction of study is in the development of approximation algorithms or heuristics for this problem, meaning algorithms that produce a bipartite subgraph of a certain size, possibly parameterized by the chromatic number, for all graphs. This is another interesting and challenging problem left for future studies.

Chapter 7

Conclusions and Future Work

In this dissertation we have designed algorithms for approximate graph coloring with good performance by examining more global structures and properties of graphs than previously examined. We have investigated the role played by odd cycles of graphs in connection with graph coloring and found that the presence or absence of certain size odd cycles gives graphs more structure and hence simplifies graph coloring. More specifically, we have shown that the "local" neighborhood of a smallest odd cycle of a graph is bipartite. The locality of this bipartite neighborhood depends on the size of a smallest odd cycle – k. The larger the value k is, the larger is the bipartite neighborhood. Thus the absence of small odd cycles enables us to find large bipartite subgraphs. On the other hand, the absence of large odd cycles in a biconnected graph implies the absence of large even cycles, and these conditions imply that the graph contains special structures. Hence graph coloring is simplified due to the special structures. We have presented efficient graph coloring algorithms that improve the performance guarantee on certain graphs, sometimes by a large margin, over the existing approximate graph coloring algorithms, because we have exploited some favorable global structures and properties of graphs.

Another factor that has contributed to the success of the graph coloring algorithms presented in this dissertation is the approach, which is: find large bipartite subgraphs. However, it is the combination of identifying global structures of a graph and finding large bipartite subgraphs that gives rise to a "working" algorithm with a guaranteed performance.

Despite the improved performance guarantee, the graph coloring algorithms presented in this dissertation have their limitations. For one, the algorithms are not "general" graph coloring algorithms, for they do not provide good performance guarantee on all graphs. The algorithms in Chapter 4 are designed for triangle-free graphs, while in contrast the algorithms in Chapter 5 are designed for graphs with only

small odd cycles. How do we color graphs that have both large and small odd cycles? Can we extend the techniques developed in this dissertation to handle general graphs? Is it possible to combine different methods so that all cycle sizes are covered? Also can we describe the performance of graph coloring algorithms in terms of the chromatic number of the graph? These are some of the interesting questions and problems that arise from this work, and they pose challenges for future research.

In this dissertation, we have obtained some global structures and properties of graphs by focusing on odd cycles of graphs. However, we do not have a complete picture on the role played by odd cycles of a graph in connection with graph coloring, for example, Conjecture 5.1 is unresolved. We have begun the investigation. We hope our results will bring interest to this problem, so that the investigation continues.

Because we have been focusing on a smallest odd cycle of a graph in Chapters 3 and 4, we are interested in methods of finding one efficiently. As we commented in Chapter 2, Itai and Rodeh[1978] have reduced the problem of finding a smallest odd cycle to that of matrix multiplication. Can this bound be reduced? An open problem is whether it is possible to find a smallest odd cycle in a graph more efficiently, possibly in $O(n^2)$ time without using matrix multiplication. In contrast, a smallest even cycle in a graph can be found in $O(n^2)$ time. Monien[1983] has presented an algorithm that finds a smallest even cycle of a graph in time $O(n^2 \cdot \min\{A(n), \lambda(G)\})$, where A is the inverse Ackermann-function and $\lambda(G)$ is the length of a smallest even cycle of G. The presence of inverse Ackermann's function in the running time is due to the use of an Union-Find algorithm. An observation we make is that the structure of the ''union tree'' in Monien's algorithm is fixed, so that the structure of the unions is known in advance, hence we can apply the algorithm of Gabow and Tarjan[1985] to obtain a linear time bound for disjoint set union in Monien's algorithm. Therefore we can find a smallest even cycle of a graph in time $O(n^2)$ by combining the algorithms of Monien[1983] and of Gabow and Tarjan[1985].

In Chapter 5, we have designed algorithms for coloring graphs with only small odd cycles by analyzing the structures of those graphs. For graphs with only 3-cycles in odd cycles, we have shown a complete characterization of their structures. For other graphs we used breadth-first search as a tool to help us simplify the structures. To analyze the structures of graphs, we performed case studies. As the complexity of the structures of graphs increases, however, we find the technique of case analysis unsatisfactory, for it does not extend well. An approach that is general and can be uniformly applied is certainly

more desirable in this case. Further study is needed in this direction.

In this dissertation, we have investigated the interplay of three different topics: odd cycles, bipartite subgraphs, and approximate graph coloring. Certainly other combinations of topics are possible. Are there other structures of graphs which we can exploit to aid graph coloring? This is an interesting question and is left for future work.

We have been interested in more theoretical issues in this dissertation. We have presented coloring algorithms with guaranteed performance, rather than heuristics which seem to run well. We have designed algorithms that find maximum bipartite subgraphs efficiently, asymptotically speaking. In practice, however, what does one expect in terms of performance of these algorithms? Are they still efficient? We leave this as an open problem.

References

Aho, A. V., J. E. Hopcroft, and J. D. Ullman [1974]

The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA.

Akers, S. B. [1972]

Routing, in *Design Automation of Digital Systems - Vol. 1: Theory and Techniques*, M. A. Breuer (ed.), Prentice-Hall Inc., Englewood Cliffs, NJ, pp.283-333.

Appel K., and W. Haken [1976]

Every planar map is four colorable, *Bull. Amer. Math. Soc.*, **82**, pp.711-712.

Appel K., and W. Haken [1977]

Every planar map is four colorable: Part I: "Discharging", *Illinois J. Math.*, **21**, pp.429-490.

Appel K., and W. Haken [1986]

The four color proof suffices, *Math Intelligencer*, **8** (1), pp.10-20.

Appel K., W. Haken, and J. Koch [1977]

Every planar map is four colorable: Part II: "Reducibility", *Illinois J. Math.*, **21**, pp.491-567.

Baker, B. S. [1983]

Approximation algorithms for NP-complete problems on planar graphs, *Proc. 24th Annual Symposium on Foundations of Computer Science*, pp.265-273.

Bárány, I. [1978]

A short proof of Kneser's conjecture, *J. Combinatorial Theory A*, **25**, pp.325-326.

Berge, C. [1973]

Graphs and Hypergraphs, North Holland, Amsterdam and London.

Berger, B., and J. Rompel [1988]

A better performance guarantee for approximate graph coloring, *Manuscript*, June, 1988.

Bern, M. W., E. L. Lawler, and A. L. Wong [1987]

Linear-time computation of optimal subgraphs of decomposable graphs, *J. Algorithms*, **8**, pp.216-235.

Bollobás, B. [1978]

Extremal Graph Theory, Academic Press, New York, New York.

Bollobás, B., and N. Sauer [1976]

Uniquely colourable graphs with large girth, *Can. J. Math.*, vol **28**, no 6, pp.1340-1344.

Bondy, J. A., and U. S. R. Murty [1976]

Graph Theory with Applications, North-Holland, New York, New York.

Booth, K. S., and G. S. Lueker [1976]

Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *J. Comput. System Sci.*, **13**, pp.335-379.

Broder, S. [1964]
Final examination scheduling, *CACM*, vol **7**, no 8, pp.494-498.

Brooks, R. L. [1941]
On colouring the nodes of a network. *Proc. Cambridge Philos. Soc.* **37**, pp.194-97.

Brown, J. R. [1972]
Chromatic scheduling and the chromatic number problem, *Management Science,* vol **19**, no 4, pp.456-463.

Chaitin, G. J. [1982]
Register allocation & spilling via graph coloring, *Sigplan Not.*, vol **17**, no 6, pp.98-105.

Chaitin, G. J., M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein[1981]
Register allocation via coloring, *Computer Languages,* **6**, pp.47-57.

Christofides, N. [1975]
Graph Theory, Academic Press, New York, New York.

Coleman, T. F., and J. J. Moré [1981]
Estimation of sparse jacobian matrices and graph coloring problems, ANL-81-39, Argonne National Lab., June 1981.

Coppersmith, D., and S. Winograd [1987]
Matrix multiplication via arithmetic progressions, *Proc. 19th Annual ACM Symposium on Theory of Computing*, pp.1-6.

Curtis, A. R., M. J. D. Powell, and J. K. Reid [1974]
On the estimation of sparse jacobian matrices, *J. Inst. Maths. Appl.*, **13**, pp.117-119.

Dailey, D. P. [1980]
Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete, *Discrete Math.*, **30**, pp.289-293.

Descartes, B. [1947]
A three-colour problem, *Eureka,* **9**, April, 1947.

Descartes, B. [1948]
Solutions to problems in Eureka No. 9, *Eureka,* **10**, March, 1948.

Descartes, B .[1954]
Solution to advanced problem No. 4525, *Amer. Math. Monthly,* **61**, pp.532.

Dyer, M. E., and A. M. Frieze [1986]
Fast solution of some random NP-hard problems, *Proc. 27th Annual Symposium on Foundations of Computer Science*, pp.331-336.

Edwards, K. [1986]
The complexity of colouring problems on dense graphs, *Theoretical Computer Science,* **43**, pp.337-343.

Eilon, S., and N. Christofides [1971]
The loading problem, *Management Science,* **17**, pp.259.

Erdös, P. [1959]

Graph theory and probability, *Canad. J. Math.*, **11**, pp.34-38.

Erdös, P. [1961]

Graph theory and probability II, *Canad. J. Math.*, **13**, pp.346-352.

Erdös, P. [1962]

On circuits and subgraphs of chromatic graphs, *Mathematika*, **9**, pp.170-175.

Erdös, P. [1979]

Problems and results in graph theory and combinatorial analysis, in *Graph Theory and Related Topics*, edited by J. A. Bondy and U.S.R. Murty, Academic Press, pp.153-163.

Erdös, P., and A. Hajnal [1966]

On chromatic numbers of graphs and set systems, *Acta math. Sci. Hungar.*, **17** (1-2), pp.61-99.

Erdös, P., and A. Hajnal [1985]

Chromatic number of finite and infinite graphs and hypergraphs, *Discrete Math.*, **53**, pp.281-285.

Erdös, P., and H. Sachs [1963]

Reguläre graphen gegenebener teillenweite mit minimaler knotenzahl, Wiss. Z. Univ. Halle - Wittenberg, *Math. Nat. R.*, **12**, pp.251-258.

Frank, A. [1980]

On chain and antichain families of a partially ordered set, *J. Combin. Theory Ser. B*, **29**, pp.176-184.

Fredman, M. L. [1976]

New bounds on the complexity of the shortest path problem, *SIAM J. Comput.*, **5**, pp.83-89.

Gabow, H. N., and R. E. Tarjan [1985]

A linear-time algorithm for a special case of disjoint set union, *J. Comput. and Sys. Sci.*, **30**, pp.209-221.

Gallai, T. [1973]

Kritische graphen, *Publ. Math. Inst. Hungar. Acad. Sci.*, **8**, pp.165-192.

Garey, M. R., and D. S. Johnson [1976]

The complexity of near-optimal graph coloring, *JACM*, vol **23**, no 1, pp.43-49.

Garey, M. R., D. S. Johnson, and H. C. So [1976]

An application of graph coloring to printed circuit testing, *IEEE Trans. on Circuits and Systems*, **23**, pp.591-598.

Garey M. R., D. S. Johnson, and L. J. Stockmeyer [1976]

Some simplified NP-complete graph problems, *Theor. Comp. Sci.*, **1**, pp.237-267.

Gavril, F. [1972]

Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph, *SIAM J. Comput.*, **1**, pp.180-187.

Golumbic, M. [1980]

Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York.

Grimmett, G. R., and C. J. H. McDiarmid [1975]
On colouring random graphs, *Math. Proc. Camb. Phil. Soc.*, **77**, pp.313-324.

Grötschel, M., L. Lovász, and A. Schrijver [1981]
The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, **1**, pp.169-197.

Grötzsch, H. [1958]
Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg. Math.-Nat. Reihe*, **8**, pp.109-119.

Gupta, U. I., D. T. Lee, and Y.-T. Leung [1982]
Efficient algorithms for interval graphs and circular-arc graphs, *Networks*, **12**, pp.459-467.

Hall, A. D., and F. S. Acton [1967]
Scheduling university course examinations by computer, *CACM*, vol **10**, no 4, pp.235-238.

Harary, T. [1969]
Graph Theory, Addison-Wesley, Reading, Massachusetts.

Imrich, W. [1984]
Explicit construction of regular graphs with no small cycles, *Combinatorica*, **4**, pp.53-59.

Itai, N., and M. Rodeh [1978]
Finding a minimum circuit in a graph, *SIAM J. Comput.*, vol **7**, no 4, pp.413-423.

Johnson, D. S. [1974]
Worst case behavior of graph coloring algorithms, in: *Proceedings of the 5th S-E Conference on Combinatorics, Graph Theory and Computing, Congr. Num.*, **10**, pp.513-527.

Johnson, D. S. [1985]
The NP-completeness column: an ongoing guide, *J. Algorithms*, **6**, pp.434-451.

Karp, R. M. [1972]
Reducibility among combinatorial problems, in: *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (eds.), Plenum Press, New York, pp.85-103.

Kelly, J. B., and L. M. Kelly [1954]
Paths and circuits in critical graphs, *Amer. J. Math.*, **76**, pp.786-792.

Kierstead, H. A., E. Szemerédi, and W. T. Trotter [1984]
On coloring graphs with locally small chromatic number, *Combinatorica*, **4**, pp.183-185.

Kneser, M. [1955]
Aufgabe 300, *Jahresbericht. Deutschen Math.-Verein.*, **58**, pp.27.

König, D. [1936]
Theorie der endlichen und unendlichen Graphen, Leipzig, 1936, Reprinted Chelsea, New York, 1950.

Kučera, L. [1977]
Expected behavior of graph coloring algorithms, in *Fundamentals of Computation Theory*, Springer-Verlag Lecture Notes in Computer Science, no 56, pp.447-451.

Lewis, J. M., and M. Yannakakis [1980]
The node-deletion problem for hereditary properties is NP-Complete, *J. Comp. System Science,* **20**, pp.219-230.

Lovász, L. [1968]
On chromatic number of finite set-systems, *Acta Math. Acad. Sci. Hungar.*, **19**, pp.59-67.

Lovász, L. [1978]
Kneser's conjecture, chromatic number, and homotopy, *J. Combinatorial Theory A,* **25**, pp.319-324.

Lovász, L. [1983]
Self-dual polytopes and chromatic number of distance graphs on the sphere, *Acta Sci. Math. Szeged.*, **45**, pp.317-323.

Lubotzky, A., R. Phillips, and P. Sarnak [1986]
Explicit expanders and the Ramanujan conjectures, *Proc. of 18th ACM Symp. on Theory of Computing,* pp.240-246.

Lubotzky, A., R. Phillips, and P. Sarnak [1988]
Ramanujan graphs, *Combinatorica,* **8** (3), pp.261-277.

Manber, R., and G. Narasimhan [1987]
Algorithms for the maximum induced bipartite subgraph problem on interval and circular-arc graphs, U. of Wisconsin - Madison CS Technical Report #696, April, 1987.

Manvel, B. [1985]
Extremely greedy coloring algorithms, in *Graphs and Applications, Proceedings of the First Colorado Symposium on Graph Theory*, edited by F. Harary and J. S. Maybee, John Wiley, pp.257-270.

Margulis, G. A. [1982]
Graphs without short cycles, *Combinatorica,* **2**, pp.71-78.

Matula, D. W., G. Marble, and J. D. Isaacson [1972]
Graph coloring algorithms, in *Graph Theory and Computing*, edited by R. C. Read, Academic Press, pp.109-122.

May, K. O. [1965]
The origin of the four-color conjecture, *Isis,* **56**, pp.346-348.

McDiarmid, C. J. H. [1979]
Colouring random graphs badly, in *Graph Theory and Combinatorics*, edited by R. J. Wilson, Pitman Research Notes in Mathematics, **34**, pp.76-86.

Monien, B. [1983]
The complexity of determining a shortest cycle of even length, *Computing,* **31**, pp.355-369.

Müller, V. [1979]
On colorings of graphs without short cycles, *Disc. Math.*, **26**, pp.165-176.

Mycielski, J. [1955]
Sur le coloriage des graphes, *Colloq. Math.*, **3**, pp.161-162.

Nešestřil, J. [1966]

On k-chromatic graphs without circuits of a length ≤ 7 (Russian), *Comm. Math. Univ. Carolinae*, **7**, 3.

Nešestřil, J., and V. Rödl [1979]

A short proof of the existence of highly chromatic hypergraphs without short cycles, *J. Combinatorial Theory B*, **27**, pp.225-227.

Neufield, B. A., and J. Tartar [1974]

Graph coloring conditions for the existence of solutions to the time-table problem, *CACM*, **17**, pp.450.

Ng, T., and L. Johnsson [1985]

Generation of layouts from circuit schematics; A graph theoretic approach, Research report YaleU/DCS/RR-378, March, 1985.

Peck, J. E. L., and M. R. Williams [1966]

Examination scheduling, *CACM*, vol **9**, no 6, pp.433-434.

Roschke, S. I., and A. L. Furtado [1973]

An algorithm for obtaining the chromatic number and an optimal colouring of a graph, *Information Processing Letters*, **2**, pp.34-38.

Rose, D. J., R. E. Tarjan, and G. S. Lueker [1976]

Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.*, **5**, pp.266-283.

Sarrafzadeh, M., and D. T. Lee [1987]

A new approach to topological via minimization, Northwestern University, Center for Integrated Microelectronic Systems Technical Report CIMS-87-01, November, 1987.

Shamir, E., and E. Upfal [1984]

Sequential and distributed graph coloring algorithms with performance analysis in random graph spaces, *J. Algorithms*, **5**, pp.488-501.

Schäuble, M. [1968]

Beiträge zum problem der existenz und konstruktion endlicher graphen gegebener chromatischer zahl, die gewissen strukturbedingungen genügen, und zu verwandten problemen, Dissertation, Ilmenau.

Strassen, V. [1969]

Gaussian elimination is not optimal, *Numerische Mathematik*, **13**, pp.354-356.

Takamizawa, K., T. Nishizeki, and N. Saito [1982]

Linear-time computability of combinatorial problems on series-parallel graphs, *JACM*, vol **29**, no 3, pp.623-641.

Tarjan, R. E. [1972]

Depth first search and linear graph algorithms, *SIAM J. Computing*, **1**:2, pp.146-160.

Tucker, A. [1971]

Matrix characterization of circular-arc graphs, *Pacific J. Math.*, **39**, pp.535-545.

Tucker, A. [1980]

An efficient test for circular-arc graphs, *SIAM J. Comput.*, **9**, pp.1-24.

Tucker, A. C., and L. Bodin [1976]

A model for municipal street sweeping operations, *Case Studies of Applied Mathematics*, Math. Assoc. of America, Washington, pp.251-295.

Turner, J. S. [1984]

On the probable performance of graph coloring algorithms, *22nd Allerton Conf. on Communications, Control, and Computing*, pp.281-290.

Turner, J. S. [1988]

Almost all k-colorable graphs are easy to color, *J. Algorithms*, **9**, pp.63-82.

Welsh, D. J. A., and M. B. Powell [1967]

An upper bound for the chromatic number of a graph and its applications to timetabling problems, *The Computer Journal*, **10**, pp.85-86.

Wigderson, A. [1983]

Improving the performance guarantee for approximate graph coloring, *JACM*, vol **30**, no 4, pp.729-735.

Wilf, H. S. [1984]

Backtrack: an O(1) expected time algorithm for the graph coloring problem, *Inform. Process. Lett.*, **18**, pp.119-121.

Wood, D. C. [1968]

A system for computing university examination timetables, *The Computer Journal*, **11**, pp.41.

Wood, D. C. [1969]

A technique for coloring a graph applicable to large scale timetabling problems, *The Computer Journal*, **12**, pp.317.

Yannakakis, M. [1987]

Personal communication.

Yannakakis, M., and F. Gavril [1987]

The maximum k-colorable subgraph problem for chordal graphs, *IPL* **24**, pp.133-137.

Zykov, A. A. [1949]

On some properties of linear complexes (in Russian), *Mat. Sbornik N.S.*, **24**, pp.163-188. English translation in: *Amer. Math. Soc. Transl.*, **79**, (1952), reissued in: *Translations Series 1*, **7**, *Algebraic Topology*, pp.418-419 (Amer. Math. Soc. 1962).