8-2003

Topics in Graph Algorithms: Structural Results and Algorithmic Techniques, with Applications

Faisal Nabih Abu Khzam
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Faisal Nabih Abu Khzam entitled "Topics in Graph Algorithms: Structural Results and Algorithmic Techniques, with Applications." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Michael A. Langston, Major Professor

We have read this dissertation and recommend its acceptance:

Bradley Vander Zanden, Bruce MacLennan, Jens Gregor, Don W. Bouldin

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Faisal Nabih Abu Khzam entitled "Topics in Graph Algorithms: Structural Results and Algorithmic Techniques, with Applications." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Michael A. Langston
Major Professor

We have read this dissertation
and recommend its acceptance:

Bradley Vander Zanden

Bruce MacLennan

Jens Gregor

Don W. Bouldin

Accepted for the Council:

Anne Mayhew
Vice Provost
and Dean of Graduate Studies

(Original signatures are on file with official student records)

Topics in Graph Algorithms: Structural Results and Algorithmic Techniques, with Applications

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Faisal Nabih Abu-Khzam

August 2003

To Salma

Acknowledgments

I wish to express my sincere gratitude and appreciation to my advisor, Professor Michael A. Langston for his expert guidance and support. I would also like to thank Professors D. Bouldin, J. Gregor, B. Mclennan and B. Vander zanden for reading this dissertation and providing constructive comments.

Abstract

Coping with computational intractability has inspired the development of a variety of algorithmic techniques. The main challenge has usually been the design of polynomial time algorithms for $\mathcal{NP}$-complete problems in a way that guarantees some, often worst-case, satisfactory performance when compared to exact (optimal) solutions.

We mainly study some emergent techniques that help to bridge the gap between computational intractability and practicality. We present results that lead to better exact and approximation algorithms and better implementations. The problems considered in this dissertation share much in common structurally, and have applications in several scientific domains, including circuit design, network reliability, and bioinformatics.

We begin by considering the relationship between graph coloring and the immersion order, a well-quasi-order defined on the set of finite graphs. We establish several (structural) results and discuss their potential algorithmic consequences.

We discuss graph metrics such as treewidth and pathwidth. Treewidth is well studied, mainly because many problems that are $\mathcal{NP}$-hard in general have polynomial time algorithms when restricted to graphs of bounded treewidth. Pathwidth has many applications ranging from circuit layout to natural language processing. We present a linear time algorithm to approximate the pathwidth of planar graphs that have a fixed disk dimension.

We consider the face cover problem, which has potential applications in facilities location and logistics. Being fixed-parameter tractable, we develop an algorithm that solves it in time $O(5^k + n^2)$ where k is the input parameter. This is a notable improvement over the previous best known algorithm, which runs in $O(8^k n)$.

In addition to the structural and algorithmic results, this text tries to illustrate the practicality of fixed-parameter algorithms. This is achieved by implementing some algorithms for the vertex cover problem, and conducting experiments on real data sets. Our experiments advocate the viewpoint that, for many practical purposes, exact solutions of some $\mathcal{NP}$-complete problems are affordable.

Contents

List of Figures

Chapter 1

Introduction and Background

The notion of *Fixed-Parameter Tractability* has recently changed the landscape in the search for efficient and practical algorithms for $\mathcal{NP}$-hard problems. The story started two decade ago, when Neil Robertson and Paul Seymour undertook the task of proving a conjecture of Wagner [57]. They were able to establish that graphs are well-quasi ordered by some containment relations. Their work, considered by many to be the deepest discrete mathematics project of this century, opened the door for many applications and inspired Fellows and Langston, who were the first to notice and thoroughly study the algorithmic consequences [28, 29, 30, 27]. They proved that some parameterized $\mathcal{NP}$-complete problems become tractable when the corresponding input parameter is fixed. This led to the emergence of Parameterized Complexity Theory, pioneered mainly by Downey and Fellows [21].

It is really due to this new classification of "difficulty" that many researchers around the globe strive nowadays to obtain "faster" exact algorithms for some $\mathcal{NP}$-complete problems, once used to be dismissed as intractable. The problems of interest are classified as *fixed-parameter tractable*. These are parameterized

problems that become easy (tractable) when their input parameter is fixed. The endeavor for proving that a certain parameterized problem is fixed-parameter tractable is sometimes associated with obtaining better exact algorithms for the problem. Such a "positive" impact is a lot more beneficial than negative answers provided, only for the research community, by some hardness result.

In this first chapter, we briefly review some well-quasi ordering background from a graph-theoretic angle. We discuss graph containment relations and single out two of them that define well-quasi-orders: the minor order and the immersion order. We also introduce (formally) the concept of fixed-parameter tractability and survey some problems that are relevant to our work.

In chapter 2, we focus our attention on immersion containment and discuss its relationship to graph coloring. Chapter 3 is devoted to the discussion of common and new algorithmic techniques developed along the road for better and faster algorithms. Some of these techniques are examined in more details in chapter 4 as we present sequential and parallel implementations for the Vertex Cover problem. We also introduce a new algorithm to solve the Face Cover problem for plane graphs. This is the context of chapter 5.

Some graph metrics are also discussed in this text. We focus our attention on the treewidth and pathwidth of graphs. The pathwidth metric has been studied recently because of its applications in VLSI design. In fact, the author's research started by trying to obtain fast approximation algorithms for the pathwidth of planar graphs that have a fixed disk dimension. A new algorithm, presented in chapter 6, nicely reduces the task of approximating the pathwidth of a graph of disk dimension k to that of an outerplanar graph by loosing only an additive constant.

1.1 Notation

We mainly restrict our attention to finite, simple and undirected graphs. We neglect loops and multiple edges that may be introduced by applying some operations to graphs under discussion. For a given graph G, $V(G)$ and $E(G)$ denote the sets of vertices and edges of G respectively. For simplicity, we often use V and E, without refering to G, when no confusion results. The cardinality of V (also termed the *order* of G) is denoted by n. Edges of a simple graph are uniquely identified by their endpoints. We use uv to denote an edge whose endpoints are u and v.

The neighborhood of a vertex v of $G = (V, E)$ is defined by: $N_G(v) = \{u \in V : uv \in E\}$. The subscript G will be dropped and $N(v)$ will be used when there is no ambiguity about the graph under discussion. The cardinality of $N(v)$ is the *degree* of v. The smallest and largest degrees taken over all vertices of a graph are denoted by $\delta(G)$ and $\Delta(G)$ respectively.

We denote by K_s the complete graph on s vertices. $K_{s,t} = (A, B)$ will be used to denote a complete bipartite (or bichromatic) graph where $\{A, B\}$ is the corresponding partition of $V(G)$. Complete graphs play an important role in our applications. Particularly, we will be interested in finding complete subgraphs (AKA cliques) of maximum size in a given graph.

Part of this work is restricted to planar graphs. A planar graph is one that can embed in the plane without edge crossings. A particular planar embedding (or simply, drawing) of a graph is called a plane graph. The family of planar graphs has many interesting subfamilies. An outerplanar graph, for example, is a planar graph that has at least one embedding in the plane minus an open disk so that vertices lie on the boundary of the disk and edges are drawn in the exterior (or, equivalently, interior) of the disk without crossing. Outerplane graphs are

particular outerplanar embeddings or drawings of outerplanar graphs.

1.2 Graph Metrics

We briefly describe some properties of graphs that are commonly known as graph metrics. Among several well known graph metrics, the ones reviewed in this section are those relevant to our work.

1.2.1 Treewidth

Definition 1 A tree decomposition of a graph G is a pair (T, Y), where T is a tree and $Y = \{Y_i : i \in V(T)\}$ is a collection of subsets of $V(G)$ satisfying the following:

(i) $\forall uv \in E(G), \exists i$ such that $\{u, v\} \subseteq Y_i$.

(ii) $\forall i, j, k \in V(T)$, if j is on the path between i and k in T, then $Y_i \cap Y_k \subseteq Y_j$.

The *width* of a tree decomposition (T, Y), denoted by $w((T, Y))$, is $max\{|Y_i| : Y_i \in Y\} - 1$. The *treewidth* of G, denoted by $tw(G)$, is $min\{w((T, Y)) : (T, Y)$ is a tree decomposition of $G\}$.

Tree decompositions of a graph G whose width is the same as $tw(G)$ are considered optimal tree decompositions. Figure 1.1 shows a graph and two tree decompositions of which one is optimal.

1.2.2 Pathwidth

Definition 2 A path decomposition of a graph G is a pair (P, X), where P is a path and $X = \{X_i : i \in V(P)\}$ is a collection of subsets of $V(G)$ satisfying the following:

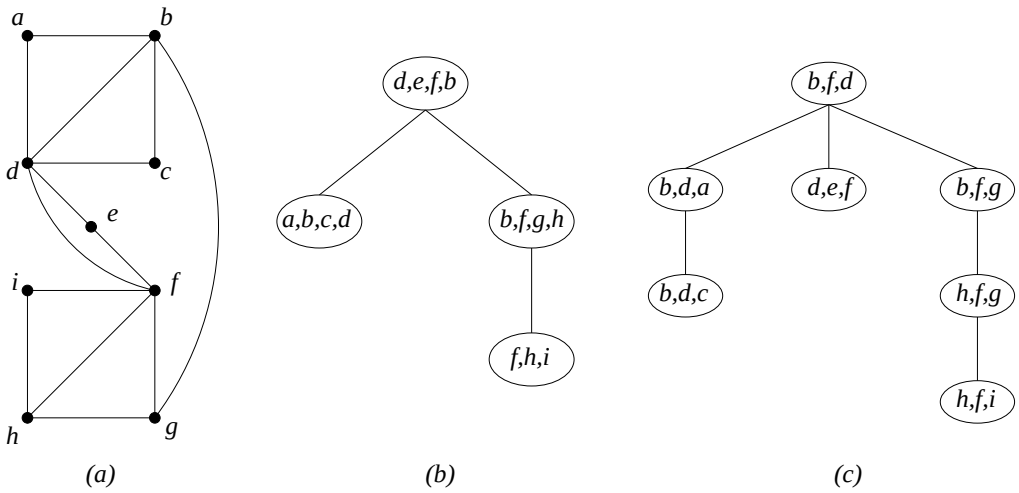


Figure 1.1: (a) A graph G. (b) A tree decomposition of G. (c) An optimal tree decomposition of G.

(i) $\forall uv \in E(G), \exists i$ such that $\{u, v\} \subseteq X_i$.

(ii) $\forall i, j, k \in V(P)$, if $i \leq j \leq k$, then $X_i \cap X_k \subseteq Y_j$.

The *width* of a path decomposition (P, X), denoted by $w((P, X))$, is $max\{|X_i| : X_i \in X\} - 1$. The *pathwidth* of G, denoted by $pw(G)$, is $min\{w((P, X)) : (P, X)$ is a path decomposition of $G\}$.

As for tree decompositions, optimal path decompositions of a graph, G, are those whose width agree with $pw(G)$. Figure 1.2 shows a graph and two path decompositions of which one is optimal.

The pathwidth of any path is one. The pathwidth of a complete graph, K_t, is $t - 1$ (since any pair of vertices of K_t must belong to at least one X_i in any path decomposition (P, X)). Moreover, trees could have arbitrarily large pathwidth, as

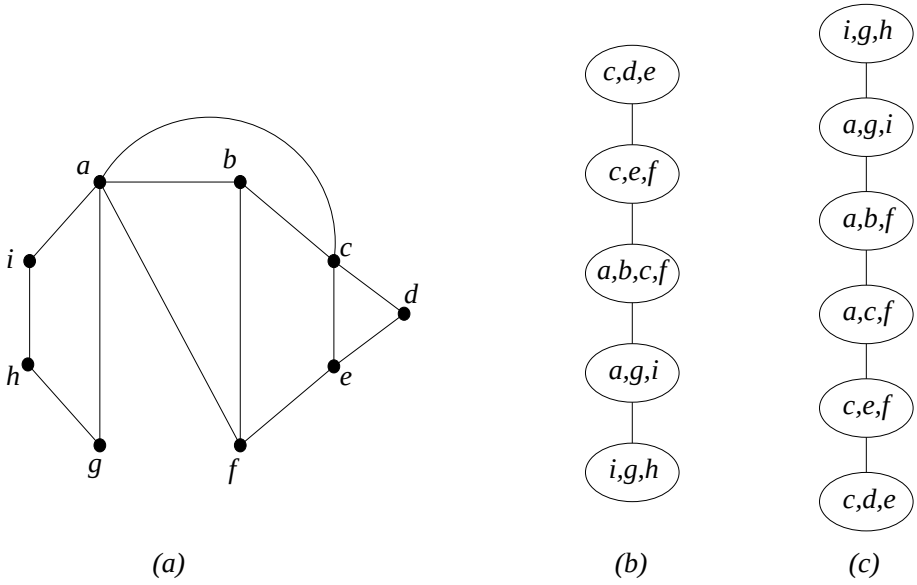


Figure 1.2: (a) A graph G. (b) A path decomposition of G. (c) An optimal path decomposition of G.

witnessed by ternary trees[1]. This being true since the pathwidth of a ternary tree grows with its height.

Since a path is a special tree, any path decomposition of a graph G is a tree decomposition. Hence, $\forall G$, $tw(G) \leq pw(G)$. This inequality is strict in general. The fact that ternary trees could have arbitrary pathwidth shows that the gap between the treewidth and the pathwidth of a graph could be arbitrarily large.

1.2.3 Cutwidth

A *linear layout* of a graph G is a permutation, L, of the vertices of G. In other words, L is a function from $V(G)$ to $\{1, 2, \ldots, |V(G)|\}$. We think of a linear layout as a drawing of G so that vertices lie on a horizontal line while edges are free to cross (even if the graph is planar). The cutwidth of a linear layout of a graph G is the maximum number of edges of G that are cut by a vertical line drawn between consecutive vertices in the layout. The cutwidth of G is the minimum cutwidth taken over all possible linear layouts of G. Figure 1.3 shows a graph and two possible linear layouts of which one is optimal.

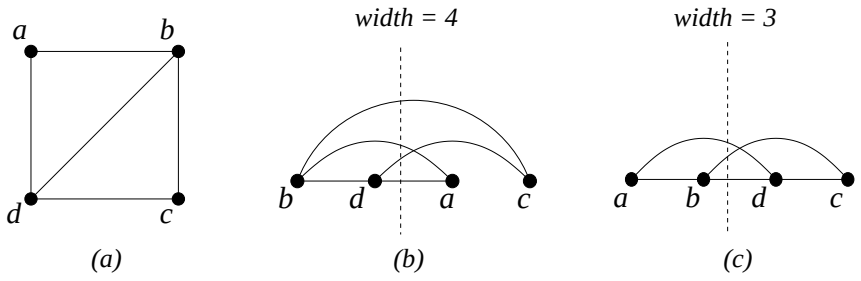


Figure 1.3: (a) A graph G. (b) A linear layout of G. (c) An optimal linear layout of G.

[1]trees whose internal vertices are all of degree 3

1.3 Well-Quasi-Ordering Theory and Graphs

Let $\leq$ be a binary relation defined on a set X. $(X, \leq)$ is called a *quasi-order* if $\leq$ is reflexive and transitive in X. It is sometimes more convenient to say: $\leq$ defines a quasi-order on X, or X is quasi-ordered by $\leq$. An *antichain* is a set of elements of X that are pairwise uncomparable under $\leq$[2]. A descending chain is a sequence of distinct elements, $a_1, a_2, a_3, \ldots$, of X such that $a_{i+1} \leq a_i$ for $i \geq 0$. A quasi-ordered set $(X, \leq)$ is a *well-quasi-order* (or, for simplicity, *wqo*) if (1) it has no infinite antichain, and (2) there exist no infinite descending chain.

We shall see that finding a *wqo* relation on the set of graphs may have important algorithmic consequences. There are many well known quasi-order relations defined on graphs. The subgraph relation, denoted $\leq$, being the first to notice. Unfortunately, $\leq$ is not a *wqo* since, for example, the set of all cycle graphs presents an infinite antichain. (see figure 1.4).

1.3.1 Topological Containment

Let $e = uv$ be an edge of a graph G. Subdividing e is the operation that replaces e by a path with endpoints u and v. The length of the introduced path may be zero or more. A *subdivision* of a graph H is a graph obtained from H (or a graph isomorphic to H) by edge subdivision.

A graph H is said to be *topologically* contained in a graph G, written $H \leq_t G$, if a subgraph, H' of G is isomorphic to a subdivision of H. H' is called an *H-mode* in G and vertices of H' that correspond to those of H are called *corners* of the model. Figure 1.5 shows that K_4 is topologically contained the three-dimensional cube, Q_3.

[2] a and b are comparable under $\leq$ if either $a \leq b$ or $b \leq a$

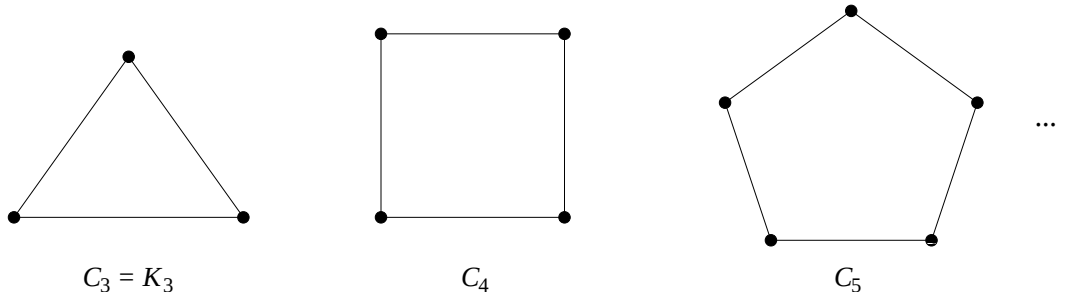


Figure 1.4: An infinite antichain for the subgraph relation

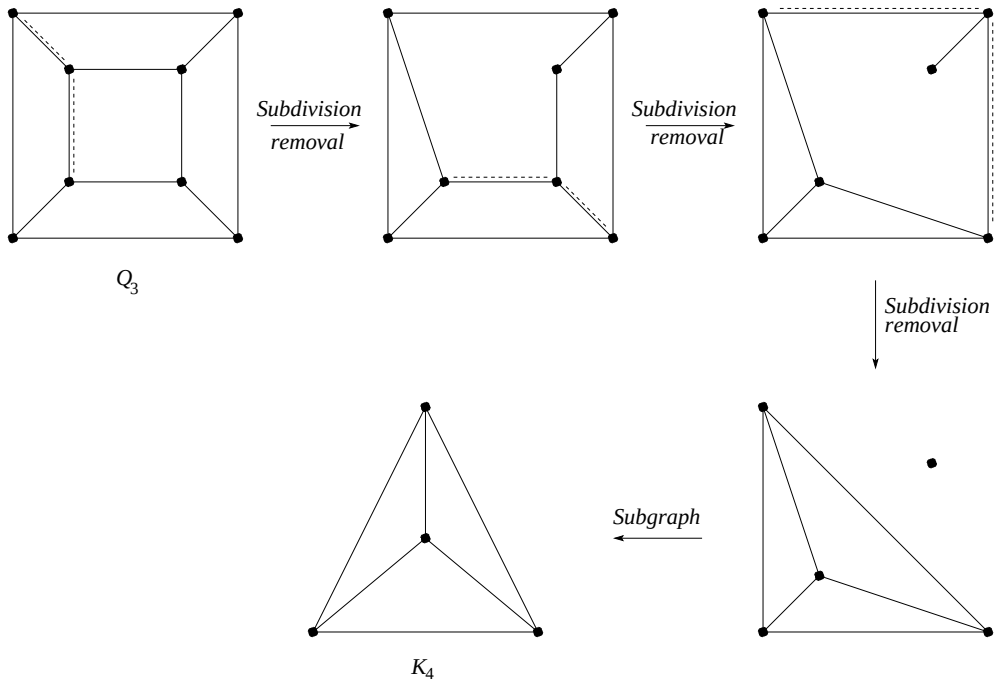


Figure 1.5: $K_4 \leq_t Q_3$

Remark 1 $H \leq_t G$ if and only if there is a one-to-one mapping from $V(H)$ to $V(G)$ under which edges of H are mapped to vertex-disjoint paths between the images of the corresponding endpoints.

Topological containment induces another quasi-order relation on graphs, usually called the *topological order*. It is not, however, a *wqo*. The infinite antichain for subgraph containment, depicted in figure 1.4, does not serve as an antichain for $\leq_t$ since $C_3 \leq_t C_4 \leq_t C_5 \leq_t ....$ A well known example is the infinite sequence of $double - cycle$ graphs, depicted in figure 1.6 below.

1.3.2 Minor Containment

Let $e = uv$ be an edge of a graph G. Contracting e is the operation that replaces e, u, and v by one vertex w, and sets $N(w) = N(u) \cup N(v)$. A graph H is said to be a *minor* of a graph G, written $H \leq_m G$, if a graph isomorphic to H can be obtained from a subgraph of G by contracting edges. This notion is illustrated in figure 1.7, which shows that K_5 is a minor of the Peterson graph.

Minor containment defines a quasi-order on the set of graphs. In their deepest discrete mathematics work of the century, Robertson and Seymour proved what is now known as the *Graph Minor Theorem* [55]:

Theorem 1 *Graphs are well-quasi-ordered by minor containment.*

Remark 2 Minor containment can also be characterized as follows: H is a minor of G if and only if there exists a one-to-one mapping from the vertices of H to connected subgraphs of G for which the images of adjacent elements of H are connected in G by vertex-disjoint paths. It follows that the minor order is a generalization of the topological order, because a single vertex is a trivially connected component.

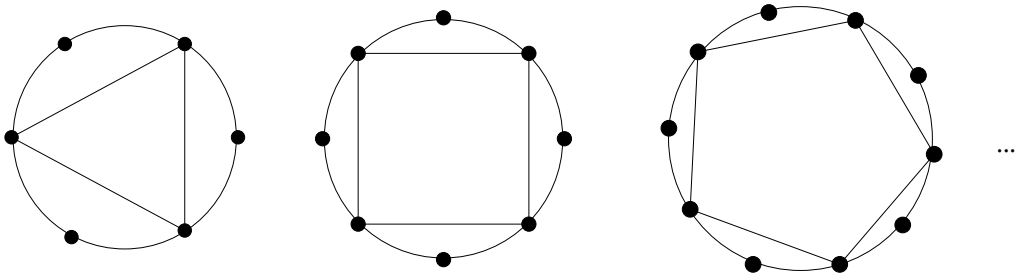


Figure 1.6: An infinite antichain for the topological order

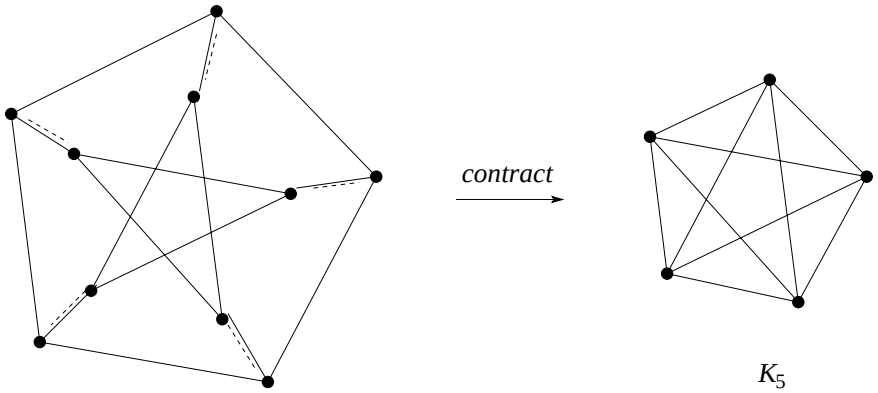


Figure 1.7: K_5 is a minor of the Peterson graph

Remark 3 It was shown in [28] that, if $\Delta(H) \leq 3$, then, for any graph G, $H \leq_m G$ if and only if $H \leq_t G$.

1.3.3 Immersion Containment

A pair of adjacent edges uv and vw, with $u \neq v \neq w$, is *lifted* by deleting the edges uv and vw, and adding the edge uw. A graph H is said to be *immersed* in a graph G if and only if a graph isomorphic to H can be obtained from G by taking a subgraph and lifting pairs of edges. This notion is illustrated in figure 1.8, which shows that K_4 is immersed in $K_1 + P_5$.

Another result of the work of Robertson and Seymour is the following theorem.

Theorem 2 *[56] Graphs are well quasi-ordered by immersion containment.*

Remark 4 H is immersed in G if and only if there exists a one-to-one mapping from the vertices of H to the vertices of G for which the images of adjacent elements of H are connected in G by edge-disjoint paths. Under such an injection, an image vertex is called a *corner* of H in G; all image vertices and their associated paths are collectively called a *model* of H in G. Therefore, like the minor order, the immersion order is also a generalization of the topological order. This is trivially due to the fact that vertex-disjoint paths are edge-disjoint.

Remark 5 if $\Delta(G) \leq 3$, then, for any graph H, $H \leq_i G$ if and only if $H \leq_t G$. Combining this with remark 3, we obtain: if $\Delta(H) \leq 3$ and $\Delta(G) \leq 3$ then $H \leq_t G \iff H \leq_m G \iff H \leq_i G$.

We revisit the immersion order in more details in the next chapter by discussing its relationship with graph coloring.

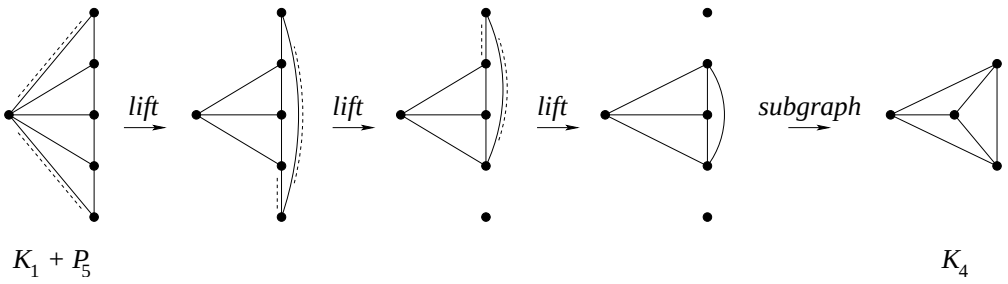


Figure 1.8: K_4 is immersed in $K_1 + P_5$

1.4 Algorithmic Consequences of $\mathcal{WQO}$ Theory

A family F of finite graphs is closed in the minor order if $G \in F$ implies that any minor of G is in F. The obstruction set of F, denoted by $obs(F, \leq_m)$, consists of minor-minimal elements in F's complement. Therefore, F can be characterized by means of $obs(F, \leq_m)$ as follows: $G \in F \iff \forall H \in obs(F, \leq_m), H \not\leq_m G$. Due to $\mathcal{WQO}$ theory, we know that $obs(F, \leq_m)$ is finite. (Otherwise, it would be an infinite anti-chain.) Thanks for the following theorem of Robertson and Seymour [55], such F must have a polynomial time recognition test.

Theorem 3 *For every fixed graph H, there exists a polynomial time algorithm that, given a graph G, decides whether $H \leq_m G$.*

Remark 6 The run time of the algorithm mentioned in Theorem 3 is $O(n^3)$. It will be $O(n)$ if the graph G is of bounded treewidth. Despite the large constant hidden by the big O notation, such run time is interesting since the order of H did not show up in the exponent of n. We study this type of (fixed-parameter) algorithms in the next section.

It would be useful (algorithmically) if one is able to determine the elements of $obs(F, \leq_m)$. Unfortunately, $\mathcal{WQO}$ theory does not provide such information.

Theorem 4 *[30] There is no algorithm to compute, from a finite description of a minor-closed family F of graphs as represented by a Turing machine that accepts precisely the graphs in F, the set of obstructions for F.*

Note that families of finite graphs that are closed in the immersion order also have finite obstruction sets. Moreover, due to the following theorem of Fellows and Langston [29], they can be recognized in polynomial time.

Theorem 5 *For every fixed graph H, the problem that takes as input a graph G and determines whether $H \leq_i G$ is solvable in time $O(n^{h+3})$, where h is the order of H.*

Remark 7 For an immersion closed family, F, the membership test is $O(n^{h+3})$, where h is the order of the largest graph in $obs(F, \leq_i)$. It was shown in [29] that, if F is of bounded treewidth, then it has an $O(n^2)$ membership test. This membership test used the result of Robertson and Seymour that testing for immersion containment can be done in linear time on graphs of bounded treewidth [55]. We note that such membership test should, in principle, run in linear time since its quadratic time was due to an algorithm for constructing tree decompositions of bounded width, which now takes linear time[6].

Let $\leq_Q$ be any quasi-order defined on the set of finite graphs. A family F of finite graphs is said to exclude a graph H in $\leq_Q$, if $H \not\leq_Q G$ for all $G \in F$ (in other words, if $H \notin obs(F, \leq_Q)$). For example, planar graphs are characterized by Kuratowski theorem as the family of graphs that exclude K_5 and $K_{3,3}$ in the topological order [42]. The same characterization holds if topological containment is replaced by minor containment [65].

A family F of finite graphs is said to have a bounded treewidth if for all $G \in F$, $tw(G) \leq c$, where c is a constant. For example, the family of outerplanar graphs

has treewidth two. In fact, graphs of treewidth two are exactly the *series-parallel graphs*. These are planar graphs that exclude K_4 in the minor order[3].

Graph families that are closed in the minor order and exclude a planar graph are of particular interest, again, due to the following results of Robertson and Seymour [55].

Theorem 6 *For every planar graph H, there exists a constant C_H, such that every graph G that doesn't have an H-minor satisfies $tw(G) \leq C_H$.*

Theorem 7 *If a family F of finite graphs is minor-closed and excludes a planar graph, then F has a polynomial-time membership test.*

Remark 8 Theorem 7 predated the linear-time tree decomposition algorithm of Bodlaender [6]. It is now well known that graph families that are closed in the minor order and exclude a planar graph can be recognized in linear time.

1.5 Parameterized Complexity

Many problems of practical interest are $\mathcal{NP}$-complete [32]. In other words, it is very unlikely that a polynomial-time algorithm exists for any of these problems. Various algorithmic techniques were invented solely to cope with this computational intractability problem. Common classical methods include approximation algorithms, randomized algorithms, and heuristic techniques. Once a problem is determined to be $\mathcal{NP}$-Complete, exact algorithms used to be dismissed and classical methods were the only adopted methods of approach.

Despite the fact that exact solutions are not usually sought, many exact exponential algorithms appeared for $\mathcal{NP}$-complete problems [44, 24, 59]. Practicality,

[3]By remark 3, series-parallel graphs are also characterized by those that exclude K_4 in the topological order.

however, was not usually pursued. We can list several "fastest" exact exponential algorithms that were never implemented.

Not all applications benefit from approximate or heuristic solutions. Especially if, in practice, exact algorithms could deliver desired solutions in real time. In fact, in some practical applications of some $\mathcal{NP}$-complete problems, either the input instance is of modest size or some input parameter is fairly small. In chapter 4, we empirically show that exact exponential algorithms for such applications are affordable.

The input of most $\mathcal{NP}$-complete problems consists of a pair (I, k), where I is the input instance and k is a parameter. These problems are called *parameterized problems*. In this text, we adopt the notation $\pi(k)$ for a parameterized problem π, and we assume that n denotes the size of I.

As we mentioned earlier, the dawn of fixed-parameter tractability appeared to have started with Fellows and Langston. They considered only graph problems whose yes instances are closed under the minor and the immersion orders. With this approach, they showed that some parameterized problems must have fast exact algorithms when corresponding input parameters are fixed. Their (nonconstructive) proofs relied on Theorem 1 and the arguments in remark 7. This opened a door for more scrutiny, later, by Downey and Fellows (then many others). As a result, *Parameterized Complexity* was born and a new hierarchy, termed the $\mathcal{W}$-hierarchy, of classes within $\mathcal{NP}$ was defined.

At the bottom of the new hierarchy lies the class of problems that become tractable when the relevant parameter is fixed.

Definition 3 Let (I, k) be a parameterized problem, where I is the problem instance and k is the parameter. (I, k) is fixed parameter tractable ($\mathcal{FPT}$) if it has an algorithm that runs in time $O(f(k)|I|^c)$, where $|I|$ is the size of I, $f(k)$ is

an arbitrary function, and c is a constant.

When k is fixed, $f(k)|I|^c$ is, simply, a polynomial. Thus the given problem is tractable even if the growth of $f(k)$ is very fast (factorial for example). The corresponding run time is called *uniform poly-time*.

Constructively proving that a problem is $\mathcal{FPT}$ is useful since it usually provides an algorithm that runs in uniform poly-time. In fact, this is usually not the end. Many researchers around the globe are challenged to obtain faster exact algorithms for $\mathcal{FPT}$ problems. This is achieved by obtaining an algorithm whose run-time, $O(f(k)|I|^c)$, exhibits a new function of k ($f(k)$) that is slower in variation then the best (previously) known function.

Algorithms for $\mathcal{FPT}$ problems are called *fixed-parameter algorithms* . It was believed that such algorithms work only when the parameter in question is fixed. The algorithm described in [13], for example, was designed only to determine if the input graph has a vertex cover of size at most 5. It is now clear, at least to our research group at the University of Tennessee, that most fixed-parameter algorithms designed for a given $\mathcal{FPT}$ problem are nothing but exact exponential algorithms for the optimization version of the problem.

1.6 Some Fixed Parameter Tractable Problems

We discuss briefly a set of well known $\mathcal{FPT}$ problems that we considered in our research. $\mathcal{WQO}$ theory had the honor to be the first tool used for classifying some of these problems. For a minimization (maximization) parameterized problem $\pi(p)$, $Y_k(\pi)$ denotes the family of yes instances of $\pi(p)$ for all $p \leq k$ ($p \geq k$).

1.6.1 The Vertex Cover Problem (VC)

The inputs to Vertex Cover are an undirected graph G and a parameter k. The question asked is whether a set S of k or fewer vertices covers every edge in G. (An edge is covered if either or both of its endpoints are in S.)

The parameterized Vertex Cover problem ($VC(k)$) is $\mathcal{N}P$-complete for arbitrary k. When k is fixed, $Y_K(VC)$ is closed under taking minors. To see this, note that both the operation of taking a subgraph and the one of contracting edges cannot lead to more edges to cover. It follows (by the argument of Remark 6) that $VC(k)$ is $\mathcal{F}PT$.

There has been tremendous progress in the process of obtaining better exact algorithms for this problem. It's now solvable in $O(1.2852^k + kn)$ [15].

1.6.2 The 3-Hitting Set Problem (3-HS)

Given a collection C of subsets of a finite set S and a positive integer k. The Hitting Set problem is to decide whether there exists a subset S' of S such that $|S'| \leq k$ and every subset in C has a nonempty intersection with S'. Hitting Set is NP-complete and the parameterized version of the problem ($HS(k)$) is $W[2]$-hard in general. In other words, the problem does not seem to have a practical solution when k is fixed (see [21]). When every subset in C is of size not exceeding a fixed positive integer d, the problem is called d-Hitting Set (or d-HS) and, while still $\mathcal{N}P$-complete, its parameterized version, d-$HS(k)$, is known to be FPT. When $d = 2$, the problem reduces to that of finding a Vertex Cover for a graph whose vertices and edges correspond to elements of S and C respectively. The $d = 3$ case received some attention lately, mainly because of its potential applications in computational biology. The simplest approach would solve 3-$HS(k)$ in $O(3^k n)$ where n is the input size. Most recently, Niedermeier and Rossmanith presented

an $O(2.270^k + n)$ algorithm[4] [53].

1.6.3 The Planar Dominating Set Problem (PDS)

A dominating set of a graph $G = (V, E)$ is a subset, V' of V, that has a nonempty intersection with the neighborhood of every vertex that belongs to its complement. Deciding whether a given graph has a dominating set of size not exceeding k is $\mathcal{NP}$-complete. Moreover, this problem is $\mathcal{W}[2]$-hard in general [21, 20]. It is fixed-parameter tractable when the input is a planar graph [1, 21]. There are two recent algorithms for the parameterized Planar Dominating Set problem ($PDS(k)$) that use different approaches. The first has a run time of $O(4^{6\sqrt{(34k)}}n)$ [1] and is based on dynamic programming on tree decompositions. Of course, this algorithm is not practical but is of theoretical interest because of its subexponential behavior. The second, assumed to be more practical, runs in $O(8^k n)$ [2] and is simpler to implement but a lot harder to analyze.

1.6.4 The Face Cover Problem (FC)

A face cover of a plane graph is a set of faces whose boundaries contain all vertices. For arbitrary $k > 0$, determining whether a plane graph G has a face cover of size k is an $\mathcal{NP}$-complete problem [5]. Existence of algorithms that solve the problem in time $O(c^k n)$ [5, 1] proves that the problem $FC(k)$ is $\mathcal{FPT}$. The algorithm described in [1] solves $FC(k)$ in $O(3^{36\sqrt{(34k)}}n + n^2)$ by reduction to the planar dominating set problem. It is believed that Face Cover has a better algorithm that runs in $O(8^k n)$ time using the same reduction and the $PDS(k)$ algorithm of [2]. The latter being more practical for small k. In chapter 5, we present a direct

[4]Here n is the whole input size which is assumed to be quadratic in the number of elements of the set S.

algorithm with a run time of $O(5^k + n^2)$.

1.6.5 The Disk Dimension Problem (DD)

The *disc dimension* of a planar graph G, denoted by $dd(G)$, is the least positive integer k such that G embeds in the plane minus k open discs, $\{d_i\}_{i=1}^{k}$, so that every vertex of G lies on the boundary, C_i, of some disc d_i. It is not hard to see that the disk dimension of a planar graph is the minimum face cover taken over the set of all possible planar embeddings of the graph. Despite this relationship, the two problems are totally different. This is because no solution of one could help solving the other.

The parameterized Disc Dimension problem, $DD(k)$, is to decide for a given planar graph G whether the disk dimension of G is not more than k.

$DD(k)$ was originally introduced in [27], where it was shown to be decidable in $O(n^2)$ for any fixed k, since $Y_k(DD)$ is closed in the minor order and excludes a planar graph (see Remark 6). Hence $DD(k)$ is $\mathcal{FPT}$. The decision version of the general problem is $\mathcal{NP}$-complete [26].

1.6.6 Graph Metrics Problems

Each graph metric described in section 1.2, gives rise, naturally, to a parameterized (decision) problem. The problems $TW(k)$, $PW(k)$ and $CW(k)$ are to determine, respectively, whether some input graph has treewidth, pathwidth and cutwidth that is not greater than k.

All three problem are $\mathcal{NP}$-complete, even when restricted to planar graphs of maximum degree three [3, 49].

The families $Y_k(TW)$ and $Y_k(PW)$ are closed in the minor order. Therefore, $TW(k)$ and $PW(k)$ are $\mathcal{FPT}$.

As for $CW(k)$, $Y_k(CW)$ is closed in the immersion order. Observing that $Y_k(CW)$ has bounded treewidth, the argument of Remark 7 was used in [29] to show that $CW(k)$ is $\mathcal{FPT}$.

1.6.7 Within k Vertices of F

Let F be a family of graphs that is closed under taking subgraphs. The set of all graphs that have k or fewer vertices whose removal produces an element of F is denoted by $W_k(F)$, and is known as the family of graphs that are *within k vertices of F*. The corresponding problem of deciding membership in $W_k(F)$ is $\mathcal{NP}$-hard in general. It provides a generalization of many known problems. For example, if F is the set of edgeless finite graphs, then membership in $W_k(F)$ is exactly $VC(k)$. When F is closed in the minor order, membership in $W_k(F)$ is $\mathcal{FPT}$ [28].

Chapter 2

Graph Coloring and the Immersion Order

The relationship between graph coloring and the minor order has been studied for over sixty years, mainly due to the four color theorem and the famous long standing conjecture of Hadwiger [35]. We study the relationship between graph coloring and the immersion order by examining properties of graphs that are t-chromatic and exclude K_t in the immersion order. Our investigation of this relationship led us to conjecture that if G requires at least t colors, then $K_t \leq_i G$.

We present evidence in support of our proposition by studying properties of counterexamples that are minimal w.r.t $\leq_i$, aiming at proving their existence is impossible. We establish several properties of such minimal counterexamples. In particular, we prove that, if any exist, they must be both 4-vertex-connected and t-edge-connected. The $t = 5$ case is examined in additional detail.

The structural results obtained in this chapter are also of algorithmic importance. Proving that graphs that exclude K_t in the immersion order are $(t-1)$-colorable (as our conjecture claims) could lead, due to Theorem 5, to useful

polynomial-time preprocessing for the very hard coloring problem.

2.1 Background

Coloring a graph G is assigning colors to vertices of G so that adjacent vertices receive different colors. For example, K_t requires exactly t colors. The chromatic number of G, denoted by $\chi(G)$, is the minimum number of colors required by G in any proper coloring of its vertices. Of course it is well known that determining $\chi(G)$ is $\mathcal{NP}$-hard. Since K_t is the smallest t-chromatic graph, it is tempting to try to associate $\chi(G)$ with some sort of clique contained within G. After all, if G contains K_t as a subgraph, then it is easy to show that G can be colored with no fewer than t colors. To see that the presence of a K_t subgraph is not necessary, however, one needs only to observe that C_5, the cycle of order five, requires three colors yet does not contain K_3 as a subgraph.

Nevertheless, perhaps some weaker form of K_t is present. One possibility is topological containment (as a motivation, note that C_5 contains K_3 topologically). Sometime in the 1940s Hajós conjectured that if $\chi(G) \geq t$, then G must contain a topological K_t [36]. The conjecture is trivially true for $t \leq 3$. In 1952 Dirac proved it true for $t = 4$ [19]. It was not until Catlin's work in 1979 that Hajós' conjecture was finally settled, and negatively, with a family of counterexamples for $t \geq 7$ [12]. One such counterexample is the 13-vertex graph depicted in figure 2.1. It requires seven colors and contains no topological K_7. Subsequently, Erdős and Fajtlowicz were able to prove the rather surprising result that almost all graphs are counterexamples [25]. Thus Hajós' conjecture remains open only for $t \in \{5, 6\}$.

Another possibility is the minor order, which is a generalization of the topological order. Hadwiger conjectured in 1943 that, if $\chi(G) \geq t$, then G must contain a K_t minor [35].

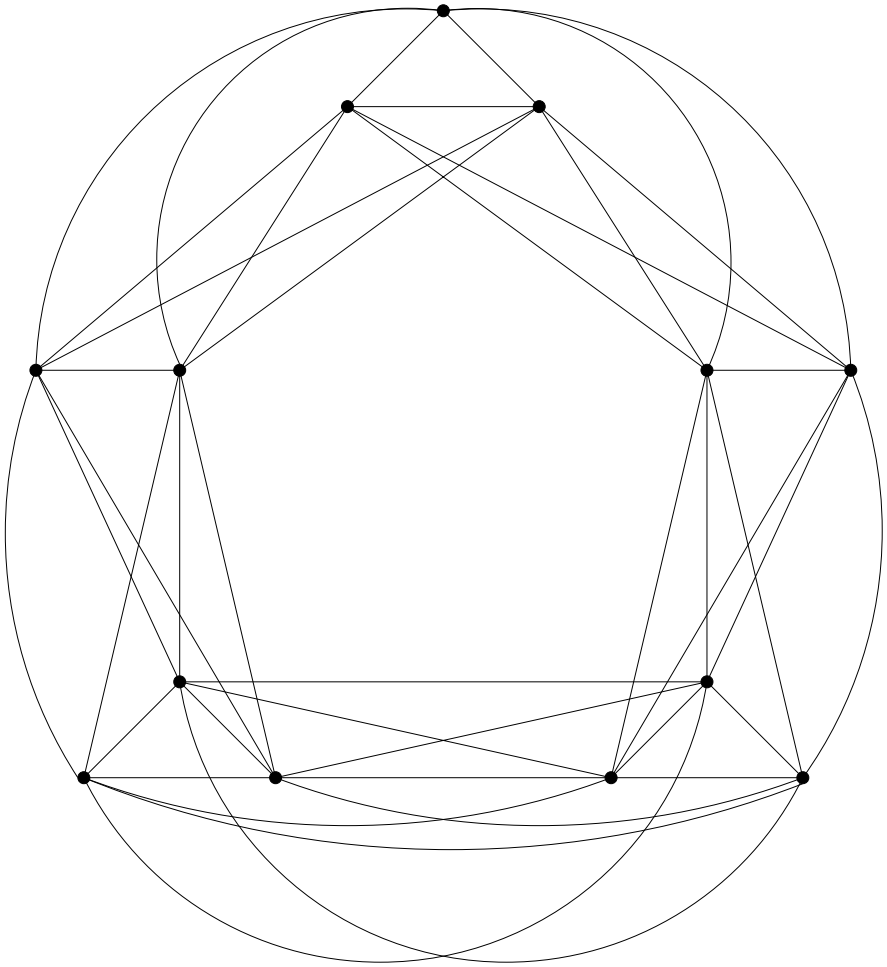


Figure 2.1: A 7-chromatic graph that does not contain a topological K_7.

By remark 3, this conjecture equates to Hajós' conjecture for $t \leq 4$. Wagner proved in 1964 that, for $t = 5$, Hadwiger's conjecture is equivalent to the four color theorem [66]. In 1993 Robertson, Seymour and Thomas proved it true for $t = 6$ [58]. Whether Hadwiger's conjecture holds true in general, however, has thus far not been decided. This is in spite of decades of research, hordes of supporting evidence and a multitude of results on many of its variants and restrictions [9, 22, 39, 62, 64, 68]. As of this writing, a resolution of Hadwiger's conjecture seems distant.

We focus instead on the immersion order. Immersion containment is quite distinct from topological and minor containment. Recalling Figure 1.8, for example, we observe that K_4 is contained in $K_1 + P_5$ in neither the topological nor the minor order. Previous investigations into the immersion order have generally been conducted from a purely algorithmic standpoint. We refer the reader to [10, 28, 29, 30, 43] for examples and applications. In contrast, here we mainly consider structural issues.

2.2 Preliminaries

Recall that we restrict our attention to finite, simple undirected graphs (multiple edges and loops that may arise from lifting are irrelevant to coloring). G is said to be *t-vertex-connected* if at least t vertex-disjoint paths connect every pair of its vertices. A *vertex cutset* is a set of vertices whose removal breaks G into two or more nonempty connected components. The cardinality of a smallest vertex cutset in G is equal to the largest t for which G is t-vertex-connected (unless G is a complete graph, which can have no vertex cutset). G is said to be *t-edge-connected* if at least t edge-disjoint paths connect every pair of its vertices. An *edge cutset* is a set of edges whose removal breaks G into two or more nonempty

connected components. The cardinality of a smallest edge cutset in G is equal to the largest t for which G is t-edge-connected.

If $\chi(G) \leq t$, then G is said to be *t-colorable*. If $\chi(G) = t$, then G is said to be *t-chromatic*. If $\chi(G) = t$ and $\chi(H) < t$ for every proper subgraph H of G, then G is said to be *t-color-critical*. A t-coloring of G is realized by a map c from the vertices of G to the set $\{1, 2, .., t\}$ so that, if G contains the edge uv, then $c(u) \neq c(v)$. Given such a map, c_{ij} is used to denote the subgraph induced by the vertex set $\{u : c(u) \in \{i, j\}\}$. A path contained within c_{ij} is termed a *Kempe chain* [69], so-named in honor of the foundational work done on them by Kempe in [40]. Of course c_{ij} need not be connected, and so for any $u \in c_{ij}$ we employ $c_{ij}(u)$ to denote the set $\{v : v$ resides in the same connected component of c_{ij} as does $u\}$. Such sets have useful properties.

Observation 1 *If $\{i, j\} \neq \{k, l\}$, then c_{ij} and c_{kl} are edge disjoint.*

Although the immersion order is traditionally defined in terms of taking subgraphs and lifting pairs of edges, Kempe chains and Observation 1 make it helpful for us to utilize as well the alternate characterization described in remark 4.

Suppose vertex u has degree $t-2$ or less in a t-chromatic graph G. Then $G-u$ must also be t-chromatic. Otherwise $G-u$ could be colored with $t-1$ colors, and u assigned one of the $t-1$ colors unused within $N(u)$.

Observation 2 *If G is t-color-critical, then $\delta(G) \geq t-1$.*

If G is t-chromatic but $G-u$ is only $(t-1)$-chromatic, then it is possible to consider only colorings in which u is assigned a unique color.

Observation 3 *If G is t-color-critical, then for any vertex u there exists a coloring c in which $c(u) = 1$ and $c(v) \neq 1$ for every vertex $v \in G-u$.*

2.3 Motivation

Given the assorted connections between graph coloring, connectivity and the immersion order, we seek to determine just how $\chi(G)$ is related to immersion containment. A main purpose of this chapter is an attempt to formalize this relationship with the following conjecture, which we find both plausible and appealing.

Conjecture 1 If $\chi(G) \geq t$, then G contains an immersed K_t.

In the sequel, we present compelling preliminary evidence in support of this conjecture. Our investigations to date suggest that it merits more detailed scrutiny. Its complete resolution, however, is well beyond the scope of this text.

Our conjecture, like Hadwiger's, is trivially true for $t \leq 4$. This is because the immersion order generalizes the topological order, for which Hajós' conjecture is long known to hold when $t \leq 4$. We address the $t = 5$ case in some detail, and come to within one edge of proving the conjecture holds for it.

Before proceeding, we introduce a notion of immersion criticality and show how it relates to the possible existence of counterexamples. G is said to be *t-immersion-critical* if $\chi(G) = t$ and $\chi(H) < t$ whenever H is properly immersed in G. Because $\chi(K_t) = t$, any counterexample must either be t-immersion-critical or have properly immersed within it another t-immersion-critical counterexample. Similarly, any t-immersion-critical graph distinct from K_t must be a counterexample. Thus our conjecture is equivalent to the statement that K_t is the only t-immersion-critical graph for every t. Although we have thus far fallen short of establishing this one way or the other, we can show that there are at most a finite number of them. To do this, we rely on properties of well-quasi-orders and immersion order obstruction sets.

Theorem 8 *There are finitely many t-immersion-critical graphs for each fixed t.*

Proof Consider the family of graphs $F = \{G : \chi(G) < t$ and $\chi(H) < t$ for every $H \leq_i G\}$. Then, by definition, F is closed in the immersion order. Because graphs are well-quasi-ordered by the immersion relation, it follows that F's obstruction set is finite. This set contains precisely the t-immersion-critical graphs. ∎

2.4 Properties of t-Immersion-Critical Graphs

Connectivity of minimal counterexamples played an important role in the long endeavor for a settlement of Hadwiger's conjecture. G is said to be *t-minor-critical*[1] if $\chi(G) = t$ and $\chi(H) < t$ whenever H is a proper minor of G. K_t is of course both $(t-1)$-vertex-connected and $(t-1)$-edge-connected. Thus, if any t-minor-critical graph is not as strongly connected, then Hadwiger's conjecture is false for all $t' \geq t$. So suppose G denotes a t-minor-critical graph other than K_t (in which case the conjecture fails). Some 35 years ago [46], Mader showed that G must be at least 7-vertex-connected whenever $t \geq 7$. This provides evidence in support of the conjecture for $t \in \{7, 8\}$. A few years later [62], Toft proved that G must also be t-edge-connected. This provides additional supporting evidence for all t. Very recently, Kawarabayashi has shown that G must be at least $\lceil \frac{t}{3} \rceil$-vertex-connected as well [38].

Following the same trend, we study both the vertex and edge connectivity of t-immersion-critical graphs. Because the immersion order includes the taking of subgraphs, we know that t-immersion-critical graphs are also t-color-critical. They are, therefore, 2-vertex-connected and $(t-1)$-edge-connected [63]. We now establish that such graphs are either isomorphic to K_t or 4-vertex-connected and t-edge-connected. We assume $t \geq 5$ unless stated otherwise. Kempe chains play a pivotal role in our investigation.

[1] This notion has sometimes been termed *t-contraction-critical*.

2.4.1 Vertex Connectivity

We begin with three easy and useful lemmas about cutsets, paths and coloring. Lemmas 1 and 2 are probably well known, although they may not be formulated elsewhere in precisely the same way we state them in this treatment. Lemma 2, which we dub *The Patching Lemma*, is especially helpful. Lemma 3 is certainly well known, and mentioned in a variety of sources (e.g., [9, 64, 68]). We include the proofs of these lemmas here both for completeness and, more importantly, to illustrate and clarify their utility in subsequent results.

Lemma 1 *Let S denote a minimum-cardinality vertex cutset in a 2-vertex-connected graph G, and let C denote a connected component of $G\backslash S$. Then any two elements of S must be connected by a path whose interior vertices lie completely within C.*

Proof Because G is 2-vertex-connected, $|S| \geq 2$. Let a and b denote any two distinct elements of S. It must be that a is adjacent to some vertex u in C, since otherwise S is not minimal ($S\backslash\{a\}$ defines a vertex cutset of cardinality $|S| - 1$). Similarly, it must be that b is adjacent to some vertex v in C. If $u = v$, then we are done. If $u \neq v$, then the connectedness of C ensures that there is a subpath P from u to v lying completely within C. Thus $au \cup P \cup vb$ is the desired path. ∎

Two colorings are said to be *equivalent* if the partitions induced by their respective color classes are identical.

Lemma 2 (The Patching Lemma) *Let S denote a vertex cutset of G, and let G_1 and G_2 denote a pair of induced subgraphs for which $G_1 \cup G_2 = G$ and $G_1 \cap G_2 = S$. If G_1 and G_2 admit t-colorings whose restrictions to S are equivalent, then G is t-colorable.*

Proof Let S, G, G_1 and G_2 be as defined in the statement of the lemma. Let c and d denote the specified t-colorings of G_1 and G_2, respectively. Modify one

coloring, say d, by renaming its color classes so that each element of S is assigned the same integer under both colorings. Patched together, c and d now provide a t-coloring of G. ∎

The Patching Lemma can be used to establish the following well-known fact.

Lemma 3 *No vertex cutset of a t-color-critical graph can be a clique.*

Proof Suppose otherwise for some G with cutset S. Remove S from G. Let C denote one resultant connected component, and let G_1 be the subgraph induced by $C \cup S$. Let G_2 denote the subgraph induced by $G \backslash C_1$. Because G is t-color-critical, G_1 and G_2 must each be $(t-1)$-colorable. And because S is a clique, any $(t-1)$-coloring of G_1 and any $(t-1)$-coloring of G_2 must be equivalent when restricted to S. But now by the Patching Lemma, G must also be $(t-1)$-colorable, and thus not t-color-critical. ∎

To simplify matters, we shall adopt the following conventions for the remainder of this subsection:

t is at least five,

G denotes a t-immersion-critical graph,

S denotes a minimum-cardinality vertex cutset in G,

C denotes a connected component of $G \backslash S$,

G_1 denotes the subgraph induced on $C \cup S$, and

G_2 denotes the subgraph induced on $G \backslash C$.

Lemma 4 *Every t-immersion-critical graph is 3-vertex-connected.*

Proof Suppose otherwise, as witnessed by some G with $S = \{a, b\}$. We know from Lemma 3 that the edge ab is not present in G. Let $i \in \{1, 2\}$. By Lemma 1,

there must be a path, P_i, with endpoints a and b, whose vertices lie completely within G_i. Lifting the edges of P_{3-i} to form the single edge ab, and then taking the subgraph induced by the vertices of G_i, produces a graph H_i properly immersed in G. It follows that H_i is $(t-1)$-colorable. Because ab is present in H_i, any such coloring of H_i assigns different colors to a and b. But G_i is a subgraph of H_i. Thus, there are $(t-1)$-colorings of G_1 and G_2 that each assign different colors to a and b. By the Patching Lemma, this ensures a $(t-1)$-coloring of G, which is a contradiction. ∎

Lemma 4 applies to t-topological-critical graphs as well. To see this, note that the two paths defined in the proof are vertex-disjoint. An analog of Lemma 4 does not hold, however, if the graph is only known to be t-color-critical. Such graphs are guaranteed only to be 2-vertex-connected. A t-color-critical graph that is not be 3-vertex-connected can be constructed as follows. Begin with a pair of non-adjacent vertices, u and v, a copy of K_{t-1} and a copy of K_{t-2}. Connect u to every vertex but one in the copy of K_{t-1}. Connect v to every vertex but some different one in the copy of K_{t-1}. Now connect both u and v to every vertex in the copy of K_{t-2}. Such a graph is depicted in Figure 2.2, with $t = 4$. Note that these graphs are not t-immersion-critical. In the instance shown, for example, K_4 is properly immersed using a model whose corners are u plus the vertices of K_3.

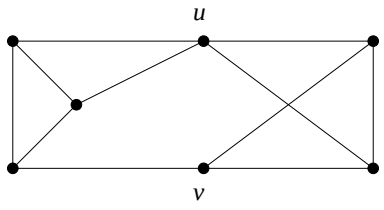


Figure 2.2: A 4-color-critical graph that is not 3-vertex-connected.

Lemma 5 *If $|S| = 3$, then G_1 and G_2 admit $(t-1)$-colorings that assign more than one color to the elements of S.*

Proof Let $S = \{u, v, w\}$, and consider the case for G_1. By Lemma 1, there is a path between u and v in G_2. Lifting this path and taking the subgraph induced by the vertices of G_1 produces a graph H properly immersed in G. Because G is t-immersion-critical, and because H contains the edge uv, H must admit a $(t-1)$-coloring that assigns different colors to u and v. As a subgraph of H, G_1 can likewise be colored. A symmetrical argument handles the case for G_2. ∎

What we have really just shown is that if G is only 3-vertex-connected, then G_1 admits a $(t-1)$-coloring that assigns different colors to any fixed pair of elements of S. This raises the possibility that a single coloring of G_1 may suffice, simultaneously assigning different colors to all three elements of S. We now show that this cannot happen, and that two distinct colorings are involved. It follows that the same must then be true for G_2.

Let a and b denote vertices of G, and let c denote a coloring of G in which $c(a) = i \neq j = c(b)$. If a and b belong to the same connected component of c_{ij}, then they are connected by some Kempe chain P_{ij} contained within c_{ij}. In this event, we say that a and b are *c-chained*.

Lemma 6 *If $|S| = 3$, then neither G_1 nor G_2 admits a $(t-1)$-coloring that assigns three different colors to the elements of S.*

Proof Suppose otherwise, as witnessed by a $(t-1)$-coloring c of G_1. Let $S = \{u, v, w\}$ and assume, without loss of generality, that $c(u) = 1, c(v) = 2$ and $c(w) = 3$. Let d denote some $(t-1)$-coloring of G_2. By Lemma 5 and the Patching Lemma, it must be that d assigns exactly two colors to the elements of S. So assume, again without loss of generality, that $d(u) = d(v)$. If u and v

are not c-chained, then we can exchange colors 1 and 2 in $c_{12}(v)$ to produce a $(t-1)$-coloring c' of G_1 that assigns color 1 to both u and v and leaves the color of w set to 3. This means that the restrictions of c' and d to S are equivalent. But now, by the Patching Lemma, G is $(t-1)$-colorable, which is impossible.

Thus it must be that u and v are c-chained by some P_{12} in G_1. Lifting this chain and taking the subgraph induced by the vertices of G_2 produces a graph H properly immersed in G. H contains uv, and so must admit a $(t-1)$-coloring d' that assigns different colors to u and v. G_2 is likewise colored by d'. By the Patching Lemma, d' cannot assign a third color to w. So assume, again without loss of generality, that $d'(w) = d'(u)$. If u and w are not c-chained, then (as in the previous argument) we can construct a $(t-1)$-coloring c'' of G_1 so that the restrictions of c'' and d' to S are equivalent, which is impossible.

Thus it must be that u and w are c-chained by some P_{13} in G_1. Because they are edge disjoint, P_{12} and P_{13} can be lifted simultaneously. Lifting these two chains and taking the subgraph induced by the vertices of G_2 produces a graph H' properly immersed in G. H' contains both uv and uw, and so must admit a $(t-1)$-coloring d'' that assigns different colors to u and v and different colors to u and w. G_2 is likewise colored by d''. By the Patching Lemma, d'' cannot assign three colors to the elements of S. So it must be that $d''(v) = d''(w)$. If v and w are not c-chained, then (as in the previous arguments) we can construct a $(t-1)$-coloring c''' of G_1 so that the restrictions of c''' and d'' to S are equivalent, which is impossible.

Thus it must be that v and w are c-chained by some P_{23} in G_1. Because they are edge disjoint, P_{12}, P_{13} and P_{23} can be lifted simultaneously. Lifting these three chains and taking the subgraph induced by the vertices of G_2 produces a graph H'' properly immersed in G. H'' contains uv, uw and vw, and so must admit a $(t-1)$-coloring d''' that assigns three different colors to S. G_2 is likewise colored

by d'''. This means that the restrictions of c and d''' to S are equivalent, which is impossible, contradicting the supposition that c exists. ∎

We are now ready to prove that minimum-cardinality vertex cutsets of t-immersion-critical graphs have at least four elements. The use of Kempe chains in Lemma 6 has been especially effective, so much so that we need only paths not chains in what follows.

Theorem 9 *Every t-immersion-critical graph is 4-vertex-connected.*

Proof Suppose otherwise, as witnessed by some G with $S = \{u, v, w\}$. Let c and d denote $(t-1)$-colorings of G_1 and G_2, respectively. By Lemmas 5 and 6, we restrict our attention to the case in which both c and d assign exactly two colors to elements of S. Without loss of generality, assume $c(u) = c(v)$ and $d(u) = d(w)$. By Lemma 1, there is a path P_1 in G_1 whose endpoints are u and w. Similarly, there is a path P_2 in G_2 whose endpoints are u and v. Lifting P_i and taking the graph induced by the vertices of G_{3-i} produces a graph H_{3-i} properly immersed in G. H_1 contains uv, and so must admit a $(t-1)$-coloring c' that assigns different colors to u and v. G_1 is likewise colored by c'. By Lemma 6, c' cannot assign a third color to w. Lest the restrictions of c' and d to S be equivalent, it must be that $c'(w) = c'(v)$. H_2 contains uw, and so must admit a $(t-1)$-coloring d' that assigns different colors to u and w. G_2 is likewise colored by d'. By Lemma 6, d' cannot assign a third color to v. But if $d'(v) = d'(u)$, then the restrictions of c and d' to S are equivalent. And if $d'(v) = d'(w)$, then the restrictions of c' and d' to S are equivalent. Thus, under some pair of colorings of G_1 and G_2, the Patching Lemma ensures that G is $(t-1)$-colorable, a contradiction. ∎

2.4.2 Edge Connectivity

We begin with a pair of well-known observations (see, for example, [68]).

Observation 4 *A minimum-cardinality edge cutset separates a graph into exactly two connected components.*

Observation 5 *If H is obtained by deleting the edge uv from a t-color-critical graph, then H is $(t-1)$-colorable and, under any $(t-1)$-coloring, u and v are assigned the same color.*

The next lemma plays an essential role in our edge-connectivity arguments. It is probably also well known, but may not be formulated elsewhere in exactly the same way we state it here.

Lemma 7 *Let H be obtained by deleting the edge uv from a t-color-critical graph. Let c denote a $(t-1)$-coloring of H with $c(u) = c(v) = 1$. Then $v \in c_{1i}(u) \forall i \in \{2, 3, \ldots, t-1\}$.*

Proof Let H and c be defined as stated. Assume the lemma is false, then there exists a color $i \in \{2, 3, \ldots, t-1\}$, such that $v \notin c_{1i}(u)$. Swapping colors 1 and i in $c_{1i}(u)$ produces c', another $(t-1)$-coloring of H. But then u and v are assigned different colors under c', which is impossible. ∎

For simplicity, we adopt the following conventions in the remainder of this subsection:

t is at least 5,

G denotes a t-immersion-critical graph,

S denotes a minimum-cardinality edge cutset in G,

C_1 and C_2 are "the" two components of $G \backslash S$,

S_1 and S_2 denote the sets of endpoints of elements of S contained in C_1 and C_2, respectively,

uv denotes an element of S, with $u \in S_1$ and $v \in S_2$, and

H denotes $G \backslash \{uv\}$.

Lemma 8 *If G is not t-edge-connected, then every $(t-1)$-coloring of H assigns either one color to S_1 and all $t-1$ colors to S_2 or vice versa.*

Proof Assume G is not t-edge-connected. Then S must have cardinality $t-1$. Being a subgraph of G, H can be $(t-1)$-colored by some coloring c that assigns the same color to u and v. WLOG, let $c(u) = c(v) = 1$. By Lemma 7, $v \in c_{1i}(u)$ for all $i \in \{2, 3, \ldots, t-1\}$. It follows that u and v are the endpoints of $t-2$ Kempe chains $\{P_{1i}\}_{i=2}^{t-1}$ such that $P_{1i} \subseteq c_{1i}$. Let $S' = S \backslash \{uv\}$. Then S' is a cutset of cardinality $t-2$ of H. By Observation 1, the chains are edge disjoint, and so each contains at least one distinct element of S'. Every path between u and v in H contains at least one edge of S'. Hence, there is a one-to-one correspondence between $\{P_{1i}\}_{i=2}^{t-1}$ and elements of S'. This means that every element of S' has an endpoint assigned color 1 by c. If c assigns only color 1 to S_1, then it must assign all $t-1$ colors to S_2. Similarly, if c assigns all $t-1$ colors to S_1, then it must assign only color 1 to S_2. The only remaining case to consider occurs if c assigns more than one but fewer than $t-1$ colors to S_1. To show that this cannot happen, we now proceed by contradiction, and suppose S_1 contains vertex w assigned color i, but no vertex assigned color j, where $\{i, j\} \subseteq \{2, 3, \ldots, t-1\}$. Since every edge of S' has an endpoint of color 1, w has no neighbor colored j in C_2. Then, it must be that $c_{ij}(w)$ is completely contained within C_1. Swapping colors i and j in $c_{ij}(w)$ produces c', another $(t-1)$-coloring of H with $c'(u) = c'(v) = 1$. By this construction, c' assigns color j to endpoints of two different elements of S', and

accordingly assigns color i to the endpoint of no element of S'. We conclude that $v \notin c'_{1i}(u)$, contradicting Lemma 7. ∎

Theorem 10 *Any t-immersion-critical graph other than K_t is t-edge-connected.*

Proof Suppose otherwise, then there exist some G that is t-immersion-critical, not isomorphic to K_t, and not t-edge-connected. By Lemma 8 and, without loss of generality, H admits a $(t-1)$-coloring c that assigns color 1 to $S_1 \cup \{v\}$. Thus all $t-1$ colors are assigned to S_2, and we index the elements of S_2 by $\{v = v_1, v_2, .., v_{t-1}\}$, where $c(v_i) = i$.

Let i and j denote distinct elements of $\{1, 2, .., t-1\}$. Then $c_{ij}(v_j)$ must contain v_i since, otherwise, we can exchange colors i and j in $c_{ij}(v_j)$ to produce a $(t-1)$-coloring of H in which the elements of S_2 are assigned $t-2$ colors, thereby contradicting Lemma 8. It follows that v_i and v_j are the endpoints of a Kempe chain, P_{ij} contained within $c_{ij}(v_j)$. Moreover, even if i or j is 1, the elements of S_1 are excluded from this chain. This being true since, if P_{ij} contains an element of S_1, then it will use at least two edges of $S' = S \backslash uv$ (one to reach S_1 and another to go back to C_2). This is impossible by Lemma 8.

Therefore P_{ij} is completely contained within C_2. Because such a chain exists for each pair of vertices in S_2, and because the chains are edge disjoint, K_{t-1} is immersed in C_2 using a model whose corners are the elements of S_2.

Now let i denote an element of $\{2, .., t-1\}$. From the proof of Lemma 8, we know that u and v are the endpoints of a Kempe chain containing v_i. This means that u and v_i are the endpoints of a subchain completely contained within $C_1 \cup S$. Because such a chain exists for each vertex in $S_2 \backslash \{v\}$, because the chains are edge disjoint, and because $uv \in G$, K_t is immersed in G using a model whose corners are $u \cup S_2$. This is the desired contradiction. ∎

Corollary 1 *If G is t-immersion-critical and not K_t, then $\delta(G) \geq t$.*

Proof Immediate from Theorem 10 and the fact that $\delta(G)$ is an upper bound on G's edge connectivity. ∎

Corollary 2 *If G is t-color-critical with a vertex u of degree $t-1$, then K_t is immersed in G via a model whose corners are $u \cup N(u)$.*

Proof Follows from the proof of Theorem 10 by letting S be the set of edges incident on u. ∎

2.5 On Settling the $t = 5$ Case

Let n denote the number of vertices in G. Mader has shown recently that any graph with at least $3n-5$ edges must contain a topological (and hence an immersed) K_5 [47]. A 5-immersion-critical graph G can therefore contain at most $3n-6$ edges, and so $\delta(G) \leq \lfloor 2(3n-6)/n \rfloor = 5$. We know from Corollary 1 that a 5-immersion-critical graph other than K_5 satisfies $\delta(G) \geq 5$.

Observation 6 *If G is 5-immersion-critical and not K_5, then $\delta(G) = 5$.*

Let K_5^- denote the graph obtained by deleting one edge from K_5. Recall that a pair of vertices is said to be c-chained if they are connected by some Kempe chain under coloring c.

Theorem 11 *If $\chi(G) \geq 5$, then K_5^- is immersed in G.*

Proof Any graph requiring five or more colors has a 5-immersion-critical graph immersed within it. Moreover, if that immersed graph is K_5, then we are done. Thus it suffices to consider only the case in which G itself is 5-immersion-critical and not K_5. By Observation 6, G has a vertex, u, of degree five. By Observation 3, G has a coloring c in which $c(u) = 1$ and $c(v) \neq 1$ for every vertex $v \in G - u$.

Let the neighborhood of u be denoted by the set $\{v, w, x, y, z\}$. It must be that c assigns colors 2 through 5 across this collection of five vertices, since otherwise u could be reassigned another color producing a 4-coloring of G. Thus, without loss of generality, we assume that v, w and x receive distinct colors and that y and z are colored the same. Every pair of vertices in $\{v, w, x\}$ must be c-chained, else one of them could be recolored. Similarly, every vertex in $\{v, w, x\}$ must be c-chained to at least one vertex in $\{y, z\}$. It follows that some vertex in $\{y, z\}$, say y, is c-chained to at least two elements of $\{v, w, x\}$. Each of the chains so identified is edge-disjoint with the others, as well as with the edges incident on u. Therefore the set $\{u, v, w, x, y\}$ forms the corners of K_5^- model immersed in G. ∎

It might be possible that a resolution of the $t = 5$ case could be based on properties that are indirectly associated with coloring or immersion containment. Note that the $t = 4$ case of Hajós' conjecture follows simply from the fact that $\delta(G) \geq 3$ for any 4-color-critical graph G. We suspect that every simple 5-edge-connected graph contains K_5 in the immersion order. If this suspicion is true, then of course the $t = 5$ case of our conjecture is settled, because 5-immersion-critical graphs are 5-edge-connected. It is easy to see that our suspicion may be justified only as long as we restrict our attention to simple graphs. See Figure 2.3.

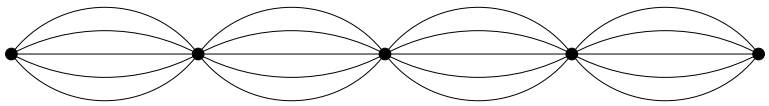


Figure 2.3: A 5-edge-connected multigraph of order five with no immersed K_5.

2.6 Graph Coloring and Cutwidth

For any graph G, $\Delta(G)$ and $cw(G)$ are related by the inequality: $cw(G) \geq \frac{\Delta(G)}{2}$. There is also a simple relation between $\delta(G)$ and $cw(G)$.

Lemma 9 *For any graph G, if $\delta(G) \geq r-1$ then $cw(G) \geq (\frac{r}{2})^2$.*

Proof Let G be a graph satisfying $\delta(G) \geq r-1$. Then G has at least r vertices. The lemma follows immediately from the simple fact that, in any linear layout of G, each of the first $\lceil r/2 \rceil$ vertices have at least $r/2$ neighbors among the rest of the graph. ∎

Theorem 12 *For any graph G, if $\chi(G) \geq t$ then $cw(G) \geq (\frac{t}{2})^2$.*

Proof Let G be a t-chromatic graph. Then G has a t-color-critical subgraph, H. Since $\delta(H) \geq t-1$, and by Lemma 9, $(\frac{t}{2})^2 \leq ch(H) \leq cw(G)$. ∎

Remark 9 We know that $Y_k(CW)$ is closed in the immersion order. We also know that the minimum degree of a t-immersion-critical graph, other than K_t, is t. Let G be a t-chromatic graph that does not contain K_t in the immersion order. Then G has immersed in it, a t-immersion-critical graph of minimum degree t. Therefore, by Lemma 9, $cw(G) \geq (\frac{t+1}{2})^2$.

2.7 Potential Applications

Despite the theoretical nature of our conjecture, it may have practical applications. As an example, consider the conjecture of Hadwiger along with ours. Together, they may form the basis of an efficient preprocessing strategy for the "very hard" graph coloring problem. This is because, at least in principle, both the immersion

and the minor containment of K_t in an arbitrary graph G have polynomial-time order tests for any fixed t.

To illustrate, consider the case $t = 5$. Assuming our conjecture is true (which we strongly believe), if we test and find that K_5 is absent from G in either the immersion or the minor order, then four or fewer colors are required to color G. Moreover, it seems that the two orders are rather orthogonal. See Figure 2.4. K_5 is a minor of the graph in Figure 2.4(a). It is found by contracting the edge uv. In Figure 2.4(b), K_5 is obtained by lifting uw and wy to form uy and lifting vw and wx to form vx. It is hoped, therefore, that when used together these two tests would provide a better "coloring filter" than when either is used alone. We note also that practical order tests have already been developed for the case $t \leq 4$ in the immersion order [10], and $t \leq 5$ in the minor order [48].

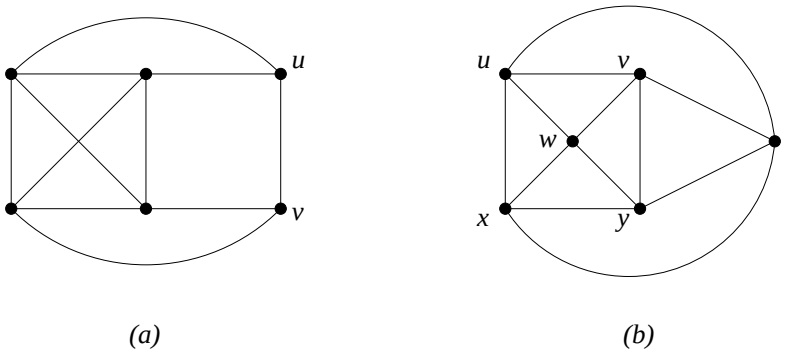


Figure 2.4: Graphs containing K_5 in the minor but not the immersion order, and vice versa.

Chapter 3

Algorithmic Techniques for $\mathcal{FPT}$ Problems

3.1 Kernelization

Fast preprocessing techniques have proved useful for many problems that require computationally expensive algorithms. In general, the goal is to reduce the size of any given input instance as much as possible, and guarantee that an equivalent instance is produced. In other words, solving the problem for the original input instance is equivalent to solving it for the preprocessed one.

Common Preprocessing Actions

When dealing with graph algorithms, common preprocessing techniques include the following actions:

(i) Checking if the input graph is not connected. If so, dealing with each (smaller) connected component is usually easier. The process of detecting connected components in a graph takes linear time since it's based on

breadth-first search. For this reason, most algorithms assume the input graph is already connected.

(ii) Dealing with isolated vertices. This may be regarded as a special case of (i) since an isolated vertex is nothing but a singleton connected component of the given graph. Such singletons are simply deleted with a possible change to the prospective solution. For example, in problems like Vertex Cover and Disk Dimension, isolated vertices can be automatically deleted (whenever detected). On the other hand, for problems like Face Cover and Dominating Set, deleting isolated vertices must be accompanied with adding an element to the solution (and decrementing the parameter).

(iii) Dealing with low degree vertices. We shall see in the next chapter that if an instance, (G, k), of the Vertex Cover problem has vertices of degree less than 3, then it can be preprocessed into another one, (G', k') such that $\delta(G') > 2$ and $k' < k$. Moreover, a pendant vertex can be deleted in almost all problem instances. It is often accompanied with a change to the problem instance and, if dealing with a search problem, the prospective solution needs to be updated.

(iv) Dealing with high degree vertices. Such vertices play an important role in many problems. For example, if (G, k) is an instance of the Cutwidth problem, the presence of a vertex of degree larger than $2k + 1$ in G implies automatically that it is a no instance.

(v) Detecting Special subgraphs. Some special subgraphs that can be quickly detected could have a dramatic impact on the final output answer. As an example, consider the Vertex cover problem. If the input instance is (G, k),

then the presence of a simple path[1] of length $2k+1$ in G automatically implies that (G,k) is a no instance.

For many $\mathcal{FPT}$ problems, some preprocessing actions are guaranteed to reduce, in polynomial time, the size of any input instance to a function of the parameter. This type of preprocessing, termed *Kernelization*, is now a common practice in the design of algorithms for $\mathcal{FPT}$ problems. It is conceived as the process of reducing a given problem instance to its combinatorial core, the problem *kernel* .

Example 1 The Vertex Cover problem has a well known simple kernelization that produces, for a given instance (G,k), a kernel (G',k') such that $|V(G')| \leq k'^2 + k'$. The idea is the following: if a vertex, v, has more than k neighbors then it must be in every vertex cover of size k. Otherwise, all of its neighbors must be used to cover the (at least $k+1$) edges connecting them to v. Iterative application of this simple rule yields an instance (G',k') such that every vertex of G' has at most k' neighbors. If a cover, C of size k' (or less) exists, then $G' \backslash C$ has at most k'^2 vertices (since it contains the neighbors of at most k' vertices).

We prove in a subsequent chapter that, along with some preprocessing methods, this kernelization of Vertex Cover produces an instance (G',k') such that $|V(G')| \leq \frac{k'^2}{3} + k$.

3.2 Bounded Search Trees

The bounded search tree technique is a simple and effective approach that proved useful for many interesting problems. It basically consists of an exhaustive search in a hypothetical tree whose size is bounded by a function of the parameter.

[1] not self intersecting

We say hypothetical since it is never actually constructed. The search is usually conducted recursively via *depth-first search*. The root of the search tree consists of the (usually kernelized) given instance of the problem. Each visited node is constructed in polynomial time (when needed). We illustrate with the following example from [21].

Example 2 Let (G, k) be a kernelized instance of the Vertex Cover problem. The search for a solution (or just an answer) proceeds using the search tree technique as follows. The root consists of the instance (G, k). If G is edgeless, then we are done since no edges are to be covered anymore. Now let uv be any any edge of G. Then either u or v (or both) belongs to the sought cover. Assuming u does, we construct a node corresponding to the instance $(G - u, k - 1)$. The search now proceeds recursively: If $(G-u, k-1)$ is a yes instance then we add u to the solution of $G - u$ and halt. Otherwise, it is still possible that a suitable solution can be found by adding v to the cover. So we construct another instance $(G - v, k - 1)$ and proceed similarly.

The height of the binary tree in the above example is bounded above by k since each time a node is constructed, its corresponding parameter value is one minus the parameter associated with its parent node. So if level k is reached, the parameter is zero and there is no need to proceed. It follows that the number of nodes in this particular search tree is bounded above by 2^k.

Note that we are dealing only with $\mathcal{FPT}$ problems. If the parameterized problem is not $\mathcal{FPT}$ then, unless the $\mathcal{W}$-hierarchy collapses, any search tree for the problem is not bounded by a function of the parameter.

Some efficient algorithms have been designed lately that use bounded search trees in a clever way. The most recent Vertex Cover algorithm uses a search tree whose size is bounded above by 1.2852^k [15]. This type of improvement is often

due to *branching* and *pruning* techniques. While branching is a rule by which children of a certain node are spawned, pruning occurs mainly in two cases: (1) If it is possible to foretell that one of the branches (at least) has no solution, and (2) if the existence of a solution for one branch implies that another branch has a solution (so it is safe to prune this branch). Algorithms that use a search tree are often called branching algorithms. Since they are usually preceded by kernelization, the terms kernelization and branching are very popular in the $\mathcal{FPT}$ community.

3.3 The Use of Tree Decomposition

Problems that are NP-hard in general are often solvable in polynomial-time when restricted to graphs of bounded treewidth. Once a tree decomposition, (T, Y), of width w is given for a graph G, it is often possible to use dynamic programming and the structure of the tree T to obtain algorithms whose run times are of the form $O(f(w)n^c)$. When a parameterized problem $\pi(k)$ admits this type of algorithm, it will be interesting to obtain a relationship between its parameter and the treewidth of its yes instances. For if $f(w)$ is $g(k)$, $\pi(k)$ is trivially $\mathcal{FPT}$.

A generic technique for the use of constant-width tree decompositions to obtain polynomial-time algorithms is described in [4]. We illustrate with a concrete example, by showing how to use the technique to obtain a uniform-poly-time algorithm for Vertex Cover. Before we proceed, we need the following, probably well known, Lemma.

Lemma 10 *The treewidth of any element of $Y_k(VC)$ is bounded above by k.*

Proof Let G be a graph that admits a vertex cover, C, of size k. Then a width-k path decomposition, (P, X), of G is constructed by setting $X_i = C \cup \{v_i\}$, where

$\{v_i\}_{i=1}^{n-k}$ is the set of vertices that are in the complement of C in $V(G)$. Thus $tw(G) \leq pw(G) \leq k$.∎

Let G be a yes instance of $VC(k)$ and let (T, Y) be an optimal tree decomposition of G. Then, by Lemma 10, $w(T, Y) \leq k$. We now show how to to use (T, Y) to solve VC optimally in time $O(4^k n)$. To do so, we make use of the following notations:

- r denotes the root of T
- t denotes the current (visited) node of T
- $G(t)$ is the subgraph of G induced by vertices of X_t
- $Sub(t)$ denotes the subtree of T whose root is t.
- $H(t)$ is the subgraph of G induced by vertices that appear in $Sub(t)$. For example, $H(r)$ is G itself.
- For each subset Z of G_t, $f(Z, t)$ denotes the size of the smallest vertex cover that contains Z.
- $f_t = min\{f(Z, t) : Z \subset V(G(t))\}$. In other words, f_t is the size of an optimal vertex cover of $H(t)$, provided such cover of size $\leq k$ exists.
- A table, M_t, is associated with each node t of T. Each row of M_t corresponds to a subset Z of $G(t)$ and is used to store two items: the characteristic vector α_Z of Z, and the value of $f(Z, t)$.

Since $w(T, Y) \leq k$, the number of rows of M_t is at most 2^{k+1}. The number of columns is at most $k+2$ and can be augmented by $|V(G)|$ to store the characteristic vector of the cover whose size if $f(Z, t)$ if dealing with the search version of the problem.

We start by initializing the tables M_t. If $Z \subset V(G(t))$ is a vertex cover of

$G(t)$, then the initial value assigned to $f(Z,t)$ is the size of Z[2]. This initialization step is performed by any traversal of T.

The next step is to update each table as we move up the tree T, in such a way that each node is visited after all its descendents. This task is shown in the function $UPDATETABLE$, shown below.

Function $UPDATETABLE$

Input: Table M_t corresponding to an internal node t of T, tables $\{M_{t'} : t'$ is a child node of $t\}$.

Output: Updated M_t if at least one row Z has $f(Z,t) \leq k$. Otherwise, exit and report that no global solution exists.

```
Begin function
flag ← 0
for each row Z in M_t do
    if f(Z,t) ≤ k
        for each child node t' of t do
            f(Z,t) ← min{f(Z',t) + |Z\Z'| : Z' ⊂ V(G(t'))}
            if f(Z,t) ≤ k
                flag ← 1
if flag = 0
    return 0
return M_t
End function
```

The pseudo-code for $UPDATETABLE$ does not show all the details. For

[2]It is easy to see that not all vertices of $G(t)$ are needed in the cover. So we assume that $|Z| \leq k$.

example, some flag can be used to guarantee that all children of a node have their tables updated and ready, so updating the current table becomes possible. The variable flag is used in the code to check whether the table M_t has at least one "potential" solution.

$UPDATETABLE$ is called first on nodes whose children are all leaves. Then it is called on all nodes whose (non-leaf) children have their tables updated, until it either stops to declare that no solution is possible, or reaches the root of the tree. If the size of the optimal vertex cover of G is not more than k, then f_r is this size.

The time required by $UPDATETABLE$ is quadratic in the number of rows of M_t. This is easily seen because of the number of comparisons needed for computing $f(Z,t)$ on line 5[3]. Therefore, it takes $O((2^k)^2) = O(4^k)$ time. Moreover, the number of nodes in a tree decomposition is $O(n)$, where n is the order of G. It follows that $VC(k)$ can be decided in $O(4^k n)$ time.

One may argue that the algorithm shown in example 2 is better. This is not the case in general. The upper bound of k on the treewidth of $Y_k(VC)$ graphs is very loose. In practice one would expect this treewidth to be a lot less. For example, planar graphs that belong to $Y_k(VC)$ have treewidth $O(\sqrt{k})$. However, although treewidth is $\mathcal{FPT}$, algorithms that construct optimal tree decomposition of graphs of fixed treewidth are still impractical.

[3]It is possible to do this computation in linear time by a careful ordering of the rows in each table. Our intention, however, is to illustrate the method and not to obtain the best possible algorithm

3.4 Pseudo-Kernelization

We now introduce a new technique for the design of efficient algorithms for $\mathcal{FPT}$ problems.

A *pseudo-reduction rule* is a reduction rule that applies only when some condition $P(\pi(k))$ is placed on the input instance of the parameterized problem $\pi(k)$. Applying this rule must provably reduce the size of any input instance to a function of the parameter k. Of course, we are only interested in obtaining a bound on the size of the kernel that improves on the one given by normal kernelization.

A *pseudo-kernelization algorithm* for a problem $\pi(k)$ is a fixed-parameter algorithm that uses a condition $P(\pi(k))$ in its search tree phase as follows: If $P(\pi(k))$ holds then reduce the size of the current instance. Otherwise, a better branching rule is applied due to the absence of P.

Pseudo-kernelization applies well to 3-$HS(k)$. The best algorithm for the problem is due to Niedermeier and Rossmanith and runs in $O(2.270^k + n^2)$. Their reduction rules lead to a kernel of size $O(k^3)$.

Recall that the input of 3-$HS(k)$ consists (in addition to the parameter k) of a collection of subsets of a set S where each subset is of size three or less. It is to be determined whether there exists a set, S', of k or fewer elements of S that has non-empty intersection with all elements of the collection C. Now consider the following condition:

$P(3HS(k))$: The input has no one single pair of variables (elements of S) that appear in more than three sets (i.e., three elements of C).

If P holds, then we have the following *pseudo-reduction* rule: If an element, u, of S appears in more than $3k$ sets, then add u to S'. The reason why we place u in S' is simple: If $u \notin S'$, then at most all k elements of S' would be needed to hit the $3k+1$ sets containing u. By the pigeon hole principle, at least one element of

S' would share 4 sets with u, violating P.

The pseudo-reduction rule (above) leads to a pseudo-kernel of size not exceeding $3k^2$. To see this, note that, every element of C has to contain one of the k elements of S'. Moreover, none of these k variables can appear in more than $3k$ sets. It follows that, after the pseudo-reduction rule is applied, C has at most $3k^2$ elements.

Now assume P doesn't hold. Then at least one pair $\{u, v\}$ appears in (at least) 4 sets. So there are (at least) 4 elements of S that appear with $\{u, v\}$ in a set. While searching for a solution we could branch with the following 3 cases: Either $u \in S'$, or $v \in S'$, or all (at least 4) elements appearing with $\{u, v\}$ are added to S'. Let $T(k)$ be the size of the subtree of the search tree that is rooted at a node associated with the current value of k. Assuming that P does not hold in this subtree, then we can branch according to the above branching rule and get: $T(k) = 2T(k-1) + T(k-4) = 2.107^k$ In other words, as long as P is absent, the branching rule we use is better than rules that do not assume its absence.

This technique is plausible in the sense that: if P holds then we have good news for kernelization and re-kernelization and when P doesn't hold then we have good news for branching.

Another nice feature of pseudo-kernelization is flexibility. It is possible to use another pseudo-kernelization condition P' that places more (less) restriction on the input to obtain a smaller (larger) pseudo-kernel but the speedup promised for branching would be slower (faster). As an example, if we replace 3 by 4 in P, we could still get a pseudo-kernel of size bounded above by $4k^2$ and a branching rule that leads to $T(k) = 2T(k-1) + T(k-5)(= 2.056^k)$.

3.5 Interleaving

Interleaving is a technique that is only used in a branching algorithm. The idea is simple: before applying a branching rule and moving to a child node in the search process, check if the current instance is amenable to kernelization (if a kernelization algorithm is known). If so, it is re-kernelized.

Let A and B be kernelization and branching algorithms (respectively) for a $\mathcal{FPT}$ problem $\pi(k)$. The run time of B is $O(f(k)g(k))$ where $g(k)$ is the (upper bound of the) size of the kernel produced by A and $f(k)$ is the exponential function obtained due to branching rules. It is proved in [52] that interleaving kernelization and branching as explained above, reduces the run time of B to $O(f(k) + g(k))$.

3.6 Other Techniques

Other kernelization and branching techniques appeared in recent fixed-parameter algorithms. We list some of them briefly;

Catalytic Branching: used in a branching algorithm for the max-leaf-spanning-tree problem [31]. It simply consists of using one vertex of the graph as a catalyst by assuming it to be in (or not in) the potential solution. To find such vertex, all $|V(G)|$ vertices are tried prior to branching.

Coordinatized Kernelization: Appeared also in [31] and is currently used in some new algorithms. The main idea is to assume the input, I, of a problem $\pi(k)$ satisfies an extreme condition: $I \in Y_k(\pi)$ and $I \notin Y_{k+1}(\pi)$ if π is a maximization problem, $I \notin Y_{k-1}(\pi)$ if it is a minimization problem. Accordingly, some reduction rules are drawn.

Crown Decomposition: A kernelization technique introduced recently by M. Fel-

lows [26] and is useful in at least a couple of $\mathcal{F}PT$ problems. In particular, it applies to the vertex cover problem, where a kernel of size not exceeding $3k$ can be obtained.

Chapter 4

A Case Study: The Vertex Cover Problem

We describe implementations of sequential and parallel algorithms for the Vertex Cover problem. This work is part of a general project aimed at the implementations of exact algorithms for $\mathcal{FPT}$ problems. The motivations behind this project are:

(i) Instances of $\mathcal{FPT}$ problems are, in general, amenable to reduction in size by kernelization techniques. The size of the kernelized instance is bounded above by a function of the input parameter. This property is believed to be the fundamental difference between $\mathcal{FPT}$ problems and other $\mathcal{NP}$-hard ones.

(ii) The use of search trees in most $\mathcal{FPT}$ algorithms caught our attention. Our intuition is: in such search spaces, solutions (when they exist) are not necessarily scarce. Moreover, some solutions tend to be in subtrees that are of modest size. This suggests that balanced decompositions of search trees

among several processors could sometimes lead to unusual speedups over sequential algorithms.

(iii) We were encouraged by the amount of resources present at the University of Tennessee. In particular, parallel implementations could benefit from the large number of processors that are available.

The asymptotically fastest known algorithm would solve the parameterized Vertex Cover problem ($VC(k)$) in $O(1.2852^k + kn)$ [15]. Earlier sequential implementations could solve the problem for $k \leq 200$ in less than an hour. Most recently, a parallel implementation raised this (so called) klam value[1] to more than 400 but with graphs of size not exceeding 800 [14]. We obtain better run times and raise the klam value considerably. Moreover, our codes run on inputs of arbitrary size.

The first version of our code is divided into four modules: Preprocessing Techniques, kernelization based on linear programming, sequential branching with repreprocessing (aka interleaving), and parallel branching based on search tree decomposition. We discuss these modules and present experimental results obtained by running our code on real and synthetic data sets.

4.1 Notations and Data Structures

Throughout this chapter, we assume the following:

G denotes the input graph of our $VC(k)$ algorithm,

k denotes the parameter,

[1]The klam value, associated with a problem $\pi(k)$, is defined in [21]. In this text, it is regarded as the current estimated upper bound on k for which solutions could be obtained in real time.

k' denotes the current value of k during the different steps of the algorithm

n denotes the order of G,

$n^{'}$ denotes the (current) order of $G^{'}$,

C is a potential vertex cover of size $\leq k$ (if any), and

vertices of G are indexed from 0 to $n-1$.

The adjacency matrix representation is used for graphs in the code. Because search tree algorithms are recursive, any implementation should take into account the memory overflow problem that could result. For best use of memory resources, we don't make physical changes to the adjacency matrix of G while searching for a solution. It is kept unchanged and is stored as a global variable, called $GRAPH$. All changes made to G are recorded in a vector of size n called the *status* vector.

The *status* vector keeps track of the status of vertices of G. Each vertex, v_i, is either in the cover ($status[i] = 1$), is not in the cover but still active ($status[i] = 0$) or it has been deleted ($status[i] = -1$). So, if G and *status* are given, we can easily obtain C and the current graph, G', obtained by a sequence of operations made to G up to the current stage.

Another useful vector is the *degree* vector, which keeps track of the current degree of each vertex. This is useful in the branching stage where vertices of high degree are preferred. So we don't have to compute the degrees at each node of the tree. Moreover, *degree* is used to delete isolated vertices (those whose degree drops to 0) and detect low degree vertices who play a major role in preprocessing (and repreprocessing). The use of *status* and *degree* is illustrated in functions $REMOVE\text{-}VERTEX$ and $ADD\text{-}VERTEX$ below.

Function $REMOVE\text{-}VERTEX$

Input: G', given by vector *status*, and $v \in V(G')$ given by its index s.

Action: Delete v from G'.

Begin function

If $status[s] \neq 0$

    error message and exit

$status[s] \leftarrow -1$

for i=0 to n-1 do

    if ($status[i] = 0$ and $GRAPH[i][s] = 1$)

        $degree[i] \leftarrow degree[i] - 1$

        if $degree[i] = 0$

            $status[i] \leftarrow -1$

            $n' \leftarrow n' - 1$

$n' \leftarrow n' - 1$

End function

Function $ADD\text{-}VERTEX$

Input: G', C (given by $status$), and v given its index s.

Action: Add v to C.

Begin function

if $status[s] \neq 0$

    error message and exit

$status[s] \leftarrow 1$

for i=0 to n-1 do

    if ($status[i] = 0$ and $GRAPH[i][s] = 1$)

        $degree[i] \leftarrow degree[i] - 1$

        if $degree[i] = 0$

            $status[i] \leftarrow -1$

$n' \leftarrow n' - 1$

$n' \leftarrow n' - 1$

$k' \leftarrow k' - 1$

End function

The status vector is useful in constructing the final C. This construction is needed since some operations, made to the graph in preprocessing and kernelization, change the graph (G') and the value of the parameter k', without adding vertices to the cover. For example, the vertex folding operation, described in the next section, changes G' and decrements k', but does not provide a clue upon what vertex is added to C until a final cover of G' is obtained. This folding operation requires a special stack of folded vertices represented by a $3 \times n$ array, $folded$, and an integer, $foldindex$, which holds the index of the top of the stack. $folded$ is used in order to undo some changes made to G' while constructing C.

4.2 Reduction Techniques

We describe a few preprocessing and reduction rules for $VC(k)$. Then we show how linear programming can be used for further reduction. The goal is to find a set $C \subset V(G)$ such that, if G has a vertex cover of size k, then at least one such cover must contain C.

The following lemma proved useful in obtaining strict upper bounds on the size of a preprocessed graph. Moreover, it helps us decide when it is unnecessary to use our costly linear programming kernelization.

Lemma 11 Let $G = (V, E)$ be a graph admitting a vertex cover of size k. Then

$$k(1 + \frac{1}{\Delta(G)}) \leq |V| \leq k(1 + \frac{\Delta(G)}{\delta(G)}) \tag{4.1}$$

Proof Let C be a vertex cover of size k of G. The complement, $\bar{C}$, of C is an independent set consisting of $n-k$ vertices. Let F be the set of edges that have endpoints in $\bar{C}$. Since each element of $\bar{C}$ has at least $\delta(G)$ neighbors in C, the number of edges in F is at least $|\bar{C}|\delta(G)$. In other words, $|F| \geq (n-k)\delta(G)$. The number of edges incident on elements of C is not smaller than $|F|$ and is not larger than $|C|\Delta(G)$. So $k\Delta(G) \geq |F| \geq (n-k)\delta(G)$. This proves that $n \leq k(1+\frac{\Delta(G)}{\delta(G)})$. Moreover, every element of C has at least one neighbor in $\bar{C}$. Which implies that $k = |C| \leq |F| \leq (n-k)\Delta(G)$. ∎.

A regular graph is one whose vertices are all of the same degree. Lemma 11 implies the following property of regular graphs.

Corollary 3 *If G is a regular graph of order n, then the size of an optimal vertex cover of G is at least $\frac{n}{2}$.*

4.2.1 Preprocessing

Preprocessing rules are often easy and fast. The rules described in this section are applied iteratively until none of them could further apply.

Rule 1 A pendant vertex may be deleted while adding its unique neighbor to the cover. This is done as follows: Let u be a pendant vertex and let v be its unique neighbor, then

- Add v to C
- decrement the degree of each neighbor of v
- delete those neighbors whose degree drops to zero
- decrement k'
- update the value of n' according to the number of vertices that are deleted.

Rule 2 Adjacent neighbors of degree-two vertices are included in the cover. To illustrate, let u be of degree two and let v and w be its neighbors. If vw is an edge of G', then at least two of the three vertices (u, v, and w) is in any vertex cover. So choosing v and w is not going to produce a cover that is larger than optimum.

Rule 3 (Introduced in [15]) Degree-two vertices whose neighbors are not adjacent can be "folded" as follows: (Again, let $N(u) = \{v, w\}$).

- $N_{G'}(u) \leftarrow N_{G'}(v) \cup N_{G'}(w)$
- delete v and w.
- decrement k'
- $n' \leftarrow n' - 2$
- $foldindex \leftarrow foldindex + 1$[2]
- Update folded to keep track of the current change (by storing indices of u, v, and w in $folded[foldindex][0]$, $folded[foldindex][1]$, and $folded[foldindex][2]$ respectively.

To see why this folding operation works, note the following:

(i) if a neighbor of u is placed in C, then u would be pendant and we can place its second neighbor in C. Thus it is safe to assume that either $u \in C$ or $\{v, w\} \subset C$.

(ii) If u is in any cover C of size k', then the new graph has a cover of size $k' - 1$: when u is in C, neither of v and w is in C. Therefore all neighbors of v and w must be in C. So, after folding u, the edges incident on u in the new graph are double-covered by u and by its new neighbors (formerly neighbors of v and w), which allows us to exclude u from the cover of the

[2]The initial value of $foldindex$ is -1.

new graph and decrement k'. In this case, the resulting graph has a cover of size $k' - 1$ if and only if the previous one has a cover of size k'.

(iii) if v and w are in any cover C of size k'. Then their edges can be covered by u in the new graph which would then have a cover of size $k' - 1$.

It follows that folding u results in a new graph whose optimal vertex cover is one less than that of the original graph.

Rule 4 (See [11]) Consider vertices of degree at least $k + 1$. Let v be any such vertex, then we add v to C since otherwise, C would contain all ($> k$) neighbors of v.

The procedure that takes care of all of the preprocessing rules is called $PREPROCESS$. After finding a solution, the cover is generated by reversing the folding operation. This is described by function $UNFOLD$ below.

Function $UNFOLD$

Input: $status$, $folded$ and the size, n, of $status$.

output: a vertex cover of G given by its characteristic vector $cover$.

Begin function

for $i = 0$ to $n - 1$ do

    if $status[i] = 1$

        cover[i] = status[i]

for $i = foldindex$ downto 0 do

    if $cover[(folded[i][0])] = 1$

        $cover[(folded[i][0])] \leftarrow 0$

        $cover[(folded[i][1])] \leftarrow 1$

        $cover[(folded[i][2])] \leftarrow 1$

else

$cover[(folded[i][0])] \leftarrow 1$

return cover

End function

Theorem 13 There is an $O(kn + n^2)$ algorithm that takes as input a graph G and a positive integer k and produces a graph, G' and an integer $k' \leq k$ such that: G has a k-Vertex-Cover if and only if G' has a k'-Vertex-Cover. Moreover: $|V(G')| \leq \frac{k'^2}{3} + k'$.

Proof Apply each of the above mentioned preprocessing actions iteratively. At each step, the value of k is decremented. Thus the $O(kn)$ run time claimed can be easily seen since all operations can be dealt with using the *status* and *degrees* vectors, each of linear size. Reading the graph (at least for initializing the *degrees* vector) takes $O(n^2)$. The resulting graph G' satisfies $3 \leq \delta(G') \leq \Delta(G') \leq k'$. Thus by Lemma 11, $|V(G')| \leq \frac{k'^2}{3} + k'$. ∎

4.2.2 Kernelization Based on Linear Programming

The optimization version of Vertex Cover can be stated as follows: assign a value $X_u \in \{0, 1\}$ for each vertex u of the input graph G, so that the sum of all vertices assigned the value 1 is minimized, and at least one of the endpoints of each edge is assigned a value of 1. In other words:

(1) Minimize: $\Sigma_{u \in V} X_u$.

(2) Subject to: $X_u + X_v \geq 1$ whenever $uv \in E$.

(3) Under the constraints: $X_u \in \{0, 1\}$ for all $u \in V$.

This is (obviously) an *integer programming* formulation of the problem. We refer to it by $VCIP$. In this context, the size of an optimal vertex cover is called

the objective function, and the set of all functions from $V(G)$ to $\{0,1\}$ that satisfy condition (2) is called the region of $feasible$ solutions.

Since Integer Programming (IP) is an $\mathcal{NP}$-complete problem, it is often useful to replace the constraints $X_u \in \{0,1\}$ by $X_u \geq 0$. Thus obtaining a *linear programming* (LP) formulation, $VCLP$, of the problem. This transformation from IP to LP is usually called an LP relaxation of the problem since the resulting problem has a "wider" range of $feasible$ solutions.

The value of the objective function returned by a $VCLP$ solver (which we call OPT_{LP}) is always a lower bound on the one returned by a $VCIP$ solver (called OPT_{LP}). One of the achievements of the work of Nemhauser and Trotter is the fact that $OPT_{IP} \leq 2 * OPT_{LP}$ [51, 37, 41]. A simple proof of this results appeared in [41]. We include a slightly modified version of this proof here as it helps clarifying our kernelization strategy of $VC(k)$.

Let $LPSOLVER$ be, as its name suggests, a program that takes as input a $VCLP$ instance and returns an optimal LP solution. Let $\{X_u : u \in V\}$ be the set of values assigned by $LPSOLVER$ to vertices of G.

Define:

$P = \{u \in V(G) : X_u > 0.5\}$,
$Q = \{u \in V(G) : X_u = 0.5\}$, and
$R = \{u \in V(G) : X_u < 0.5\}$.

Our LP-based kernelization is simple. We just add into the cover all elements of P and delete all elements of R. The remaining graph, G', is the one induced by the set Q. We now show why these operations are sound by proving that at least one optimal vertex cover contains P and, therefore, has an empty intersection with R.

Let A be the set of vertices of P that are not selected by some optimal Vertex Cover solution, OPT_{IP}. And let B be the subset of R that are selected by OPT_{IP}. We may assume $|A| \neq |B|$, since, otherwise, replacing B by A gives the desired optimal cover. Note that elements of B have no neighbors in Q. So, If $|A| < |B|$ then replacing A by B in OPT_{IP} gives a better cover. Moreover, if $|A| > |B|$ then a better LP solution would be obtained by setting $X_u \leftarrow X_u + \epsilon$ for all $u \in B$ and $X_v \leftarrow X_v - \epsilon$ for all $v \in A$, where $\epsilon = min\{X_v - 0.5 : v \in A\}$. This is impossible because $LPSOLVER$ must have delivered an optimal solution.

Notice that the size of any optimal vertex cover of G' is bounded below by $\Sigma_{u \in Q} X_u$, which is at least $0.5|Q|$. This is true since the optimal value returned by $LPSOLVER$ is bounded above by OPT_{IP} (in minimization problems). The kernelization algorithm tries to find vertices that belong to a cover C of size $\leq k$. This is achieved as follows:

- $k' \leftarrow k - |P|$.
- if $|Q| > 2k'$ then No solution (return 0). Otherwise:
- Add P to the cover C.
- Construct G' from the vertices of Q.

To illustrate, we only need to justify the second step. Clearly, $|C| \geq \Sigma_{u \in V} X_u = \Sigma_{u \in P} X_u + \Sigma_{u \in Q} X_u = |P| + \frac{|Q|}{2}$. Therefore, $k' = k - |P| \geq |C| - |P| = \frac{|Q|}{2}$.

When dealing with huge and dense graphs, the above LP formulation may not be practical, especially since the number of constraints (rows) is the number of edges in the graph, and we have to deal with graphs having millions of edges. For this purpose we use a code that solves the dual of our LP problem [18]. The dual of a minimization LP problem is a maximization one. Moreover, the number of constraints in the dual problem is exactly equal to the number of vertices of the

input graph (see [16] for more details about obtaining the dual of an LP problem). Other methods to speed up this LP kernelization appear in [37].

4.3 Search Techniques

We implemented sequential and parallel search techniques to solve $VC(k)$. For this first version, we chose a simple approach based on selecting a vertex, v, of highest degree and proceeding recursively by either adding v or all its neighbors to the cover. This operation is often referred to by "branching at vertex v". This is why search tree algorithms are sometimes called branching algorithms. Figure 4.1 shows a graph G and the (hypothetical) search tree that corresponds to running a $VC(4)$ branching on G. The dotted edge is never used.

The branching algorithm used in our implementation is described in [15]. Some of the branching rules are not incorporated yet because they rely on folding of degree 2 vertices, which is not implemented in the branching phase of our code at this point. Interleaving of folding is due to appear in the next version. The current version includes interleaving of preprocessing rules 1 and 4.

Recall the description of pruning in the previous chapter. For simplicity, let us say that a node is pruned if we can make it a leaf and not consider its instance for a solution. Several new pruning techniques are incorporated in our code over these proposed in [15]. In particular:

(i) At each node in the search tree, we check whether n' becomes larger than $k(1 + \frac{\Delta(G')}{\delta(G')})$. If so, the current node is pruned.

(ii) If the number of edges is larger than $k'\Delta(G')$ then the current node is pruned. (In a yes instance, k' vertices cover all edges. Each vertex of the k' vertices covers at most $\Delta(G')$ edges, so there are at most $k'\Delta(G')$ of them).

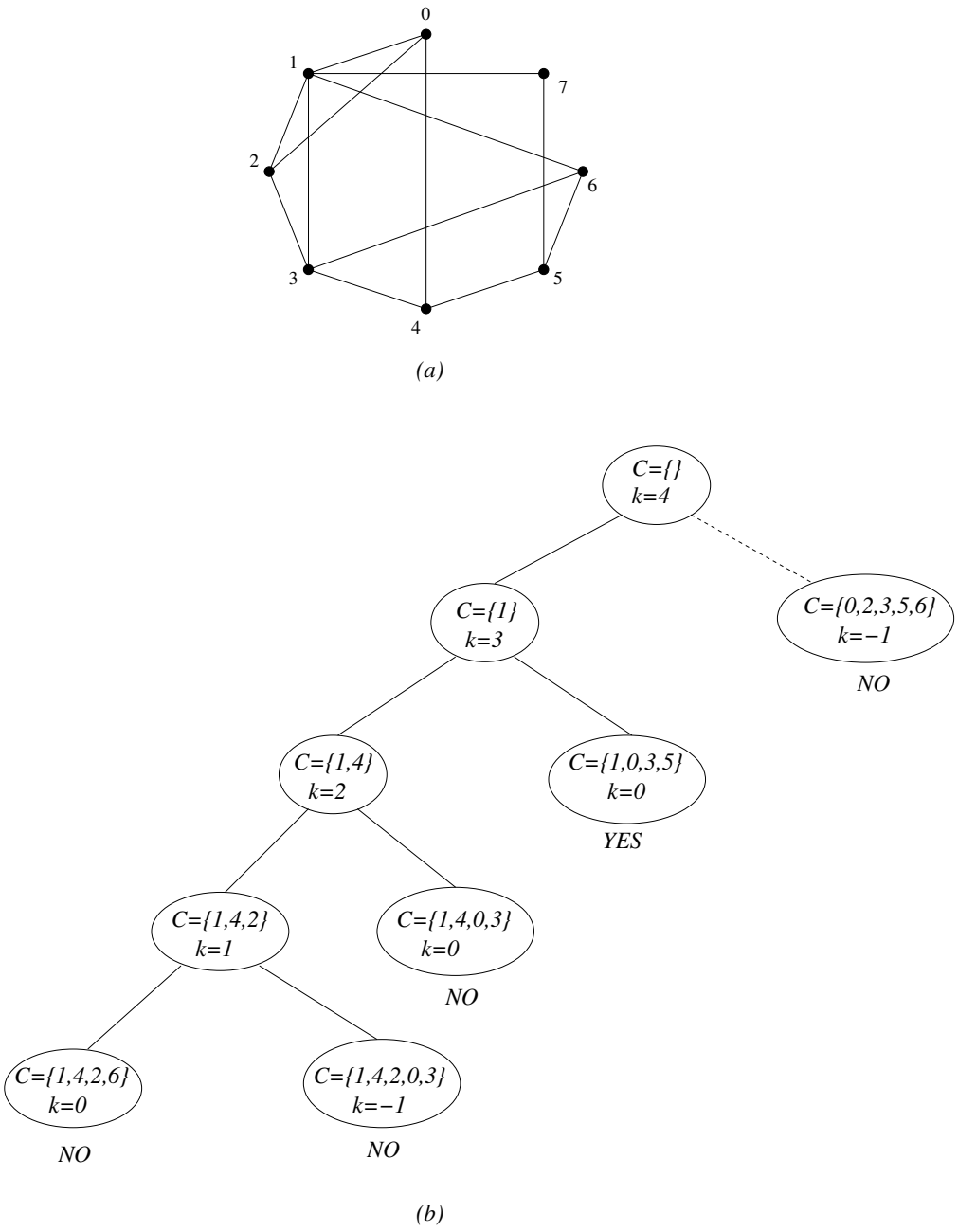


Figure 4.1: (a) The input graph, G, to $VC(4)$. (b) the corresponding search tree

(iii) If $\Delta(G')$ drops to 2, we just search for a solution of size k' in linear time. If none exists, the node is pruned, otherwise a solution is returned.

Our parallel implementation relies on a decomposition of the search tree: the search tree is decomposed into subtrees and each subtree is assigned to a different processor using, simply, SSH to the different machines available.

To illustrate how our search tree decomposition code works, consider a (general) search tree that corresponds to an input of $VC(k)$. Suppose we can use 4 processors. The first step in our decomposition would actually build the tree up to level 2 (since, if it were full, it would have 4 leaves). Then, if the number of leaves that do not correspond to no instances is less than 4, the algorithm proceeds to more levels by branching at the leaf with the largest value of k', since such leaf would correspond to the largest instance.

4.4 Experimental Results

4.4.1 Nonsynthetic Data Sets

Each real data set corresponds to a family of protein sequences who share a common domain. The sequences are obtained from NCBI [50] and, using ClustalW [17], pairwise alignments are performed and corresponding scores are recorded. A score between two sequences determines how aligned they are. The goal of biologists is usually to obtain a best representative phylogenetic tree for this protein family. It was noted in [14] that better phylogenetic trees are obtained by considering a subset of this data set consisting of sequences whose pairwise scores are higher than a certain threshold value, t. To accomplish this task, a graph is constructed so that vertices and edges correspond to protein sequences and pairwise

scores respectively. To get the desired subset, edges of scores below t are deleted and the resulting graph, G_1, is simple and unweighted. It is now easy to see that the desired set corresponds to a clique (i.e., complete subgraph) of maximum size in G_1. Since the *clique* problem is not $\mathcal{FPT}$, we use the fact that a clique in G_1 is nothing but the complement of a vertex cover in the complement, G_2, of G_1 (see figure 4.2). G_2 is often called the conflict graph (w.r.t. t). Our vertex cover code then comes into play, and is used iteratively until an optimal vertex cover is obtained for G_2. Once a solution is obtained, the sequences corresponding to cover vertices are removed from the data set and the remaining sequences are fed into ClustalW to get a phylogenetic tree.

Aiming at obtaining real data sets, we followed the same approach as in [14] and downloaded several sets of protein sequences from [50].

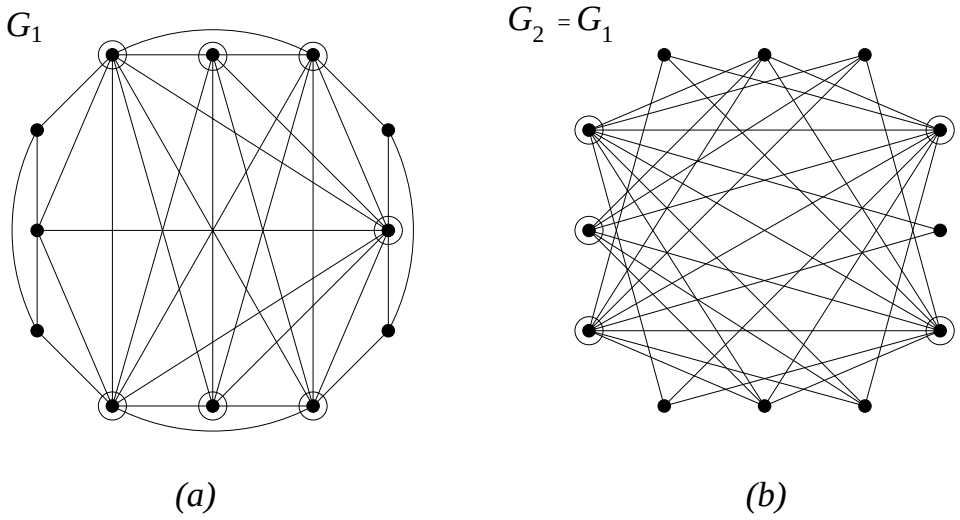


Figure 4.2: (a) A clique in G_1 is the complement of (b) a vertex cover in $G_2 = \bar{G}_1$

The following tables (Table 1 and Table 2) display some of our results. *PREPROCESS* and *LPSOLVER* are the codes that correspond to the previously described preprocessing and kernelization algorithms. The number of machines used in these experiments is 32, each is a 500 Mhz Sun UltraSPARC IIe processor. In all the tables below, time is measured in seconds unless otherwise stated.

Note that, since the data sets obtained are very dense, the decomposition strategy used in the parallel algorithm of this first version did not help in getting a satisfactory speedup over the sequential code. We observed that most processors finish too early being assigned subtrees corresponding to easily detectable no instances (the value of k is already decreased tremendously in most subtrees whose roots are close to the global root). Notice, however, the total run time required to find a solution.

Although we can't declare an upper bound on our klam value at this time, we are sure that it is at least 1000. In fact, we were able to obtain covers of size more than 2000 for data sets corresponding to sequences sharing the SH3 domain. The number of those sequences (at the time we downloaded them) is 2466. The graphs have, depending on the chosen threshold value, a number of edges ranging between 1.5 million and 2.2 million. For threshold 15, our code could finish in 17 minutes. The other cases were harder at this time and the current version is not appropriate for them. In fact, many experiments on the graph corresponding to threshold 5 could finish in one to two hours but, for threshold 10, the answer did not appear in real time (took more than a day). Based on the number of ways we can improve this $VC(k)$ code, our second version, which is due to appear soon, should be able to deal easily with the hard instances, and is expected to scale up to larger graphs as well.

Table1. Conflict Graphs of 972 sequences from the Globin protein family.

Threshold	k	Y/N	*PREPROCESS*			*LPSOLVER*			Sequential	Parallel
			Time	n'	k'	Time	n'	k'	Branching	Branching
5	284	Y	1	99	43	1	0	0	-	-
5	283	N	1	99	42	1	-	-	-	-
10	391	Y	1	867	289	9	221	117	1	not needed
10	390	N	1	867	288	9	221	116	9	not needed
15	427	Y	1	240	107	2	174	96	1	not needed
15	426	N	1	240	106	2	174	95	1	not needed

Table 2. Conflict graphs corresponding to 839 sequences with SH2 domain.

Threshold	k	Y/N	*PREPROCESS*			*LPSOLVER*			Sequential	Parallel
			Time	n'	k'	Time	n'	k'	Branching	Branching
5	399	Y	1	803	387	45	494	250	6	not needed
5	398	N	1	803	386	45	494	249	154 mins.	120 mins.
10	547	Y	1	726	435	69	616	389	19	15
10	546	N	1	725	433	66	615	387	62	55
15	606	Y	1	547	314	54	547	314	15	11
15	605	N	1	538	304	61	538	304	18	14

4.4.2 Synthetic Data Sets

Our synthetic data sets were obtained in several ways. A graph generator was written only to produce graphs with a vertex cover of size k and whose vertices have degrees in a given range. We generated many graphs and tested the code. The results were always impressive.

Our best experiments were obtained on data sets from Carleton University, where the parallel code was unusually faster than the sequential one. This really verified our intuition about the behavior of parallel codes that are based on the search tree decomposition technique.

Of course, the speedup shown in table 3 is not always possible. We report it only to show how high a speedup can be when a decomposition of search tree is used for parallelism. In fact, worst cases where very small (even negligible) speedup could happen. Especially when dealing with very dense graphs. In such an extreme case, the search tree looks like a path near its root and gets "thicker" at lower levels until it becomes almost full. Thus, all processors but one could finish quickly because they were assigned easy instances.

Although the extreme case described above is rare, it actually happened with a particular family of real data sets. Future versions of this code are expected to deal with this difficulty, mainly by developing techniques for reassigning jobs to processors that finish early.

Table 3. Graphs of size 600 obtained from Carleton University.

Graph	k	Y/N	*PREPROCESS*			*LPSOLVER*			Sequential	Parallel
name			Time	n'	k'	Time	n'	k'	Branching	Branching
RG30	400	Y	1	500	300	skipped	-	-	days	5
RG31	400	Y	1	500	275	skipped	-	-	days	4
RG32	400	Y	1	500	250	skipped	-	-	days	4

4.4.3 Observations

While running our first version on numerous graphs, we distinguished three types of behaviors:

(i) Powerful preprocessing and kernelization finish most (if not all) of the job (as shown in table 1).

(ii) Powerful sequential kernelization and branching (armed with interleaving) can do very well (see table 2). In fact, interleaving did really make a huge difference when incorporated. More preprocessing and kernelization rules will be interleaved in the next version.

(iii) On some hard instances, the sequential code could be very slow. These are instances that will benefit from our search space decomposition technique and would benefit a lot from increasing the number of processors. The results displayed in table 3 show how important is the incorporation of this parallel approach, which has the potential of achieving super-linear speedups.

Chapter 5

Fast Algorithms for the Face Cover Problem

We now turn our attention to the Face Cover problem. This work further illustrates the "kernelization and branching" methodology, and elucidates how the planarity of a graph can be exploited in this context.

In the optimization version of the face cover problem (FC), we are given a plane graph G, and seek the least number k of faces whose boundaries contain all the vertices of G. The corresponding decision version is $\mathcal{NP}$-complete [5].

When k is fixed and a face cover of size at most k exists, finding such a cover can be accomplished in linear time. All recent $FC(k)$ algorithms use a reduction to the parameterized Planar Dominating Set problem ($PDS(k)$). This reduction first appeared in [21] where an $O(12^k)$ flawed algorithm was presented.

$PDS(k)$ has two recent algorithms that are based on different approaches. The first, which runs in $O(8^k n)$ by using the bounded search tree technique, appeared in [2]. Although Face Cover was not mentioned, it is believed that the best $FC(k)$ algorithm would follow from this work and should have a run time of $O(8^k n)$ [26].

The second $PDS(k)$ algorithm uses dynamic programming on tree decomposition, and is used to solve $FC(k)$ (explicitly) in $O(3^{36\sqrt{(34k)}}n + n^2)$ [1]. Of course, this algorithm has a better asymptotic run time. Yet, an algorithm with a run time of $O(8^k n)$ would be preferred in practice.

Face cover has a highly practical linear-time algorithm when $k = 1$, since this is just recognition of outerplane graphs. However, even for $k = 2$, none of the known algorithms seems to be encouraging.

In this chapter, a direct $O(5^k + n^2)$ algorithm, $FACECOVER$, for $FC(k)$ is presented. Our algorithm is simple to implement, but its analysis is intricate. Most of this chapter is involved with the proofs of correctness and time complexity analysis.

The notation $G = (V, F)$ is used when referring to a plane graph. For a face $f \in F$, the set of vertices appearing in the ordered tuple associated with f is called the boundary of f and is denoted by $V(f)$. For a vertex $v \in V$, the set of faces whose boundaries contain v is denoted by $F(v)$.

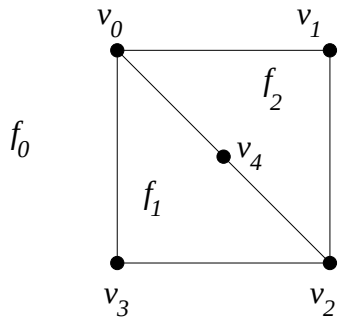
For a planar graph G, $\delta(G) \leq 5$. Which implies that we can always find a vertex that belongs to the boundaries of at most five faces. So, if we use the search tree technique, branching at such vertex introduces at most five "smaller" instances of the problem. The question now is, after performing this branch operation, is there any guarantee that another vertex of degree ≤ 5 is present in the resulting graph? The answer is usually no. We shall see, later in this chapter, that some reduction rules can be used to modify the graph and always guarantee (in the worst case) the existence of a vertex belonging to no more than five faces that qualify for membership in the cover.

Face Cover is a special case of the Hitting Set Problem (HS). To see why, note that a face cover of $G = (V, F)$ is a hitting set of the sets $\{F(u) : u \in V\}$. We mentioned earlier that $HS(k)$ is not $\mathcal{FPT}$, unless we fix the number of elements

allowed in each of the input sets. We denoted this restricted form of $HS(k)$ by d-$HS(k)$, where d is the (fixed) upper bound on the size of input sets. Thus, a d-HS algorithm can be used to solve the face cover problem when the degree of any vertex of the input plane graph is bounded by d. We deal, however, with the general face cover problem. Moreover, preprocessing techniques used for the general HS algorithm in [67] apply well to $FC(k)$ instances.

5.1 Preliminaries

We shall assume that our input is like the input of a HS algorithm. In fact, this is why we chose to characterize plane graphs by vertices and faces. The data structure used for a plane graph consists, essentially, of two lists corresponding to vertices and faces of the graph. The list associated with a vertex v starts with the number of faces in $F(v)$ then a list of the indices of these faces. Similarly, the list associated with face f contains the number of vertices in $V(f)$ followed by the indices of elements of $V(f)$. Figure 5.1 shows a plane graph and the corresponding input to our $FC(k)$ algorithm.



F(0): 3; 0,1,2
F(1): 3; 0,1,2
F(2): 3; 0,1,2
F(3): 3; 0,1,2
F(4): 2; 1,2

V(0): 4; 0,1,2,3
V(1): 4; 0,4,2,3
V(2): 4; 0,1,2,4

Figure 5.1: A plane graph and the corresponding input representation

The number of vertices of the given plane graph is denoted by n. When dealing with graphs, the size of input is often quadratic in n. This is due to the popular adjacency matrix representation. In our case, the graph is plane and, thanks to Euler's formula, has a linear number of edges and faces. We show that, according to our representation, the size of input is linear in n.

Lemma 12 *Let $G = (V, F)$ be a given plane graph, and let $V(f)$ and $F(v)$ be as defined above. Then the sets $\{F(v) : v \in V\}$ and $\{V(f) : f \in F\}$ have linear size.*

Proof Consider the bipartite graph $H = (A, B)$ constructed as follows:

(i) A and B consist of the vertices and faces of G respectively.

(ii) An edge of H connects a vertex v of A to a vertex f of B if and only if $f \in F(v)$ (if and only if $v \in V(f)$).

The number of edges of H is the same as the number of edges in G since, for any vertex v of G, $d_G(v)$ is the number of faces in $F(v)$ which is the same as $d_H(v)$. It follows that $|E(H)| \leq 3n - 6$[1]. Therefore: $|\{F(v) : v \in V\}| = \Sigma_{v \in V} d(v) = 2|E(G)| = 2|E(H)|$. This proves that $|\{F(v) : v \in V\}|$ is of linear size. Moreover, $|\{V(f) : f \in F\}| = \Sigma_{f \in B} d_H(f) = 2|E(H)| = |\{F(v) : v \in V\}|$.∎

We use the bounded search tree technique in our $FC(k)$ algorithm. During the search process, the vertex set is partially covered by the already selected faces. We shall, then, reduce the graph at each step by eliminating covered vertices. While this action is easy (and safe) when dealing with a general HS instance, it must be performed carefully in our case. Especially because we need to be assured that every node in the search tree has at most 5 children. Moreover, deleting a covered vertex from a plane graph might change the size of an optimal face cover

[1] we show in the next chapter that, if G is a yes instance of $FC(k)$ then $|E(G)| \leq 2n - 3k + 6$.

and produce incorrect results. As an example, consider the graph shown in figure 5.2. If the algorithm selects the outer face and deletes all its vertices, the resulting graph has a face cover of size one, while the original graph has a minimum face cover of size three. Our algorithm deals carefully with covered vertices. Before deleting a covered vertex, v, we modify all faces containing it by, simply, deleting v from their (ordered) lists. This face compaction will be called "the surgical operation" later in this chapter. It is depicted in figure 5.3.

5.1.1 Annotated Plane Graphs Representation

For a given plane graph $G = (V, F)$, vertices and faces are of two types: active and marked. Active vertices are those to be covered (i.e., not covered yet) and active faces are those that can be used to cover active vertices. A plane graph with active and marked vertices and/or faces will be called *annotated* in this text.

A general version of the face cover problem was presented in [5], where not all vertices of the graph are to be covered. Our algorithm will be ready to deal with this version as well. In fact, if a vertex is not to be covered, then we may assume that it has been covered earlier during the process of finding the face cover and it will be marked.

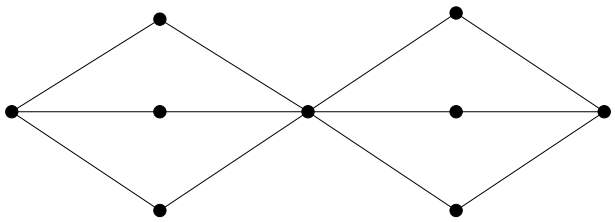


Figure 5.2: Deleting the vertices that are covered by selecting the outer face produces a wrong answer.

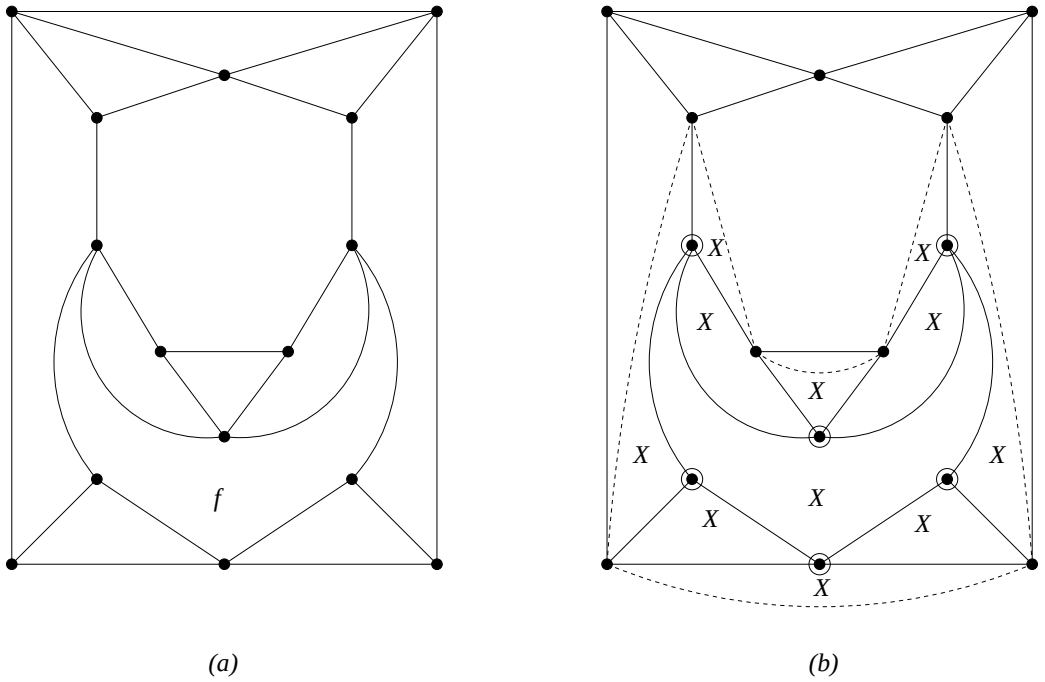


Figure 5.3: (a) Before selecting face f. (b) After selecting f. The circled vertices are marked. For simplicity, edges between marked vertices are not contracted in this figure. Dotted edges and the regions marked with X are "implicitly" present but not actually added to the graph.

Marking a face f is done by deleting $V(f)$ after replacing the index of f by -1 in the list(s) $F(v)$ of each vertex $v \in V(f)$. This process takes $O(|\{F(v) : v \in V(f)\}|)$. Thus it is $O(n)$. We will refer to this procedure by $MARK\text{-}FACE(f)$.

Similarly, marking a vertex v is done by deleting $F(v)$ after replacing the index of v by -1 in the list(s) $V(f)$ of each $f \in F(v)$. This procedure, denoted by $MARK\text{-}VERTEX(v)$, takes time $O(|\{V(f) : f \in F(v)\}|)$ which is $O(n)$. To show that such simple operation is sound, we prove it to be equivalent to a surgical operation on the graph.

5.1.2 A Surgical Operation

Two neighbors, u and w, of a vertex, v, are consecutive if some face, f, contains the three vertices such that one if the two ordered tuples (u, v, w) and (w, v, u) is a sub-tuple of the ordered tuple of f.

If a vertex, v, is of degree at least 2, and is marked, then active faces that are adjacent to it will not be needed to cover it. Deleting v could lead to wrong answers as shown in Figure 5.2. So, prior to removing v, we make sure that the marking operation avoids such cases. The marking operation simply consists of (1) contracting edges between v and all marked neighbors of v, then (2) adding edges between consecutive active neighbors of v (even if they are adjacent) and marking all faces that contain v in the resulting graph. This action is safe in the following sense: every face that is adjacent to v and is used to cover a neighbor of v, must contain two consecutive neighbors of v. Thus, adding an edge between consecutive neighbors preserves active faces that might later be needed to cover neighbors of v (and possibly other vertices). We refer to this operation on the graph by the surgical operation. In our implementation of the algorithm, we did not need to perform the surgical operation. It is used only to show that the

algorithm has the claimed performance and the operations of marking vertices and faces are sound.

Notice also the case where only one vertex of a face is left active at a certain point. In this case, the face is still active unless the vertex belongs to some other face. We shall see that preprocessing rules detects this case and deals with it, leaving no such faces in the graph. Figure 5.3, which shows (a snapshot of) the effect of the surgical operation, depicts some pendant faces.

5.2 Preprocessing

For simplicity, we assume the given graph is connected. Due to the chosen input representation, however, our code works in both cases. Our preprocessing consists of two main rules:

> The Dominated Face Rule: If two faces, f and f', of G are such that $V(f) \subset V(f')$, then f is said to be dominated by f'. In such case, f can be marked since every face cover of G that contains f can be replaced by a (possibly better) cover that contains f'.
>
> The Dominated Vertex Rule: If two vertices, u and v, are such that $F(u) \subset F(v)$, then v is said to be dominated by u, and v is marked since any face that covers u will cover v.

Checking if a vertex (face) is dominated is done by a search through all the sets of $\{F(v) : v \in V\}$ ($\{V(f) : f \in F\}$). It thus takes linear time. It follows that checking if the graph has dominated vertices or faces can be accomplished in $O(n^2)$. Each dominated vertex or face is then marked in $O(n)$ time. Therefore, the total run time of preprocessing is $O(n^2)$. We shall refer to this preprocessing algorithm by procedure $PREPROCESS$ later in this chapter.

We noticed that many preprocessing actions described in our previous version of the algorithm (as well as subsequent face cover kernelization techniques by other authors) were just particular cases of the above two rules. For example, if all interior vertices of a path, P, of length more than two are of degree two, then we used to contract all interior edges (edges between interior vertices). Thus keeping one (necessary) interior vertex of degree two. This turns out to be the dominated vertex rule since two interior degree-two vertices have the same set of active faces. Moreover, the (easy) pendant vertex rule, which simply adds the unique face containing a pendant vertex to the cover, is also a special rule of the dominated vertex rule since the (unique) neighbor of a pendant vertex is dominated.

5.3 Kernelization

This section is devoted to the proof of the following theorem.

Theorem 14 Let $G = (V, F)$ be an instance of $FC(k)$. There is an algorithm that runs in $O(n^2)$ and produces another, instance $G' = (V', F')$, of $FC(k')$ such that:

(i) G is a yes instance of $FC(k) \iff G'$ is a yes instance of $FC(k')$,

(ii) $k' \leq k$, and

(iii) $|V(G')| \leq 2k'^3$.

Lemma 13 Three or more vertices of a plane graph, G, may not be common to more than two faces of G.

Proof Assume vertices u_1, u_2, and u_3 belong to three faces, f_1, f_2, and f_3. Construct another plane graph, G', by drawing three vertices, v_1, v_2, and v_3 such that

vertex v_i is placed inside face f_i and joined to all vertices that lie on the boundary of f_i. The subgraph of G' induced by vertices $\{u_i\}_{i=1}^3$ and $\{v_i\}_{i=1}^3$ is isomorphic to $K_{3,3}$. This is a contradiction. ∎

Lemma 14 Let G be a a yes instance of $FC(k)$. If two faces of G have more than $2k$ common vertices, then any face cover of size k contains at least one of them.

Proof Let C be a face cover of size k. Assume faces f_1 and f_2 contain $2k+1$ common vertices and are not elements of C. Then, by the pigeon hole principle, some element of C must cover at least three of the $2k+1$ common vertices of f_1 and f_2. This is impossible by lemma 13. ∎

Corollary 4 Let G be a yes instance of $FC(k)$. If no pair of faces of G have $2k+1$ common vertices, then every face whose length exceeds $2k^2$ is in any optimal face cover of G.

Corollary 5 Let G be a yes instance of $FC(k)$. If $f \in F$ has more than $2k$ vertices with more than k faces, then f must be in any optimal face cover of G.

We now describe the kernelization process and use it in a proof of Theorem 14. This process is referred to, later, by procedure $KERNELIZE$.

The first step in $KERNELIZE$ is a simple search for all faces of length $> 2k'^2$, where k' is originally equal to k. This is done in $O(n)$ time since the length of a face is captured when reading input. All faces that are found whose length exceeds $2k'^2$ are kept in a list, L.

The second step is the following: For each face $f \in L$, the number of common vertices with all other faces of L is computed. (We may only consider faces that contain at least one common vertex with f). If a face f' has more than $2k'$

common vertices with f then their common vertices are all added to a list M of vertices that are to be marked[2], and a virtual new vertex, v, is added to the list of vertices such that $F(v) = \{f, f'\}$. This operation is equivalent to adding a (virtual) degree-two vertex that dominates all common vertices of f and f'. If no such face f' exists, then, because of corollary 5, f is added to a list C of faces that are in the cover and are to be marked. Moreover, if the number of faces that share more than $2k'$ common vertices with f exceeds k', then f is also selected in the cover and, thus added to list C. Otherwise, more than k' faces will have to be in the cover. Note that, because of Lemma 14, we stop the search if more than k' disjoint pairs of faces have (at least) $2k' + 1$ common vertices.

The last step deals with a cleanup of the lists and an update of the parameter: vertices in M are marked, faces in C are marked together with their boundary vertices, and the parameter k' is replaced by $k - |C|$ (k is unchanged while C, being originally empty, is filled with cover faces). Then list L is emptied and the process is repeated from step one until no more faces of length $> 2k'^2$ are found (or no solution can be found).

Proof of Theorem 14:

Algorithm $KERNELIZE$ just described satisfies the three conditions of the theorem. In particular, condition (iii) follows from corollary 4. As for the time complexity, we note the following:

(i) Step one takes $O(n)$ since it's a simple filling of list L by a one pass through the list of active faces.

(ii) Step (ii) takes $O(kn)$: for each face $f \in L$, we either find another face f' that shares $2k' + 1$ vertices with f or we add f to C. By Lemma 14, no more

[2]Note that, by the surgical operation, active faces do not have marked vertices on their boundaries.

than k pairs of faces can share $2k+1$ vertices, and no more than k faces can be added to C. Hence, the number of iterations in step two is at most k throughout the whole process.

(iii) The last (cleanup) step takes time $O(n^2)$ since it deals with marking vertices and faces that are in M and C respectively. The proof of Theorem 14 is now complete.

5.4 A Direct Face Cover Algorithm

Our direct algorithm is represented by the procedure $FACECOVER$ shown below. Subroutines $PREPROCESS$, $KERNELIZE$ and $MARK\text{-}FACE$ correspond (obviously) to the processes previously described in detail.

Procedure $FACECOVER$

Input: A plane graph $G = (V, F)$ given by $\{F(u) : u \in V\}$ and $\{V(f) : f \in F\}$, and an integer $k \geq 1$.

Output: A face cover, C of size $\leq k$ of G if one exists. NULL otherwise.

Begin procedure

$k' \leftarrow k$

$(G, C, k') \leftarrow PREPROCESS(G, k')$

$(G, C, k') \leftarrow KERNELIZE(G, k')$

Select an active vertex $v \in V$ such that $|F(v)| = min\{F(u)|u \text{ is active in } V\}$.

For every $f \in F(v)$ do

$C_1 \leftarrow C \cup \{f\}$;

$G_1 \leftarrow MARK\text{-}FACE(f)$;

$(G_1, C_1, k') \leftarrow KERNELIZE(G_1, k' - 1)$;

$C_2 \leftarrow FACECOVER(G_1, k')$;

if($C_2 \neq NULL$)

return $C_1 \cup C_2$

return NULL

End procedure

We shall prove that, at every call to $FACECOVER$, the selected vertex, v, has no more than five active faces in its $F(v)$ list. We know that the first call is guaranteed to select such vertex. As a remark, we note that, at least three such vertices are present in the graph. This is guaranteed by virtue of Euler's formula and the following lemma, which first appeared as a corollary in [54].

Lemma 15 If G is a planar graph, then G has at least three vertices of degree ≤ 5.

Proof Let $m = |M = \{v \in V(G) : d(v) > 5\}|$ and $l = |L = \{v \in V(G) : d(v) \leq 5\}|$. Then: $3n - 6 \geq e(G) \geq \frac{\sum_{\mathrm{v\in L}} \mathrm{d(v)} + \sum_{\mathrm{v\in M}} \mathrm{d(v)}}{2} \geq \frac{\sum_{\mathrm{v\in L}} \mathrm{d(v)} + 6\mathrm{m}}{2} \geq \frac{\sum_{\mathrm{v\in L}} \mathrm{d(v)} + 6(\mathrm{n-l})}{2} \geq \frac{\sum_{\mathrm{v\in L}} \mathrm{d(v)}}{2} + 3n - 3l$. Therefore $l \geq 2 + \frac{\sum_{\mathrm{v\in L}} \mathrm{d(v)}}{6} > 2$ ∎

If a vertex, v, is the only active vertex of face f, then f will only be selected (and marked) if v doesn't belong to any other face. Otherwise, it would be dominated (thus marked). We can, therefore, assume that every active face has at least two active vertices.

Faces of length two may exist due to the surgical operation which could introduce multiple edges between two vertices. This case is easily handled by $KERNELIZE$ since two vertices cannot belong to more than one face of length two (by the dominated face rule).

Lemma 16 Let v be an active vertex of an annotated plane graph, G. Then no marked neighbor of v belongs to an active face of v.

Proof The lemma follows immediately from the surgical operation.

Theorem 15 $FACECOVER$ runs in $O(5^k + n^2)$ time.

Proof At each call to $FACECOVER$, the (plane) subgraph induced by active vertices of G_1 must have a vertex, v, of degree ≤ 5. By Lemma 16, the active faces in $F(v)$ are faces that are common to v and its active neighbors. Thus the number of such active faces would exceed five only if v has multiple edges with at least one of its neighbors. Which means that v belongs to faces of length two. However, a face of length two will only be active if it's the unique face that is common to its two vertices. This proves that each node in the search tree has at most five children. Having no more than k levels, the search tree has at most $O(5^k)$ nodes. Thus pure branching would take $O(5^k k^3)$ (after the ($O(n^2)$ kernelization). since interleaving is used as in [52], the run time of branching reduces to $O(5^k + k^3)$ and the overall run time is $(5^k + k^3 + n^2)$. This completes the proof. ■

5.5 Remarks

We implemented our direct FC algorithm. Our code was tested on input plane graphs of size up to 200. The purpose of testing the code at this time was checking for correctness but we noted that answers were obtained in seconds especially after incorporating interleaving.

So far, we don't know of any other code for FC. It is interesting to see how ours would compare to the other indirect algorithm of Alber et al. [1]. As noted earlier, we anticipate that their use of tree decompositions and dynamic programming may impose some overhead. This justifies our direct approach, which is a lot faster when the size of the cover is small.

Chapter 6

Approximation Algorithms for Planar Graphs of Fixed Disk Dimension

We now turn our attention to planar graphs. We derive some useful properties of planar graphs whose optimal disk dimension is fixed. This work originally aimed at extending the work of Govindan et al. [33] to obtain a fast approximation algorithm for the pathwidth of planar graphs that have disk dimension two or more. We present a linear time algorithm for graphs of any fixed disk dimension.

The proofs in this chapter rely heavily on Kuratowski's theorem [42]. So we will assume from this point on that neither K_5 nor $K_{3,3}$ could be topologically contained in a planar graph. We also use the facts that outerplanar graphs exclude only $K_{2,3}$ and K_4 in both the minor and the topological orders, while series-parallel graphs exclude only K_4 in both orders.

Let DD_k denote the family of planar graphs whose disk dimension is at most k. We refer to such planar graphs by DD_k graphs. They enjoy several interesting

properties that are discussed in the following two sections.

6.1 Disk Dimension and Face Cover

Given a DD_k graph G, we know that G has an embedding in the plane so that all vertices lie on the boundaries of at most k disks. Such disks can be treated as faces since they have empty interior.

On the other hand, let H be a graph that has at least one embedding in which k faces cover all vertices. Since the interiors of the k faces are empty, one would treat them as the k disks in a DD_k layout of H. But where do we draw a vertex which belongs to more than one of the cover faces? We ask this question because we would like to draw the disks so that no two of them share a common vertex. The answer is easy. Since the boundaries of the k disks in a DD_k layout need not be cycles of the graph, we dictate that such vertex belongs to the boundary of exactly one disk. So it will be possible to treat the set $\{C_i : C_i$ is the boundary of disk $d_i\}$ as a partition of G. This is illustrated in figure 6.1 where vertex u belongs to two cover faces but is drawn on one of the two disks.

A DD_k layout of a graph G whose disk dimension is k is considered an optimal layout. Similarly, if a planar graph H has no embedding in which fewer than k faces cover all vertices, then any embedding that has a face cover of size k is considered an optimal embedding of H. Figure 6.1 shows a DD_2 layout of ${K_5}^-$. Noting that DD_1 graphs are the outerplanar graphs and that K_5- is not outerplanar (since it contains a K_4 subgraph), the layout shown in this figure is optimal.

Despite the similarity between disk dimension and face cover, it is important to notice the difference between the problems $DD(k)$ and $FC(k)$ defined earlier. Knowing that a given embedding of a planar graph, G, has a face cover of size

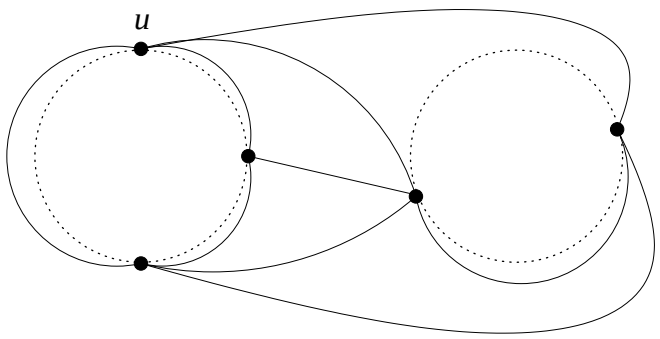


Figure 6.1: A DD_2 layout of ${K_5}^-$

k implies that its disk dimension is at most k, but does not provide any lower bound. Figure 6.2 shows two embeddings of an outerplanar graph of which one is optimal and the other has an optimal face cover of size four.

6.2 Fundamentals

We use c_i to denote the center of disk d_i. The wheel graph of G with respect to $\{d_i\}_{i=1}^k$ is defined by: $G^w = (V \cup \{c_i\}_{i=1}^k, E \cup \bigcup_{i=1}^k \{uc_i : u \in C_i\})$. Note that G^w is defined in terms of a particular layout and, therefore, it is not necessarily unique. The planarity of G^w is useful. We will employ it in many of our proofs.

The distance between a pair of vertices in a graph G is the length of the shortest path connecting them. The diameter of G is the maximum of all distances between pairs of vertices of G. It is easy to visualize that, if G is a connected DD_k graph given with some optimal DD_k layout, then the diameter of the corresponding G^w is bounded above by $3k-1$. To see this, let c_0 and c_1 be the centers of two adjacent disks (d_0 and d_1 respectively). Then some pair of vertices, $(u, v) \in C_i \times C_j$ are adjacent in G. The path $c_0u \cup uv \cup vc_1$, in G^w, connects the two centers and is of

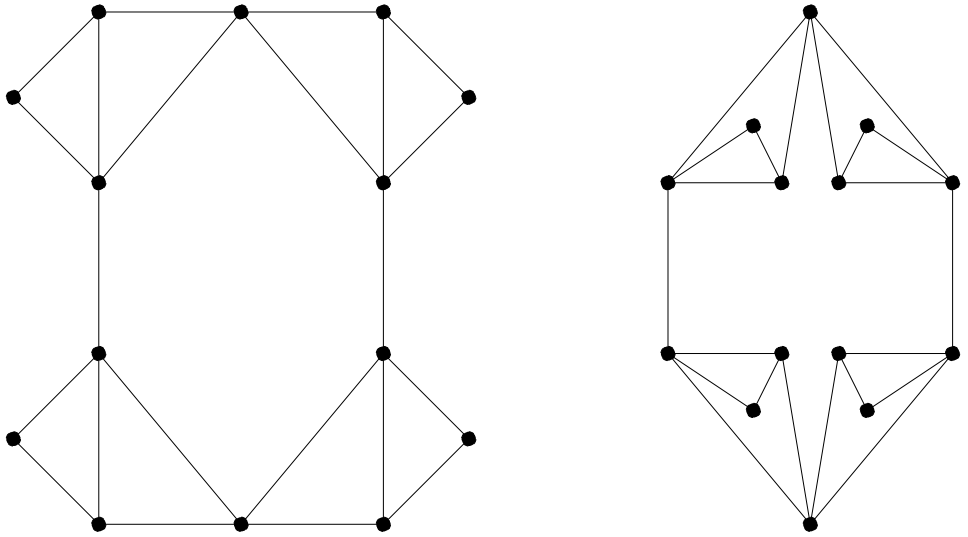


Figure 6.2: (a) An outerplane embedding of an outerplanar graph. (b) Another planar embedding of the same graph

length 3. It follows that, in the worst case, the distance between any two centers c_i and c_j does not exceed $3(k-1)$. Finally, since any vertex of G is a neighbor of (exactly) one of the centers of G^w, the distance between any two vertices is at most $2 + 3(k-1) = 3k - 1$.

A planar graph G is maximal planar if $\forall u, v \in V(G), uv \notin E(G) \Rightarrow G' = (V(G), E \cup \{uv\})$ is not planar. The disk dimension of such a graph is at least $n/3$, where n is the number of vertices. To see this, we note that in any DD_k layout of a maximal planar graph, none of the k disks can contain more than three vertices (but, a disk may have to contain fewer than three). In fact, $n/3$ is only a lower bound, since some graphs have disk dimension more than $n/3$. For example, it follows from Theorem 17 (to follow) that $dd(K_{2,11}) = 6 > \lceil (13/3) \rceil$.

Thanks to Euler's formula, we know that a planar graph of order n can have at most $3n - 6$ edges. And an outerplanar graph of order n has at most $2n - 3$ edges. Since outerplanar graphs are exactly the DD_1 graphs the following result generalizes this to DD_k graphs.

Theorem 16 *Let G be a planar graph having n vertices and e edges and satisfying $dd(G) = k$. Then $e \leq 2n + 3k - 6$*

Proof Let n_i be the number of vertices in C_i. Then $\sum_{i=1}^{k} n_i = n$. G is not maximal planar since more edges can be drawn in the interior of each disk d_i between vertices that lie on the boundary. For each d_i, the maximum number of such edges is $n_i - 3$. This gives a total of $n - 3k$ edges that can be added to G to get another planar graph G' on n vertices satisfying $G \subseteq G'$. It follows that the number of edges of G' is bounded above by $3n - 6$ and is equal to $e + n - 3k$.∎

Lemma 17 If $(A, B) = K_{2,3} \leq_t G$, and $i \geq dd(G)$, then the three vertices of B do not lie on the boundary of a single disk in any DD_i layout of G.

Proof Extending G to G^w allows us to get $K_{3,3} = (A \cup \{c_i\}, B) \leq_t G^w$. Which is impossible since G^w is planar.∎

Theorem 17 If $K_{2,3m+1} \leq_t G$, then $dd(G) > m$.

Proof Let $(A, B) = K_{2,3m+1} \leq_t G$. If $dd(G) \leq m$, then at least 3 vertices of B must lie on the boundary of some disk d_i. This contradicts the assertion of lemma 17.∎

Observation 7 Let $G = (V, E)$ be an element of DD_2. Then G has two vertices u and v such that $dd(G - uv) = 1$. To see this, note that removing two adjacent vertices that lie on the two disks of a DD_2 layout of G would lead to unifying the two disks. (It's like thickening edge (u, v) and opening a tunnel between the two disks.)

Lemma 18 Let G be a planar graph satisfying $dd(G) = k > m > 0$. Then G has at most $2m$ vertices $\{u_1, u_2, ...u_l\}, l \leq 2m$, such that $dd(G - \{u_1, u_2, ...u_l\}) \leq k - m$.

Proof Let G be given with an optimal (DD_k) layout. There are at least two disks, d_i and d_j in the layout that have an edge joining some u on d_i to a v on d_j. By Observation 7, removing u and v results in replacing d_i and d_j by one disk. This can be repeated m times or until we get a DD_{k-m} graph.∎

6.3 A Reduction Algorithm

We present an algorithm that, when given any planar graph G and a positive integer k, tries to remove at most $2k - 2$ vertices of G in order to obtain an outerplanar subgraph H of G. If the algorithm fails to produce any such H, then $dd(G) > k$. Hence, our algorithm proves the following theorem.

Theorem 18 *The family of graphs whose disk dimension is at most k is a sub-family of* $W_{2k-2}(outerplanar)$.

According to Lemma 18, given a DD_k graph G, there are pairs of adjacent vertices whose removal reduces the disk dimension of G. The question is, how do we find such pair when G is not given by a DD_k layout?

Theorem 19 Let G be a DD_k graph satisfying $K_4 \leq_t G$. Let u be any of the 4 corners of the K_4 model in G. Then the three neighbors of u in the model can't all belong to the boundary of the same disk as u.

Proof Note first that the four corners $\{u_i\}_{i=1}^4$ of a K_4-model cannot all lie on the boundary of the same disk. Otherwise G^w would contain a K_5 in the topological order. Let v_2, v_3, and v_4 be the neighbors of u_1 in the model. Assume also that, either $v_i = u_i$ or v_i is on the $u_1 - u_i$ path of the K_4 model. Let $\{d_i\}_{i=1}^k$ be a DD_k layout of G such that $u_1 \in C_1$. Assume $\{v_2, v_3, v_4\} \subset C_1$. If $u_2 \neq v_2$, then $(\{u_1, u_2\}, \{v_2, v_3, v_4\}) = K_{2,3} \leq_t G$. To see this, note that $u_2 - v_2, u_2 - u_3 - v_3$, and $u_2 - u_4 - v_4$ are vertex disjoint paths in the model of $K_4 \leq_t G$. Which is impossible by Lemma 17. ∎

Theorem 20 Let G be a DD_k graph satisfying $(A, B) = K_{2,3} \leq_t G$. Let u be any of the 2 corners corresponding to A in the $K_{2,3}$ model. Then the 3 neighbors of u in the model can't all lie on the same disk as u.

Proof Let v_1, v_2 and v_3 be the neighbors of u in the model. Then $(A, \{v_3, v_4, v_5\})$ is another $K_{2,3}$ model in G. Result follows by Lemma 17.∎

We are ready now to present our reduction algorithm. Procedure $REDUCE$, below, uses the following assumptions and notations: We use the expression 2-corner of a $K_{2,3}$-model to denote any of the two elements of A when $(A, B) =$

$K_{2,3} \leq_t G$. Function outerplanar is an outerplanarity test. If G is not outerplanar, and K_4 is a minor of G, a corner of the K_4-model is returned together with its three neighbors in the model. If K_4 is not a minor of G but $K_{2,3}$ is, a 2-corner is returned together with its three neighbors in the model.

Procedure REDUCE

Input: A planar graph G with n vertices and e edges, and an integer $k \geq 1$.

Output: A set S of "at most" $2k-2$ vertices of G such that $G-S$ is outerplanar. If no such set exists, return NULL.

Begin procedure
If($e > 2n - 3k + 6$)
    return NULL;
$S \leftarrow \phi$;
If ($k = 0$) return NULL;
If (outerplanar(G)) return S;
If ($K_4 \leq_t G$)
    $u \leftarrow$ corner of a K_4-model in G;
else
    $u \leftarrow$ 2-corner of a $K_{2,3}$-model in G;
$\{v_0, v_1, v_2\} \leftarrow$ neighborhood of u in the model;
for ($i = 0; i < 3; i++$)
    $S' = REDUCE(G - \{u, v_i\}, k - 1)$;
    if ($S' \neq NULL$)
        return $S \cup S'$
return NULL
End procedure

Note that corners of a K_4-model can be found by the linear-time algorithm described in [45]. Corners of a $K_{2,3}$-model can also be found in linear time [61].

Lemma 19 *Let (G, k) be the input to procedure $REDUCE$, where G is any planar graph of order n. The output of $REDUCE$ is either an outerplanar subgraph of order $n - 2k + 2$ or the fact that G is not in DD_k. Moreover, $REDUCE$ runs in $O(3^k n)$.*

Proof The preceding lemmas and discussion explain the output of $REDUCE$. As for its time complexity, note that each time vertex u is found (u is the corner of a K_4-model or a 2-corner of a $K_{2,3}$-model), we branch with three cases in the search tree. The height of the search tree is at most k. So it has at most 3^k nodes. All other statements in the code are done in linear time.∎

6.4 Tree Decompositions of DD_k Graphs

Theorems 17 and 6 imply the following:

Corollary 6 For fixed k, graphs that belong to DD_k have bounded treewidth.

We have already observed that, if G is connected, the diameter of G^w is bounded above by $3k-1$. It is shown in [23], that, for a planar graph G, $tw(G)$ is $O(D)$ where D is the diameter of G. This provides an alternate proof of corollary 6. By Lemma 18, given a DD_k graph G, there are at most $2k-2$ vertices whose removal produces an outerplanar graph. And outerplanar graphs have treewidth two or less. We therefore have an explicit upper bound on the treewidth of DD_k graphs:

Theorem 21 Let G be a DD_k graph. Then $tw(G) \leq 2k$.

The following procedure uses the same technique as in $REDUCE$ to remove at most $2k-2$ vertices of the graph. It outputs NULL only if $dd(G) > k$. There is no need, however, to go all the way until the resulting graph is outerplanar. It is well known that series-parallel graphs are "the" graphs that have treewidth at most two.

Procedure ddk_tw

Input: A planar graph G.

Output: A width $2k$ tree decomposition (T, X) of G

Begin procedure
If (series-parallel(G))
    $(T, X) \leftarrow sp\_tw(G)$;
    return (T, X);
If $(k = 1)$
    return NULL; // G can't be outerplanar and not series-parallel
$u \leftarrow$ corner of a K_4-model in G;
$\{v_0, v_1, v_2\} \leftarrow$ neighborhood of u in the model;
for $(i = 0; i < 3; i++)$
    $(T, X_1) \leftarrow ddk\_tw(G - \{u, v_i\}, k-1)$
    if(width(T) $< 2k-1$)
        $X \leftarrow \{X_i \cup \{u, v_i\} : X_i \in X_1\}$;
        return (T, X);
return NULL;
End procedure

ddk_tw runs in $O(3^k n)$ since it uses the same concept as $REDUCE$. It should be faster than $REDUCE$ in general because it doesn't check for the presence of

a $K_{2,3}$-model.

So far, we showed constructively that the treewidth of DD_k graphs is bounded above by $2k$. We can further show that it is in fact $O(\sqrt{k})$. To do this, we rely on the work of Alber et al. [1] on the Planar Dominating Set problem.

Theorem 22 Treewidth of DD_k graphs is $O(\sqrt{k})$.

Proof Define: $PDS_k = \{G : G \text{ is planar and has a dominating set of size} \leq k\}$, and $DD_k^w = \{G^w : G \in DD_k\}$. Clearly, $DD_k^w \subset PDS_k$. And, by the work of Alber et al., treewidth of PDS_k graphs is $O(\sqrt{k})$. Let G be a DD_k graph, then $G \subset G^w$ corresponding to a suitable DD_k layout.∎

Using the above theorem to obtain a tree decomposition for DD_k graphs would not be easy unless we have a (relatively) fast algorithm that solves the $DD(k)$ problem. However, the current best algorithm for $DD(k)$ is highly impractical.

6.5 Pathwidth Approximation

Optimal path decompositions of DD_k graphs can be obtained (theoretically) in polynomial time. This is due to [8], which asserts that optimal path decompositions can be found in polynomial time for graphs of bounded treewidth. The algorithm suggested is not practical, however, because it operates on sets of size $O(n^{11})$ in its first step.

Motivated by fast approximation algorithms for the pathwidth of DD_1 graphs, we show how to get similar algorithms for DD_k graphs. In fact, we rely on Lemma 18 (by using procedure $REDUCE$) to show how to obtain a linear time algorithm whose performance ratio is 3 on general DD_k graphs and 2 on some biconnected ones.

Take, for example, a DD_2 graph G. Delete a (suitable) pair $\{u, v\}$ of vertices. The resulting graph, H, is outerplanar. Using the work of [34], we can find a path decomposition (P, X) that is not more than $3pw(H)+2$ in $O(n)$ time. (The bound stated in [34] is only O($nlogn$), because the algorithm described there relies on obtaining optimal path decompositions of trees. It has more recently been shown that such decompositions can be computed in linear time [60]. Thus the algorithm of [34] runs in linear time as well.) Adding the two vertices to every element of X gives a path decomposition of G of width $\leq 3pw(H) + 4 \leq 3pw(G) + 4$ because $H \subseteq G$. If H were biconnected, we could use the algorithm of [7] to obtain a path decomposition of width $\leq 2pw(H) + 1$.

Procedure *ddk_pw*, below, assumes that a path decomposition of a given outerplanar graph, H, can be obtained by function *outpl_pw* in linear time. The width of the path decomposition returned by *outpl_pw* is not more than $3pw(H) + 2$. Thus, given graph G as input, *ddk_pw* returns a path decomposition of width not exceeding $3pw(G) + 2k$. It outputs NULL only if $dd(G) > k$.

Procedure ddk_pw

Input: A planar graph G.

Output: A path decomposition (P, X) of G

Begin procedure

If (outerplanar(G))

    $(P, X) \leftarrow outpl\_pw(G)$;

    return (P, X);

$S \leftarrow REDUCE(G, k)$

if($|S| \leq 2k - 2$)

    $(P, X_1) \leftarrow outpl\_pw(G \backslash S)$;

    $X \leftarrow \{X_i \cup S : X_i \in X_1\}$;

return (P, X);

return NULL;

End procedure

Theorem 23 *If G is a DD_k graph of order n, a path decomposition of G with width at most $3pw(G) + 2k$ can be constructed in $O(3^k n)$ time.*

Proof Obtaining the path decomposition of width at most $3pw(G)+2k$ has been described in details in the preceding paragraphs. The claimed time complexity follows from lemma 19 since the time consuming part of function ddk_pw is the call to $REDUCE$. Other parts of the code run (obviously) in linear time.■

6.6 Remarks

We showed that graphs of disk dimension k or less are within $2k-2$ vertices of outerplanar graphs. The containment of DD_k in $W_{2k-2}(outerplanar)$ is proper. In fact, for arbitrary $r > 0$, a graph, G_r, that has disk dimension r and is within one vertex of outerplanar can be constructed from K_1 and $(r-1)$ copies of K_3 by connecting a vertex u representing K_1 to all vertices of $(r-1)K_3$. The resulting graph is $G_r = K_1 + (r-1)K_3$ is shown in figure 6.3. G_r is (obviously) planar and has disk dimension r. Procedure $REDUCE$ could still do a good job on this graph since, unless the edges connecting u to other vertices are subdivided, it would only take out two vertices of which one is u.

Our constructive proof, manifested in procedure $REDUCE$, allows for a family of approximation algorithms for DD_k graphs. To illustrate, if problem π is easy (or, as in our case, has a fast approximation) on outerplanar graphs then, for some fixed k, it has an absolute approximation algorithm for DD_k graphs since the quality of solution will only be affected by an additive constant.

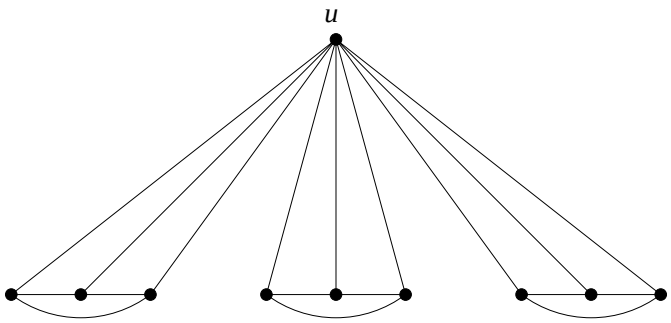


Figure 6.3: A "within one vertex from outerplanar" graph whose disk dimension is four.

Chapter 7

Summary and Directions for Future Research

We have explored the relationship between graph coloring and the immersion order, and have conjectured that if $\chi(G) \geq t$, then G contains an immersed K_t. We proved that, for each fixed value of t, there can be only a finite number of t-immersion-critical graphs. Our most important results are that t-immersion-critical graphs other than K_t must, if any exist, be both 4-vertex-connected and t-edge-connected. Since the immersion containment generalizes topological containment, the absence of an immersed K_t in a graph implies that it does not have a topological K_t. So, for $t \in \{5, 6\}$, the existence of any counterexample to our conjecture would prove that Hajós conjecture is also false for (any of) these cases, a problem that has been open for about 60 years.

We considered the different techniques used in exact algorithms for problems that are $\mathcal{NP}$-complete but fixed parameter tractable. Aiming at showing that such exact algorithms should not be dismissed, we implemented a parameterized Vertex Cover algorithm. Based on experimental results, we judge that recent

developments of fixed-parameter algorithms did really make a breakthrough in bridging the gap between notoriously hard problems and practicality. The role played by preprocessing techniques is essential and, based on our experience, we find that any preprocessing idea counts. We continue, therefore, to explore and add new preprocessing algorithms to our dynamically improving code. Our linear programming method for kernelization of $VC(k)$ already obtains a kernel of size $\leq 2k$. We suspect, however, that the crown reduction method (which obtains a kernel of size $\leq 3k$) and the linear programming method are orthogonal and, together, could lead to further reduction.

Our implementation of Vertex Cover used the search tree technique, called branching. We implemented a parallel version of branching, based on a decomposition of the search tree. This method (which we call the "search tree decomposition" technique) is very promising since it could lead to super-linear speedups on some input instances. The (sequential and parallel) branching phase of our algorithm includes interleaving of some preprocessing rules. With all the new techniques and technologies used and "to be used" in this project, it is hoped that the code will keep scaling up to accommodate larger input instances and cope with its rich areas of applications.

We considered the $\mathcal{FPT}$ Face Cover problem and presented a novel direct algorithm that runs in $O(5^k + n^2)$ time. We think that indirect approaches that use dynamic programming on tree decompositions are still far from practical. An open problem is whether a small tree decomposition could be obtained for the input plane graph by exploiting (only) the fact that a face cover of size k is present.

We considered the pathwidth metric. Because exact fixed-parameter algorithms for this problem are still far from practical at this time, and because approximation algorithms appeared for the pathwidth of several subfamilies of

planar graphs, we extended previous work by presenting an approximation algorithm for the pathwidth of planar graphs with fixed disk dimension. Our algorithm is highly practical and could take as input any planar graph. It could fail only when the input is not of the given fixed disk dimension.

To get the pathwidth approximation algorithm, we showed that the family, DD_k, of planar graphs of disk dimension k or less is properly contained in the family, $W_{2k-2}(outerplanar)$, of graphs that are within $2k-2$ vertices of outerplanar. Our constructive proof allows also for a family of approximation algorithms for DD_k graphs. In fact, since most hard problems have fast solutions on outerplanar graphs, a general approximation strategy for a problem on DD_k would be to use our reduction algorithm and obtain an outerplanar subgraph by removing no more than $2k-2$ vertices, then solve the problem and append the $2k-2$ vertices to the solution.

We provided a tree decomposition algorithm that guarantees a width that is not larger than twice the disk dimension of the input planar graph. This algorithm should be useful as a heuristic for constructing tree decompositions of planar graphs in general. It might be possible to use tree decompositions and dynamic programming to obtain a fixed-parameter disk dimension algorithm. It would be very challenging, however, to obtain a practical algorithm for this problem.

Bibliography

Bibliography

[1] J. Alber, H. L. Bodlaender, H. Fernau, T.Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.

[2] Jochen Alber, Hongbing Fan, Michael R. Fellows, Henning Fernau, Rolf Niedermeier, Fran Rosamond, and Ulrike Stege. Refined search tree technique for DOMINATING SET on planar graphs. *Lecture Notes in Computer Science*, 2136:111–122, 2001.

[3] S. Arnborg and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[4] D. Bienstock and M. Langston. Algorithmic implications of the graph minor theorem. In *Handbook of Operations Research and Management Science*, volume 7, pages 481–502. Elsevier Science, 1995.

[5] D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17:53–76, 1988.

[6] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small width. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[7] H. L. Bodlaender and F. V. Fomin. Approximation of pathwidth of outerplanar graphs. *Journal of Algorithms*, 43(2):190–200, 2002.

[8] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21:358–402, 1996.

[9] Béla Bollobás. *Extremal Graph Theory*. Academic Press, New York, 1978.

[10] H. D. Booth, R. Govindan, M. A. Langston, and S. Ramachandramurthi. Sequential and parallel algorithms for K_4 immersion testing. *Journal of Algorithms*, 30:344–378, 1999.

[11] J.F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22:560–572, 1993.

[12] Catlin. Hajös graph-coloring conjecture: variation and counterexamples. *JCTB*, 26:268–274, 1979.

[13] Kevin Cattell and Michael J. Dinneen. A characterization of graphs with vertex cover up to five. In *ORDAL*, pages 86–99, 1994.

[14] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse grained parallel machines. Technical report, Department of Computer Science, Carleton University, Ottawa, Canada, 2002.

[15] J. Chen, I. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.

[16] Vasek Chv´tal. *Linear Programming*. W.H.Freeman, New York, 1983.

[17] ClustalW. See `http://helix.nih.gov/apps/bioinfo/clustalw.html`.

[18] W. Cook. Private communication, 2003.

[19] G. A. Dirac. A property of 4-chromatic graphs and some remarks on critical graphs. *J. London Math. Soc.*, 27:85–92, 1952.

[20] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992.

[21] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[22] P. Duchet and H. Meyniel. On hadwiger's number and the stability number. *Annals of Discrete Math.*, 13:71–74, 1982.

[23] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 1999.

[24] David Eppstein. Small maximal independent sets and faster exact graph coloring. In Frank Dehne, Jörg-Rudiger Sack, and Roberto Tamassia, editors, *Proc. 7th Worksh. Algorithms and Data Structures*, number 2125 in Lecture Notes in Computer Science, pages 462–470. Springer-Verlag, August 2001.

[25] P. Erdös and S. Fajtlowicz. On the conjecture of hajös. *Combinatorica*, 1:141–143, 1981.

[26] M. Fellows. Private communication, 2003.

[27] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Information Processing Letters*, 26:157–162, 1987.

[28] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM*, 35:727–739, 1988.

[29] M. R. Fellows and M. A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM Journal on Discrete Mathematics*, 5:117–126, 1992.

[30] M. R. Fellows and M. A. Langston. On search, decision and the efficiency of polynomial-time algorithms. *Journal of Computer and Systems Sciences*, 49:769–779, 1994.

[31] M. R. Fellows, Catherine McCartin, Frances A. Rosamond, and Ulrike Stege. Coordinatized kernels and catalytic reductions: An improved fpt algorithm for max leaf spanning tree and other problems. *FSTTCS*, pages 240–251, 2000.

[32] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman, New York, 1979.

[33] R. Govindan, M. A. Langston, and S. Ramachandramurthi. A practical approach to layout optimization. In *Sixth International Conference on VLSI Design*, pages 222–225, 1993.

[34] R. Govindan, M. A. Langston, and X. Yan. Approximating the pathwidth of outerplanar graphs. *Information Processing Letters*, 68:17–23, 1998.

[35] H. Hadwiger. Über eine klassifikation der streckenkomplexe. *Vierteljahrsschr. Naturforsch. Ges. Zürich*, 88:133–142, 1943.

[36] G. Hajös. Über eine konstruktion nicht n-farbbarer graphen. *Wiss. Z. Martin Luther Univ. Halle-Wittenberg Math. Naturwiss. Reihe,*, 10:116–117, 1961.

[37] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS, 1997.

[38] K. Kawarabayashi. On the connectivity of minimal-counterexamples to Hadwiger's conjecture, to appear.

[39] K. Kawarabayashi and B. Toft. Any 7-chromatic graph has K_7 or $K_{4,4}$ as a minor. *Combinatorica*, to appear.

[40] A. B. Kempe. How to color a map with four colours without coloring adjacent districts the same color. *Nature*, 20:275, 1879.

[41] S. Khuller. The vertex cover problem. *ACM SIGACT News*, 33:31–33, June 2002.

[42] K. Kuratowski. Sur le problème des courbes gaushes en topologie. *Fund. Math.*, 15:271–283, 1930.

[43] M. A. Langston and B. C. Plaut. Algorithmic applications of the immersion order. *Discrete Mathematics*, 182:191–196, 1998.

[44] E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.

[45] P. C. Lui and R. C. Geldmacher. An O(max(m, n)) algorithm for finding a subgraph homeomorphic to K_4. *Congressius Numeratium*, 29:597–609, 1980.

[46] W. Mader. Homomorphiesätze für graphen. *Math. Ann.*, 178:154–168, 1968.

[47] W. Mader. 3n-5 edges do force a subdivision of K_5. *Combinatorica*, 18:569–595, 1998.

[48] P. J. McGuinness and A. E. Kezdy. An algorithm to find a K_5 minor. In *Proceedings, Third ACM-SIAM Symposium on Discrete Algorithms*, pages 345–356, 1992.

[49] B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, 58:209–229, 1988.

[50] NCBI. See `http://www.ncbi.nlm.nih.gov/`.

[51] G.L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[52] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter tractacle algorithms. *Information Processing Letters*, 73:125–129, 2000.

[53] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 2001.

[54] Chiba Nishizeki. Planar graphs: Theory and algorithms. *Annals of Discrete Mathematics*, 32, 1988.

[55] N. Robertson and P. D. Seymour. Graph minors XIII. the disjoint paths problem. *J. Combin. Theory Ser. B*, 63:65–110, 1995.

[56] N. Robertson and P.D. Seymour. Graph minors IV. tree-width and well-quasi-ordering. *Journal of Combinatorial Theory, Series B*, 48:227–254, 1990.

[57] N. Robertson and P.D. Seymour. Graph minors XVI: Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, to appear.

[58] N. Robertson, P.D. Seymour, and R. Thomas. Hadwiger's conjecture for K_6-free graphs. *Combinatorica*, 13:279–361, 1993.

[59] J. M. Robson. Finding a maximum independent set in time $o(2^{n/4})$. Technical report, Université Bordeaux, January 2001.

[60] K. Skodinis. Computing optimal linear layouts of trees in linear time. Technical Report MIP-9904, Department of Computer Science, University of Passau, Passau, Germany, 1999.

[61] R. Thomas. Private communication, 2001.

[62] B. Toft. On separating sets of edges in contraction-critical graphs. *Math. Ann.*, 196:129–147, 1972.

[63] B. Toft. On critical subgraphs of colour critical graphs. *Discrete Math.*, 7:377–392, 1974.

[64] B. Toft. Colouring, stable sets and perfect graphs. In *Handbook of Combinatorics*, volume 2, pages 233–288. Elsevier Science B.V., 1995.

[65] K. Wagner. Über eine eigenschaft der ebenen komplexe. *Math. Ann.*, 114:570–590, 1937.

[66] K. Wagner. Beweis einer abschwächung der hadwiger-vermutung. *Math. Ann.*, 153:139–141, 1964.

[67] K. Weihe. Covering trains by stations or the power of data reduction. In *Proceedings, International Conference on Algorithms and Experiments*, pages 1–8, 1998.

[68] D. B. West. *Introduction to Graph Theory.* Prentice Hall, 1996.

[69] H. Whitney and W.T. Tutte. Kempe chains and the four colour problem. *Utilitas Math.*, 2:241–281, 1972.

Vita

Faisal N. Abu Khzam finished his BS in computer science in 1990 and his MS in mathematics in 1997 from the American University of Beirut. In the fall of 1998, Faisal joined the Computer Science department at the University of Tennessee, to pursue a doctoral degree.