To the Graduate Council:

I am submitting herewith a dissertation written by Siddharthan Ramachandramurthi entitled "Algorithms for VLSI Layout Based on Graph Width Metrics". I have examined the final 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 Allen Langston, Major Professor

We have read this dissertation
and recommend its acceptance:

Jean R. S. Blair

Heather Booth

Donald W. Bouldin

Accepted for the Council:

Associate Vice Chancellor
and Dean of the Graduate School

Algorithms for VLSI Layout Based on Graph Width Metrics

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Siddharthan Ramachandramurthi

August 1994

Acknowledgements

I thank my advisor Dr. Mike Langston for making this dissertation possible. He has been a constant source of help and encouragement, and I have learnt a lot from him. I am also grateful to Dr. Jean Blair, Dr. Heather Booth, and Dr. Don Bouldin for serving on my dissertation committee. It has been a pleasure to work with them.

I thank my family and all my friends for their support and for making the last six years memorable. Especially, I thank Rajeev Govindan who has been an invaluable companion both in research and in tennis.

I am thankful to the Department of Computer Science and the Joint Institute for Computational Science at the University of Tennessee for providing me with a pleasant work environment. I also acknowledge the National Science Foundation and the Office of Naval Research for providing partial funding for this research.

Abstract

Layout permutation is a frequently used process in VLSI design whereby some of the circuit elements are arranged in order to obtain an optimum layout. Our aim is to develop practical algorithms for well-known layout permutation problems such as Gate Matrix Layout and Minimum Cut Linear Arrangement. Because these optimization problems are $\mathcal{NP}$-Hard, and also because in practice the problems tend to be of bounded size, we address their fixed-parameter variants.

Although recent advances in the field of algorithms have yielded powerful nonconstructive tools to tackle these problems, such algorithms have remained impractical. The main feature of these algorithms is their reliance on a finite set of forbidden structures known as "obstructions" to determine whether a layout is feasible. We study the applicability of these novel techniques to problems in VLSI design and present a practical approach for finding optimal solutions for fixed-parameter versions of problems such as Gate Matrix Layout and Minimum Cut Linear Arrangement.

We model circuits by graphs and transform layout optimization problems to graph width minimization problems. This transformation extends the applicability of our algorithms to a wide variety of problems in VLSI design and other domains.

To demonstrate our approach, we design a linear-time decision algorithm for 3-track Gate Matrix Layout using only a few obstructions and devise fast self-reductions to find a layout if any exist. Ours is the first practical self-reduction algorithm based on obstruction sets. These algorithms have been implemented and experimental results show that they yield correct results "almost always."

In an effort to extend this method for gate matrix layouts with more than 3 tracks, we present several lower and upper bounds for the treewidth and pathwidth metrics of a graph. We also present fast algorithms to compute some of the bounds in practice. These bounds give us a better understanding of the structure and number of obstructions and we develop a general method to construct some of the dense obstructions for

gate matrix layout. We find that there are numerous dense obstructions and explore ways to devise tests for them.

Based on the cutwidth metric of graphs, we also develop algorithms for the Minimum Cut Linear Arrangement problem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In VLSI circuit design, the input consists of a description of a circuit in terms of its components and their interconnections, and the goal is to obtain a layout of the circuit within the limits of a given fabrication technology. There are various criteria associated with a layout that affect the cost and performance of a circuit, such as the total area and the length of the longest interconnection. Naturally, we would like to minimize the cost and maximize the performance of the circuit, and this is known as layout optimization. Layout permutation problems are those in which we potentially have to consider all possible permutations of certain components of the circuit in order to obtain an optimum layout.

With the advent of computer aided design (CAD), automatic design tools have grown in popularity and are now the mainstay of most circuit designers. Therefore, there is a great emphasis on developing better algorithms for circuit layout. Unfortunately, nearly all versions of the VLSI layout optimization problem are $\mathcal{NP}$-Hard [Le]. In particular, all the layout permutation problems that we consider in this dissertation are $\mathcal{NP}$-Hard.

1.1 Fixed-Parameter Problems

Although there is a real need for practical algorithms, the intractability of the optimization problem poses a formidable challenge to those who develop CAD tools. However, in practice, we have to work with a limited amount of resources, which in VLSI terms could mean a chip of fixed area or a bus of constant width. Therefore, it is reasonable to address only the "fixed-parameter" version of the layout problem. In this version, instead of trying to find a layout with the least cost, the goal is to obtain a layout whose cost is less than a certain fixed amount. This is also expedient because unlike optimization, at least in principle, the fixed-parameter problem is often solvable in polynomial time.

In this thesis, we develop a new approach to developing practical algorithms for fixed-parameter versions of problems in VLSI design.

1.2 Our Approach

Our work incorporates novel techniques based on powerful non-constructive tools with a view to developing a new approach to the design of practical algorithms. We make the distinction between *decision* algorithms, which only determine whether or not any layouts exist, and *search* algorithms, which also find a layout whenever possible. Just as one might expect, decision algorithms are almost always faster than search algorithms. It is often possible to develop a search algorithm by employing a decision algorithm. Such a technique is known as *self-reduction.*

An important feature of non-constructive tools is that they can prove the existence of a polynomial-time decision algorithm without actually presenting or even showing how to find such an algorithm. For this they rely on the fact that, in order to have a layout whose cost does not exceed a predetermined amount, the circuit must exclude a finite number of forbidden structures called *obstructions.*

By examining the number and nature of the obstructions, we are able to invent practical algorithms for both decision and search. We demonstrate our approach on some typical layout permutation problems. Experimental results of the performance of our algorithms are also presented as evidence of their potential for practical utility.

Throughout this dissertation, we represent circuits by graphs in such a way that a layout optimization problem on circuits is transformed into a problem of minimizing a width metric of graphs. Such a transformation serves two purposes. First, it allows us to take advantage of the non-constructive techniques mentioned above. Next, since these graph width metrics can be used to model problems from a wide variety of applications, it expands the scope of our algorithms.

1.3 Sample Problems

We demonstrate our general method for two well-known layout permutation problems encountered in VLSI design. First we focus on the Gate Matrix Layout problem, where the goal is to find a layout with minimum area, in gate matrix style, for a given circuit. We transform the Gate Matrix Layout problem to the pathwidth problem on graphs. The underlying combinatorial problem is encountered in numerous other VLSI problems, including multiple folding of PLAs and Weinberger arrays [FL1]. It even arises in other diverse areas, such as natural language processing [KoT].

In the Minimum Cut Linear Arrangement problem, the goal is to layout the components of a circuit on a horizontal line such that the maximum number of wires cut by a vertical line between adjacent components is minimized. We transform the Minimum Cut Linear Arrangement problem to the cutwidth problem on graphs. This problem arises in several different contexts, such as one dimensional linear arrays [FHKY], datapath design [CNSD], backplane design [GCT], and minimizing network congestion [Lan].

1.4 An Overview

The rest of this dissertation is organized as follows. In Chapter 2, we introduce a general approach to the design of practical algorithms for VLSI circuit layout permutation problems. We demonstrate this method in Chapter 3, where we develop algorithms for the Gate Matrix Layout problem in VLSI design with the number of tracks fixed at 3. As a means of developing practical algorithms for gate matrix layout with more tracks, we develop several lower and upper bounds for the treewidth and pathwidth metrics of a graph in Chapter 4. These bounds are of independent interest as well. The structure and the number of obstructions for the fixed-parameter versions pathwidth and treewidth are studied in Chapter 5. Besides developing new tools to construct some of the dense obstructions, we also present a lower bound on the size of the obstruction set. In Chapter 6, we use the cutwidth metric of a graph to apply our method to linear layout problems like Minimum Cut Linear Arrangement in VLSI design. Conclusions and directions for future research are presented in Chapter 7.

Chapter 2

An Obstruction-Based Approach to Algorithm Design

In this chapter, we describe a new approach to designing practical algorithms for layout permutation problems in VLSI design. This approach was inspired by the recent availability of powerful mathematical tools to model and analyze the structure of layout problems. These tools are based on the concept of well-quasi-ordering of graphs. In particular, the work of Robertson and Seymour [RS1–RS5] on graph minors is fundamental to our research.

2.1 Notation

All graphs we use are finite and undirected. Unless stated otherwise, loops and multiple edges are not allowed. Let $G = (V, E)$ denote a graph with n vertices and e edges. Then n is the *order* of G and e is the *size* of G. Note that if G is a connected graph, then $e \geq n - 1$, i.e. $n = O(e)$.

If v_i is a vertex in G, then $N(v_i) = \{v_j : j \neq i, (v_i, v_j) \in E\}$ is the set of all neighbors of v_i, and the degree of v_i is $\delta(v_i) = |N(v_i)|$. The minimum degree of G is $\delta(G) = \min_{1 \leq i \leq n} \delta(v_i)$. When there is no ambiguity, we write N_i for $N(v_i)$, δ_i for $\delta(v_i)$, and δ for $\delta(G)$. Sometimes, for clarity, we write $\delta(v_i; G)$ and $N(v_i; G)$ for the degree and neighborhood of v_i in G.

The vertex-connectivity (henceforth simply connectivity) of G is denoted by $\kappa(G)$. If H is a subgraph of G, we write $H \subseteq G$. If H is a proper subgraph of G, we write $H \subset G$. $G \cong H$ means that G is isomorphic to H.

A complete graph is one in which there exists an edge between each pair of distinct vertices. K_n denotes the complete graph of order n. If G is a graph of order n, then the *complement* of G, written $\bar{G}$, is $K_n - E(G)$.

2.2 Graph Minors and Well-Quasi-Ordering

A *quasi-order* is a reflexive and transitive binary relation. Let Q be a set quasi-ordered by a binary relation $\leq$. The quasi-order $\leq$ is *well-founded* if there is no infinite descending chain $q_0 > q_1 > q_2 > \ldots$ of elements of Q. A set $A \subseteq Q$ is an *antichain* if $a \not\leq b$ and $b \not\leq a$ for every pair of distinct elements a and b of A.

Definition [Mi] A set Q is *well-quasi-ordered* by a binary relation $\leq$ if and only if Q is well-founded and has no infinite antichain.

In our research, we shall be concerned with sets of finite graphs. The work of Robertson and Seymour on graph minors is the theoretical basis for our algorithms. First we define some binary relations on graphs G and H.

Definition H is *topologically* contained in G, denoted $H \leq_t G$, if and only if a graph isomorphic to H can be obtained from a subgraph of G by removing subdivisions. If H is topologically contained in G but H is not isomorphic to G, then we write $H <_t G$.

An edge (v_i, v_j) of a graph is contracted by identifying its two endpoints. The resulting vertex has neighborhood $N(v_i) \cup N(v_j) - \{v_i, v_j\}$.

Definition H is a *minor* of G, denoted $H \leq_m G$, if and only if a graph isomorphic to H can be obtained from a subgraph of G by contracting edges. If H is a minor of

G and not isomorphic to G, then we write $H <_m G$.

A pair of adjacent edges (v_i, v_j) and (v_j, v_k) of a graph G is lifted by removing the two edges from G and introducing a new edge (v_i, v_k).

Definition H is *immersed* in G, written $H \leq_i G$, if and only if a graph isomorphic to H can be obtained from G by taking subgraphs and *lifting* pairs of adjacent edges. If $H \leq_i G$ and $H \ncong G$, then we write $H <_i G$.

Definition A family $\mathcal{F}$ of graphs is *closed* in the minor (immersion) order if, for every G in $\mathcal{F}$, if H is a minor of (immersed in) G then H is also in $\mathcal{F}$.

Definition A graph G is a *minimal* element of set Q in the minor (immersion) order if and only if for every $H <_m G$ ($H <_i G$), H is not in Q.

Definition If $\mathcal{F}$ is a minor (immersion) closed family, then the *obstruction set* for $\mathcal{F}$, written $obs(\mathcal{F})$, is the set of all graphs in the complement of $\mathcal{F}$ that are minimal in the minor (immersion) order.

Therefore, if $\mathcal{F}$ is a minor (immersion) closed family, then $G \in \mathcal{F}$ if and only if $H \not\leq_m G$ ($H \not\leq_i G$) for every $H \in obs(\mathcal{F})$.

It is easy to see that $\leq_m$ and $\leq_i$ are quasi-orders. The following theorem, proven by Robertson and Seymour, shows that graphs are well-quasi-ordered by $\leq_m$ and $\leq_i$.

Theorem 2.2.1 [RS5] If $\mathcal{F}$ is a minor (immersion) closed family of finite graphs, then $obs(\mathcal{F})$ is a finite set.

The following theorem is what makes well-quasi-orders interesting from an algorithmic perspective.

Theorem 2.2.2 [RS4, FL5] For every fixed graph H, there exists polynomial-time algorithm that when given an input graph G, decides whether H is a minor of (immersed in) G.

These two theorems together imply a polynomial-time algorithm to decide membership in any minor or immersion closed family. The algorithm would proceed as follows. Given a graph G whose membership in $\mathcal{F}$ is to be decided, we check to see if G contains an obstruction to $\mathcal{F}$. If G does not contain any of the obstructions, then it belongs to $\mathcal{F}$. Otherwise it does not belong to $\mathcal{F}$. Since there are only a finite number of obstructions, and there exists a polynomial-time algorithm to test for the containment of each, membership in $\mathcal{F}$ can be decided in polynomial time as well. Note how this algorithm is based on prior knowledge of $obs(\mathcal{F})$.

Next, we shall see how these theorems can be applied in practice.

2.3 Non-Constructive Techniques

Even though Theorems 2.2.1 and 2.2.2 are very powerful, they cannot be used directly by the algorithm designer. The reasons for this are twofold. First, Theorem 2.2.1 is non-constructive in the following sense: it proves that the number of obstructions is finite without actually showing what these obstructions are, or even how they can be found. It is known that any proof of Theorem 2.2.1 must be inherently non-constructive [FRS] and there can be no general scheme to compute the obstructions to a given family [FL6]. Second, although the general algorithm for minor testing proposed by Theorem 2.2.2 has time complexity $O(n^3)$ where $n = |V(G)|$, it involves enormous constants of proportionality and is highly impractical (see [FL2]). If the family $\mathcal{F}$ excludes even one planar graph, then the graphs in $\mathcal{F}$ have bounded treewidth [RS3] as a result of which the minor containment test can be done in $O(n)$ time [Bod1]. This algorithm also has extremely large constants. The immersion testing algorithm has time complexity $O(n^{h+3})$ where $h = |V(H)|$ and is also impractical.

Clearly, these general schemes are not practical. Putting them to good use is a formidable challenge. This does not deter us, because the obstructions to the specific problems of interest to us may still be computable. Fellows and Langston [FL3] have

proposed a way to compute the obstructions for many different problems. This issue has been addressed by others as well (see [LA] for example).

The unifying theme of our work is our motive to explore new avenues for developing practical algorithms for layout permutation problems in VLSI design by tapping into the ideas inherent in theorems 2.2.1 and 2.2.2, and adapting them for realistic applications.

2.4 Our General Approach

Suppose we have a layout permutation problem Π, whose objective is to obtain a layout of a given circuit $\mathcal{C}$ with small cost. Assuming that the problem is intractable in general, we turn to the fixed-parameter version Π_k. Thus we would like to know whether $\mathcal{C}$ has a layout with cost no more than some fixed constant k. If $\mathcal{C}$ has such a layout, then we say that $\mathcal{C} \in \Pi_k$. We impose four requirements.

Our first requirement is one of problem transformation. We need to be able to find a function g that maps the circuit $\mathcal{C}$ to a graph G, a function $f : N \rightarrow N$, and a graph metric $\mathcal{M}$ such that if $\mathcal{M}(G) = \mathcal{M}(g(\mathcal{C})) \leq f(k)$, then $\mathcal{C}$ has a layout with cost at most k. Since our aim is to obtain fast algorithms, the functions g and f should be quickly computable. We cannot expect $\mathcal{M}(G)$ to be significantly easier to solve than the original layout permutation problem itself. Therefore, this first step can be viewed as a pre-processing step where we transform the input to a convenient graph form. The choice of the function g is dictated by our next requirement.

The second requirement is that the family of graphs $g(\Pi_k)$, representing all circuits $\mathcal{C} \in \Pi_k$, be closed either in the minor or the immersion order. This property is fundamental to our approach. For any practical problem, it is usually easy to determine whether the corresponding family of graphs is closed. Once we have closure, we proceed to the next step.

The third requirement is knowledge of the set of obstructions for the closed fam-

ily. Theorem 2.2.1 only guarantees that any minor (immersion) closed family has a finite number of obstructions. The proof itself does not tell us how to find these obstructions.

In our approach we work with only a small subset of the obstruction set. There are several reasons for this. For many problems encountered in practice, it is the case that only some of the obstructions are relevant. Also, certain obstructions tend to occur more frequently than others. Often, even when all the obstructions are known, fast containment tests may not be available for all of them. Thus, for a combination of theoretical and practical reasons, we might consider only a small subset of the obstruction set. Naturally, such an algorithm cannot be expected to give the correct answer always. Of course, it is difficult to identify the important obstructions. We are mainly guided by the structural properties of the problem at hand and the availability of fast containment tests.

The fourth requirement is fast algorithms to test whether the input graph contains an obstruction. For any known obstruction, Theorem 2.2.2 ensures the existence of a polynomial-time test. Unfortunately, such an algorithm would be highly impractical due to the enormous constants of proportionality. Therefore, we need truly practical algorithms to test for the obstructions.

Given all four ingredients, we can tackle problem Π_k. We shall illustrate our approach in the next chapter.

Chapter 3

Gate Matrix Layout and Pathwidth

We use the Gate Matrix Layout problem (GML) as a prototypical example to illustrate our general approach to developing practical algorithms. The combinatorial problem underlying GML is also encountered in several other VLSI design optimization problems. Examples include multiple folding of PLAs and Weinberger arrays. This was first reported in [FL5]. [Mõ] surveys the relationship between these problems in more detail. Therefore, we address the GML problem in its full generality and demonstrate how it can be tackled using our method. For this we use a graph metric known as pathwidth. Later we discuss some of the issues specific to gate matrix layouts.

3.1 Background

We start with a statement of the problem and a survey of its history.

3.1.1 Problem Statement

The gate matrix layout style was introduced by Lopez and Law [LL] in 1980 as an elegant way to layout MOS circuits. A generalization of the Weinberger array layout style, gate matrix has been found to be particularly good for static CMOS technology.

Regularity of structure and high density of devices are the salient features of this style, which is well suited for implementing complex gates, functional cells, and random logic. A gate matrix consists of intersecting rows and columns to provide transistor placement and interconnections. The columns are realized in polysilicon and serve both as transistor gates and as interconnections. The rows are implemented with metal and diffusion. The diffusion layer forms a transistor at each intersection with a polysilicon gate (column) and the metal lines are used for interconnections. Metal and diffusion can also be used to interconnect different rows. A second metal layer is available solely for the power and ground connections. Each row in the layout is called a *track*, and consists of one or more nets of the circuit.

An instance of the Gate Matrix Layout problem (GML) consists of a set of *nets* and their respective connections to a set of *gates*. A gate matrix layout consists of any assignment of tracks to the nets subject to the restriction that no two nets incident on a common gate can share the same track. The goal is to minimize the number of tracks utilized thereby minimizing the area required for a layout.

Figure 3.1 on page 13 shows a CMOS circuit and two different layouts in gate matrix style for the circuit. The circuit consists of nets N_1, N_2, and N_3, and gates A, B, C, D, E, F, and Z. Net N_1 is connected to gates A, B, and F; net N_2 is connected to gates C, D, and E; net N_3 connects gates C, F, and Z.

Each net consists of one or more transistors connected together either in series or parallel using metal and diffusion. Since each net is typically connected to only a small subset of the gates, assigning each net to a distinct track would be wasteful of chip area. If several nets shared the same track, that would help reduce the total area of the layout. Observe that the number of nets that can share a track depends on the linear ordering of the gates and the assignment of the nets to tracks. Therefore, by permuting the gates we may be able to obtain further savings in the number of tracks. For example, in Figure 3.1, the gate sequence $A - B - C - D - E - F - Z$

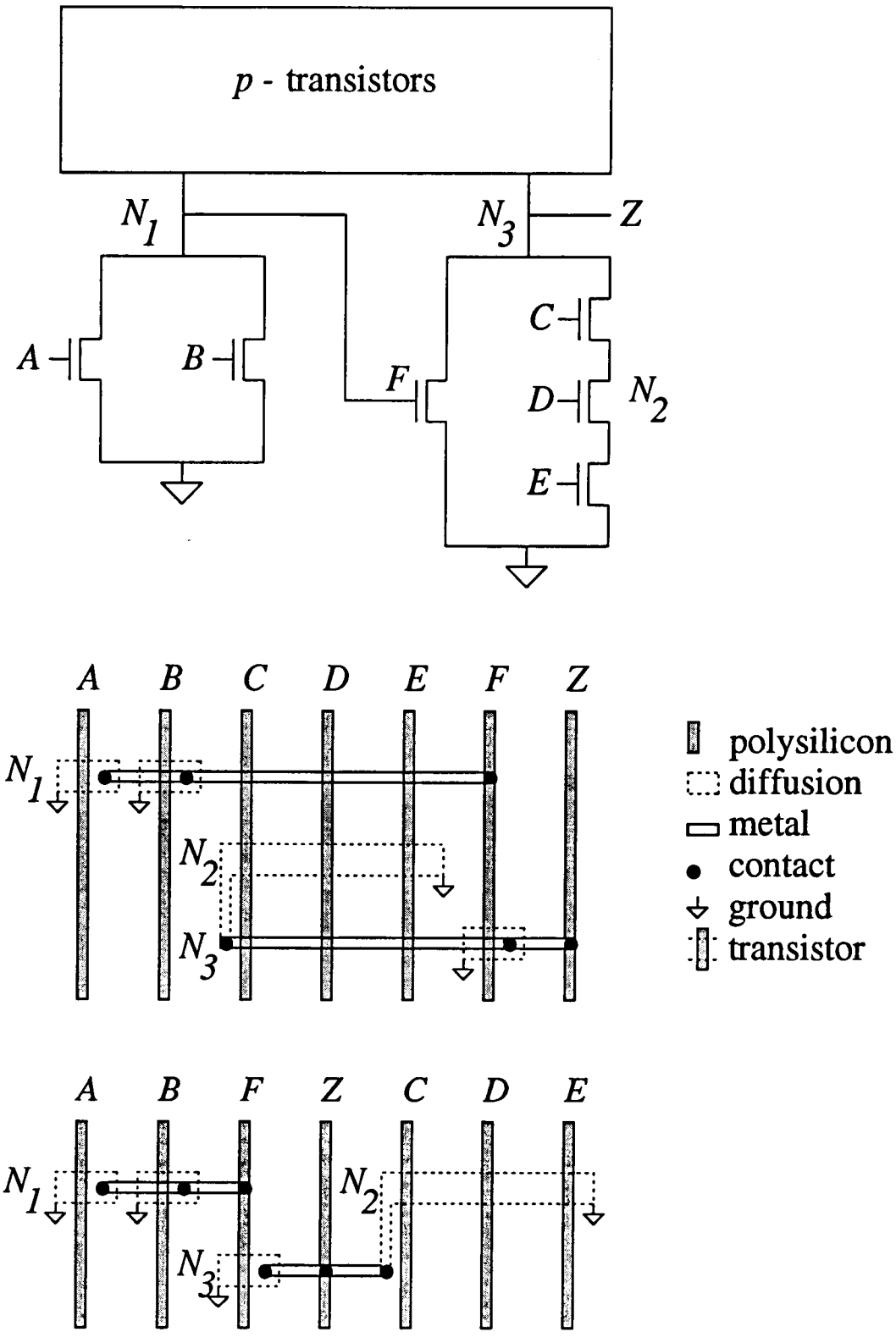


Figure 3.1: A CMOS circuit and its gate matrix layout

results in a layout requiring 3 tracks. On the other hand, the gate sequence $A - B - F - Z - C - D - E$ allows nets N_1 and N_2 to share a track and results in a layout requiring only 2 tracks for the same circuit.

Given an instance of the GML problem, we wish to know if there is a permutation of the gates which will allow the circuit to be laid out in k or fewer tracks, where k is the input parameter. In the corresponding fixed parameter problem, denoted GML(k), the number of tracks k is a fixed constant and not part of the input. The problem is to obtain a layout in no more than k tracks if possible.

3.1.2 Previous Work

Since its introduction, the gate matrix layout style has grown in popularity. Owing to their regular structure, it was expected that gate matrix layouts would be easy to generate using computers. However, since the GML problem was proven to be $\mathcal{NP}$-Hard [KF], the emphasis has been on finding near-optimal solutions. The GML problem has been studied by numerous people, employing a variety of techniques. Most of these algorithms are heuristics based on methods such as the greedy strategy and branch and bound technique [HPK, Li, WHW, Wi1, Wi2]. It is shown in [DKL] that these heuristics can produce arbitrarily bad results, and unless $\mathcal{P} = \mathcal{NP}$, there can be no algorithm that will always produce a layout that is within a constant additive factor of the optimal.

Dynamic programming [DKL] and simulated annealing [DN] methods have also been proposed for the GML problem. Although dynamic programming is an exact method that finds the optimum layout by trading space for time, it is still prohibitively slow. The dynamic programming algorithm presented in [DKL] has time complexity $O(m^2 2^m)$. Its space requirement also grows exponentially with m. Notice that this is still an improvement over the factorial time complexity of the exhaustive search algorithm that would consider all $m!$ column permutations. On the other hand, the

merits of simulated annealing tend to be controversial. In order to obtain an efficient simulated annealing algorithm, great care is required in the formulation of the problem [DN] and this is non-trivial.

Unlike GML, the fixed parameter problem is tractable. That is, for any fixed k, polynomial-time algorithms to solve GML(k) are known. Before discussing the fixed parameter version of GML, we develop some requisite theoretical background.

3.2 Problem Transformation

In this section, we show that the Gate Matrix Layout problem can be transformed to the interval graph augmentation problem. Even though this transformation has been widely used (for example [OMKKF], [Wil], [DKL]), most previous descriptions do not provide complete detail. We present a proof here both for the sake of completeness and for clarity of exposition.

Given a circuit consisting of n nets and m gates, we can represent it as a 0-1 matrix M called the *incidence matrix*, where

$$M(i,j) = \begin{cases} 1 & \text{if net } i \text{ is incident on gate } j \\ 0 & \text{otherwise} \end{cases}$$

The incidence matrix for the circuit in Figure 3.1 would look like this:

	A	B	C	D	E	F	Z
N_1	1	1	0	0	0	1	0
N_2	0	0	1	1	1	0	0
N_3	1	0	1	0	0	1	1

The GML problem is related to determining whether the incidence matrix M has the consecutive 1's property or not.

Definition A 0-1 matrix is said to have the *consecutive 1's* property if there exists a permutation of the columns such that all the 1's in each row occur consecutively.

Even though it is not obvious at a first glance, a closer examination reveals that the example incidence matrix shown above has the consecutive 1's property because

the column permutation $A - B - F - Z - C - D - E$ will make all the 1's on each row appear consecutively.

First, we show the relationship between the GML problem and the consecutive 1's property of the incidence matrix.

Lemma 3.2.1 A circuit has a gate matrix layout in k tracks if and only if there exists a (possibly empty) set of 0's in the corresponding incidence matrix M that, when changed to 1's, give a matrix M' with the consecutive 1's property and the maximum column sum in M' does not exceed k.

Proof

($\Rightarrow$): Suppose the circuit has a k-track layout. Then it is safe to assume that each net is incident on every gate between its left-most and right-most gates in such a layout. By first permuting the columns of M in accordance with the order of the gates in the layout and then, in the permuted matrix by treating every 0 between the leftmost and the rightmost 1's in a row as a 1, we see that M has the consecutive 1's property. Moreover, the maximum column sum of M is k.

($\Leftarrow$): Suppose we can find a set of 0's to change into 1's such that M has the consecutive 1's property, and the maximum column sum of M is k. Then it is easy to assign tracks to the nets by sorting the nets according following a simple greedy rule such that the circuit has a layout in k tracks (see [HS] for a proof of this). ∎

If M has the consecutive 1's property, the number of tracks required for a layout equals the maximum column sum. We can test whether M has the consecutive 1's property and also find a column permutation in $O(mn)$ time using the algorithm of Booth and Lueker [BL]. Once we have a column permutation, we can complete the track assignment as mentioned in the above proof. Notice how the 2-track layout of Figure 3.1 is based on the gate sequence $A - B - F - Z - C - D - E$ which gives the consecutive 1's property.

If M does not have the consecutive 1's property, then the GML problem is $\mathcal{NP}$-Hard. We must find a column permutation such that

i. M has the consecutive 1's property when we treat every 0 between the left-most and right-most 1's in each row as a 1, and

ii. the maximum column sum is minimized.

In the fixed parameter problem GML(k), instead of minimizing the maximum column sum, we try to find a column permutation (if one exists) such that the number of 1's in any column is at most k.

We can also model the circuit by means of a graph as follows.

Definition The *intersection graph* corresponding to an incidence matrix M is a graph $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ is a set of n vertices, one for each net, and E is a set of e edges such that, $(v_i, v_j) \in E$ if and only if both net i and net j are incident on the same gate.

In other words, the 1's in each column of M form a clique in G. For example, the intersection graph for the circuit in Figure 3.1 is simply the complete graph on 3 vertices.

Notice that the mapping from an incidence matrix to an intersection graph is many-to-one. Fellows and Langston have shown that the incidence matrix M can be expanded to a format with exactly two 1's per column (see [FL1]). They also show that each column in the expanded incidence matrix corresponds to an edge of the intersection graph G. Therefore, given a graph G, we can construct the expanded incidence matrix for it.

Even though the intersection graph G seems like an abstraction of the circuit, it captures all the essential information about the circuit for us to find a layout if one exists. In the sequel, we show that interval graphs play a key role in the gate matrix layout problem.

Definition The *clique matrix* of a graph G is a 0-1 matrix whose columns correspond to the maximal cliques of G, the rows correspond to the vertices of G, and the entry in row i of column j is either 1 or 0 depending on whether vertex i is part of the maximal clique j or not.

The following lemma presents a well known characterization of interval graphs.

Lemma 3.2.2 [FG] A graph G is an interval graph if and only if its clique matrix has the consecutive 1's property.

Proof See Chapter 8 of [Go] for a very readable proof. ∎

There is an interesting connection between the intersection graph version and the incidence matrix version of the gate matrix layout problem.

Lemma 3.2.3 If an incidence matrix M has the consecutive 1's property, then the corresponding intersection graph G is an interval graph and the order of largest clique subgraph of G equals the maximum column sum of M.

Proof We construct G by creating a vertex for each row of M and add an edge between two vertices if and only if they are both incident on the same gate. In order to prove that G is an interval graph, we show how to assign an interval to each vertex of G so that two vertices of G are adjacent if and only if their corresponding intervals overlap. We find a column permutation of M so that the rows of M have the consecutive 1's property and number the columns from 1 to m. Each vertex v of G is assigned an interval $[l, r]$, where l is the column number of the leftmost 1 and r is the column of the rightmost 1 in the row corresponding to v. If the intervals of two vertices overlap then there exists some column where the rows corresponding to both vertices are incident. From the construction of G it is clear that if two intervals overlap, then the two vertices are adjacent and every edge in G was added this way. Therefore, G is an interval graph.

Now, let k be the order of the largest maximal clique subgraph of G. Clearly, $k \geq$ the maximum column sum of M. We use induction to prove that $k =$ the maximum column sum of M. The case when $k = 1$ is trivial. When $k = 2$, G has at least one edge and we know that each edge in G is a result of some column in M having at least two 1's. Therefore, the maximum column sum of $M = 2$.

Suppose the induction hypothesis holds for all $1 \leq k \leq c$. If the claim does not hold for $k = c+1$, then there exist at least $c+1$ distinct cliques of order c in G each a subgraph of the $c+1$ clique. This implies that there are exactly $c+1$ distinct columns in M each with c ones which together result in the $c+1$ clique. However, there is no permutation of these $c+1$ columns that will give them the consecutive 1's property. This contradicts the fact that M has the consecutive 1's property. Therefore, the hypothesis must hold for $k = c+1$ as well. ∎

The next lemma is an important converse of the previous lemma.

Lemma 3.2.4 If the intersection graph G of an incidence matrix M is an interval graph, then there exists a (possibly empty) set of 0's in M, which when changed to 1's gives the clique matrix (potentially with some redundant columns) of G.

Proof If each column of M corresponds to a maximal clique of G, then we are done according to Lemma 3.2.2. Otherwise we have to change some 0's to 1's. If M is an $n \times m$ matrix, then this set of 0's can be determined in $O(n^2m)$ time by exhaustively checking each 0 in M and changing to a 1 only those 0's that would not require us to introduce a new edge in G. The matrix M that results might have multiple copies of some columns, but that does not affect the consecutive 1's property. ∎

We are now ready to show that the gate matrix layout problem is equivalent to the interval graph augmentation problem.

Theorem 3.2.5 A circuit has a gate matrix layout in k tracks if and only if the intersection graph G can be augmented to an interval graph (by adding zero or more

edges) such that the order of the largest clique subgraph of G does not exceed k.

Proof

($\Rightarrow$): If the circuit has a k-track layout, then according to Lemma 3.2.1 the incidence matrix M has the consecutive 1's property and the maximum column sum of M does not exceed k. The rest follows from Lemma 3.2.3

($\Leftarrow$): This implication follows from Lemmas 3.2.4 and 3.2.1. ∎

In other words, if the intersection graph G is an interval graph, then the size of the maximum clique in G is the least number of tracks required for a layout. A graph G can be tested for being an interval graph and if so, the size of the maximum clique can be determined, all in $O(n + e)$ time. On the other hand, if G is not an interval graph, then to minimize the number of tracks we must find an appropriate set of edges to add to G so that it becomes an interval graph and the size of the maximum clique is minimized. This problem is $\mathcal{NP}$-Hard.

3.3 The Pathwidth Metric

We define the pathwidth metric of a graph and relate it to the gate matrix layout problem.

Definition [RS2] Given a graph G, a pair (T, Y) is a *tree-decomposition* of G if T is a tree and $Y = \{X_i\}$ is a family of subsets of $V(G)$ indexed by $V(T)$ such that:

i. $\cup X_i = V(G)$,

ii. for every edge $(v_a, v_b) \in E(G)$, $\exists i \in V(T) \ni \{v_a, v_b\} \subseteq X_i$, and

iii. for $i, j, k \in V(T)$, if j is on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.

Definition The *width* of a tree-decomposition (T, Y) is $\max_{i \in V(T)}\{|X_i| - 1\}$ and $treewidth(G)$ is the minimum width over all possible tree-decompositions of G.

Note that $treewidth(K_n) = n - 1$.

Definition A *path-decomposition* of G is just a tree-decomposition (T, Y), where T is a simple path, and the *pathwidth* of G is the minimum width over all path-decompositions of G. The *length of a path-decomposition* is $|V(T)| - 1$.

The next fact follows trivially from the definition of pathwidth.

Fact 3.3.1 For any graph G, $1 \leq treewidth(G) \leq pathwidth(G) \leq |V| - 1$.

The following lemma, which relates the gate matrix layout problem to the pathwidth metric of a graph, is fundamental to our algorithms for gate matrix layout.

Lemma 3.3.2 [FL5] A circuit has a gate matrix layout in no more than k tracks if and only if the corresponding intersection graph has pathwidth at most $k - 1$.

Let PW(k) denote the set of all graphs of pathwidth at most k, and TW(k) be the set of all graphs of treewidth at most k. The next lemma shows that PW(k) is a minor-closed family.

Lemma 3.3.3 [RS1] If $H \leq_m G$, then $pathwidth(H) \leq pathwidth(G)$.

Lemmas 3.3.2 and 3.3.3, together with Lemmas 2.2.1 and 2.2.2 imply the existence of a polynomial-time algorithm to decide whether a k-track layout is possible.

It is known that for every k, there is a tree obstruction to PW(k) [FL1]. Moreover, for every planar graph H, there is a constant w such that, every graph with no minor isomorphic to H has treewidth at most w [RS3]. Using these facts, Fellows and Langston proved the existence of an $O(n^2)$ time algorithm for GML(k) [FL1]. Recently, Bodlaender has lowered the time complexity of this algorithm to $O(n)$ [Bod1]. Neither of these algorithms is practical.

3.4 Three-track Gate Matrix Layout

In this section, we demonstrate our approach for the Gate Matrix Layout (GML) problem. We already know this is a difficult problem to solve optimally. In the fixed-parameter version of GML, denoted GML(k), the maximum number of tracks allowed for a layout is fixed at some constant k. The goal is to decide whether the input circuit has a layout using k or fewer tracks, and then to obtain a layout if possible. When $k = 1$, the problem is trivially solved and for $k = 2$, GML(2) can be solved in $O(nm)$ time using an algorithm of Booth and Lueker [BL], where n is the number of rows and m is the number of columns of the matrix.

The smallest non-trivial case is when $k = 3$, and that is what we address here. Although using only three tracks might seem overly restrictive, it serves as a good test-bed for our approach, and as will be evident later, captures a large family of circuits encountered in practice. We know that each circuit with a three-track gate matrix layout can be transformed to a graph whose pathwidth is at most two. Therefore, GML(3) is equivalent to the problem of determining whether a graph has pathwidth at most two.

3.4.1 The Six Smallest Obstructions

It is known that there are exactly 110 obstructions for GML(3) [KL]. The number of vertices in these obstructions ranges from 4 to 22. Consider the six smallest obstructions, which we call A, B, C, D, E, and F (see Figure 3.2). Graph A is K_4 and graph B is also known as the Hajós graph. We are able to use the structure and relative density of these six obstructions to design fast minor containment tests for them. If we can develop fast minor containment tests for all 110 obstructions, then we would also have a fast decision algorithm for GML(3). However, the larger obstructions are also sparser and it is very difficult to design tests for them.

Our algorithms for GML(3) are based on testing for the six obstructions alone.

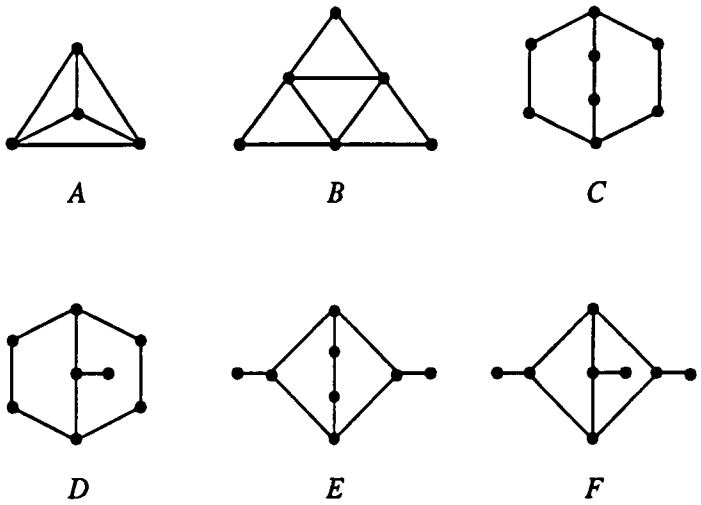


Figure 3.2: The six smallest obstructions to GML(3)

We have implemented these algorithms and the experimental results suggest that it is reasonable to expect that a circuit that cannot be laid out in three tracks will contain one of them.

3.4.2 Testing for the Obstructions

Let G be a connected graph and let H be another graph.

Definition [RS4] A *model of H in G* is a set $\{G_i : i \in V(H)\}$ of mutually disjoint, non-empty, connected subgraphs of G, together with a set $\{f_j : j \in E(H)\}$ of edges of G, such that

i. $\forall i \in V(H)$ and $\forall j \in E(H)$, $f_j \notin E(G_i)$

ii. $\forall j \in E(H)$, both endpoints of f_j are in $\cup_{i \in V(H)} V(G_i)$.

Observe that if there exists a model of H in G, then by contracting each G_i to a single vertex, we obtain a graph isomorphic to H. Moreover, every minor of G can be obtained this way.

Detecting Obstruction A

Given an intersection graph G corresponding to a circuit whose gate matrix layout is sought, we first check to see whether A is a minor of G. The following well known lemma is crucial to our algorithm. We shall prove a generalization of it in the next chapter.

Lemma 3.4.1 If G is a graph of order at least 4 and $K_4 \not\leq_m G$, then there exist a pair of non-adjacent vertices each with at most two neighbors in G.

In order to detect graph A, we use the following three graph-modifying rules.

The rules used for detecting obstruction A:

A1: **if** $(|N(v)| = 1)$ **then** delete vertex v;

Assumption: Let $\{u, w\} \subseteq N(v)$;

A2: **if** $(|N(v)| = 2$ and $(u, w) \notin E)$ **then**
begin
add edge (u, w) to E;
delete edge (u, v), edge (v, w), and vertex v;
end;

A3: **if** $(|N(v)| = 2$ and $(u, w) \in E)$ **then**
delete edge (u, v), edge (v, w), and vertex v;

Lemma 3.4.2 If G' is obtained from G by an application of rule A1, A2, or A3, then $K_4 \leq_m G'$ if and only if $K_4 \leq_m G$.

Proof Suppose G' is obtained from G by an application of rule A1, A2, or A3.
($\Rightarrow$): If G' is obtained from G by using rule A1 or A3, then $G' \subset G$. Rule A2 is equivalent to contracting v to u along the edge (u, v). Therefore, $G' \leq_m G$ in all three cases. Hence, $K_4 \leq_m G' \Rightarrow K_4 \leq_m G$.
($\Leftarrow$): Let $v \in V(G)$ and $\delta(v) \leq 2$. Suppose v belongs to the connected subgraph G_i in the model M of a K_4. Then a neighbor u of v and the edge (u, v) are also in G_i because $\delta(K_4) = 3$. Observe that each of the rules A1, A2, and A3 simply contracts

the edge (u, v) to u. After this contraction, G_i becomes a connected graph G_i'. By replacing G_i with G_i' in M, we obtain a model M' of K_4 in G'. ∎

We repeatedly modify G using rules A1, A2, and A3, until we are left with a graph G^* to which none of the rules will apply. This algorithm is well known (see [LG] for instance). If G^* is empty, then by Lemma 3.4.2 we know that $K_4 \not\leq_m G$. Otherwise, $\delta(G^*) \geq 3$ and, according to Lemma 3.4.1, $K_4 \leq_m G$.

In the sequel, we assume that $K_4 \not\leq_m G$. This will allow us to take advantage of Lemma 3.4.1 and help to simplify our algorithms for obstructions B, C, D, E, and F.

Detecting Obstruction B

Our minor containment test for obstruction B is based on the following lemma.

Lemma 3.4.3 Obstruction $B \leq_m G$ if and only if there exist three disjoint, connected subgraphs G_1, G_2, and G_3 in G such that, contracting each G_i to a single vertex v_i results in a multigraph in which there are six vertex-disjoint paths, two between each pair v_i, v_j.

Proof

($\Rightarrow$): Let v_1, v_2, and v_3 be the three vertices of degree four in B. Label the other three vertices v_4, v_5, and v_6, so that v_4 is adjacent to v_1 and v_2, and v_5 is adjacent to v_2 and v_3. $B \leq_m G \Rightarrow$ there is a model of B in G consisting of non-empty connected subgraphs $G_{v_i}, \forall i, 1 \leq i \leq 6$. We see that $G_1 = G_{v_1} \cup G_{v_4}$, $G_2 = G_{v_2} \cup G_{v_5}$, and $G_3 = G_{v_3} \cup G_{v_6}$, satisfy the requirements of the lemma.

($\Leftarrow$): If such G_1, G_2, and G_3 exist then, since G is a simple graph, there is a model of B in G. ∎

We refer to the three vertices v_1, v_2, and v_3 of this lemma as the *corner* vertices.

In order to test whether the obstruction B is a minor of an input simple graph G, our algorithm uses an internal representation with weights on the edges. If the

vertices of an edge-weighted graph are viewed as the contraction of mutually disjoint, connected subgraphs of a simple graph, then the weights denote the number of edges between these subgraphs. For the purpose of this algorithm, edge-weighted graphs will be distinguished with a hat ($\hat{}$). Initially, each edge of $\hat{G}$ is assigned weight 1. Since we only need two vertex-disjoint paths between any pair of corners, the weight of an edge is either 1 or 2.

We use the following three rules, defined only for edge-weighted graphs, in order to detect B.

The rules used for detecting obstruction B:

B1: **if** $(|N(v)| = 1)$ **then** delete vertex v;

Assumption: Let $\{u, w\} \subseteq N(v)$ and weight$(u, v) \leq$ weight(v, w);

B2: **if** $(|N(v)| = 2$ and weight$(u, v) = 1$ and $(u, w) \notin E)$ **then**
begin
introduce edge (u, w);
weight$(u, w) \leftarrow$ weight(v, w);
delete edge (u, v), edge (v, w), and vertex v;
end;

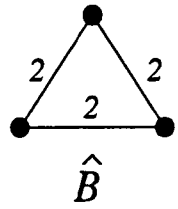
B3: **if** $(|N(v)| = 2$ and weight$(u, v) = 1$ and $(u, w) \in E)$ **then**
begin
weight$(u, w) \leftarrow 2$;
delete edge (u, v), edge (v, w), and vertex v;
end;

Definition The *simplification* of an edge-weighted graph $\hat{G}$ is the simple graph G obtained as follows:

i. Let G be the simple graph obtained by ignoring the edge weights of $\hat{G}$.

ii. For each edge (u, w) with weight 2 in $\hat{G}$, introduce a new vertex v in G and make v adjacent to both u and w.

Note that every edge-weighted graph has a unique simplification. For example, the simplification of graph $\hat{B}$ shown in Figure 3.3 is obstruction B.



$\hat{B}$

Figure 3.3: A graph with edge weights

Lemma 3.4.4 If $\hat{G}'$ is obtained from $\hat{G}$ by an application of rule B1, B2, or B3, then B is a minor of the simplification of $\hat{G}'$ if and only if B is a minor of the simplification of $\hat{G}$.

Proof Suppose $\hat{G}'$ is obtained from $\hat{G}$ by an application of rule B1, B2, or B3. Let G and G' be the simplification of $\hat{G}$ and $\hat{G}'$ respectively.

($\Rightarrow$): Suppose $B \leq_m G'$. If rule B1 was used, then $\hat{G}'$ is the same as $\hat{G} - \{v\}$. Therefore, $G' \leq_m G$. Applying rule B2 is equivalent to contracting vertex v to u along edge (u, v) both in $\hat{G}$ and in G. Hence, $G' \leq_m G$. If rule B3 was used, then $\hat{G}' - (u, w) \subset \hat{G} - (u, w)$. Since weight$(u, w)$ in $\hat{G}'$ is 2, its simplification results in an edge (u, v) and a path of length two between u and v in G'. The intermediate vertex on this path is adjacent only to u and w. The edge (u, w) exists in G also. Moreover, since u and v are neighbors of v in G, the edges $(u, v), (v, w)$ constitute a path of length two between u and w. Since $v \notin G'$, $G' \leq_m G$.

($\Leftarrow$): Let $B \leq_m G$. Then G must satisfy Lemma 3.4.3. Consider the structure of the mutually disjoint, connected subgraphs G_i of that lemma. Since v has at most two neighbors in G, none of $V(G_1), V(G_2)$, and $V(G_3)$ consists solely of v. Assume that if $u \in G_i$, then $i = 1$.

If rule B1 is used to obtain $\hat{G}'$, then every path from v passes through u in G. Therefore, if $v \in G_i$, then $u \in G_i$ also. Then, the subgraphs $G_1 - \{v\}, G_2, G_3$ exist in G' and satisfy Lemma 3.4.3. Hence, $B \leq_m G'$.

Suppose rule B2 or B3 is used to obtain $\hat{G}'$. Then, deleting the edge (u, w) from $\hat{G}'$ results in the same graph as deleting vertex v and edge (u, w) from $\hat{G}$. The rest

of the proof is based on this fact.

Since w and u are connected in G, we can assume that if $u \in G_1$, then w is in some G_i. If u and w are both in G_1, then the subgraphs G_2 and G_3 also exist in G'. In G', the subgraph induced by $V(G_1)$ can be taken instead of G_1. Since u and w are connected in G', $B \leq_m G'$.

If $u \in G_1$ and $w \in G_2$, then the two vertex-disjoint paths between G_1 and G_2 in G that are necessary to form obstruction B also exist between $G_1 - \{v\}$ and G_2 in G'. Moreover, these two paths are disjoint from the other four paths required by Lemma 3.4.3, both in G and G'. Therefore, $B \leq_m G'$.

If neither u nor w is in any G_i, then the subgraphs G_i also exist in G'. Moreover, at most one path between some G_i and G_j can include both u and w in G. Since the edge (u, w) exists in G', G' satisfies Lemma 3.4.3. Therefore, $B \leq_m G'$. ∎

We repeatedly modify $\hat{G}$ using rules B1, B2, and B3 until no further modification is possible. If a vertex v has exactly two neighbors and both edges incident on v have weight greater than one, then we do not modify it.

Lemma 3.4.5 If none of the rules B1, B2, and B3 is applicable to $\hat{G}$, then B is a minor of the simplification of $\hat{G}$.

Proof The proof is by induction on $|V(\hat{G})|$. In order for none of B1, B2, and B3 to be applicable, $|V(\hat{G})| \geq 3$. Let G be the simplification of $\hat{G}$. If $|V(\hat{G})| = 3$, then each edge of $\hat{G}$ has weight 2 and $B \leq_m G$. Assume that the lemma holds whenever $3 \leq |V(\hat{G})| \leq n$.

Suppose $\hat{G}$ is a graph of order $n + 1$. Since rule B1 cannot be applied to $\hat{G}$, each vertex in $\hat{G}$ has at least two neighbors. Let v be a vertex with exactly two neighbors u and w in $\hat{G}$ (v exists because $K_4 \not\leq_m G$). Both edges incident on v have weight 2, because rules B2 and B3 cannot be applied. If $(u, w) \notin E$, then contracting v to u along edge (u, v) results in a graph $\hat{G}'$ to which none of the rules can be applied. Let

G' be the simplification of $\hat{G}'$. Since $|V(\hat{G}')| = n$, it follows that $B \leq_m G'$. Further, since $G' \leq_m G$, we have $B \leq_m G$.

If the edge (u, w) exists and weight$(u, w) \geq 2$, then the graph $\hat{B}$ (see Figure 3.3, page 27) is a subgraph of $\hat{G}$. Hence, $B \leq_m G$. The only remaining possibility is that weight$(u, w) = 1$.

We know that any graph has a biconnected component that contains only one of the articulation points. Let $\hat{H}$ be such a biconnected component of $\hat{G}$ and let H be the simplification of $\hat{H}$. We will show that $B \leq_m H$.

We know that $K_4 \not\leq_m H$, because $K_4 \not\leq_m G$ and $H \subseteq G$. According to Lemma 3.4.1, there exists a vertex $v \in \hat{H}$, such that v has exactly two neighbors in $\hat{H}$ and v is not an articulation point of $\hat{G}$. Let $N(v) = \{u, w\}$. Then weight(u, v) = weight(v, w) = 2, and weight$(u, w) = 1$. One of u and w, say w, is not an articulation point of $\hat{G}$. Since rules B2 and B3 are not applicable, w has a third neighbor x (besides u and v) in $\hat{H}$. Since $\hat{H}$ is biconnected, there exist vertex-disjoint paths connecting u and w to x. These two paths and the edge (u, w) together make two vertex-disjoint paths between u and w that are disjoint from v in $\hat{H}$. Therefore, according to Lemma 3.4.3, $B \leq_m H$. Since $H \subseteq G$, we conclude that $B \leq_m G$. ∎

3.4.3 Detecting Four Obstructions in One Pass

Testing for obstructions C, D, E, and F is more complicated. We take advantage of the structural similarities among these four obstructions and device a single algorithm that will detect the presence of any of them.

Definition A *T-link* between two vertices u and v in a graph is a path of length two between u and v such that the intermediate vertex w on the path has a neighbor x distinct from u and v.

All four vertices u, v, w, and x, as well as the three edges $(u, w), (v, w)$, and (w, x) are considered to be part of the *T-link*.

Definition A *link* between two vertices u and v in a graph is either

i. a path of length at least three between u and v, or

ii. a *T-link* between u and v.

Observe that each of the obstructions C, D, E, and F has maximum degree three. Therefore, we can use the following lemma.

Lemma 3.4.6 [FL2] If H is a graph whose maximum degree does not exceed three, then $H \leq_m G$ if and only if $H \leq_t G$.

Our test for minor containment of C, D, E, and F is based on the next lemma. The proof is based on the fact that in each of these obstructions, there exist only two vertices with exactly three neighbors such that each neighbor in turn has degree at least two. These two vertices will be called *corners*. Then it is clear that there exist three mutually vertex-disjoint links between the corners of each obstruction.

Lemma 3.4.7 A graph G contains one of the obstructions C, D, E, or F as a minor if and only if G contains connected subgraphs G_1, G_2, and G_3 such that

i. $V(G_i) \cap V(G_j) = \{x, y\}$ for every $1 \leq i < j \leq 3$, and

ii. each G_i can be contracted to a link between x and y.

Proof ($\Rightarrow$): Suppose G contains such an obstruction as a minor. Then according to Lemma 3.4.6, G also contains the obstruction in the topological order. If $H \leq_t G$, then by subdividing zero or more edges of H we can obtain a graph $H' \subseteq G$. Hence, there exists a one to one mapping of the vertices of H into the vertices of G such that the edges of H map to vertex-disjoint paths in G. Let x and y be the corners of the obstruction. The subgraph consisting of the images of all the vertices and edges of each link between the corners is a connected subgraph of G. Since the images of the

edges are vertex-disjoint paths, the three subgraphs corresponding to the three links intersect only at the corners.

($\Leftarrow$): Suppose there exist connected subgraphs G_1, G_2, and G_3 in G that satisfy conditions (i) and (ii) of the lemma. Then, depending on the type of link to which each subgraph can be contracted, G contains at least one of the obstructions C, D, E, and F. ∎

For the purpose of testing whether any of the obstructions C, D, E, and F is a minor of a simple graph G, our algorithm uses an internal representation with weights on the vertices as well as the edges of G. Recall that to test for obstruction B, we only used edge weights. Here, we also assign weights to the vertices in order to find T-links. Graphs with both vertex and edge weights will be distinguished with a check ($\check{}$). Thus, the input simple graph G is stored as $\check{G}$ whose vertex and edge weights are initialized as follows.

The weight of a vertex is either 0 or 1. Initially, every vertex with three or more neighbors is assigned weight 1. All other vertices are assigned weight 0. The vertex weights are not changed during the algorithm.

The weight of an edge is an integer between 1 and 4. All edge weights are initialized to 1. Edge weight 2 denotes a path of length two between the end-points. A weight of 3 means that we have found one link. A weight 4 means that we have found two vertex-disjoint links between the end-points. The edge weights never decrease during the algorithm.

We use the following five rules, defined only for graphs with both vertex and edge weights, in order to detect obstructions C, D, E, and F.

The rules used for detecting obstructions C, D, E, and F:

C1: **if** $(|N(v)| = 1)$ **then** delete vertex v;

Assumption: Let $\{u, w\} \subseteq N(v)$ and weight$(u, v) \leq$ weight(v, w);

C2: **if** $(|N(v)| = 2$ and weight$(v, w) \leq 3$ and $(u, w) \notin E)$ **then**
begin
add edge(u, w);
weight$(u, w) \leftarrow$ minimum{3, weight(u, v)+weight(v, w)+weight(v)};
delete edge (u, v), edge (v, w), and vertex v;
end;

C3: **if** $(|N(v)| = 2$ and weight$(v, w) \leq 3$ and weight$(u, w) < 3)$ **then**
begin
weight$(u, w) \leftarrow$ minimum{3, weight(u, v)+weight(v, w)+weight(v)};
delete edge (u, v), edge (v, w), and vertex v;
end;

C4: **if** $(|N(v)| = 2$ and weight$(v, w) \leq 3$ and weight$(u, w) = 3$ and
weight(u, v)+weight(v, w)+weight$(v) \geq 3)$ **then**
begin
weight$(u, w) \leftarrow 4$;
delete edge (u, v), edge (v, w), and vertex v;
end;

C5: **if** $(|N(v)| = 2$ and weight(u, v)+weight(v, w)+weight$(v) < 3$ and
weight$(u, w) \geq 3)$ **then** delete edge (u, v), edge (v, w), and vertex v;

Definition An *expansion* of a weighted graph $\check{G}$ is a simple graph G obtained using the following sequence of five steps:

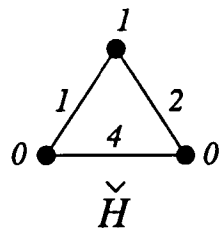
i. Let G be the simple graph obtained by ignoring the weights of $\check{G}$.

ii. If an edge of $\check{G}$ has weight 2, then subdivide the corresponding edge in G.

iii. If an edge has weight 3 in $\check{G}$, then replace that edge in G with a link of any kind.

iv. If an edge has weight 4 in $\check{G}$, replace that edge in G with a pair of vertex-disjoint links of any kind.

v. Finally, if a vertex v of weight 1 in $\check{G}$ has less than three neighbors in G, then add a new vertex to G and make it adjacent to v.



$\check{H}$

Figure 3.4: A graph with vertex and edge weights

In general, a graph $\check{G}$ can have more than one expansion. For example, each of the obstructions C, D, E, and F is a minor of an expansion of graph $\check{H}$ shown in Figure 3.4. The expansion of a graph is unique only if its maximum edge weight is 2.

Lemma 3.4.8 If $\check{G}'$ is obtained from $\check{G}$ by an application of rule C1, C2, C3, C4, or C5, then an expansion of $\check{G}'$ contains one of the obstructions C, D, E, and F as minor if and only if an expansion of $\check{G}$ does.

Proof

($\Rightarrow$): Suppose G' is an expansion of $\check{G}'$ that contains one of C, D, E, and F as a minor. If $\check{G}'$ was obtained using rule C1 or C5, then $\check{G}'$ is the same as $\check{G} - \{v\}$. Therefore, G' is a minor of an expansion of $\check{G}$.

If one of the rules C2, C3, and C4 was used, then deleting the edge (u, w) from $\check{G}'$ results in a graph identical to that obtained by deleting vertex v and edge (u, w) from $\check{G}$. If rule C2 or C3 was used, then weight(u, w) in $\check{G}' \leq$ weight(u, v) + weight(v, w) + weight(v) in $\check{G}$. Therefore, G' is a minor of an expansion of $\check{G}$.

Let $\check{H}'$ be the subgraph induced by u and w in $\check{G}'$, and let $\check{H}$ be the subgraph induced by u, v, and w in $\check{G}$. Consider the expansion $H' \subseteq G'$ of $\check{H}'$ and an expansion H of $\check{H}$.

If rule C4 was used, then weight$(u, w) = 4$ in $\check{G}'$. Hence, there are two vertex-disjoint links between u and w in H'. Since weight$(u, w) = 3$ in $\check{G}$, there exists a link in H between u and w that excludes vertex v. Further, since

weight(u,v)+weight(v,w)+weight$(v) \geq 3$, a link between u and w in H that includes vertex v can be obtained by contracting zero or more edges. Therefore, G' is a minor of an expansion of $\check{G}$.

($\Leftarrow$): Let G be an expansion of $\check{G}$ that contains one of the four obstructions. We show that an expansion G' of $\check{G}'$ also contains an obstruction.

If an expansion of $\check{G} - \{v\}$ contains an obstruction, then so does some expansion of $\check{G}'$. This is because for rules C1 and C5 $\check{G}'$ is identical to $\check{G} - \{v\}$. For rule C2, the edge (u,w) does not exist $\check{G}$. For rules C3 and C4, the weight of edge (u,w) in $\check{G}'$ is not less than its weight in $\check{G}$. Therefore, every expansion of $\check{G} - \{v\}$ is a subgraph of an expansion of $\check{G}'$.

Suppose no expansion of $\check{G} - \{v\}$ contains an obstruction. Let G_1, G_2, and G_3 be the connected subgraphs, and let x and y be the corners of an obstruction in G that satisfy Lemma 3.4.7. Notice that only those vertices that were originally in $\check{G}$ and none of the vertices introduced during expansion can serve as corners in G. Since v has at most two neighbors in $\check{G}$, it cannot be a corner in G. Thus v must be an essential part of a link between the corners of any obstruction in G. Therefore, assume that v is in the connected subgraph G_1 of G.

If v is part of a T-link between the corners, then it must be the intermediate vertex of the T-link. Otherwise, the intermediate vertex of the T-link would have weight 1 in both $\check{G}$ and $\check{G} - \{v\}$ and as a result, an expansion of $\check{G} - \{v\}$ would contain the same obstruction as G. Therefore, if v has only one neighbor in $\check{G}$, then it cannot be part of a link between the corners. Hence, v must have at least two neighbors in $\check{G}$, and rule C1 is not applicable. Let $N(v) = \{u, w\}$.

Suppose v is part of a T-link between the corners. Then v is the intermediate vertex of T-link between u and w, and weight$(v) = 1$. Therefore, u and w must also be in G_1 besides v. Also, weight(u,v) + weight(v) + weight$(v,w) \geq 3$ in $\check{G}$.

If v does not form a T-link, then it must be part of a path of length at least three

between the corners. Therefore, u and w must also be in G_1 besides v. Observe that G_1 can be contracted to a path of length three between u and w that also includes v if and only if weight(u,v)+ weight$(v,w) \geq 3$ in $\check{G}$.

Suppose weight(u,v)+ weight(v)+ weight$(v,w) \geq 3$. In this case, rule C5 cannot be applied. Let $\check{G}'$ be obtained by applying one of the rules C2, C3, and C4. Since the number of vertex-disjoint links between u and w in any expansion G' of $\check{G}'$ is the same as the number in G, we conclude that G' also contains an obstruction.

Suppose weight(u,v)+ weight(v)+ weight$(v,w) < 3$. In this case, rule C4 cannot be applied. Since v has to be part of a path of length at least three between the corners, u and w cannot both be corners. Hence, if the edge (u,w) exists, then weight$(u,w) = 1$. Therefore, rule C5 cannot be applied. Let $\check{G}'$ be obtained by applying rule C2 or C3. Then the length of the longest path between u and w in G' is the same as that in G. Therefore, G' contains an obstruction also. This completes the proof. ∎

Lemma 3.4.9 If none of the rules C1, C2, C3, C4, and C5 can be applied to $\check{G}$, then an expansion of $\check{G}$ contains one of the obstructions C, D, E, and F as a minor.

Proof Since rule C1 cannot be applied, each vertex of $\check{G}$ has at least two neighbors. Since K_4 is not a minor of any expansion of $\check{G}$, there exists a vertex v with exactly two neighbors in $\check{G}$. Let $N(v) = \{u,w\}$. As in the rules C2–C5, assume that weight$(u,v) \leq$ weight(v,w). There are two cases to consider.

Case-1: weight$(u,v) \leq$ weight$(v,w) \leq 3$.

In order for v not to meet the conditions of rules C2–C5, the edge $(u,w) \in E$, weight$(u,w) > 3$, and weight(u,v) + weight(v,w) + weight$(v) \geq 3$. Consider the subgraph $\check{H}$ induced by vertices u, v, and w in $\check{G}$. There exist two links between u and w, and the vertex v along with its two edges is the third link. Therefore, $\check{H}$ can be expanded into an obstruction.

Case-2: weight$(u, v) \leq$ weight$(v, w) = 4$.

The proof of this is by induction on $|V(\check{G})|$. If $|V(\check{G})| = 3$, then every vertex of $\check{G}$ has two neighbors. If weight$(u, v) \leq 3$ and weight$(u, w) \leq 3$, then this reduces to Case-1. Otherwise, at least one of the edges (u, v) and (u, w) has weight 4. As a result, an expansion of $\check{G}$ contains an obstruction.

Suppose the lemma is true for all $3 \leq |V(\check{G})| \leq n$. Let $|V(\check{G})| = n + 1$, and let $v \in V(\check{G})$ with $N(v) = \{u, w\}$.

If weight$(u, v) < 3$, then we contract the edge (u, v) to obtain $\check{G}'$. Since $|V(\check{G}')| = n$, an expansion G' of $\check{G}'$ contains an obstruction. Moreover, there exists an expansion G of $\check{G}$ such that $G' \leq_m G$. Therefore, G also contains the obstruction.

If weight$(u, v) \geq 3$, let $\check{H}$ be a biconnected component of $\check{G}$ containing only one articulation point of $\check{G}$. There exists a vertex $v \in V(\check{H})$ such that $N(v) = \{u, v\}$ and v is not an articulation point of $\check{G}$. Moreover, there exists a path from u to w in $\check{H}$ that does not include v. Since weight$(v, w) = 4$ and weight$(u, v) \geq 3$, the subgraph of $\check{H}$ induced by u, v, and w, along with this path between u and w can be expanded to a simple graph that contains an obstruction. Therefore, an expansion of $\check{G}$ also contains the obstruction. ∎

3.4.4 A Linear Time Test for Obstructions

The algorithm that we use to test for minor containment of the graphs A, B, C, D, E, and F is called MINOR-TEST and is shown in Figure 3.5. There are three distinct phases to this algorithm. The first phase is used to detect obstruction A, the second phase is used for B, and the third stage is used to detect the presence of any of C, D, E, and F. The algorithm terminates as soon as an obstruction is detected. In each phase, MINOR-TEST initializes the vertex and edge weights of the graph and also the set of graph modification rules R as appropriate and invokes the procedure REDUCE-GRAPH.

Algorithm MINOR-TEST

Input : A connected simple graph G.
Output: YES, if any of the obstructions A, B, C, D, E, and F is a minor of G (see Figure 3.2 on page 23);
NO, otherwise.

begin procedure

1. Read the input graph G in uninitialized adjacency matrix form.
2. $R \leftarrow$ {A1,A2,A3} (see page 24);
   if (REDUCE-GRAPH(G, R) = NO) **then** output YES and **stop**;
3. Initialize the weight of each edge to 1;
   Initialize the weight of each vertex to 0;
4. $R \leftarrow$ {B1,B2,B3} (see page 26);
   if (REDUCE-GRAPH(G, R) = NO) **then** output YES and **stop**;
5. Initialize the vertex weights as follows:
   if ($|N(v)| \leq 2$) **then** weight(v) $\leftarrow$ 0;
   else weight(v) $\leftarrow$ 1;
6. R $\leftarrow$ {C1,C2,C3,C4,C5} (see page 32);
   if (REDUCE-GRAPH(G, R) = NO) **then** output YES;
   else output NO;

end procedure.

Figure 3.5: Algorithm MINOR-TEST

Algorithm REDUCE-GRAPH (shown in Figure 3.6) is the general scheme used to modify the input graph according the set of specified rules. First a queue of all vertices that have only one or two neighbors is created. Then each vertex in the queue is visited sequentially. During a visit, the algorithm checks if any of the rules from the set R can be applied to that vertex. If a rule is applicable to the vertex, then the edges are modified before the vertex is deleted. If this deletion causes any vertex to have exactly 2 neighbors then that vertex is added to the queue. Therefore, the order of the graph decreases with each application of a rule. If none of the rules can be applied then according to Lemmas 3.4.1, 3.4.2, 3.4.5, and 3.4.9, the graph is either empty or contains an obstruction. The output of REDUCE-GRAPH is YES if the graph is empty, and NO otherwise. Therefore, Algorithm MINOR-TEST will always terminate with the correct result.

Algorithm REDUCE-GRAPH

Input : A connected graph G, and a set R of rules.
Output: YES, if G can be reduced to the empty graph using R;
NO, otherwise.

```
begin procedure
1. Create a graph H isomorphic to G.
2. Create a queue of all vertices with only one or two neighbors.
3. while (the queue is not empty) do
   begin
      remove a vertex v from the queue;
      if (a rule in R applies to v) then
      begin
         L ← list of neighbors of v with exactly three neighbors;
         modify H according to the rule;
         if (a vertex in L has degree 2) then enqueue it;
      end;
      if (no rule in R applies to v) then mark v as visited;
   end;
4. if (H is empty) then output YES;
   else output NO;
end procedure.
```

Figure 3.6: Algorithm REDUCE-GRAPH

Lemma 3.4.10 Algorithm MINOR-TEST has a running time of $O(n)$.

Proof All operations performed by algorithm MINOR-TEST take constant time except for the three calls to algorithm REDUCE-GRAPH. The basic operations performed by algorithm REDUCE-GRAPH are: searching for vertices of minimum degree, deleting vertices and edges, checking for the existence of specific edges, and adding edges. Using an uninitialized adjacency matrix representation for the graph [AHU], each of these operations can be performed in constant amortized time. It is known that $K_4 \not\leq_m G \Rightarrow e \leq 2n-3$ (see [Bol] for a proof). Therefore, each operation is used $O(n)$ times, guaranteeing that algorithm REDUCE-GRAPH runs in linear time. Therefore, MINOR-TEST also has a time complexity of $O(n)$. ∎

3.5 Layout Algorithms

Testing for the presence of obstructions is sufficient only to tell us whether or not a layout is possible. How can we find a layout when we know that it exists? We address this question here.

3.5.1 A Self-Reduction Technique

Self-reduction is a process used to a build a search algorithm for a problem by making repeated calls to a decision algorithm for the problem. Recall that a decision algorithm only tells us whether a solution exists and need not necessarily find a solution. It is known that any $\mathcal{NP}$-Complete problem is self-reducible [BG]. However, there are problems for which no self-reduction has been found even though a polynomial-time decision algorithm is known to exist [BFL]. The fact that a decision algorithm for GML(k) could be used to find a layout was first shown by Brown, Fellows, and Langston [BFL].

For any fixed k, the obstruction set for GML(k) is finite and there exists an $O(n)$ time minor-test for each obstruction. Thus, there is an $O(n)$ time algorithm to decide whether a graph contains an obstruction to GML(k). If the intersection graph contains an obstruction then no k-track layout is possible. Otherwise, a layout exists. In order to construct a layout, we modify the incidence matrix M by changing every possible 0 to a 1 without introducing an obstruction. This is accomplished sequentially by consulting the decision algorithm each time we change a 0 to a 1 to ensure that the graph corresponding to the modified matrix does not contain an obstruction. Let us call the final matrix obtained by self-reduction the *full* matrix.

Lemma 3.5.1 The full matrix M' obtained from the incidence matrix M by self-reduction based on an obstruction testing algorithm has the consecutive 1's property.

Proof Since the intersection graph G' corresponding to M' does not contain an

obstruction, there exists a layout in k tracks. Since M' is full, changing any 0 in M' to a 1 introduces an obstruction in G'. Therefore, according to Lemma 3.2.1, M' has the consecutive 1's property. Moreover, the maximum column sum of M' is at most k because otherwise $K_{k+1} \subseteq G'$ and we know that $K_{k+1} \in obs(\text{GML}(k))$. ∎

Our version of the self-reduction algorithm, which we call SELF-REDUCE, is shown in Figure 3.7.

Algorithm SELF-REDUCE

```
Input  : An n × m Boolean matrix M and
         the corresponding intersection graph G.
Output: M and G after self-reduction for k-track gate matrix layout.
begin procedure
for j = 1 to m do
begin
    colsum ← number of 1's in column j of M
    i ← 1;
    while ((colsum ≤ k) and (i ≤ n)) do
    begin
        if (M(i,j) = 0) then
        begin
            M(i,j) ← 1;
            Add edges to G if necessary and remember them;
            Call the decision algorithm on G;
            if a layout is possible then colsum ← colsum + 1;
            else
            begin
                M(i,j) ← 0;
                Delete the edges that were added in G;
            end;
        end;
        i ← i + 1;
    end;
end;
end procedure.
```

Figure 3.7: Algorithm SELF-REDUCE

Since we can decide in $O(n)$ time whether a layout exists, by sequentially attempt-

ing to change each of the at most nm 0's in the matrix M to a 1, we make $O(nm)$ calls to the decision algorithm. Therefore, algorithm SELF-REDUCE runs in time $O(n^2m)$.

The complete algorithm for k-track gate matrix layout for any fixed k, based on self-reduction using a known (but possibly faulty) decision algorithm, is called GATE-MATRIX-LAYOUT and is shown in Figure 3.8.

Algorithm GATE-MATRIX-LAYOUT

Input : The netlist of a circuit consisting of n nets and m gates.
Output: A k-track gate matrix layout for the circuit, or
NO, if a k-track layout could not be found.
begin procedure
1. Read the netlist of the circuit.
2. Obtain the Boolean matrix M representing the circuit.
   Transform the matrix M to the intersection graph G.
3. Call the decision algorithm on G.
   if a layout is not possible **then**
   output NO and **stop**;
4. Call the self-reduction algorithm on M and G.
5. Test whether M has the consecutive 1's property.
   if M does not have the consecutive 1's property **then**
   output NO and **stop**;
   else find a good column permutation of M;
6. Sort the n nets into m buckets according to their left end-point.
   Assign a track to each net using a greedy strategy.
7. Output the layout.

end procedure.

Figure 3.8: Algorithm GATE-MATRIX-LAYOUT

Once the self-reduction is completed, finding a column permutation that gives the consecutive 1's property takes only $O(nm)$ time [BL]. We can sort the nets by their left end-point and use this sorted list to assign them to tracks in a greedy manner using the rule that two nets can share a track only if they do not overlap or intersect [HS]. The n nets can be sorted into m buckets in $O(n + m)$ time and the track assignment also takes the same amount of time.

If we use a correct decision algorithm in step 3, and call algorithm SELF-REDUCE in step 4, then matrix M will always have the consecutive 1's property in step 5. In this case, we have the following theorem.

Theorem 3.5.2 [BFL] If a circuit has a gate matrix layout in k tracks, then such a layout can be found in $O(n^2m)$ time, where n and m are the number of nets and gates, respectively.

In the next section we explore self-reductions based on faulty decision algorithms.

3.5.2 Imperfect Self-Reductions

In general, self-reduction is based on a correct decision algorithm for the problem (i.e., an algorithm that will always correctly decide whether a solution exists to the given instance). What happens if we used an imperfect decision algorithm? Consider for example, algorithm MINOR-TEST which tests for only six out of the 110 obstructions for GML(3). When this algorithm finds an obstruction, it says "YES" meaning that a layout is not possible. If it fails to find one of the six obstructions, then it outputs "NO". In this case, our self-reduction would prematurely assume that a layout is possible. Suppose we base our 3-track gate matrix layout algorithm on self-reduction using MINOR-TEST. How would such an algorithm behave?

With a perfect decision algorithm, the matrix M is always guaranteed to have the consecutive 1's property at the end of self-reduction. This is no longer true for an imperfect self-reduction. Clearly, if the original graph contains an obstruction to GML(3) other than the six that algorithm MINOR-TEST detects, then the obstruction will not be detected and the self-reduction would be futile. This is the case of the "false positive" because a 3-track layout never existed.

More interestingly, suppose the circuit indeed has a 3-track layout, but during self-reduction by changing a 0 to a 1, we introduce an extraneous obstruction in the

graph. This would be an error and by failing to detect our folly, algorithm MINOR-TEST would lead us astray because at the end of the self-reduction we would be unable to find the layout. This is the case of the "false negative" because the self-reduction fails to find a layout even though one exists. However, when an imperfect self-reduction fails to find a layout, we cannot distinguish between a false positive produced by the decision algorithm and more importantly, a false negative produced during self-reduction.

Observe that false positives are inevitable. However, the occurrence of false negatives should be minimized. One way to avoid false negatives entirely is to use a form of self-reduction where we change a 0 to a 1 only if it does not amount to introducing a new edge in the graph. We call this algorithm SR1 (see Figure 3.9). Let M be an $n \times m$ gate matrix and G be the intersection graph. Notice that SR1 never calls the decision algorithm MINOR-TEST and is a rather degenerate form of self-reduction where the final matrix that we obtain is simply the clique matrix of G. Therefore, SR1 is only checking whether the graph G is an interval graph as per Lemma 3.2.2.

Algorithm SR2 is similar to algorithm SELF-REDUCE except for the fact that we now use MINOR-TEST as the decision algorithm. Another variation of self-reduction that we attempt is called SR3 (see Figure 3.10 on page 45). In this case, each time we change a 0 to a 1 we test whether the matrix has the consecutive 1's property and if so, we stop immediately. Recall that a graph of order n and pathwidth at most k can have only $O(n)$ edges. Moreover, the number of 1's in any column of the corresponding incidence matrix does not exceed $k+1$. Therefore, during self-reduction, we can add at most $O(n)$ new edges to the graph, and change at most $O(m)$ 0's to 1's. Therefore, for GML(3), algorithm SR1 takes $O(nm)$ time, SR2 takes $O(n^2m)$ time, and SR3 would take $O(n^2m + nm^2)$ time. Note that if $m = O(n)$, then SR3 has the same time complexity as SR2 even though in reality it involves more work.

Algorithm SR1

Input : A Boolean matrix M and the corresponding intersection graph G.
Output: M and G after self-reduction for k-track gate matrix layout.

```
begin procedure
for j = 1 to m do
begin
    L ← list of all rows with a 1 in column j of M;
    colsum ← number of 1's in column j of M;
    i ← 1;
    while ((colsum ≤ k) and (i ≤ n)) do
    begin
        if (M(i,j) = 0) then
        begin
            flag ← FALSE;
            for (each row r ∈ L) do
                if (edge(i,r) ∉ G) then flag ← TRUE;
            if (flag = FALSE) then
            begin
                M(i,j) ← 1;
                colsum ← colsum + 1;
                add row i to list L;
            end;
        end;
        i ← i + 1;
    end;
end;
end procedure.
```

Figure 3.9: Algorithm SR1

Algorithm SR3

Input : A Boolean matrix M and the corresponding intersection graph G.
Output: M and G after self-reduction for k-track gate matrix layout.

```
begin procedure
for j = 1 to m do
begin
    colsum ← number of 1's in column j of M;
    i ← 1;
    while ((colsum ≤ k) and (i ≤ n)) do
    begin
        if (M(i,j) = 0) then
        begin
            M(i,j) ← 1;
            Add edges to G if necessary and remember them.
            Call algorithm MINOR-TEST on G.
            if (a layout is not possible) then
            begin
                M(i,j) ← 0;
                Delete the edges that were added in G.
            end;
            else
            begin
                colsum ← colsum + 1;
                if (M has the consecutive 1's property) then stop;
            end;
        end
        i ← i + 1;
    end ;
end ;
end procedure.
```

Figure 3.10: Algorithm SR3

How well do these imperfect self-reductions perform in practice? We explore this question in the next section.

3.6 Experimental Results

In order to study the behavior of our imperfect self-reduction, we implemented algorithm MINOR-TEST. We also implemented the algorithm of Fulkerson and Gross [FG] to determine whether a 0-1 matrix has the consecutive 1's property. Even though this algorithm takes $O(n^2m)$ time in contrast to the $O(nm)$ time of [BL], we chose it for ease of implementation. These programs were written in the C language on Sun SPARC stations. We compare our results with those produced by Kinnersley's [Ki1] implementation of the exact algorithm of [DKL] which is based on dynamic programming and is extremely slow.

3.6.1 Randomly Generated Inputs

We tested our algorithm on large numbers of randomly generated inputs. For each fixed value of n and probability p, we generated a thousand random graphs of order n with uniform edge probability p and considered each graph to be the intersection graph of a circuit with n nets. The incidence matrix M corresponding to such a random graph G was obtained by creating a row (net) for each vertex in G, a column (gate) for each edge in G, and putting two 1's in each column corresponding to the two end-points of the edge. The rest of the matrix consists of 0's. Note that the number of 1's in each row is the same as the degree of the corresponding vertex.

Given such a matrix M and the corresponding graph G, we first use algorithm MINOR-TEST to determine whether a 3-track layout is possible and then we self-reduce. The same matrix M is also input to the dynamic programming algorithm and the exact result is found. Tables 3.1, 3.2, and 3.3 on page 48 show some representative results from our experiments. Figures 3.11, 3.12, and 3.13 on the subsequent pages

present these results in graphical form.

The first column of each table shows the number of nets in the circuit. Since we test for all obstructions of order less than 8, ours is an exact algorithm for circuits with 7 nets or less. Therefore, we start our experiments with 8 nets. The second column shows the exact result obtained by the exponential time algorithm of [DKL]. The third column shows the result of using algorithm MINOR-TEST to decide whether the circuit has a 3-track layout or not. The difference between the third and second columns is due to the occurrence of false positives.

Each circuit that passes MINOR-TEST is self-reduced using three different algorithms, namely SR1, SR2, and SR3 for comparison. Notice that the performance of SR1 is the worst of the three self-reductions. In trying to avoid false negatives, we find that the clique matrix of a large number of the graphs generated fail to have the consecutive 1's property and therefore are not interval graphs. This means that they do need to be augmented with some new edges before we can find a layout. SR2 is a tremendous improvement over SR1 and in fact is fairly close to the exact algorithm. Contrary to expectations, our experiments show that SR3 is only a marginal improvement over SR2 and may not justify the additional computing time.

From Table 3.4 on page 52 it is clear that as the number of nets increases, so does the number of obstructions that algorithm MINOR-TEST does not detect. Therefore, for larger circuits, due to the increased possibility of a false negative, it seems unlikely that our imperfect algorithms would do very well. For instance, consider the case when $n = 15$. Even though there exist 60 obstructions of order 15 or less for GML(3) (see Table 3.4), for $p = 0.25$ and $n = 15$, using SR2 we are able to find a layout for 11 out of the 14 circuits that do have a layout. From the tables it is clear that for each p and over all values of n, when compared to the exact algorithm, SR2 is successful well over 90% of the time.

Table 3.1: Experimental results for edge probability = 0.5

number of vertices	circuits with a 3-track layout*	MINOR-TEST: layout possible	layouts found by Self-Reduction		
			SR1	SR2	SR3
8	97	97	29	96	96
9	12	12	7	12	12
10	5	5	0	5	5
11	1	1	0	1	1
12	0	0	0	0	0

*determined by dynamic programming

Table 3.2: Experimental results for edge probability = 0.33

number of vertices	circuits with a 3-track layout*	MINOR-TEST: layout possible	layouts found by Self-Reduction		
			SR1	SR2	SR3
8	711	711	402	702	703
9	454	455	208	440	440
10	221	222	95	199	201
11	94	95	26	83	83
12	16	17	8	13	13
13	6	6	4	5	6
14	0	0	0	0	0

Table 3.3: Experimental results for edge probability = 0.25

number of vertices	circuits with a 3-track layout*	MINOR-TEST: layout possible	layouts found by Self-Reduction		
			SR1	SR2	SR3
8	928	928	658	913	914
9	789	790	495	751	753
10	620	624	338	575	578
11	449	453	208	389	392
12	245	251	99	213	214
13	121	125	51	96	96
14	36	39	16	28	28
15	14	15	4	11	11
16	3	3	1	2	2
17	0	0	0	0	0

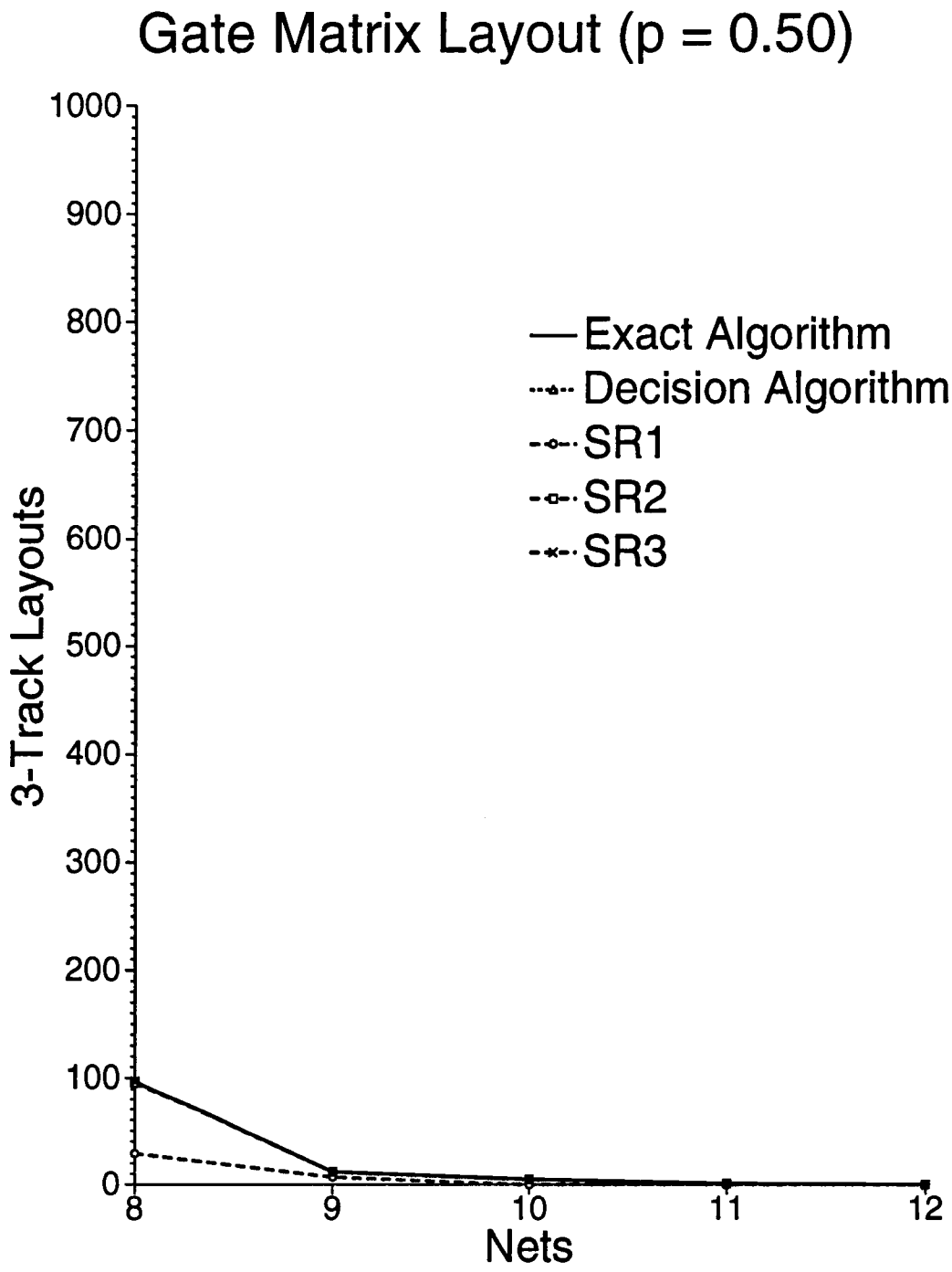


Figure 3.11: Graphical representation of Table 3.1

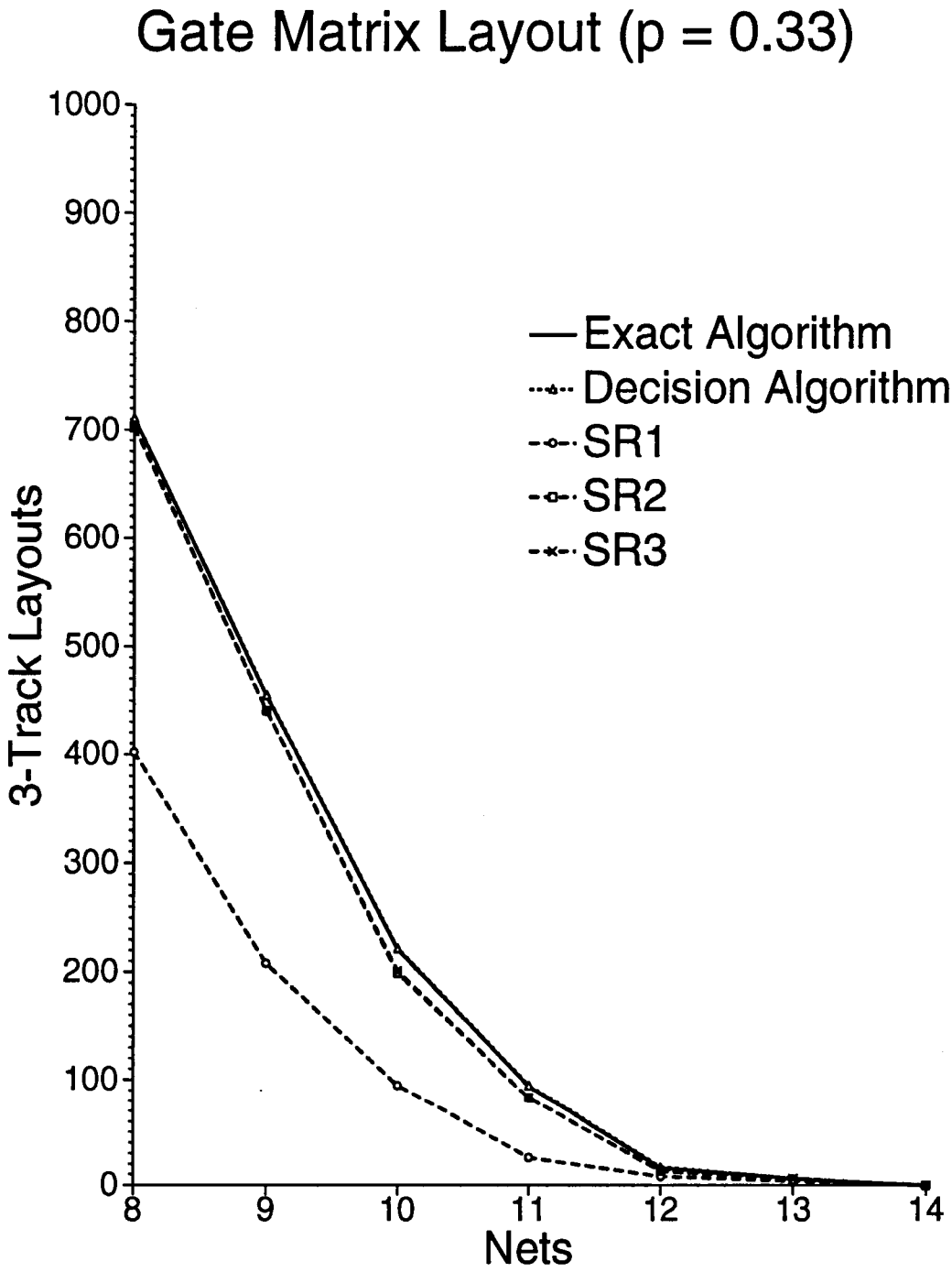


Figure 3.12: Graphical representation of Table 3.2

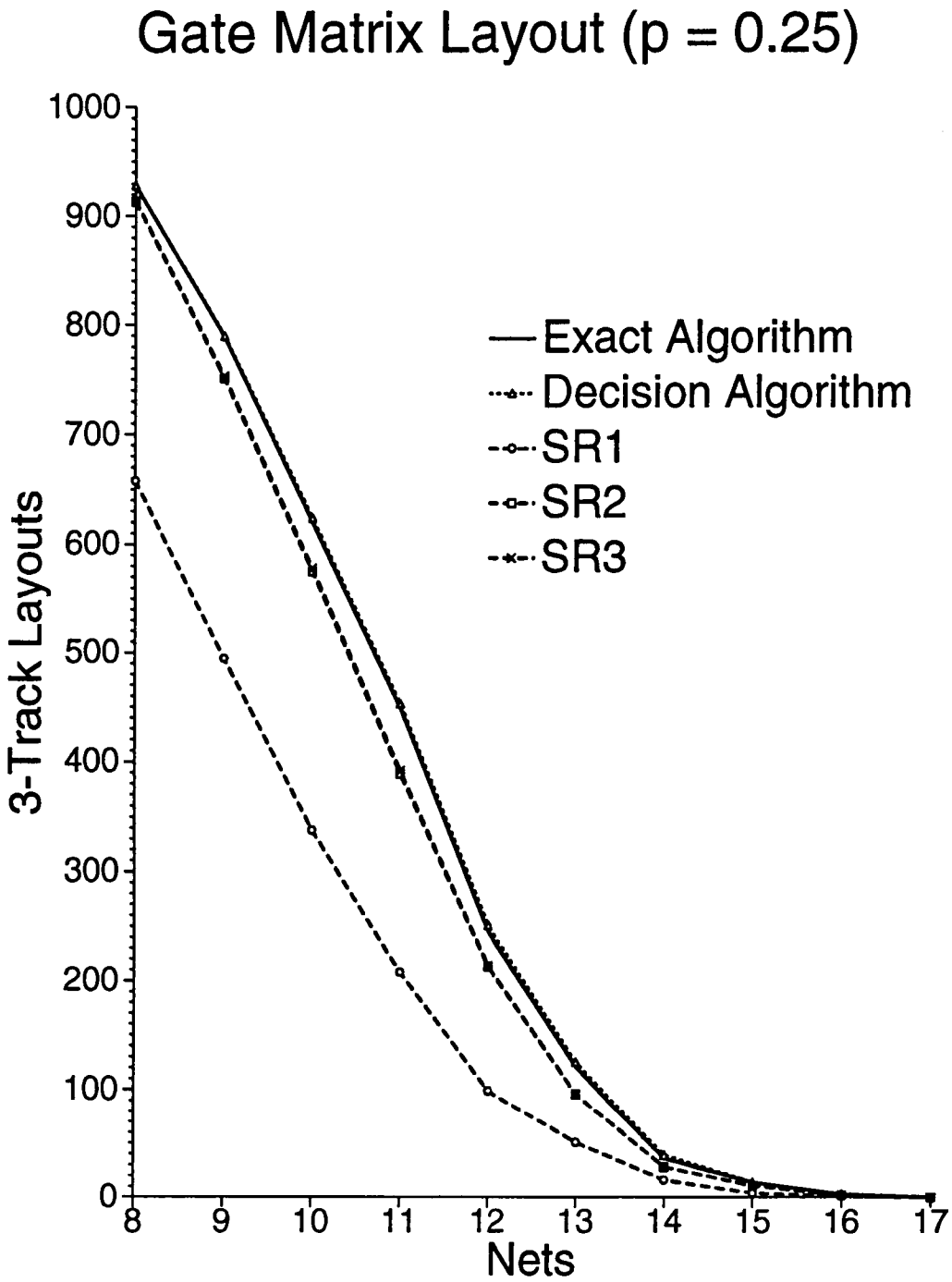


Figure 3.13: Graphical representation of Table 3.3

Table 3.4: The Distribution of Obstructions for GML(3)

number of vertices	number of obstructions	cumulative number
4	1	1
5	0	1
6	1	2
7	0	2
8	5	7
9	1	8
10	1	9
11	3	12
12	2	14
13	10	24
14	19	43
15	17	60
16	9	69
17	10	79
18	13	92
19	5	97
20	2	99
21	1	100
22	10	110

3.6.2 Comparison With Greedy Heuristics

A family of matrices that demonstrate the worst-case behavior of published greedy heuristics for the GML problem is presented in [DKL]. Even though 4 tracks are sufficient for this family, those heuristics that use a greedy strategy to find a good gate ordering would produce a layout that requires $n-1$ tracks for n nets. By deleting the bottom row from this family, we have adapted the 4-track example to 3-track layouts.

We start with a 0-1 matrix with n rows and $2(n-2)$ columns as shown in Figure 3.14, where each * is treated as a 0.

1	*	1	*	1	*		1	0
1	1	0	0	0	0		0	0
0	0	1	1	0	0	...	0	0
0	0	0	0	1	1		0	0
			⋮					
0	0	0	0	0	0		1	1
0	1	*	1	*	1		*	1

Figure 3.14: A worst-case matrix

It is easy to see that this matrix requires at least 3 tracks for a layout. When each * is treated as a 1, the matrix has the consecutive 1's property and 3 tracks are sufficient for a layout. Only the first and the last rows need to be on separate tracks. The rest of the rows can all share the same track.

Many heuristics for the GML problem (for example [Li],[Wi2]) build the final gate sequence by iteratively appending a new gate to a partially constructed sequence. The choice of the new gate is based on some greedy criterion such as minimizing the intersection between the current sequence and the new candidate. If the matrix has n rows, then such a greedy heuristic would obtain a layout that requires $n-1$ tracks. Clearly, the number of tracks for such a layout is not bounded by any constant times

the optimal.

1	1	1	1		1	0	0		0
1	*	*	*		*	1	0		0
0	1	*	*		*	*	1		0
0	0	1	*		*	*	*		0
0	0	0	1	···	*	*	*	···	0
		⋮							
0	0	0	0		1	*	*		1
0	0	0	0		0	1	1		1

Figure 3.15: The layout found by a greedy heuristic

The layout in Figure 3.15 is the result of the column permutation found by such a greedy heuristic. Each * in this matrix is treated as a 1 in order to obtain the consecutive 1's property.

Algorithm SR1 produces a 3-track layout for the matrix of Figure 3.14 for all n. Only those locations in the matrix with a * are changed to 1 during self-reduction thereby resulting in the consecutive 1's property. The number of tracks is independent of the initial column permutation.

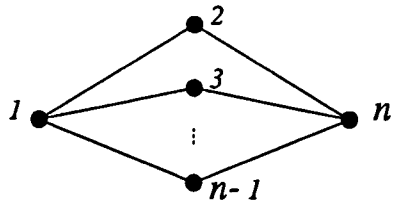


Figure 3.16: The intersection graph for the worst-case matrix

The intersection graph for a matrix of n rows, shown in Figure 3.16, does not contain any of the six obstructions detected by MINOR-TEST. Moreover, the only edge that is added during self-reduction by algorithm SR1 is between vertices 1 and n. Adding any other edge would introduce a K_4 minor in the graph.

3.6.3 Even Better Algorithms

There are at least two ways to make the self-reduction based layout algorithm faster. One way is to improve the decision algorithm itself. Since our imperfect decision algorithm already requires only $O(n)$ time and is very fast in practice, there is very little room for improvement here. The other way is to make fewer calls to the decision algorithm. Taking the latter approach, [FL4] describes a general method called *scaffolding* that only makes $O(n \log n)$ calls to the decision algorithm in contrast with the $O(nm)$ calls made by SELF-REDUCE. In order to make fewer calls to the decision algorithm, this method transforms a problem with parameter k into a problem with parameter k' where k' is generally larger than k. In other words, scaffolding requires a more powerful decision algorithm and hence is currently impractical. Moreover, scaffolding only reduces the time complexity of the layout algorithm. It does not guarantee better results when we only have an imperfect decision algorithm.

3.7 Some Practical Considerations

In this section, we consider some of the common issues that arise in the practical implementation of a CAD tool for gate matrix layout, and discuss how they can be addressed using our approach.

3.7.1 Circuit Representation

Our first concern is the impact of the method used by an algorithm to represent the circuit, on the quality of the final layout produced. Most design automation tools for VLSI use a description of the circuit in a *net list* format. However, the term *net* is very loosely defined and tends to vary depending on the application. For gate matrix layout, a net is traditionally taken to mean a node in the circuit i.e. a junction of two or more transistor terminals. Since each transistor is controlled by a gate, each net in a circuit is associated with the gates corresponding to the transistors incident on

the net. Under this convention, a net list is simply a listing of each net in the circuit together with the set of gates associated with it.

Consider a CMOS circuit with two or more transistors connected in series. The function of the circuit remains the same even if the order of these transistors is changed. If we assume that the position of each transistor in a schematic diagram is fixed, then the net list will not reflect the flexibility inherent in the circuit. Since each initial ordering of these transistors will give us a different net list, the mapping from the logic representation of the circuit to a net list representation is one-to-many. This is particularly troublesome for gate matrix layout because the number of tracks required depends on the gate permutation that we choose. It has been observed that the initial ordering of the transistors can significantly affect the final gate matrix layout ([WHW], [HFK]).

3.7.2 Transistor Reordering

Suppose a set of two or more transistors in a circuit is connected in series. The function of the circuit is independent of the relative ordering of these transistors. Therefore, a maximal set of series connected transistors can be represented by a single net that does not impose any inital ordering of the gates. It is sufficient to specify the external nets that connect to the end transistors of the series net. The notion of *dynamic nets* introduced in [HFK] represents series connected transistors in this manner. For our algorithm, we require that the gate signals of the end-transistors that connect to the external nets be bound initially. Under this requirement, the incidence matrix would have a 1 in the column corresponding to the gates of the two end-transistors for the series net and the external nets. Besides having the freedom to rearrange the internal transistors, we can also redefine the end-transistors to our convenience after finding a column permutation.

Consider the circuit in Figure 3.1 on page 13. Notice that net N_2 consists of three

transistors in series, with gate C incident on both N_2 and N_3. The gate sequence $A-B-C-D-E-F-Z$ gives us a 3-track layout. If we redefine the end-transistor of N_2 to be E, then N_3 would connect to E rather than to C. This would reduce the number of tracks to two because now N_2 and N_3 can share a track. In this case, redefining the end-transistor of N_2 is equivalent to reordering the transistors in net N_2 so that the external net N_3 is connected to E.

Recall that the incidence matrix is simply a 0-1 matrix that shows on which gates each net is incident. A 1 entry in the incidence matrix could denote either a transistor or a via, and we have no way to distinguish between the two. For instance, in Figure 3.1, even though both N_2 and N_3 are incident on gate C, only N_2 has a transistor on gate C. Therefore, in order to take advantage of transistor reordering and end-transistor redefining possibilities, we need more information than just the incidence matrix or the intersection graph.

Carrying the idea of rearrangement beyond just individual transistors, we could also reorder larger portions of a circuit without modifying its function. In an attempt to take advantage of these possibilities, a recently published heuristic [SC] uses logic equations as input and synthesizes a CMOS cell in gate matrix layout style to realize the function. However, it is not at all clear when such a rearrangement would be beneficial. We show how such rearrangements can be performed to advantage during net merging, after a gate sequence has been found.

3.7.3 Net Merging

In a layout, any series net will have a transistor on each gate on which it is incident. The external net of a series net will only be connected to an end-transistor by a vertical diffusion line. This presents an opportunity to decrease the number of tracks in the final layout. After we find a column permutation, we can merge the external net with the series net whenever possible. Besides making the layout more compact,

this also eliminates the need to use a vertical diffusion run between the two nets had they been assigned to different tracks. This process is known as *net merging* ([WHW], [HFK], [SWK] etc.).

Let N_i be a series net and N_j be a parallel net and let N_i and N_j be connected in series. Suppose that in the final gate sequence all the gates of N_j occur entirely between two consecutive gates of N_i. Then we can merge N_i and N_j because this would not affect the function of the circuit and would be equivalent to reordering two series connected portions of the circuit as discussed in [SC]. This merging also makes the layout more compact.

3.7.4 Track Assignment

Typically in a gate matrix layout the polysilicon gates are evenly spaced. Since a transistor is formed at each intersection of a diffusion line and a polysilicon line, when used to connect different rows together a diffusion line must run parallel to the gates and in between them. Sometimes, there might be some intervening rows between two rows to be connected with vertical diffusion. In such a case, we must ensure that there is sufficient space between the polysilicon lines so that the diffusion line is not too close to a transistor in an intervening row. This is easily accomplished by introducing additional space between the two polysilicon gates. Since the number of columns in the gate matrix usually far exceeds the number of rows, extra space between some of the columns does not significantly affect the total area of the layout.

If more than one vertical diffusion line must share the channel between two gates then we must route the lines so that they do not intersect. In order to accomplish this, we might have to permute the rows of the matrix. The problem of permuting the rows in order to realize the layout is mentioned in [WHW] and heuristics are developed. However, in some rare cases, there may be no row permutation that allows us to realize all the interconnections between rows for a given column permutation.

After the gate sequence has been determined, most algorithms complete the track assignment using the greedy method of [HS]. This method is optimal in terms of the number of tracks it uses for a given ordering of the gates. However, it does not take into account the feasibility of routing the vertical diffusion lines. In [HW2], a heuristic for track assignment is presented that is claimed to be better at realizing the vertical diffusion lines.

3.7.5 Fully Complementary CMOS Circuits

A CMOS circuit consists of a pMOS portion and an nMOS portion, and they are laid out in distinct portions of the chip. In fully complementary CMOS, the p and n portions are duals of each other. When dealing with fully complementary CMOS circuits, many algorithms for gate matrix layout consider only the pMOS or the nMOS portion of the circuit. These algorithms assume that a gate permutation that minimizes the number of tracks for either one would also minimize the cost of the entire layout. As pointed out in [CH], this assumption is not always true.

We observe that, if each net has a layout in exactly one track, then due to the complementary nature of the p and n portions of the circuit, it is sufficient to consider either the pMOS or the nMOS circuit alone. In order to ensure that each net fits on one track, it may be necessary for some tracks to be wider than usual because additional space might be required to interconnect the transistors in a net. If the p and n parts are not fully complementary, then optimizing one alone may not be sufficient. In this case, if we find separate layouts for the p and n parts then they might have different gate sequences and additional routing will be required to connect the gates of the two parts together.

Considering the entire circuit at once is the ideal way to obtain an optimum layout. However, besides increasing the size of the problem, this further complicates many other issues. Since the p and n wells are located in separate areas of the chip, the

p transistors cannot share a track with n transistors. Therefore, we would have to distinguish between the nets for the pMOS part and the nets for the nMOS part and the track assignment for each part should be done separately. Furthermore, the cost function used by the optimization algorithm would need modification. If we use the incidence matrix representation, then the total number of tracks required would be the sum of the maximum column sum for the p nets and the maximum column sum for n nets.

3.7.6 Other Issues

The routing of power and ground connections is an important step of the layout process. Besides the primary metal and diffusion layers, there is also a second metal layer available for this purpose. Still, the use of vias for power or ground connections can cause the layout area to increase slightly (see [HW2]).

In a large circuit, most of the gates tend to be incident on only a small fraction of the nets. This has motivated the idea of compacting the layout by folding the gates whereby, more than one gate can occupy the same column of the matrix. An algorithm for gate matrix folding based on simulated annealing is presented in [DN]. Heuristics for partitioning a circuit into blocks to layout the individual blocks separately are explored in [HW1].

3.8 Summary

The design automation of gate matrix layout is a multi-stage process. In this chapter, we addressed the most crucial step which is the determination of a good gate sequence. We developed novel algorithms for the 3-track layout problem and presented experimental data in support of its utility. From the discussion on some of the practical considerations that ensued, it is clear that several further optimizations are possible and there are also additional constraints that must be met. One of the

ways to use our algorithms for gate matrix layout with more than 3 tracks would be to partition a large circuit into smaller functional cells and combine the layouts of the constituent cells in order to obtain the final layout of the large circuit.

The main intent of this chapter was to illustrate this general approach to algorithm design. Our approach is of general interest both for gate matrix layout and for other problems in VLSI design. In the chapters that follow, we develop new tools that estimate the number of tracks required for a layout and also help to extend the methods used in this chapter.

Chapter 4

Quickly Estimating Pathwidth

We know that it is difficult to determine the minimum number of tracks required for a gate matrix layout of a circuit. Can we at least quickly estimate the number of tracks necessary?

In the previous chapter, we focussed on fixed-parameter gate matrix layout mainly due to the intractability of the minimization problem. This raises two issues. What is a good choice of the value of the parameter? What do we do when the circuit does not have a layout in k-tracks? Good estimates of the number of tracks would be helpful in dealing with these two questions.

The theme of this chapter is lower bounds for the pathwidth of a graph. We start with several lower bounds for treewidth, which in turn are lower bounds for pathwidth. In light of the relation between $\mathrm{PW}(k-1)$ and $\mathrm{GML}(k)$, these bounds amount to an estimate of the number of tracks necessary for a gate matrix layout. We also present efficient algorithms to compute these lower bounds whenever possible. Previously, every lower bound known for pathwidth was also a bound for treewidth. In this chapter, we present a stronger lower bound for pathwidth that does not hold for treewidth.

We also present some upper bounds for pathwidth. Unlike the lower bounds, it appears to be much more difficult to obtain good upper bounds for pathwidth.

4.1 Smooth Decompositions

We use a special kind of tree-decomposition called a *smooth* decomposition [Bod1]. A similar notion was independently developed by Yan for path decompositions [Ya].

Definition [Bod1] A tree-decomposition (T, Y) of width k is *smooth* if
(i) for all $i \in V(T)$, $|X_i| = k + 1$, and
(ii) if (i, j) is an edge in T then $|X_i - X_j| = |X_j - X_i| = 1$.

Since every path-decomposition is also a tree-decomposition, the notion of *smooth* path-decomposition is defined similarly.

As shown in [Bod1, Ya], any tree-decomposition can be easily transformed into a smooth tree-decomposition without affecting the width of the decomposition. We prove some elementary properties of smooth decompositions. Let $G = (V, E)$ be a graph of order n and size e.

Lemma 4.1.1 [Bod1] If (T, Y) is a smooth tree-decomposition of width k for graph G, then $|V(T)| = n - k$.

Corollary 4.1.2 If (T, Y) is a smooth path-decomposition of width k for graph G, then the length of the decomposition is $n - k - 1$.

Due to their regular structure, smooth decompositions help us better comprehend the structure of tree-decompositions and also simplify many of our proofs.

4.2 Bounds for Pathwidth and Treewidth

We now present several lower bounds for pathwidth and treewidth.

4.2.1 The Impact of Complete Graphs

The presence of a complete graph affects the pathwidth and treewidth significantly. Our first lower bound for treewidth is based on complete graphs.

Lemma 4.2.1 [RS2] If $K_k \leq_m G$, then $treewidth(G) \geq k-1$.

Proof By definition, $treewidth(K_k) = k-1$. It is known that if $H \leq_m G$, then $treewidth(H) \leq treewidth(G)$. Therefore, $treewidth(G) \geq k-1$. ∎

The following lemma, which is a complement of Lemma 4.2.1, provides an upper bound for pathwidth.

Lemma 4.2.2 If $K_k \subseteq \bar{G}$, then $pathwidth(G) \leq n-k$.

Proof Let the vertices $v_1, v_2, \ldots, v_k$ form a K_k in $\bar{G}$. Then, the following is a smooth path-decomposition of width $n-k$ for G: $X_i = \{v_i, v_j : k+1 \leq j \leq n\}, 1 \leq i \leq n-k$. ∎

For any fixed integer k, it can be determined in polynomial time whether $K_k \leq_m G$ or whether $K_k \subseteq G$. However, it is $\mathcal{NP}$-Hard to determine the order of the largest complete subgraph of a given graph [GJ].

The complementary nature of Lemma 4.2.1 and Lemma 4.2.2 reminds us of Ramsey numbers, which are defined by the following theorem.

Theorem 4.2.3 [Ram] Given two positive integers x and y, there exists a smallest integer $r(x,y)$, called a *Ramsey number*, such that for every graph of order at least $r(x,y)$, either $K_x \subseteq G$ or $K_y \subseteq \bar{G}$.

If the order of a graph exceeds $r(x,y)$ for some x and y, then we know that either $x-1 \leq treewidth(G)$, or $pathwidth(G) \leq n-y$. For example, it is known [Rad] that $r(4,4) = 18$ and $r(4,5) = 25$. Consider a series-parallel graph G of order n. By definition, G does not contain K_4 as a subgraph. Therefore, if $n \geq 18$, then $K_4 \subseteq \bar{G}$, and as a consequence $pathwidth(G) \leq n-4$. Similarly, if $n \geq 25$, then $K_5 \subseteq \bar{G}$, and $pathwidth(G) \leq n-5$.

Unfortunately, it is not known how to compute the Ramsey numbers in general and only a few small cases have been solved using computers. In fact, even the value

of $r(4,5)$ was determined only very recently and the exact value of $r(5,5)$ is currently unknown [Rad].

4.2.2 The Number of Edges as a Limiting Factor

We now present some lower bounds for the treewidth of a graph based on the number of edges in the graph.

Lemma 4.2.4 [Bod1] If $treewidth(G) = k$ then $e \leq nk - \frac{1}{2}k(k+1)$.

The following corollary shows that half the average degree of a graph is a lower bound on the treewidth.

Corollary 4.2.5 For any graph G, $treewidth(G) \geq \frac{e+1}{n}$.

Proof Let $w = treewidth(G)$. Then according to Lemma 4.2.4, $e \leq nw - \frac{w(w+1)}{2}$. Since $w \geq 1$, we know that $\frac{1}{2}w(w+1) \geq 1$. Therefore, $e \leq nw - 1$ or $w \geq \frac{e+1}{n}$. ∎

This lower bound is tight for all graphs of the form $K_{1,n-1}$.

The following lemma, which gives a more refined lower bound for treewidth, was originally proven by Yan [Ya] for pathwidth. We observe that it holds for treewidth as well.

Lemma 4.2.6 For any graph G,

$$treewidth(G) \geq \frac{2n - 1 - \sqrt{(2n-1)^2 - 8e}}{2}$$

Proof Let $treewidth(G) = k$. We solve the inequality of Lemma 4.2.4, with the condition that $k \leq n-1$. The inequality of Lemma 4.2.4 can be written as $k^2 - (2n-1)k + 2e \leq 0$. The general solution of this quadratic equation is $\frac{2n-1 \pm \sqrt{(2n-1)^2-8e}}{2}$. Since $k \leq n-1$, we get $k \geq \frac{2n-1-\sqrt{(2n-1)^2-8e}}{2}$. ∎

4.3 The Metric γ of a Graph

In this section, we introduce a new metric γ of a graph, and prove that it is a lower bound for treewidth.

Lemma 4.3.1 If G is not a complete graph and $treewidth(G) = w$, then there exists a pair of non-adjacent vertices, each of degree at most w in G.

Proof Let (T, Y) be a smooth tree-decomposition of width w for G where $Y = \{X_1, \ldots, X_{n-w}\}$. Since $G \subset K_n$, $w \leq n-2$ or $n-w \geq 2$. Let X_1 and X_{n-w} be two leaves of T. Let X_2 be the neighbor of X_1 and let X_{n-w-1} be the neighbor of X_{n-w}. Also, let $\{v_1\} = X_1 - X_2$ and $\{v_2\} = X_{n-w} - X_{n-w-1}$. Since X_2 is on every path from X_1 to any vertex X_i in T and $v_1 \notin X_2$, we know that $v_1 \notin X_i$ for any $i > 1$. Similarly, $v_2 \notin X_i$ for any $i < n-2$. Therefore, $\forall i$, $|X_i \cap \{v_1, v_2\}| \leq 1$. Hence, $(v_1, v_2) \notin E(G)$. Moreover, $|X_1| = w+1 \Rightarrow \delta(v_1) \leq w$ and $|X_{n-w}| = w+1 \Rightarrow \delta(v_2) \leq w$. ∎

This lemma immediately motivates the definition of a new metric γ of a graph.

Definition If $G = (V, E)$ is a graph of order n, then

$$\gamma(G) = \min_{1 \leq i < j \leq n} \{n-1, \max\{\delta_i, \delta_j\} : (v_i, v_j) \notin E\}$$

4.3.1 γ as a Lower Bound for Treewidth

We show that γ is a lower bound for treewidth, and then compare this bound with the other known bounds.

Lemma 4.3.2 For any graph G, $treewidth(G) \geq \gamma(G)$.

Proof If G is a complete graph, then $\gamma(G) = n-1 = treewidth(G)$. Otherwise, let $w = treewidth(G)$ and let v_1, v_2 be a pair of non-adjacent vertices of degree $\leq w$ in G. Then $w \geq \max\{\delta_1, \delta_2\} \geq \gamma(G)$. ∎

Unlike the bound of Lemma 4.2.6 which is based on the total number of edges in the graph, γ is based on the neighborhood of individual vertices and hence reveals more useful structural information. There are families of graphs for which γ is the stronger of the two bounds.

For instance, if G is a complete bipartite graph of the form $K_{m,m}$, then $\gamma(G) = m$, whereas Lemma 4.2.6 gives a lower bound of $\lceil \frac{4m-1-\sqrt{8m^2-8m+1}}{2} \rceil$. In order to compare the two bounds, we solve the following inquality:

$$\begin{aligned}
\lceil \frac{4m-1-\sqrt{8m^2-8m+1}}{2} \rceil &\geq m \\
\frac{4m-1-\sqrt{8m^2-8m+1}}{2} &> m-1 \\
2m+1-\sqrt{8m^2-8m+1} &> 0 \\
(2m+1)^2 &> 8m^2-8m+1 \\
4m(m-3) &< 0 \\
0 < m &< 3
\end{aligned}$$

Therefore, for $m \geq 3$, γ is the greater of the two lower bounds. The following table lists the value of the function $\lceil \frac{4m-1-\sqrt{8m^2-8m+1}}{2} \rceil$ for some small values of m.

m	1	2	3	4	5	6	7	8	9	10	15	20	25
$\lceil \frac{4m-1-\sqrt{8m^2-8m+1}}{2} \rceil$	1	2	2	3	4	4	5	5	6	7	9	12	15

Even though in this case $\gamma(G) = \delta(G)$, as will be evident in the next chapter, in general $\gamma(G) \geq \delta(G)$. This example also shows that the bound given by Lemma 4.2.6 can even be less than δ. Unlike the minimum degree of a graph, the maximum degree seems to have no direct bearing on its treewidth or pathwidth. For instance, we can construct square grids of arbitrary treewidth or pathwidth. Simpler still are the binary trees which can have arbitrarily large pathwidth.

4.3.2 Computing γ in Linear Time

We present an algorithm that takes time linear in the size of the input graph to compute γ. First let us define γ_i for each vertex v_i.

Definition If $G = (V, E)$ is a graph of order n, then

$$\gamma_i = \min_{1 \leq j \leq n} \{n - 1, \max\{\delta_i, \delta_j\} : (v_i, v_j) \notin E\}$$

Note that $\gamma = \min_i\{\gamma_i\}$. The following lemma gives another characterization of γ_i which is used by our algorithm.

Lemma 4.3.3

$$\gamma_i = \begin{cases} \delta_i, & \text{if } \exists v_j \ni \delta_j \leq \delta_i, (v_i, v_j) \notin E \\ \min_j\{n - 1, \delta_j : \delta_j > \delta_i, (v_i, v_j) \notin E\} & \text{otherwise} \end{cases}$$

Proof For every v_j such that $\delta_j \leq \delta_i$ and $(v_i, v_j) \notin E$, $\max\{\delta_i, \delta_j\} = \delta_i$. Therefore, if such a vertex v_j exists, then $\gamma_i = \delta_i$. Otherwise, for every v_j such that $(v_i, v_j) \notin E$, $\max\{\delta_i, \delta_j\} = \delta_j$. ∎

Let $G = (V, E)$ be a graph of order n and $x_1, x_2, \ldots, x_n$ be a permutation of $1, 2, \ldots, n$ such that $\delta_{x_1} \leq \delta_{x_2} \leq \ldots \leq \delta_{x_n}$. From the definition of γ it is clear that $\gamma = \delta_{x_i}$ for some i. Our algorithm to compute $\gamma(G)$ is called **GAMMA** and is shown in Figure 4.1. It is based on the following two lemmas.

Lemma 4.3.4 $\gamma = \delta_{x_i}$ if and only if v_{x_i} is a vertex of least degree such that there exists a vertex v_{x_j} with $\delta_{x_j} \leq \delta_{x_i}$ and $(v_{x_j}, v_{x_i}) \notin E$.

Proof ($\Rightarrow$): This follows from the definition of γ.
($\Leftarrow$): It is clear that $\gamma_{x_i} = \delta_{x_i}$. Moreover, if $\delta_{x_j} = \delta_{x_i}$ then $\gamma_{x_j} = \delta_{x_j}$ else $\gamma_{x_j} = \delta_{x_i}$. In other words, $\gamma_{x_j} = \delta_{x_i}$. Also, $\gamma_{x_m} \geq \delta_{x_i}$ whenever $m \neq i$ or $m \neq j$. Therefore, $\gamma = \delta_{x_i}$. ∎

Algorithm GAMMA

Input : A graph G
Output: The integer metric $\gamma(G)$
begin procedure
1. Store the input graph G in uninitialized adjacency matrix form.
   Compute the degree of each vertex while reading the input.
2. Bucket sort the vertices in non-decreasing order of their degree.
   $\delta \leftarrow$ minimum degree;
3. **if** $(\delta = n - 1)$ **then** $\gamma \leftarrow n - 1$;
4. **else**
   begin
      Let $v_{x_1}, v_{x_2}, \ldots, v_{x_{\delta+2}}$ be the first $\delta + 2$ vertices in sorted order.
      found $\leftarrow$ FALSE;
      $i \leftarrow 2$;
      while $((i \leq \delta + 1)$ and not(found)) **do**
      begin
         $j \leftarrow 1$;
         while $((j \leq i - 1)$ and not(found)) **do**
            if edge$(v_{x_j}, v_{x_i}) \notin G$ **then** found $\leftarrow$ TRUE;
            else $j \leftarrow j + 1$;
         if (not(found)) **then** $i \leftarrow i + 1$;
      end;
      $\gamma \leftarrow \delta_{x_i}$;
   end;
5. Output γ and **stop.**
end procedure.

Figure 4.1: Algorithm GAMMA

Lemma 4.3.5 If $\gamma = \delta_{x_i}$, then $2 \leq i \leq \delta + 2$.

Proof Clearly $\delta_{x_1} = \delta$. Therefore, there exists a j, $2 \leq j \leq \delta + 2$ such that $(v_{x_1}, v_{x_j}) \notin E$. Since $\delta_{x_1} \leq \delta_{x_j}$, we have $\max\{\delta_{x_1}, \delta_{x_j}\} = \delta_{x_j}$. Therefore, $\gamma \leq \delta_{x_j}$ where $1 \leq j \leq \delta + 2$. But, we know that $\gamma = \delta_{x_i}$ for some $1 \leq i \leq n$. Consequently, we have $\gamma = \delta_{x_i} \leq \delta_{x_j}$ where $2 \leq i \leq j \leq \delta + 2 \leq n$. ∎

Lemma 4.3.4 tells us how to find γ. Lemma 4.3.5 narrows down the search space, and hence the execution time of the algorithm.

Theorem 4.3.6 If the input graph G has n vertices and e edges, then Algorithm

GAMMA computes $\gamma(G)$ in $O(n+e)$ time.

Proof Step 1 takes $O(n+e)$ time. Bucket sorting n vertices into $n-1$ buckets takes $O(n)$ time. The 'while' loops to compute γ will perform δ^2 iterations in the worst case. Since there exist at least $\delta+1$ vertices of degree $\geq \delta$, we have $\delta(\delta+1) \leq 2e$ which implies that $\delta^2 \leq 2e$. So, Algorithm GAMMA has an execution time of $O(n+e)$. ∎

4.4 Relating Pathwidth To Treewidth

We know that the treewidth of a graph never exceeds its pathwidth. But how closely are the two metrics related? The following lemma provides a relation between pathwidth and treewidth.

Lemma 4.4.1 [BGHK] If G is a graph of order n and $treewidth(G) = k$, then $pathwidth(G) \leq (k+1)\log n$.

Recently, Bodlaender and Kloks [BK2] have shown that if the treewidth of a graph is bounded, then its pathwidth can be computed in polynomial time.

Lemma 4.4.2 [BK2] For every fixed k, there exists a polynomial-time algorithm that when given a graph G of treewidth k, computes $pathwidth(G)$ and a path-decomposition of minimum width for G.

Unfortunately, this result is only of theoretical significance. Even for graphs of treewidth 2, the running time of this algorithm is very large.

Given a graph G whose treewidth and pathwidth are unknown, is it easy to decide whether $pathwidth(G) = treewidth(G)$? This question seems like a very hard one to settle. We present a sufficient condition for the pathwidth of a graph to be identical to its treewidth. This condition is based on the fact that if the treewidth is less than the pathwidth, then every tree-decomposition of minimum width has at least three leaves. First, we define a new metric β for a graph.

Definition If G is a graph of order n, then

$$\beta(G) = \min_{1 \leq i < j < k \leq n} \{n - 1, \max\{\delta_i, \delta_j, \delta_k\} : \{(v_i, v_j), (v_j, v_k), (v_i, v_k)\} \cap E = \phi\}$$

Lemma 4.4.3 For every graph G, $\gamma(G) \leq \beta(G)$.

Proof The proof follows from the definitions of β and γ, and the fact that

$$\{\max\{\delta_i, \delta_j, \delta_k\} : \{(v_i, v_j), (v_j, v_k), (v_i, v_k)\} \cap E = \phi\} \subseteq \{\max\{\delta_i, \delta_j\} : (v_i, v_j) \notin E\}$$

∎

Lemma 4.4.4 If $\beta(G) > treewidth(G)$, then $pathwidth(G) = treewidth(G)$.

Proof Let $w = treewidth(G)$ and $\beta > w$. Suppose $pathwidth(G) > w$. Then every smooth tree-decomposition (T, Y) of width w for G has at least three leaves, say X_1, X_2, and X_3, with neighbors X_i, X_j, and X_k respectively. Let $v_1 \in X_1 - X_i$, $v_2 \in X_2 - X_j$, and $v_3 \in X_3 - X_k$. Then $\max\{\delta_1, \delta_2, \delta_3\} \leq w$. But this implies that $\beta \leq w$ which is a contradiction. ∎

Observe that all the lower bounds for pathwidth that we have seen so far were actually lower bounds for treewidth. Therefore, when the treewidth of a graph is already known, these bounds do not provide any new information. In the next section, we present for the first time, lower bounds for pathwidth that are independent of treewidth.

4.5 The Role of Complete Bipartite Graphs

In this section, we show that any graph of bounded treewidth can be characterized by a special kind of connected subgraph in its complement. This leads to an even stronger characterization of pathwidth, which we use to derive some lower bounds for pathwidth that are independent of treewidth.

4.5.1 A New Characterization of Treewidth and Pathwidth

Definition A *bipartite graph* is a graph $G = (V, E)$ such that, $V = V_1 \cup V_2$, $V_1 \cap V_2 = \phi$, and $E \subseteq \{(v_i, v_j) : v_i \in V_1, v_j \in V_2\}$.

If $E = \{(v_i, v_j) : v_i \in V_1, v_j \in V_2\}$ then G is called a *complete bipartite graph.* The complete bipartite graph with $|V_1| = x$ and $|V_2| = y$ is denoted by $K_{x,y}$. When $|x - y| \leq 1$, $K_{x,y}$ is called a *balanced complete bipartite graph.*

The following lemma relates the length of a simple path in a smooth tree-decomposition of G to complete bipartite subgraphs of $\bar{G}$.

Lemma 4.5.1 If (T, Y) is a smooth tree-decomposition for a graph G and T has a simple path of length $l - 1$, then there exists a connected graph $H \subseteq \bar{G}$ such that,

i. for every $1 \leq x \leq y < l$ such that $x + y = l$ there exist $L \cong K_{x,y} \subseteq H$ and $R \cong K_{x,y} \subseteq H$, where L is identical to R only when $x = y$,

ii. $l \leq |V(H)| \leq 2(l-1)$,

iii. the maximum degree of a vertex in H is $l-1$ and there exist $2l-|V(H)|$ vertices of degree $l - 1$ in H, and

iv. there are at most two vertices of degree 1 in H.

Proof Let the sequence of vertices $1, 2, \ldots, l$ be a simple path of length $l - 1$ in the tree T. For all $1 \leq i < l$, let $\{u_i\} = X_i - X_{i+1}$, and $\{v_{i+1}\} = X_{i+1} - X_i$, where u_i and v_i need not necessarily be distinct (see Figure 4.2).

Clearly, the edge $(u_i, v_j) \notin G$ for any $1 \leq i < j \leq l$, and hence $(u_i, v_j) \in \bar{G}$. Consider the graph $H \subseteq \bar{G}$ defined as follows: $V(H) = \{u_i, 1 \leq i < l\} \cup \{v_j : 1 < j \leq l\}$ and $E(H) = \{(u_i, v_j) : 1 \leq i < j \leq l\}$. H is a connected graph because $\forall j > 1, (u_1, v_j) \in E(H)$ and $\forall i < l, (u_i, v_l) \in H$.

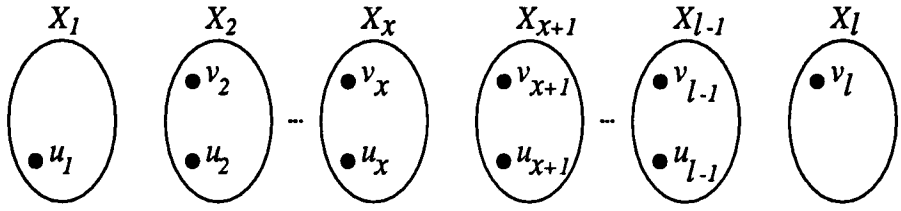


Figure 4.2: A path in a tree-decomposition

i. For each $1 \leq x \leq y < l$ such that $x + y = l$, $L \cong K_{x,y} \subseteq H$, where $V(L) = L_1 \cup L_2$ and $E(L) = L_1 \times L_2$, with $L_1 = \{u_i : 1 \leq i \leq x\}$ and $L_2 = \{v_i : x + 1 \leq i \leq l\}$. Similarly, $R \cong K_{x,y} \subseteq H$, where $V(R) = R_1 \cup R_2$ and $E(R) = R_1 \times R_2$, with $R_1 = \{v_i : y + 1 \leq i \leq l\}$ and $R_2 = \{u_i : 1 \leq i \leq y\}$. When $x = y$, $L_1 = R_2$ and $L_2 = R_1$ and as a result, L is identical to R. If $x < y$ then the edge (u_x, v_{x+1}) is in L but not in R, and consequently L is distinct from R.

ii. Since $K_{x,y} \subseteq H$ for all $x + y = l$, we know that $l \leq |V(H)|$. We can write

$$\begin{aligned} |V(H)| &= |\{u_i : 1 \leq i \leq l-1\} \cup \{v_i : 2 \leq i \leq l\}| \\ &\leq |\{u_i : 1 \leq i \leq l-1\}| + |\{v_i : 2 \leq i \leq l\}| \end{aligned}$$

Therefore, $l \leq |V(H)| \leq 2(l-1)$.

iii. We know that $\delta(u_1) = \delta(v_l) = l-1$. Also, $\delta(u_i)+\delta(v_i) = l-1$ for every $2 \leq i \leq l-1$. Therefore, the maximum degree of H is $l-1$. Moreover, if H has less than $2(l-1)$ vertices, it is because $u_i = v_i$ for some $1 < i < l$, which implies that $\delta(u_i) = l-1$. Hence, the number of vertices of degree $l-1$ in H is $2(l-1)-|V(H)|+2 = 2l-|V(H)|$.

iv. Notice that only v_2 and u_{l-1} can have degree 1. In fact, $\delta(v_2) = 1 \Leftrightarrow u_2 \neq v_2$, and $\delta(u_{l-1}) = 1 \Leftrightarrow u_{l-1} \neq v_{l-1}$. ∎

This lemma showed that the order of H might be as large as $2(l-1)$ in order to contain two distinct copies of each $K_{x,y}$ when $x \neq y$. Suppose we are only interested in the existence of these subgraphs and not it their number. How large should H

be for it to contain just one copy of each $K_{x,y}$? The following lemma answers this question.

Lemma 4.5.2 If (T, Y) is a smooth tree-decomposition for a graph G and T has a simple path of length $l-1$, then there exists a connected graph $H \subseteq \bar{G}$ such that,

i. for every $1 \leq x \leq y < l$ such that $x + y = l$ there exists a $K_{x,y} \subseteq H$,

ii. $l \leq |V(H)| \leq \lfloor \frac{3l}{2} \rfloor - 1$,

iii. the maximum degree of a vertex in H is $l-1$ and there exist $\lfloor \frac{3l}{2} \rfloor - |V(H)|$ vertices of degree $l-1$ in H, and

iv. there is at most one vertex of degree 1 in H.

Proof This proof is an adaptation of the proof Lemma 4.5.1. Consider the graph $H \subseteq \bar{G}$ defined as follows: $V(H) = \{u_i : 1 \leq i \leq \lfloor \frac{l}{2} \rfloor\} \cup \{v_j : 1 < j \leq l\}$ and $E(H) = \{(u_i, v_j) : 1 \leq i \leq \lfloor \frac{l}{2} \rfloor, 1 < j \leq l\}$. Here again, the u_i's and v_i's are from X_i as shown in Figure 4.2. The graph H is shown in Figure 4.3. H is a connected graph because $\forall j > 1, (u_1, v_j) \in E(H)$, and $\forall i \leq \lfloor \frac{l}{2} \rfloor, (u_i, v_l) \in H$.

i. In this case, for every $1 \leq x \leq y < l$ with $x + y = l$, $B \cong K_{x,y} \subseteq H$, where $V(B) = B_1 \cup B_2$ with $B_1 = \{u_i : 1 \leq i \leq x\}$ and $B_2 = \{v_j : x+1 \leq j \leq x+y\}$, and $E(B) = \{(u_i, v_j) : u_i \in B_1, v_j \in B_2$.

ii. Since $K_{x,y} \subseteq H$ for all $x + y = l$, we know that $l \leq |V(H)|$. We can write

$$\begin{aligned} |V(H)| &= |\{u_i : 1 \leq i \leq \lfloor \frac{l}{2} \rfloor\} \cup \{v_i : 2 \leq i \leq l\}| \\ &\leq |\{u_i : 1 \leq i \leq \lfloor \frac{l}{2} \rfloor\}| + |\{v_i : 2 \leq i \leq l\}| \end{aligned}$$

Therefore, $l \leq |V(H)| \leq \lfloor \frac{3l}{2} \rfloor - 1$.

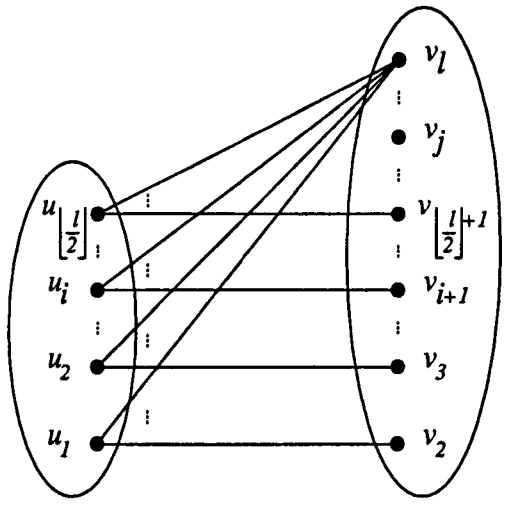


Figure 4.3: The connected graph H

iii. Notice that $\delta(u_1) = l - 1$. Moreover, for every $2 \leq i \leq \lfloor \frac{l}{2} \rfloor$, $\delta(u_i) = l - i$, $\delta(v_i) = i - 1$, and hence $\delta(u_i) + \delta(v_i) = l - 1$. Also, for every $\lfloor \frac{l}{2} \rfloor + 1 \leq j \leq l$, $\delta(v_j) \leq \lfloor \frac{l}{2} \rfloor$. As a result, the maximum degree of H is $l - 1$ and the number of vertices of degree $l - 1$ is $\lfloor \frac{3l}{2} \rfloor - 1 - |V(H) + 1 = \lfloor \frac{l}{2} \rfloor - |V(H)|$.

iv. It is clear that only v_2 can have degree 1, and that happens only when $u_2 \neq v_2$. ∎

The following corollary is the restriction of Lemma 4.5.1 to path-decompositions.

Corollary 4.5.3 If $pathwidth(G) = w$, then there exists a connected graph $H \subseteq \bar{G}$ such that,

i. for every $1 \leq x \leq y < n - w$ such that $x + y = n - w$ there exist $L \cong K_{x,y} \subseteq H$ and $R \cong K_{x,y} \subseteq H$, where L is identical to R only when $x = y$,

ii. $n - w \leq |V(H)| \leq 2(n - w - 1)$,

iii. the maximum degree of a vertex in H is $n - w - 1$, and there exist $2(n - w) - |V(H)|$ vertices of degree $n - w - 1$ in H, and

iv. there are at most two vertices of degree 1 in H.

Proof According to Corollary 4.1.2, any smooth path-decomposition of width w for G has length $n - w - 1$. Therefore, by substituting $n - w$ for l in Lemma 4.5.1, we get this result. ∎

Applying Lemma 4.5.2 to smooth path-decompositions, we get the following corollary.

Corollary 4.5.4 If $pathwidth(G) = w$, then there exists a connected graph $H \subseteq \bar{G}$ such that,

i. for every $1 \leq x \leq y < n - w$, $x + y = n - w \Rightarrow K_{x,y} \subseteq H$,

ii. $n - w \leq |V(H)| \leq \lfloor \frac{3(n-w)}{2} \rfloor - 1$,

iii. the maximum degree of a vertex in H is $n - w - 1$ and there exist $\lfloor \frac{3(n-w)}{2} \rfloor - |V(H)|$ vertices of degree $n - w - 1$ in H, and

iv. there is at most one vertex with degree 1 in H.

Proof We simply substitute $n - w$ for l in Lemma 4.5.2 to get this result. ∎

We shall show that the characterization of pathwidth given by Corollary 4.5.4 can be used to derive better bounds for the pathwidth of graph. This can in turn help to estimate the number of tracks required for a layout.

The notion of graph separators have been used by some authors to study treewidth (for example, [Go, Re]). We remark that $K_{x,y} \subseteq \bar{G}$ if and only if G has a separator of size $|V(G)| - (x + y)$. However, we find that using complete bipartite graphs reveals more structural information.

The following conjecture is a converse of Corollary 4.5.3.

Conjecture 4.5.5 If there exists a connected subgraph H of $\bar{G}$ and a constant $l \geq 2$ such that H satisfies all four conditions of Lemma 4.5.1, then $pathwidth(G) \leq n - l$.

Towards the proof of this conjecture, notice that when $|V(H)| = l$, H has l vertices of degree $l-1$. This means that $H \cong K_l$, and according to Lemma 4.2.2, $pathwidth(G) \leq n - l$. Therefore, only the situation when $l < |V(H)| \leq 2(l-1)$ remains unresolved.

4.5.2 Lower Bounds Solely for Pathwidth

All the lower bounds for pathwidth that we have seen so far were actually lower bounds for treewidth. In this section for the first time, we present non-trivial bounds for pathwidth that do not always hold for treewidth.

Using Corollary 4.5.4, we can derive a lower bound for pathwidth as follows. Let m be the largest integer such that, for every $1 \leq x \leq y$ with $x + y = m$, $K_{x,y} \subseteq \bar{G}$. Then, as a consequence of Lemma 4.5.4, we have $n - w \leq m$, or $w \geq n - m$.

Corollary 4.5.4 implies that $\bar{G}$ contains several different complete bipartite subgraphs. We can derive weaker lower bounds for pathwidth based on any of them. For example, when $x = 1$, the largest complete bipartite graph $K_{x,y}$ has y equal to the maximum degree of a vertex in $\bar{G}$, and the lower bound is just the minimum degree $\delta(G)$. Using Corollary 4.5.3 we can even show that $pathwidth(G) \geq \gamma(G)$.

The following corollary presents a lower bound based on balanced complete bipartite graphs. It should be noted that this result does not hold for treewidth.

Corollary 4.5.6 If m is the order of the largest balanced complete bipartite subgraph of $\bar{G}$, then $pathwidth(G) \geq n - m$.

Proof From Corollary 4.5.4 it is clear that if $pathwidth(G) = w \leq n - 2$, then $K_{\lceil \frac{n-w}{2} \rceil, \lfloor \frac{n-w}{2} \rfloor} \subset \bar{G}$. Therefore, $n - w \leq m$ or $w \geq n - m$. ∎

How does this bound compare with the previously known bounds for pathwidth? There are families of graphs for which this lower bound is greater than γ, whereas the treewidth equals γ.

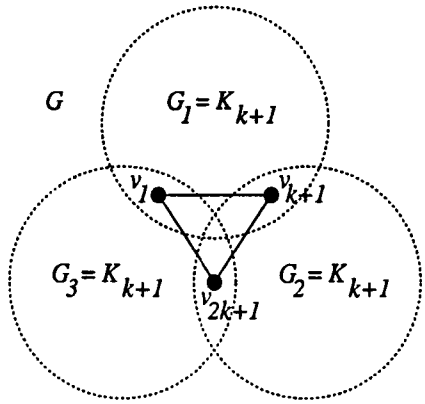


Figure 4.4: A graph with *pathwidth* > *treewidth* = γ

Consider the graph G of order $3k$ depicted in Figure 4.4. It comprises three subgraphs G_1, G_2, G_3 each isomorphic to K_{k+1}, and each pair of them shares a vertex. Let $V_1 = \{v_2, \ldots, v_k\}$, $V_2 = \{v_{k+2}, \ldots, v_{2k}\}$, and $V_3 = \{v_{2k+2}, \ldots, v_{3k}\}$. Then G_1 is induced by $V_1 \cup \{v_1, v_{k+1}\}$, G_2 is induced by $V_2 \cup \{v_{k+1}, v_{2k+1}\}$, and G_3 is induced by $V_3 \cup \{v_{2k+1}, v_1\}$. In G, $\delta_1 = \delta_{k+1} = \delta_{2k+1} = 2k$, and every other vertex has degree k.

Lemma 4.5.7 For $k \geq 2$, *treewidth*$(G) = k$, where G is the graph in Figure 4.4.

Proof Notice that $(v_2, v_{k+2}) \notin E(G)$ and $\delta_2 = \delta_{k+2} = k$. Therefore, $\gamma(G) \leq k$. But since $\delta(G) \leq \gamma(G)$ and $\delta(G) = k$ we have $\gamma(G) = k$. The following is a tree-decomposition of width k for G: $X_1 = V_1 \cup \{v_1, v_{k+1}\}$, $X_2 = V_2 \cup \{v_{k+1}, v_{2k+1}\}$, $X_3 = V_3 \cup \{v_1, v_{2k+1}\}$, and $X_4 = \{v_1, v_{k+1}, v_{2k+1}\}$. Vertices 1, 2, and 3 are leaves, and vertex 4 is an internal node of the tree. ∎

The next two lemmas show that *pathwidth*(G) > *treewidth*(G), and is in fact equal to the bound of Corollary 4.5.6.

Lemma 4.5.8 For any $k \geq 2$, $K_{k,k} \not\subseteq \bar{G}$, where G is the graph shown in Figure 4.4.

Proof The proof is by contradiction. Assume that $H \cong K_{k,k} \subseteq \bar{G}$. Let $V(H) = X \cup Y$, where $|X| = |Y| = k$ and $E(H) = \{(v_x, v_y) : v_x \in X, v_y \in Y\}$. H does not

include v_1, v_{k+1}, or v_{2k+1}, because in $\bar{G}$, $\delta_1 = \delta_{k+1} = \delta_{2k+1} = k-1$, and all other vertices have degree $2k-1$. $|V_1| = |V_2| = |V_3| = k-1$ implies that each of X and Y includes vertices from at least two different V_is. Since there are only three distinct V_is, there exists an i such that both X and Y contain a vertex from the same set V_i. Suppose $v_x \in X \cap V_i$ and $v_y \in Y \cap V_i$. But $(v_x, v_y) \notin E(\bar{G})$, because v_x and v_y belong to $G_i = K_{k+1} \subset G$. Therefore H is not a complete bipartite graph and this contradicts our initial assumption. ∎

Lemma 4.5.9 For any $k \geq 2$, $pathwidth(G) = k+1$, where G is the graph shown in Figure 4.4.

Proof Let $w = pathwidth(G)$. It follows from the previous lemma that the order of the largest balanced complete bipartite subgraph of $\bar{G}$ is at most $2k-1$. Using Corollary 4.5.6, we get $w \geq k+1$. The following is a path-decomposition of width $k+1$ for G: $X_1 = V_1 \cup \{v_1, v_{k+1}\}$, $X_2 = V_2 \cup \{v_{k+1}, v_{k+1}, v_{2k+1}\}$, and $X_3 = V_3 \cup \{v_1, v_{2k+1}\}$. ∎

From Lemmas 4.5.7 and 4.5.9 we conclude that the lower bound given by Corollary 4.5.6 can be better than γ.

The following result is the treewidth analog of Corollary 4.5.6.

Corollary 4.5.10 If m is the order of the largest balanced complete bipartite subgraph of $\bar{G}$, then the longest path in a smooth tree-decomposition of G is at most $m-1$.

Proof Let $l-1$ be the length of the longest path in the tree T over all smooth tree-decompositions (T, Y) of G. From Corollary 4.5.2 it is clear that $K_{\lfloor \frac{l}{2} \rfloor, \lceil \frac{l}{2} \rceil} \subset \bar{G}$. If $l > m$, then this would imply that the order of the largest balanced complete bipartite subgraph of G is more that m, which contradicts our basic premise. Therefore, $l \leq m$. ∎

Along the lines of Lemma 4.2.2, we show that a complete bipartite subgraph of $\bar{G}$ also gives an upper bound for $pathwidth(G)$.

Lemma 4.5.11 If $K_{x,y} \subseteq \bar{G}$, $x \leq y$, then $pathwidth(G) \leq n - x - 1$.

Proof Let $B = K_{x,y} \subseteq \bar{G}$, where $V(B) = B_1 \cup B_2$, $|B_1| = x$, $|B_2| = y$, and $E(B) = \{(v_i, v_j) : v_i \in B_1, v_j \in B_2\}$. Since $v_i \in B_1$ and $v_j \in B_2 \Rightarrow (v_i, v_j) \notin E(G)$, the following is a path-decomposition of width $n - x - 1$ for G: $X_1 = V(G) - B_1$, $X_2 = V(G) - B_2$. ∎

Note that Lemma 4.5.11 only says that $pathwidth(G) \leq n - \lfloor \frac{l}{2} \rfloor$, which is much weaker than Conjecture 4.5.5.

4.5.3 On Computing the Bounds

In order for a bound to be useful in practice, we should be able to compute it quickly. We address the practical feasibility of computing lower bounds for pathwidth based on Corollary 4.5.4. Given a graph G and a positive integer k, both the problem of determining whether G has a balanced complete bipartite subgraph of order $2k$, and the problem of determining whether G contains a complete bipartite subgraph of order k are known to be $\mathcal{NP}$-Complete [GJ]. Therefore, in general, computing a lower bound based on Corollary 4.5.4 does not seem to be any easier than determining the pathwidth itself.

For any fixed value of k however, it is possible to solve the complete bipartite subgraph problem in polynomial time. The algorithm is based on matrix multiplication and can be described as follows. Let A be the adjacency matrix for a graph G of order n. The product matrix $A \times A$ gives the number of vertex disjoint paths of length two between each pair of vertices in G. Therefore, the largest entry in $A \times A$ is the largest b such that $K_{2,b} \subseteq G$ and this can be computed in time $O(M(n))$, where $M(n)$ is the time required to multiply two $n \times n$ matrices. Currently, $M(n)$ is $O(n^{\log_2 7})$ using Strassen's method.

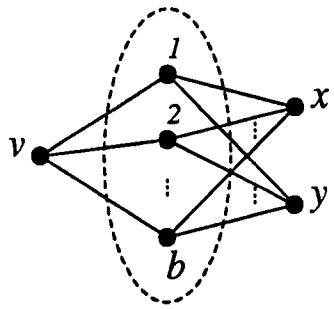


Figure 4.5: The complete bipartite graph $K_{3,b}$

Determining the largest b for which $K_{2,b} \subseteq \bar{G}$ is a practical way to obtain a lower bound on $pathwidth(G)$ because it only requires $O(M(n))$ time and the constant of proportionality is small. Given G, we first compute the adjacency matrix of G, take its complement, and then square the matrix. This algorithm can be extended to find a $K_{a,b}$ subgraph in time $O(n^{a-2}M(n))$ where $a \leq b$. We show how to find $K_{3,b}$.

Let v be a vertex in G. We simply check to see if there are b neighbors of v that are adjacent to the same two vertices x and y in $V(G) - \{v\}$. If this were true, then there exist b distinct paths of length two between v and x, and between v and y (see Figure 4.5). Moreover, the intermediate vertices on all these paths are in $N(v)$. So, we try to find all the paths of length two in G whose intermediate vertex belongs to $N(v)$. First, we determine the largest $K_{2,c}$ subgraph in the subgraph induced by $N(v)$. Next we consider the subgraph of G obtained by deleting v and every vertex and edge of G that is not adjacent to $N(v)$. Notice that this graph may include vertices not in $N(v)$, but every edge will have at least one endpoint in $N(v)$. For this graph, we use a matrix multiplication to find the maximum number d of disjoint paths of length two between a vertex not in $N(v)$ and another vertex. Since the intermediate vertices of the $K_{2,c}$ and $K_{2,d}$ are all in $N(v)$, we know that $K_{3,b} \subseteq G$, where $b = \max\{c, d\}$. The maximum b over all vertices v of G can be found in time $O(nM(n))$.

Even though this algorithm runs in polynomial time, using it to find $K_{a,b}$ for values of $a > 2$ quickly becomes impractical. The time complexity of matrix multiplication,

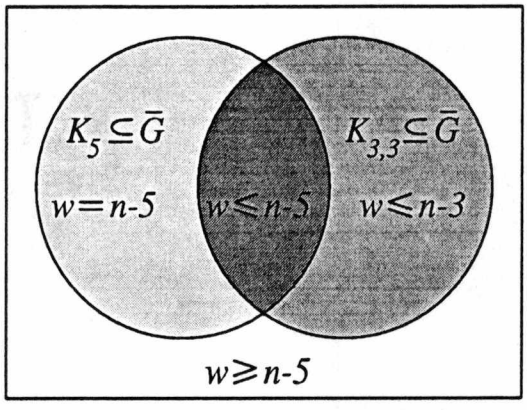


Figure 4.6: Bounds for pathwidth based on $K_{3,3}$

on which this algorithm is based, is an important factor. A faster algorithm to determine the largest graph $K_{2,b} \subseteq G$ would be a valuable tool to help speed-up these algorithms. Such an algorithm would probably have to be independent of matrix multiplication. For $5 \leq a \leq b$, the term n^{a-2} dominates the time complexity $O(n^{a-2}M(n))$ because $M(n) < n^3$. Therefore, better alternatives to exhaustive search are necessary.

For some special families of graphs, Corollary 4.5.4 does provide a practical way to obtain good bounds as shown in the following two Corollaries.

Corollary 4.5.12 If $\bar{G}$ is planar, then $pathwidth(G) \geq n - 5$.

Proof It is a well known fact that $\bar{G}$ is planar $\Rightarrow K_{3,3} \not\subseteq \bar{G}$. Therefore, $n - pathwidth(G) < 6$. ∎

Note that planarity is only a sufficient condition and is not necessary for a graph to exclude $K_{3,3}$. Currently, we do not know of an algorithm that is faster than the one based on matrix multiplication that we just described to decide whether a graph contains $K_{3,3}$ as a subgraph or not. However, there are linear-time algorithms for planarity testing.

It is interesting to compare Corollary 4.5.12 with Lemma 4.2.2. Their relationship is shown if Figure 4.6. If $K_5 \subseteq \bar{G}$, then $pathwidth(G) \leq n - 5$. Therefore, when

$K_{3,3} \not\subseteq \bar{G}$ and $K_5 \subseteq \bar{G}$, $pathwidth(G) = n - 5$. The only unresolved case is when both $K_{3,3} \subseteq \bar{G}$ and $K_5 \not\subseteq \bar{G}$.

How often does Corollary 4.5.12 help? The following lemma shows that if the order of G is at least eleven, then G must be non-planar in order for $\bar{G}$ to be planar.

Lemma 4.5.13 If $\bar{G}$ is planar and $|V(G)| \geq 11$, then G is non-planar.

Proof Euler's well known formula states that if G is planar then $e \leq 3n - 6$. In our case, $\bar{G}$ is planar. Therefore, $\bar{e} \leq 3n - 6$, and $e \geq \frac{1}{2}n(n-1) - (3n-6)$ i.e. $e \geq \frac{1}{2}n(n-7) + 6$. Clearly, for $n \geq 13$, $e \geq 3n + 6$. When $n = 11$, or $n = 12$, we can easily verify that $e > 3n - 6$. ∎

Corollary 4.5.14 If $\bar{G}$ is outer-planar, then $pathwidth(G) \geq n - 4$.

Proof $\bar{G}$ is outer-planar $\Rightarrow K_{2,3} \not\subseteq \bar{G}$. Therefore, $n - pathwidth(G) < 5$. ∎

Here again, outer-planarity is not a necessary condition for a graph to exclude $K_{2,3}$ as a subgraph. Since there exist linear-time algorithms to determine whether a graph is planar or outer-planar, the above two Corollaries can be used to obtain practical algorithms to decide whether a graph has pathwidth at most $n-5$ or $n-4$. Because we have to compute $\bar{G}$ first, and that can take $O(n^2)$ time, the final algorithms would also require $O(n^2)$ time.

4.6 Summary

In this chapter, we presented several new lower bounds for the treewidth of a graph (which in turn is a lower bound for the pathwidth), as well as lower bounds solely for pathwidth. We also found some new upper bounds for pathwidth. Table 4.1 gives a summary of the new lower bounds presented in this chapter and Table 4.2 lists all the non-trivial upper bounds known for pathwidth.

Table 4.1: Lower bounds for treewidth and pathwidth

Sufficient Condition	Lower bound	Proof
None	$treewidth(G) \geq \frac{e+1}{n}$	Corollary 4.2.5, page 65
None	$treewidth(G) \geq \frac{2n-1-\sqrt{(2n-1)^2-8e}}{2}$	Lemma 4.2.4, page 65
None	$treewidth(G) \geq \gamma(G)$	Lemma 4.3.2, page 66
$K_{x,y} \not\subseteq \bar{G}$	$pathwidth(G) \geq n-(x+y)+1$	Corollary 4.5.3, page 76

Table 4.2: Upper bounds for pathwidth

Sufficient Condition	Upper bound	Proof
$K_x \subseteq \bar{G}$	$pathwidth(G) \leq n-x$	Lemma 4.2.2, page 64
$K_{x,y} \subseteq \bar{G}, x \leq y$	$pathwidth(G) \leq n-x-1$	Lemma 4.5.11, page 80

Although some of the lower bounds can be computed quickly, all the upper bounds are impractical at present. In general, we find it more difficult to derive upper bounds than lower bounds. It should be pointed out that we do not know of any non-trivial upper bound for treewidth that does not also hold for pathwidth. Even if such a bound can be contrived, it is likely to be very difficult to compute in practice.

One way to obtain better estimates of the treewidth and pathwidth of a graph would be to find an operation that produces a proper minor with a greater lower bound than the original graph. Such an operation might be specific to the lower bound used and it is not clear how to find one. In the next chapter, we shall use some

of the bounds that we have developed here, to study the structure of obstructions for pathwidth and treewidth.

Chapter 5

Dense Obstructions for Treewidth and Pathwidth

In this chapter, we are interested in studying the structure and number of dense obstructions for PW(k). Earlier, we presented an algorithm for PW(2) based on minor-containment tests for a few dense obstructions. Since pathwidth is an upper bound on treewidth, a graph whose pathwidth is k cannot contain any of the TW(k) obstructions either. This leads us to explore the intersection of the obstruction sets for treewidth and pathwidth.

General methods to obtain some of the obstructions for PW($k+1$) by connecting together copies of obstructions for PW(k) or PW($k-1$) were presented in [KL]. The obstructions thus constructed have at least $3k+3$ vertices and are thus relatively large and sparse. In contrast, very little is known about treewidth obstructions in general. Besides the complete graph, the only non-trivial treewidth obstructions previously known were the three obstructions for TW(3) (see [APC],[ST]). Lagergren [Lag] has found a weak upper bound on the number of edges in an obstruction for TW(k). This bound is triple exponential in k^4 and is purely of theoretical significance.

We present the first general method to construct explicitly the most dense obstructions (besides the cliques) for pathwidth and treewidth. Interestingly, these graphs of order $k+3$ are obstructions for both PW(k) and TW(k) and were not previously

known to be obstructions for pathwidth. Moreover, this is the first general method to construct non-trivial obstructions to TW(k). Even though these new obstructions are very small and dense, we show that their number is an exponential function dependent on k, unlike the solitary clique. This also provides the first non-trivial lower bound for the number of obstructions for TW(k). The numerousness of even these small dense obstructions suggests that testing for them individually might be infeasible. Besides having a very rich intersection, we find new evidence that the obstruction sets for PW(k) and TW(k) also have significant differences.

5.1 Obstructions Common to TW(k) and PW(k)

The complete graph K_{k+2} is the smallest, densest, and the only graph previously known to be an obstruction both for PW(k) and TW(k). Moreover, it is the only general treewidth obstruction previously known. The next smallest obstructions should have order $n = k + 3$ and width $k + 1 = n - 2$, and using the metric γ we help shed new light on their structure. We start with a necessary and sufficient condition, in terms of γ, for pathwidth to be $n - 2$.

Lemma 5.1.1 *pathwidth*$(G) = n - 2$ if and only if $\gamma(G) = n - 2$.

Proof ($\Rightarrow$): $\gamma(G) = n - 2 \Rightarrow n - 2 \leq$ *pathwidth*(G) and $G \subset K_n$. But, $G \subset K_n \Rightarrow$ *pathwidth*$(G) < n - 1$. Therefore, *pathwidth*$(G) = n - 2$.
($\Leftarrow$): *pathwidth*$(G) = n - 2 \Rightarrow \gamma(G) \leq n - 2$. Suppose $\gamma(G) < n - 2$. Then there exist two vertices v_1 and v_2 in $V(G)$ such that the edge $(v_1, v_2) \notin E(G), \delta_1 < n - 2$, and $\delta_2 < n - 2$. $\delta_1 < n - 2$ implies that there exists $v_x \in V(G)$ such that $(v_1, v_x) \notin E(G)$. Similarly, there exists v_y (x may be equal to y) such that $(v_2, v_y) \notin E(G)$. Then the following is a smooth path decomposition of width $n - 3$ for G: $X_1 = V - \{v_2, v_x\}$, $X_2 = V - \{v_1, v_2\}$ and $X_3 = V - \{v_1, v_y\}$. This contradicts the fact that *pathwidth*$(G) = n - 2$. ∎

The following corollary is the treewidth analog of Lemma 5.1.1.

Corollary 5.1.2 *treewidth*$(G) = n - 2$ if and only if $\gamma(G) = n - 2$.

Proof If $\gamma = n - 2$, then from Lemma 5.1.1, *pathwidth*$(G) = n - 2$. Since $\gamma \leq$ *treewidth*$(G) \leq$ *pathwidth*(G), we get *treewidth*$(G) = n-2$. If *treewidth*$(G) = n-2$, then $G \subset K_n$, and $n - 2 \leq$ *pathwidth*$(G) < n - 1$. It follows from Lemma 5.1.1 that $\gamma = n - 2$. ∎

Corollary 5.1.3 *pathwidth*$(G) < n - 2$ if and only if $\gamma(G) < n - 2$.

Proof Follows directly from Lemma 5.1.1 and the fact that *pathwidth*$(G) = n-1 \Leftrightarrow$ $\gamma = n - 1$. ∎

Corollary 5.1.4 If $\gamma(G) = n - 3$ then *pathwidth*$(G) =$ *treewidth*$(G) = n - 3$.

Proof Since *pathwidth*$(G) \geq$ *treewidth*$(G) \geq \gamma(G) = n - 3$, and the previous corollary says that *pathwidth*$(G) < n - 2$, we have this result. ∎

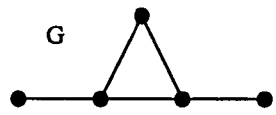


Figure 5.1: A graph with *treewidth* $= n - 3$ and $\gamma < n - 3$

Note that the converse of Corollary 5.1.4 is not true. The graph in Figure 5.1 is a counter-example, because *treewidth*$(G) = 2 = n - 3$, but $\gamma(G) = 1$.

We know that if G is an obstruction for TW(k), then there is a minor H of G which is an obstruction for PW(k). But when is a treewidth obstruction also a pathwidth obstruction? We present a sufficient condition for a pathwidth obstruction to be a treewidth obstruction also.

Lemma 5.1.5 If G is an obstruction for PW(k), and if *treewidth*(G) = *pathwidth*(G), then G is also an obstruction for TW(k).

Proof Let $H <_m G$. Then $pathwidth(H) < pathwidth(G) = treewidth(G) = k+1$. But $treewidth(H) \leq pathwidth(H)$. Therefore, for every $H <_m G$, $treewidth(H) < treewidth(G)$, and $G \in obs(\mathrm{TW}(k))$. ∎

We conclude this section by proving that the obstruction sets for pathwidth and treewidth do have a non-trivial intersection.

Lemma 5.1.6 A graph G of order n is an obstruction for $\mathrm{PW}(n-3)$ if and only if G is also an obstruction for $\mathrm{TW}(n-3)$.

Proof ($\Leftarrow$): Let $G \in obs(\mathrm{TW}(n-3))$. Then $treewidth(G) = n-2$, and Corollary 5.1.2 and Lemma 5.1.1 together give us $pathwidth(G) = n-2$. If H is a minor of G, then $treewidth(H) < treewidth(G) = n-2$, and $\gamma(G) < n-2$. Therefore, $pathwidth(H) < n-2$, and $G \in obs(\mathrm{PW}(n-3))$.

($\Rightarrow$): Let $G \in obs(\mathrm{PW}(n-3))$. Then $pathwidth(G) = n-2$ and according to Lemma 5.1.1 and Corollary 5.1.2, $treewidth(G) = n-2$. Using Lemma 5.1.5, we see that $G \in obs(\mathrm{TW}(n-3))$. ∎

5.2 The Extent of Commonality of Obstructions

After Lemma 5.1.6, we naturally wonder about the extent of intersection between $obs(\mathrm{TW}(k))$ and $obs(\mathrm{PW}(k))$. It is known (see [Ki2]) that there are a large number of tree obstructions to $\mathrm{PW}(k)$ for each $k > 0$. However, since the treewidth of a tree is 1, there cannot be a tree obstruction to $\mathrm{TW}(k)$ for any $k > 0$. Therefore, $obs(\mathrm{PW}(k)) \not\subseteq obs(\mathrm{TW}(k))$. The following lemma shows that trees are not the only obstructions that distinguish $obs(\mathrm{PW}(k))$ from $obs(\mathrm{TW}(k))$.

Lemma 5.2.1 For all $k \geq 2$, the graph G shown in Figure 4.4 on page 78 is an obstruction of order $3k$ for $\mathrm{PW}(k)$.

Proof From Lemma 4.5.7, we know that $treewidth(G) = k$. Lemma 4.5.9 showed that $pathwidth(G) = k + 1$. It can be easily verified that for the graph G shown in Figure 4.4, if $H <_m G$, then $pathwidth(H) < pathwidth(G)$. ∎

In this lemma, since $treewidth(G) = k$, clearly $G \notin obs(\mathrm{TW}(k))$. Moreover, when $k = 2, n = |V(G)| = 3k = 6 = k+4$ and $G \in obs(\mathrm{PW}(n-4))$, but $G \notin obs(\mathrm{TW}(n-4))$ (in fact, G is the obstruction of order 6 for PW(2) shown in Figure 3.2). In what follows, we generalize this observation.

Definition K_n^* is the complete graph of order n from which $\lfloor \frac{n}{2} \rfloor$ disjoint edges have been deleted. If n is odd, then an edge incident on the odd vertex is also deleted.

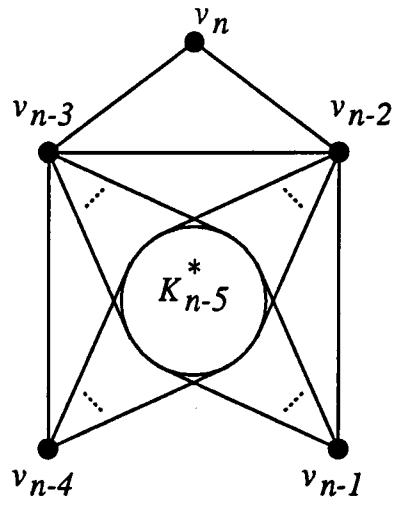


Figure 5.2: An obstruction of order n for PW$(n-4)$

Lemma 5.2.2 For every $n \geq 6$, there exists an obstruction of order n for PW$(n-4)$ that is not an obstruction for TW$(n-4)$.

Proof Consider the graph G of order n, depicted in Figure 5.2. The vertex set $\{v_1, v_2, \ldots, v_{n-5}\}$ induces a K^*_{n-5} in G. Assume that for each even i, $1 \leq i \leq n-6$, $(v_i, v_{i+1}) \notin E$ and for even n, since v_{n-5} is the odd vertex in K^*_{n-5}, $(v_1, v_{n-5}) \notin E$. For $n-4 \leq i \leq n-1$, each vertex v_i is adjacent to all the vertices of the K^*_{n-5}. Vertex v_n has only two neighbors.

The following path-decomposition shows that $pathwidth(G) \leq n-3$: $X_1 = \{v_i : 1 \leq i \leq n-3\}$, $X_2 = \{v_i, v_{n-3}, v_{n-2}, v_n : 1 \leq i \leq n-5\}$, and $X_3 = \{v_i, v_{n-2}, v_{n-1} : 1 \leq i \leq n-5\}$. When n is odd, $K_{2,2} \not\subseteq \bar{G}$ and as a consequence of Corollary 4.5.4, $pathwidth(G) > n-4$.

When n is even, we claim that there is no graph $H \subseteq \bar{G}$ that satisfies all four conditions of Corollary 4.5.3 for $w = n-4$. The only $K_{2,2} \subseteq \bar{G}$ is formed by vertices v_1, v_2, v_{n-5}, and v_n which are therefore in H. Of these four vertices, only v_n is adjacent to other vertices in $\bar{G}$. However, since the maximum degree of H must be 3, and v_n already has two neighbors in $V(H)$, it is clear that $4 \leq |V(H)| \leq 5$. This implies that there must be at least three vertices of degree 3 in H. However, this is not possible because only v_1 and v_n can have degree 3. Hence, we conclude that $pathwidth(G) = n-3$.

On the other hand, the following is a tree-decomposition of width $n-4$ for G: $X_1 = \{v_i : 1 \leq i \leq n-3\}$, $X_2 = X_1 \cup \{v_{n-2}\} - \{v_{n-4}\}$ $X_3 = \{v_i, v_{n-2}, v_{n-1} : 1 \leq i \leq n-5\}$, and $X_4 = \{v_i, v_{n-3}, v_{n-2}, v_n : 1 \leq i \leq n-5\}$. Therefore, $treewidth(G) \leq n-4$. Since every obstruction of order $n-1$ or $n-2$ for PW$(n-4)$ is also an obstruction for TW$(n-4)$, we conclude that G contains only an obstruction of order n for PW$(n-4)$. In fact, it is not hard to see that G is itself an obstruction. ∎

This shows that Lemma 5.1.6 cannot be extended even to $n-4$. Is it true then that $obs(\text{TW}(k)) \subset obs(\text{PW}(k))$? This is certainly true when $k = 1$ or $k = 2$, because in these two cases, $obs(\text{TW}(k)) = \{K_{k+1}\}$. But what about the general case?

Consider the prism graphs B_8 and B_{10} in Figure 5.3. It is known that $B_{10} \in obs(\text{TW}(3))$ (see [APC] or [ST]). In the following lemma, we prove that $B_8 \in obs(\text{PW}(3))$.

Lemma 5.2.3 The graph B_8 shown in Figure 5.3 is an obstruction for PW(3).

Proof We claim that $K_{2,3} \not\subseteq \bar{B}_8$. The proof of this claim is by contradiction. If $K_{2,3} \subseteq \bar{B}_8$, then there exist two vertices v_i and v_j in $\bar{B}_8$ such that $|N_i \cap N_j| \geq 3$ which

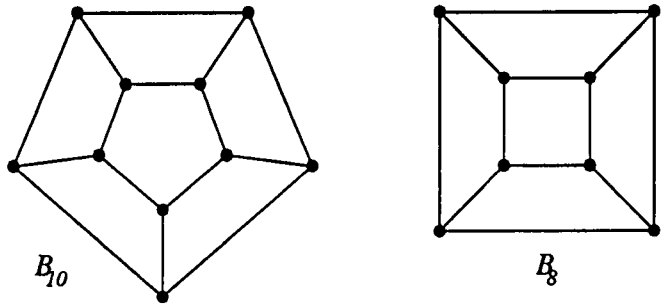


Figure 5.3: A TW(3) obstruction containing a PW(3) obstruction

in turn implies that in B_8, $|N_i \cup N_j \cup \{v_i, v_j\}| \leq 5$.

$$\begin{aligned} \text{In } B_8, |N_i \cup N_j \cup \{v_i, v_j\}| &= |N_i| + |N_j| + |\{v_i, v_j\}| + |N_i \cap N_j \cap \{v_i, v_j\}| \\ &\quad -|N_i \cap N_j| - |N_i \cap \{v_i, v_j\}| - |N_j \cap \{v_i, v_j\}| \\ &= 8 - |N_i \cap N_j| - |N_i \cap \{v_j\}| - |N_j \cap \{v_i\}| \end{aligned}$$

If the edge (v_i, v_j) exists in B_8, then $|N_i \cap N_j| = 0$; else $|N_i \cap N_j| \leq 2$. In either case, $|N_i \cup N_j \cup \{v_i, v_j\}| \geq 6$. Therefore, $K_{2,3} \not\subseteq \bar{B}_8$ and the order of the largest complete bipartite subgraph of $\bar{B}_8$ is at most 4.

Using Corollary 4.5.3, we get $pathwidth(B_8) \geq 4$. It is simple to obtain a path-decomposition of width 4 for B_8. We can also easily verify that if $H <_m B_8$, then $pathwidth(H) < 4$. ∎

Since $B_8 \leq_m B_{10}$, and $B_8 \in obs(\text{PW}(3))$, it is clear that $B_{10} \notin obs(\text{PW}(3))$. Therefore, $obs(\text{TW}(3)) \not\subseteq obs(\text{PW}(3))$. We believe that such obstructions exist for every $k \geq 3$, and consequently, $obs(\text{TW}(k)) \not\subseteq obs(\text{PW}(k))$. However, B_{10} is the only such obstruction we know and we have been unable to generalize B_{10} to values of $k > 3$. Extrapolating from the number of obstructions for PW(3), it seems likely that an obstruction of maximum order for TW(k) would contain as a proper minor an obstruction for PW(k).

Conjecture 5.2.4 For every $k \geq 3$, $obs(\text{TW}(k)) \not\subseteq obs(\text{PW}(k))$.

5.3 Obstructions of Order n for TW$(n-3)$

Having seen that $obs(\mathrm{PW}(n-3)) = obs(\mathrm{TW}(n-3))$, we now explore the structure of these obstructions. We start with an observation about their minimum degree.

Lemma 5.3.1 If $\gamma(G) = n-2$, then $\delta(G) \geq 3q-3$, where $0 \leq q \leq \lfloor \frac{n}{3} \rfloor$ is the number of vertices of degree less than $n-2$ in G.

Proof Let $V_1 = \{v_1, \ldots, v_q\}$ be the set of all vertices of degree less than $n-2$ in G, and $V_2 = V - V_1$. Suppose $v_i \in V_1$. Then there exist $\{v_x, v_y\} \subseteq V_2$ such that, $\{(v_i, v_x), (v_i, v_y)\} \cap E = \phi$. Since $\gamma(G) = n-2$, we know $\delta_x = \delta_y = n-2$. Therefore, corresponding to each vertex v_i in V_1, there exists a distinct set of at least two non-neighbors of degree exactly $n-2$. Moreover, each vertex of degree $n-2$ in V_2 is non-adjacent to exactly one vertex in V_1. In other words, there are at least $2q$ vertices of degree $n-2$ in G. Therefore, the order of G is $n \geq 3q$, or $q \leq \lfloor \frac{n}{3} \rfloor$.

$\gamma = n-2$ also implies that the vertices in V_1 are mutually adjacent. Therefore, if $v_i \in V_1$, and $\{(v_i, v_x), (v_i, v_y)\} \cap E = \phi$, then $|\{v_j \in V_1 : (v_i, v_j) \in E\}| = q-1$, and $|\{v_j \in V_2 : (v_i, v_j) \in E\}| \geq 2q-2$. Hence, $\delta_i \geq 3q-3$. Since $\delta(G) = \min\{\delta_i : v_i \in V_1\}$, $\delta \geq 3q-3$. ∎

The following theorem gives a necessary and sufficient condition for a graph of order n to be an obstruction for TW$(n-3)$.

Theorem 5.3.2 For all $n \geq 6$, a graph G of order n is an obstruction for TW$(n-3)$ if and only if G satisfies the following four conditions:

i. $\gamma(G) = n-2$,

ii. the maximum degree of G is $n-2$,

iii. $\delta(G) \geq \max\{4, 3q-3\}$, where
$0 \leq q \leq \lfloor \frac{n}{3} \rfloor$ is the number of vertices of degree $< n-2$ in G, and

iv. G is missing at least three disjoint edges

(i.e. the complement of G has a matching of size ≥ 3).

Proof ($\Rightarrow$): First we show that if $G \in obs(TW(n-3))$ then it satisfies (i)-(iv). Since $treewidth(G) = n-2$, (i) follows directly from Corollary 5.1.2. For (ii), suppose there exists v_1 with $\delta_1 = n-1$. Let v_2 be a vertex of minimum degree in G. Then $\delta_2 = \delta(G) \leq \gamma(G) = n-2$. Let $H = (V(G), E(G) - \{(v_1, v_2)\})$. $\gamma(H) = \min\{\gamma(G), \max\{\delta_1, \delta_2\}\} = \min\{\gamma(G), \delta_1\} = n-2$. This implies $treewidth(H) = n-2$, contradicting the fact that G is minor-minimal. Therefore maximum degree of $G = n-2$.

The proof of (iii) is by contradiction. Suppose $1 \leq \delta_1 = \delta \leq 3$. Let $v_2, \ldots, v_{\delta+1}$ be the neighbors of v_1, and $V_2 = \{v_i : \delta + 2 \leq i \leq n\}$ be the set of non-neighbors of v_1. Since $\gamma = n-2$, each vertex in V_2 has degree $n-2$. Therefore, each neighbor of v_1 has degree $\geq n - \delta$. If $\delta = 1$ then, $\delta_2 = n-1$ which contradicts (ii). If $\delta = 2$ then, contracting the edge (v_1, v_2) to v_2 gives us $K_{n-1} \leq_m G$ and $treewidth(K_{n-1}) = n-2 = treewidth(G)$, contradicting the minor-minimality of G. If $\delta = 3$ then, we claim that at least one of the edges $(v_2, v_3), (v_2, v_4), (v_3, v_4)$ is in G, because otherwise, $\delta_2 = \delta_3 = \delta_4 = n-3$ and $\gamma(G) < n-2$. Assume that $(v_2, v_3) \in E$. Contract edge (v_1, v_4) to v_4 to obtain $K_{n-1} \leq_m G$. Therefore $\delta \geq 4$, and using Lemma 5.3.1 we get $\delta \geq \max\{4, 3q-3\}$.

It is clear from the proof of (iii) that each vertex of degree $< n-2$ in G contributes an edge to the maximum matching in the complement of G. Thus, when $q \geq 3$, (iv) follows from (iii). Otherwise, we have three different cases.

When $q = 0$, $\delta_i = n-2, \forall i$ and, there is a matching of size $\lfloor \frac{n}{2} \rfloor$ in the complement of G which, for $n \geq 6$, is ≥ 3.

If $q = 1$, let $\delta_1 < n-2$ and $(v_1, v_2) \notin E$. Then $\delta \geq 4 \Rightarrow \exists \{v_3, v_4, v_5, v_6\} \ni \{(v_1, v_i), (v_2, v_i) : 3 \leq i \leq 6\} \subset E$ and $\{(v_3, v_4), (v_5, v_6)\} \cap E = \phi$.

If $q = 2$, let $\delta_1 \leq \delta_2 < n-2$ and $\{(v_1, v_3), (v_2, v_4)\} \cap E = \phi$. Then $\delta \geq 4 \Rightarrow \exists \{v_5, v_6\} \ni \{(v_1, v_i), (v_2, v_i) : 5 \leq i \leq 6\} \subset E$ and $(v_5, v_6) \notin E$. Thus, when $q = 1$ or 2, $(v_1, v_2), (v_3, v_4), (v_5, v_6)$ is a set of three disjoint edges missing in G. This concludes the proof of (iv).

($\Leftarrow$): Now we prove that (i)-(iv) are sufficient conditions for G to be a minor-minimal obstruction. Since (i) $\Rightarrow$ $treewidth(G) = n-2$, we only have to show that G is minor-minimal. It is easy to verify that if H is obtained by either deleting or contracting an edge of G, then $\gamma(H) < n-2$ which implies that $treewidth(H) < n-2$. Hence G is minor-minimal and the proof is complete. ∎

5.4 Enumerating the Obstructions

In this section, we enumerate the obstructions of order n for TW$(n-3)$ and obtain a lower bound for their number using the theory of partitions of integers. The following definitions are from [An].

Definition A *partition* of a positive integer n is a finite non-increasing sequence of positive integers $l_1, l_2, \ldots, l_r$ such that $\sum_{i=1}^{r} l_i = n$. The l_i are called *parts* of the partition. $p_r(n)$ is the number of partitions of n into r parts. The total number of partitions of n is $p(n) = \sum_{r=1}^{n} p_r(n)$.

We adopt the convention that $p_0(0) = 1, p_0(n) = 0$ if $n > 0$, and $p_r(n) = 0$ if $r > n$.

5.4.1 A Canonical Representation

Let $S(n)$ be the set of obstructions of order n for TW$(n-3)$ and $S(n,q)$ be the set of obstructions with q vertices of degree $< n-2$. From Theorem 5.3.2, it is clear that $|S(n)| = \sum_{q=0}^{\lfloor \frac{n}{3} \rfloor} |S(n,q)|$. We describe a canonical representation for each obstruction in $S(n,q)$ for TW$(n-3)$.

Consider an obstruction G of order n for TW$(n-3)$ with q vertices of degree $< n-2$. Let $V = V_1 \cup V_2$ with $V_1 = \{v_i : 1 \leq i \leq q\}$ where $\forall v_i \in V_1, \delta_i < n-2$ and, $V_2 = \{v_i : q+1 \leq i \leq n\}$ where $\forall v_i \in V_2, \delta_i = n-2$. Since $\forall v_i \in V_1, \delta_i < n-2$, let $\{(v_i, v_{q+i}), (v_i, v_{2q+i})\} \cap E = \phi$. Then v_{q+i} and v_{2q+i} are adjacent to all other vertices in G. The canonical form is shown in Figure 5.4. For clarity, only the missing edges are shown. All other edges are present.

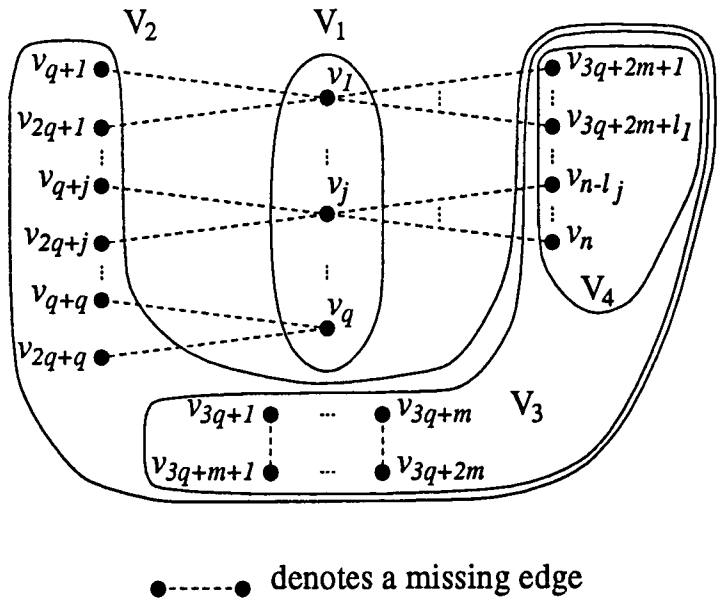


Figure 5.4: Canonical form of an obstruction in $S(n)$

Let $V_3 = \{v_i : 3q+1 \leq i \leq n\} \subseteq V_2$. If m is the size of the maximum matching in the complement of the graph induced by V_3 then $\forall i$, $1 \leq i \leq m$, let $(v_{3q+i}, v_{3q+m+i}) \notin E$. Therefore, v_{3q+i} and v_{3q+m+i} are adjacent to every other vertex in G. Clearly, $m \leq \lfloor \frac{n-3q}{2} \rfloor$.

Let $V_4 = \{v_i : 3q+2m+1 \leq i \leq n\} \subseteq V_3 \subseteq V_2$. Since every vertex in $V_2 - V_4$ already has degree $n-2$, for each vertex in V_4 there is exactly one non-neighbor in V_1. There exist $p_j(n-3q-2m)$ distinct mappings of V_4 into j elements of V_1. Suppose $l_1, \ldots, l_j$ is a j-partition of $n-3q-2m$. Then, $\forall i$, $1 \leq i \leq j$, and $\forall k$, $1 \leq k \leq l_i$, we let $(v_i, v_{3q+2m+l+k}) \notin E$ where $l = l_1 + \ldots + l_{i-1}$. Therefore, $\forall v_i \in V_1$, if $1 \leq i \leq j$

then $\delta_i = n - 3 - l_i$; else $\delta_i = n - 3$.

Observe that each graph that satisfies conditions (i)–(iv) of Theorem 5.3, and hence each obstruction in $S(n)$, has a unique representation in the form of a quintuple $(n, q, m, j, (l_1, \ldots, l_j))$. This is stated more formally in the following lemma.

Lemma 5.4.1 A graph G of order n is an obstruction for TW$(n-3)$ if and only if G can be uniquely represented by a 5-tuple $(n, q, m, j, (l_1, \ldots, l_j))$, where

i. $n \geq 6$

ii. $0 \leq q \leq \lfloor \frac{n}{3} \rfloor$

iii. $\max\{0, 3-q\} \leq m \leq \lfloor \frac{n-3q}{2} \rfloor$

iv. $\min\{1, n-3q-2m\} \leq j \leq \min\{q, n-3q-2m\}$

v. $l_1 \geq \ldots \geq l_j$ is a j-partition of $n-3q-2m$

Proof

($\Rightarrow$): The proof of this follows from the preceding description of the canonical form. Given a graph $G \in obs((\text{TW}(n-3)$, the unique 5-tuple representing G is obtained as follows: $n = |V(G)|$; $q =$ the number of vertices of degree $< n-2$ (i.e. $q = |V_1|$); $m =$ is the size of the maximum matching in the complement of the subgraph induced by those vertices of degree exactly $n-2$; $j =$ is the number of vertices of degree less than $n-3$ in G; and $n-3-l_1 \leq \ldots \leq n-3-l_j$ is the degree sequence of the set of vertices of degree less than $n-3$ in G.

($\Leftarrow$): Now we show that each 5-tuple that satisfies (i)–(v) yields a unique graph $G \in obs(\text{TW}(n-3))$. We start with a complete graph of order n and delete a set of edges from it so that the resulting graph G satisfies conditions (i)–(iv) of Theorem 5.3.2. Let $V_1 = \{v_1, \ldots, v_q\}$ and let $V_2 = V(G) - V_1$. For each $v_i \in V_1$, delete edges (v_i, v_{q+i}) and (v_i, v_{2q+i}). Let $V_3 = V_2 - \{v_i : q+1 \leq i \leq 3q\}$. Delete the m disjoint edges $(v_{3q+i}, v_{3q+m+i}), 1 \leq i \leq m$ between vertices in V_3. Let $V_4 =$

$V_3 - \{v_{3q+i} : 1 \leq i \leq 2m\}$. For each vertex $v_i, 1 \leq i \leq j$ in V_1, delete the set of edges $\{(v_i, v_{3q+2m+k}) : l_1 + \ldots + l_{i-1} + 1 \leq k \leq l_i\}$. Call the modified graph G.

We claim that G satisfies conditions (i)–(iv) of Theorem 5.3.2. Since each vertex of G has at least one non-neighbor, maximum degree$(G) = n - 2$. Moreover, only the vertices in V_1 have degree less than $n - 2$. Since the vertices in V_1 are mutually adjacent, $\gamma(G) = n - 2$. Notice that each vertex in V_1 is adjacent to the other $q - 1$ vertices in V_1, at least $2(q - 1)$ vertices in $V_2 - V_3$, and $2m$ vertices in V_3. Together, conditions (ii) and (iii) of this theorem say that $q + m \geq 3$. Therefore, $\delta(G) \geq 3q - 3 + 2m = 2(q + m) - 3 + q = 3 + q$. If $q \geq 1$, then $\delta \geq 4$. Suppose $q = 0$. We know that $\min\{1, n - 3q - 2m\} \leq j \leq q$. Since $n - 3q - 2m \geq 0$, we have $0 \leq \min\{1, n - 2m\} \leq j \leq 0$. Therefore, $n - 2m = 0$ or $m = \frac{n}{2}$. Since m is an integer, we conclude that $q = 0 \Rightarrow n$ is even and $\delta(G) = n - 2 \geq 4$. This completes the proof. ∎

5.4.2 An Exact Formula

Given n, we would like to find all the obstructions in the set $S(n)$. In order to do this, we find all the quintuples that represent an obstruction G in $S(n)$. This also enables us to count the obstructions as shown in the following theorem.

Theorem 5.4.2 The cardinality of $S(n, q)$ is as follows:

$q = 0$:
$$|S(n,0)| = \begin{cases} 1 \text{ for even } n \geq 6; \\ 0 \text{ otherwise.} \end{cases}$$

$q = 1$:
$$|S(n,1)| = \begin{cases} 0, & n \leq 6; \\ \sum_{m=2}^{\lfloor \frac{n-3}{2} \rfloor} 1 & \text{otherwise.} \end{cases}$$

$q = 2$:
$$|S(n,2)| = \begin{cases} 0, & n \leq 7; \\ \sum_{m=1}^{\lfloor \frac{n-6}{2} \rfloor} \sum_{j=0}^{2} p_j(n - 6 - 2m) & \text{otherwise.} \end{cases}$$

$3 \leq q \leq \lfloor \frac{n}{3} \rfloor$:
$$|S(n,q)| = \begin{cases} 0, & n \leq 8; \\ \sum_{m=0}^{\lfloor \frac{n-3q}{2} \rfloor} \sum_{j=0}^{q} p_j(n - 3q - 2m) & \text{otherwise.} \end{cases}$$

Proof Our proof is based on the canonical representation. Let G be a graph represented by a quintuple. In order for three disjoint edges to be missing, we need $n \geq 6$. Since each vertex in V_1 only accounts for one of the three missing edges, $m \geq 3 - q$ or $m + q \geq 3$. Moreover, $3q + 2m \leq n$, and $\delta \geq 2m + 3(q-1) = 3q + 2m - 3 = 2(q+m) + q - 3 \geq 6 + q - 3 = 3 + q$.

The following are the restrictions on the variables that determine the structure of an obstruction in $S(n)$:

n : $n \geq 6$ is the number of vertices in the graph.

q : $0 \leq q \leq \lfloor \frac{n}{3} \rfloor$ is the number of vertices of degree $< n-2$ i.e. $q = |V_1|$.

m : $\max\{0, 3-q\} \leq m \leq \lfloor \frac{n-3q}{2} \rfloor$ is the size of the maximum matching in the complement of the graph induced by V_3.

j : $1 \leq j \leq n - 3q - 2m$ is the number of parts of $|V_4|$.

When $q = 0$, every vertex in G has degree $n-2$, which implies that n should be even, and G is a complete graph of order n with $\frac{n}{2}$ disjoint edges missing. When $n \geq 6$, G satisfies all four conditions of Theorem 5.3 and, is in $S(n, 0)$.

It is clear from our canonical representation and the proof of Theorem 5.3 that, whenever $q \geq 1$, $q + m \geq 3$, and $n \geq 3q + 2m$, G is in $S(n)$. Therefore, for $q \geq 1$ and $n \geq 3q + 2m$, we have the general expression $|S(n,q)| = \sum_{m=0}^{\lfloor \frac{n-3q}{2} \rfloor} \sum_{j=0}^{q} p_j(n - 3q - 2m)$

If $q = 1$ then $m \geq 2$ and $n \geq 7$. Hence, for $n \geq 7$, $|S(n,1)| = \sum_{m=2}^{\lfloor \frac{n-3q}{2} \rfloor} p_1(n - 3q - 2m) = \sum_{m=2}^{\lfloor \frac{n-3}{2} \rfloor} 1$.

When $q = 2$, $m \geq 1$ and $n \geq 8$. Therefore, for $n \geq 8$, $|S(n,2)| = \sum_{m=1}^{\lfloor \frac{n-3q}{2} \rfloor} \sum_{j=1}^{q} p_j(n - 3q - 2m) = \sum_{m=1}^{\lfloor \frac{n-6}{2} \rfloor} \sum_{j=1}^{2} p_j(n - 6 - 2m)$

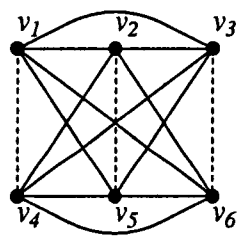
If $q \geq 3$ then $m \geq 0$ and $n \geq 9$. Moreover, when $n - 3q - 2m = 0$ we also get an obstruction corresponding to $j = 0$, where every vertex in V_1 has degree exactly $n - 3$. Since $p_0(x) = 0$ whenever $x \neq 0$, for $n \geq 9$, we can write $|S(n,q)| =$

$p_0(n-3q-2m) + \sum_{m=0}^{\lfloor \frac{n-3q}{2} \rfloor} \sum_{j=1}^{q} p_j(n-3q-2m)$. Replacing $p_0(n-3q-2m)$ with $\sum_{m=0}^{\lfloor \frac{n-3q}{2} \rfloor} p_0(n-3q-2m)$ gives us the desired formula. ∎

Table 5.1: Obstructions of order n for PW$(n-3)$ and TW$(n-3)$

$n-3$	1	2	3	4	5	6	7	8	9	10	15	20	25
$\lvert S(n)\rvert$	0	0	1	1	3	4	7	9	15	18	79	242	694

$|S(n)|$ can be computed using the recurrence relation $p_k(n) = p_k(n-k) + p_{k-1}(n-1)$. Table 5.1 lists some representative values of $|S(n)|$. For example, when $n-3 = 3$, $|S(n)| = 1$ implying that there is only obstruction of order 6 for TW(3). This obstruction is the graph shown in Figure 5.5, which is called M_6 in [APC] and $K_{2,2,2}$ in [ST]. Curiously, even though the entire obstruction set for PW(2) is known (see [KL]), the fact that $S(n) \subset obs(\text{PW}(n-3))$ was previously unsuspected. The table reveals that this is possibly because $|S(5)| = 0$ and none of the general constructions of [KL] produce pathwidth obstructions as dense as those in $S(n)$.



●------● denotes a missing edge

Figure 5.5: The obstruction of order 6 for TW(3)

5.4.3 An Exponential Lower Bound

Observe that even though we have an exact formula for $|S(n)|$, because it is not in closed form, we do not have an indication of the rate of growth of $|S(n)|$ yet. Figure 5.6 on page 102 graphically demonstrates the rate of growth of the set $S(n)$ with n. It is evident that the number of obstructions increases rapidly. In this section, we show that $|S(n)|$ grows exponentially.

Lemma 5.4.3 For $n \geq 0$, $p(n+2) \leq p(n+1) + p(n) \leq 2p(n+1)$.

Proof

$$\begin{aligned}
p(n+2) &= \sum_{j=1}^{n+2} p_j(n+2) \\
&= \sum_{j=1}^{n+2} [p_{j-1}(n+1) + p_j(n+2-j)] \\
&= \sum_{j=0}^{n+1} p_j(n+1) + \sum_{j=1}^{n+2} p_j(n+2-j) \\
&= p(n+1) + p_1(n+1) + \sum_{j=2}^{n+2} p_j(n-(j-2)) \\
&\leq p(n+1) + p_1(n) + \sum_{j=2}^{n+2} p_j(n) \\
&= p(n+1) + p(n)
\end{aligned}$$

If $l_1 \leq \cdots \leq l_k$ is a k-partition of n, then $l_1 \leq \cdots \leq l_k + 1$ is a k-partition of $n+1$. Therefore, $p_k(n+1) \geq p_k(n)$ and $p(n+1) \geq p(n)$. Hence we have $p(n+2) \leq p(n+1) + p(n) \leq p(n+1) + p(n+1)$. ∎

Theorem 5.4.4 For $n \geq 12$, $|S(n)| \geq \frac{1}{4} p(\lfloor \frac{n}{4} \rfloor)$.

Proof

$$|S(n)| \geq \sum_{q=3}^{\lfloor \frac{n}{3} \rfloor} |S(n,q)|$$

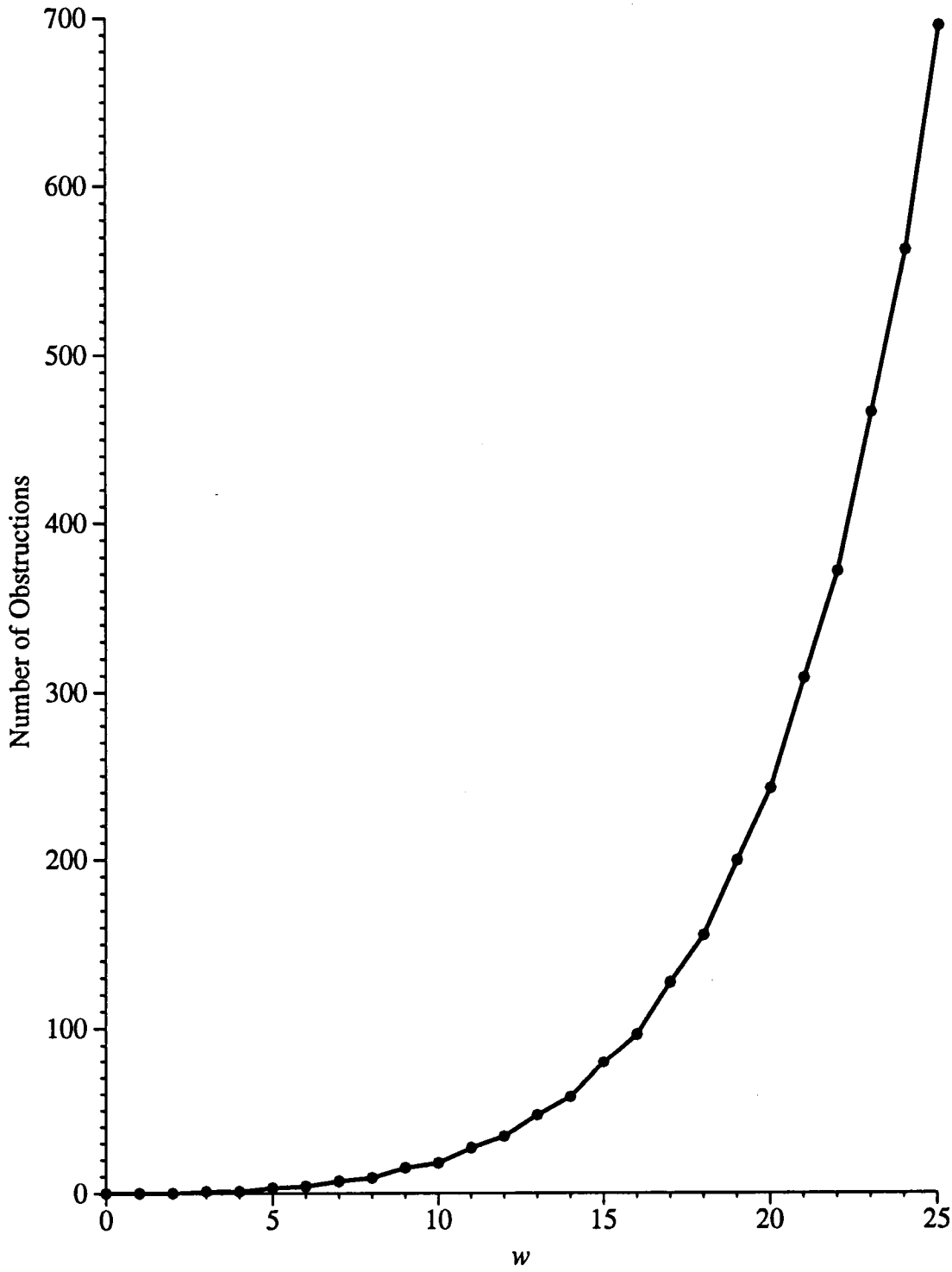


Figure 5.6: Number of Obstructions of Order $w + 3$ for PW(w) and TW(w)

Taking only the $m = 0$ terms of $S(n,q)$ for each $q \geq 3$, and using the fact that $n \geq 12 \Rightarrow \lceil \frac{n}{4} \rceil \geq 3$, we get

$$|S(n)| \geq \sum_{q=3}^{\lfloor \frac{n}{3} \rfloor} \sum_{j=1}^{q} p_j(n-3q) \geq \sum_{q=\lceil \frac{n}{4} \rceil}^{\lfloor \frac{n}{3} \rfloor} \sum_{j=1}^{q} p_j(n-3q)$$

If $q \geq \frac{n}{4}$, then $q \geq n - 3q$ and we have

$$\begin{aligned} |S(n)| &\geq \sum_{q=\lceil \frac{n}{4} \rceil}^{\lfloor \frac{n}{3} \rfloor} \sum_{j=1}^{n-3q} p_j(n-3q) = \sum_{q=\lceil \frac{n}{4} \rceil}^{\lfloor \frac{n}{3} \rfloor} p(n-3q) \\ &\geq p(n - 3\lceil \frac{n}{4} \rceil) \\ &\geq p(\lfloor \frac{n}{4} \rfloor - 2) \end{aligned}$$

Since $n \geq 12 \Rightarrow \lfloor \frac{n}{4} \rfloor - 2 \geq 1$, we can use Lemma 5.3 to get

$$|S(n)| \geq \frac{1}{2} p(\lfloor \frac{n}{4} \rfloor - 1) \geq \frac{1}{4} p(\lfloor \frac{n}{4} \rfloor).$$

∎

There is no closed form expression known to compute either $p_k(n)$ or $p(n)$. We use the following result to bound the number of obstructions.

Lemma 5.4.5 (see page 70 of [An]) As $n \to \infty$, $p(n) \sim \frac{e^{c\sqrt{n}}}{4n\sqrt{3}}$ where $c = \pi\sqrt{\frac{2}{3}}$.

From Theorem 5.4.4 and Lemma 5.4.5, it is clear that $|S(n)|$ is asymptotically exponential in $c\sqrt{n} - \log n$.

5.5 Testing for the Obstructions

Since there is an exponential number of obstructions of order n for PW$(n-3)$, testing whether an input graph G contains each one of them individually would be inefficient in general. One way to deal with such large numbers of related obstructions is to test for several of them at once. Algorithm TEST of Chapter 3 is an example of this approach.

5.5.1 Connectivity of the Obstructions

The connectivity of an obstruction is a very useful property when we try to develop a minor-containment test. Here we examine the connectivity of the dense obstructions with a view to developing tests for them.

Lemma 5.5.1 If $\gamma(G) = n - 2$, then there exist $\min\{\delta_i, \delta_j\}$ vertex-disjoint paths between each pair v_i, v_j of vertices of G.

Proof Let N_i and N_j be the set of neighbors of vertices v_i and v_j respectively. Assume, without loss of generality, that $\delta_i \leq \delta_j$. The lemma is trivial when $N_i - \{v_j\} \subseteq N_j - \{v_i\}$. Suppose otherwise. Then $\delta_j \leq n - 2$. We enumerate the paths between v_i and v_j in increasing order of their length. If the edge (v_i, v_j) exists, then it is the only path of length one. There exist exactly $|N_i \cap N_j|$ disjoint paths of length two. The number of paths of length three depends on the value of δ_j.

Case-1: $\delta_i \leq \delta_j = n - 2$

If $(v_i, v_j) \notin E$, then $N_i \subseteq N_j$. So, let $(v_i, v_j) \in E$, and $(v_j, v_y) \notin E$. If $\delta_i < n-2$, then $(v_i, v_y) \in E$, because otherwise $\gamma < n - 2$. If $\delta_i = n - 2$, then $(v_i, v_y) \in E$, because otherwise $N_i - \{v_j\} = N_j - \{v_i\}$. Therefore, $N_i - N_j = \{v_j, v_y\}$. Since $\delta_i \leq n-2$, there exists a vertex v_x such that $(v_i, v_x) \notin E$, and $\gamma = n - 2 \Rightarrow (v_x, v_y) \in E$. Therefore, $[v_i, v_y, v_x, v_j]$ is the only path between v_i and v_j that includes v_x or v_y. Hence there exist $|N_i \cap N_j| + 2 = |N_i| - |N_i - N_j| + 2 = |N_i| = \delta_i$ vertex-disjoint paths of length at most three between v_i and v_j.

Case-2: $\delta_i \leq \delta_j < n - 2$

Since $\gamma = n - 2$, the edge $(v_i, v_j) \in E$. Moreover, each vertex in $N_i - N_j - \{v_j\}$ is adjacent to each vertex in $N_j - N_i - \{v_i\}$ and, since $\delta_i \leq \delta_j$, this gives $|N_i - N_j| - 1$ paths of length three. Hence there exist $|N_i \cap N_j| + 1 + |N_i - N_j| - 1 = |N_i| = \delta_i$ vertex-disjoint paths of length at most three between v_i and v_j. ∎

Corollary 5.5.2 If $\gamma(G) = n - 2$, then $\kappa(G) = \delta(G)$.

Proof We know that $\kappa(G) \leq \delta(G)$. Lemma 5.5.1 shows that if $\gamma(G) = n - 2$ then $\kappa(G) \geq \delta(G)$. As a result, $\kappa(G) = \delta(G)$. ∎

Corollary 5.5.3 If $G \in S(n-3)$, then $\kappa(G) \geq 4$.

Proof It follows from Theorem 5.3.2 that $\delta(G) \geq 4$. Combining this with Corollary 5.5.2, we get the desired result. ∎

5.5.2 Exploiting the Connectivity

Using the connectivity of a graph is currently the best general technique available to develop a minor test for it. Almost all the known algorithms that test whether a graph H is a minor of a graph G make use of the connectivity of H (see [KM],[KPS] for example). How does the connectivity of a graph aid us in developing an efficient minor test for the graph? Clearly, if we have a connected graph H then it is sufficient to look for H in the connected components of G. The following lemma is a refinement of this fact.

Lemma 5.5.4 A 2-connected graph H is a minor of a graph G if and only if H is a minor of some 2-connected component of G.

Proof

($\Leftarrow$): Since every 2-connected component of a graph is also a subgraph, this implication is trivial.

($\Rightarrow$): The proof is by induction on the number of 2-connected components of G. If G is 2-connected then the lemma is trivially true. Assume that G is not 2-connected. We can safely assume that G is connected. Let x be an articulation point of G. Deleting x divides G into connected subgraphs $G_1, \ldots, G_k, k \geq 2$. Let G_i' denote the connected subgraph of G induced by $V(G_i) \cup \{x\}$, for each i. Clearly each G_i' contains fewer 2-connected components than G. We now have to prove that H is a minor of G_i' for some i. Assume this is not true.

Let F be a minor of G that is isomorphic to H. Since H is 2-connected, so is F. If F contains the vertex x, then x is not an articulation point of F and F is a minor of some G_i' which contradicts our assumption. If F does not contain the vertex x then we know that x was eliminated either by a subgraph or an edge-contraction operation. Eliminating x by a subgraph operation would make F either disconnected or a minor of a G_i'. In either case we have a contradiction. The other possibility is for x to be eliminated in an edge-contraction. Let (w, x) be the edge whose contraction causes the elimination of x. w now becomes an articulation point of F. This again contradicts the fact that H is 2-connected.

Therefore we conclude that our initial assumption was false. This implies that F is a minor of a G_i' for some i. Since each G_i' has fewer 2-connected components than G, this completes the induction. ∎

Consider for instance K_5, which is an obstruction for both PW(3) and TW(3). It is known that if G is non-planar and 4-connected, then $K_5 \leq_m G$ (see page 252 of [Bol] for a proof). Graph 3-connectivity and a few other structural properties have been used to develop a fast minor test for K_5 (see [KM]).

Suppose $C \subset V(G)$ separates G into p components $G_1, \ldots, G_p$. Then $G_1, \ldots, G_p$ are called the *induced components* of $G - C$. Let $K(C)$ be the complete graph constructed with the vertex set C. For each $1 \leq i \leq p$, the graph $G_i \cup K(C)$ is called an *augmented component* of G induced by C.

Theorem 5.5.5 [KM] Suppose H is a 3-connected graph, and G is a 2-connected graph with a 2-cut C. Then G has an H minor if and only if some augmented component of G induced by C has an H minor.

A generalization of this theorem follows.

Theorem 5.5.6 [KM] Let C be a k-cut ($k \geq 3$) of a k-connected graph G separating G into at least k components or at least $k-1$ components such that at least two are

non-trivial. Then each augmented component of G induced by C is a minor of G.

The availability of fast algorithms to test for k-connectivity for small k is an important reason why this approach is practical.

Conjecture 5.5.7 If $H \leq_m G$ and H is an obstruction of order n for $TW(n-3)$, then there exists a 4-connected-component with at least n vertices in G.

The following lemma can yield tests for obstructions to TW(3).

Lemma 5.5.8 [Bol] If $\delta(G) \geq 4$ then $K_5 \leq_m G$ or $K_{2,2,2} \leq_m G$.

Notice however, that $\delta(G) \geq 4$ already implies that $treewidth(G) \geq 4$. If a graph G were 4-connected, then $\delta(G) \geq 4$. Recently, the notion of *quasi 4-connectivity* has been introduced [PS] and it is reported that quasi 4-connected graphs have several interesting properties. It remains to be seen whether this yields practical algorithms for pathwidth and treewidth.

5.5.3 A Conjecture About the Number of Edges

Recall that Lemma 4.2.4 showed that for any graph G with n vertices and e edges,

$$e > nw - \frac{w(w+1)}{2} \Rightarrow treewidth(G) > w$$

Mader (see [Mad]) has shown that

$$\text{for } 1 \leq w \leq 5, e > nw - \frac{w(w+1)}{2} \Rightarrow K_{w+2} \leq_m G$$

The similarity in form between these two results is striking. Mader's result tells us that $treewidth(G) > w$ because $K_{w+2} \leq_m G$. The following lemma gives a family of graphs which show that Mader's result does not hold when $w > 5$.

Lemma 5.5.9 [Ja] For $x \geq 3$ and $y \geq 3x-1$, let J be the graph obtained by deleting $2x-1$ disjoint edges from K_{x+y-1}. Then J has the following properties:

i. $n = |V(J)| = x + y - 1$,

ii. $e = |E(J)| = n(y-2) - \frac{(y-1)(y-2)}{2} + \frac{(x-1)(x-2)}{2}$,

iii. $K_y \not\leq_m J$.

Notice that since $x \geq 3$, $e > n(y-2) - \frac{(y-1)(y-2)}{2}$ and as a result, $treewidth(J) > y-2$. In fact, it is easy to see that $\gamma(J) = \delta(J) = n-2$, which means that $treewidth(J) \geq n-2$ and J contains obstructions for TW$(n-3)$ and PW$(n-3)$ other than the clique. For $x \geq 3$ and $y = 3x-1$, the graph J obtained by deleting $2x-1$ disjoint edges from K_{x+y-1} is an obstruction for pathwidth and treewidth because it satisfies all the conditions of Theorem 5.3.2. When $y > 3x-1$, there is at least one vertex of degree $n-1$ in J and as a result, J is not minor-minimal. However, for any $x \geq 3$, if J is the graph with $y = 3x-1$ and J' has $y > 3x-1$, then $J \subset J'$.

The following conjecture can be viewed as a general statement about any counter-example to the extensibility of Mader's result.

Conjecture 5.5.10 If G is a graph with $e > nw - \frac{w(w+1)}{2}$ then either

i. $K_{w+2} \leq_m G$, or

ii. G contains as a minor an obstruction of order $w+3$ for treewidth w.

If this conjecture is true, then it would give us an upper bound on the size of a graph in which we must search for an obstruction.

5.6 Summary

In this chapter, we examined the structure and number of dense obstructions for pathwidth and treewidth and presented a general method to construct new obstructions. Surprisingly, we found that the obstruction sets for the two families have a

significant amount of overlap. Our proof that the number of dense obstructions alone grows exponentially poses a formidable new challenge in the design of practical algorithms based on obstruction tests. One of the ways to surmount this difficulty would be to develop tests for entire families of structurally related obstructions rather than testing for them individually.

The lower bounds that we presented in the previous chapter were very useful in finding the obstructions. In particular, the role of the metric γ in helping identify the dense obstructions was significant. If G is an obstruction for width k, then $1 \leq \gamma(G) \leq k+1$. Therefore, we can classify the obstructions according to their γ value. For instance, the graphs in $obs(\mathrm{PW}(k))$ range from those in $S(k+3)$ where $\gamma = k+1$, to the tree obstructions all of which have $\gamma = 1$. If results similar to Lemma 5.1.1 and Theorem 5.3.2 can be obtained for other values of γ, then it is conceivable that the obstruction set for $\mathrm{PW}(k)$ and $\mathrm{TW}(k)$ would be computable in general. However, unlike Theorem 5.3.2, separate characterizations for pathwidth obstructions and treewidth obstructions will likely be required for all other values of γ. Moreover, it is not clear whether $\{K_{k+2}\} \cup S(k+3)$ includes all the obstructions with $\gamma = k+1$ for $\mathrm{PW}(k)$.

In a broader context, we pose the following questions:
For what values of k is there an obstruction

i. of order $n+k$ for $\mathrm{PW}(n)$ and $\mathrm{TW}(n)$?

ii. of order $n * k$ for $\mathrm{PW}(n)$ and $\mathrm{TW}(n)$?

Question (i) is trivial for $k = 2$. For $k = 3$, we have answered it in the affirmative for both pathwidth and treewidth, and for pathwidth alone when $k = 4$. The answer to question (ii) is also in the affirmative for pathwidth when $k = 3$. Unfortunately, our proofs were specific to the particular value of k. It would be interesting to find more general constructions.

In conclusion, the benefits of the study of obstructions are two-fold:

1. Better comprehension of their structure and number can help us design better algorithms for the fixed-parameter problem.

2. Being minimal graphs, their structure can give us insights that can help the development of better bounds for pathwidth and treewidth.

Chapter 6

Linear Layouts and Cutwidth

Instances of linear layout arise in a variety of situations in VLSI systems. For example, during the design of individual VLSI circuits, the datapath modules are often placed in a linear fashion [CNSD]. One dimensional linear arrays are also an example of linear placement (see [FHKY], [HPK] etc.). At a much higher level, the design of backplanes for systems comprising several circuit boards is another instance of linear placement (see [GCT] for example). In all these cases, the length of the layout is determined merely by the number of components and their dimensions. The width of the layout on the other hand, is a variable that is dependent on the interconnections between the components. Considerable savings in area can be obtained if we can place the components in such a way that the width of the layout is minimized. Thus, width minimization in linear layouts is another example of a layout permutation problem. In this chapter we develop algorithms for this problem with the help of a graph metric known as cutwidth.

6.1 The Cutwidth Metric

A circuit that is specified in terms of its components and their interconnections can be represented as a graph $G = (V, E)$ where the vertices in V correspond to the components of the circuit, and the edges E represent the interconnections between

the components. Since there may be multiple connections between two components, G can be a multigraph. In fact, we will also allow graphs with loops. Note that each edge contributes 1 to the degree of each end-point. Since both endpoints of a loop are identical, each loop contributes 2 to the degree.

Definition A *linear layout* of a graph G is a one to one mapping

$$L : V \rightarrow \{1, 2, \ldots, |V|\}.$$

In other words, each linear layout L is just a permutation of the vertices of G.

Definition The *cutwidth* of a graph G *under a linear layout* L, denoted $cutwidth_L(G)$, is the maximum number of edges cut by a vertical line drawn between consecutive vertices in the linear layout.

Definition The *cutwidth* of G, written $cutwidth(G)$, is the minimum value of $cutwidth_L(G)$ over all possible linear layouts L of G.

In [MaS], it is shown that subdividing an edge does not alter the cutwidth of the graph. Therefore, any loop or multiple edge can be simplified by inserting a new vertex into it without modifying the cutwidth of the graph. Thus it is possible to restrict attention only to simple graphs. Therefore, the cutwidth of a graph with loops is defined to be the same as the cutwidth of the loop-free graph obtained by introducing a vertex of degree two into each loop.

From this definition, we see that the minimum area required for a linear layout of a circuit can be determined from the cutwidth of the corresponding graph. The problem of computing the cutwidth of a graph is the optimization version of the decision problem called Minimum Cut Linear Arrangement, known to be $\mathcal{NP}$-Complete[GJ]. It is rather complicated to compute the cutwidth even for trees, as evinced by the algorithm of [Yk] which runs in $O(n \log n)$ time. Moreover, the problem remains $\mathcal{NP}$-Hard for trees with polynomial size edge weights and also for planar graphs with

maximum degree 3 (see [MoS]).

In the fixed parameter version, the width k is a fixed constant, and we would like to decide whether the cutwidth of the graph is at most k and if so, find such a layout. The dynamic programming algorithm of [MaS] which runs in time $O(n^{k-1})$, where $n = |V(G)|$, was the fastest algorithm known for the fixed parameter problem until recently. New developments in graph theory have yielded some novel tools to tackle the cutwidth problem and these will be our focus.

6.2 Cutwidth and the Immersion Order

In this section we show how the immersion order on graphs can be used to solve the cutwidth problem. We define CW(k) to be the set of all graphs whose cutwidth is at most k. The following lemma shows that CW(k) is closed in the immersion order. This was first pointed out in [FL5]. It was observed independently in [MaS], though without the notion of immersion containment.

Lemma 6.2.1 [FL5] If $H \leq_i G$ then $cutwidth(H) \leq cutwidth(G)$.

This property, in conjunction with the finite-basis characterization of Theorem 2.2.1 shows that $obs(\text{CW}(k))$ is a finite set. Moreover, Theorem 2.2.2 says that membership in CW(k) can be decided in polynomial time. It is known that for any k, there exists a binary tree T whose cutwidth exceeds k. Since every minor of T is planar, we know that for each k, there exists at least one planar obstruction for CW(k).

The following lemma shows that under certain circumstances, minor containment is a sufficient condition for immersion containment.

Lemma 6.2.2 [FL2] If H is a graph whose maximum degree does not exceed 3, then $H \leq_m G \Rightarrow H \leq_i G$.

Therefore, a graph that does not contain the binary tree obstruction to cutwidth in the immersion order does not contain it as a minor either.

Lemma 6.2.3 [RS3] If H is a fixed planar graph, then every graph with no minor isomorphic to H has treewidth bounded by a fixed constant c_H.

From this lemma it follows that the treewidth of the graphs in CW(k) is bounded by a constant. We can combine all these results into a linear-time algorithm for cutwidth.

Theorem 6.2.4 [FL5] For every fixed k, there exists a linear-time algorithm to decide whether a graph of order n has cutwidth at most k.

Proof Let T be a planar obstruction to CW(k). If $G \in$ CW(k), then $T \not\leq_i G$ which implies that $T \not\leq_m G$, and $treewidth(G) \leq c_T$. Given a graph G, we decide whether the treewidth of G is bounded by c_T. If not, then we know that $cutwidth(G) > k$. Otherwise, we find such a tree-decomposition [Bod1] and use the dynamic programming formulation of [RS3] to decide whether G contains an obstruction to CW(k) in the immersion order. ∎

Due to the enormous constants of proportionality involved, this algorithm is highly impractical. Hence there is a need for more practical algorithms for cutwidth.

6.3 The Weak Immersion Order

The work of [MaS] can be viewed as applicable to what has been called the weak immersion order.

Definition [Fe] A graph H is *weakly immersed* in G, written $H \leq_{wi} G$, if and only if a graph isomorphic to H can be obtained from G by taking subgraphs, splitting vertices, and removing subdivisions.

Vertex splitting is the operation of replacing an existing vertex with two new vertices and assigning the edges of the original vertex to the new vertices in an arbitrary fashion. Thus, there can be several different ways to split a vertex. Note that splitting a vertex of degree one produces an isolated vertex which is rather uninteresting for most applications. Therefore, we assume that only vertices of degree greater than one are split.

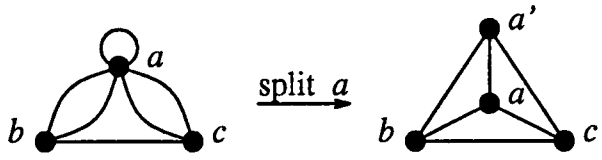


Figure 6.1: A graph with a weakly immersed K_4

Figure 6.1 shows a graph without an immersed K_4, one of whose vertices can be split to obtain a K_4.

It is not at first clear how the weak immersion order can be related to the immersion order. But note that any lift can be realized by a vertex splitting followed by a subdivision removal. Since $H \leq_i G \Rightarrow H \leq_{wi} G$, and graphs are well-quasi-ordered by immersion, it follows that they are also well-quasi-ordered by weak immersion. Makedon and Sudborough [MaS] showed that the cutwidth of a graph cannot be increased by taking subgraphs, splitting vertices, and removing subdivisions. In other words, CW(k) is also closed in the weak immersion order.

Lemma 6.3.1 [MaS] If $H \leq_{wi} G$ then $cutwidth(H) \leq cutwidth(G)$.

The weak immersion order has some interesting properties that distinguish it from the immersion and the minor orders. Observe that if $H \leq_m G$ or if $H \leq_i G$, then $|V(H)| \leq |V(G)|$ and $|E(H)| \leq |E(G)|$. However, when $H \leq_{wi} G$, we can only say that $|E(H)| \leq |E(G)|$ because it is possible that $|V(H)| > |V(G)|$ (see Figure 6.1, for example). In spite of this peculiarity, we show that there is a polynomial-time test

for weak immersion containment analogous to the test of Theorem 2.2.2 for minor and immersion containment.

The following is a well-known alternative characterization of immersion, which is very useful for algorithm design.

$H \leq_i G$ if and only if there exist

i. a one to one mapping $f : V(H) \rightarrow V(G)$, and

ii. $|E(H)|$ edge-disjoint paths of length at least 1 in G, each connecting the images of the end-points of a distinct edge in H.

Using this characterization, [FL5] proved the existence of a polynomial time algorithm to test for immersion containment as stated in Theorem 2.2.2. The time complexity of this algorithm is $O(n^{h+3})$, where $n = |V(G)|$ and $h = |V(H)|$. Using a characterization of weak immersion in terms of vertex mapping and edge-disjoint paths, we prove a similar result for weak immersion containment.

$H \leq_{wi} G$ if and only if there exist

i. a many to one mapping $w : V(H) \rightarrow V(G)$, and

ii. $|E(H)|$ edge-disjoint paths of length at least 1 in G, each connecting the images of the end-points of a distinct edge in H.

Note that G and H can be multigraphs with loops.

Theorem 6.3.2 For every fixed graph H, there exists a polynomial-time algorithm that when given an input graph G, decides whether H is weakly immersed in G.

Proof Let $n = |V(G)|$, $e = |E(G)|$, $h = |V(H)|$, and $k = |E(H)|$. Mader has shown that for every graph H there exists a constant c_H such that, if $|E(G)| > c_H|V(G)|$ then $H \leq_t G$ (see Chapter VII of [Bol]). We know that $H \leq_t G \Rightarrow H \leq_{wi} G$. Therefore, it is sufficient to consider only those graphs with $e \leq c_H n$.

There are n^h different mappings $w : V(H) \rightarrow V(G)$ to consider. Suppose a pair of adjacent vertices v_i and v_j of H are mapped to the same vertex v of G. If there is a loop available at v in G, then it can be counted as a path of length 1 between the images of v_i and v_j. But if there is no loop available at v in G, then we need a way to find a path of length at least 1 between them. For this, we specify an intermediate vertex v_m in G between the images of v_i and v_j and look for two edge-disjoint paths, one between $w(v_i)$ and v_m and, the other between v_m and $w(v_j)$.

Notice that there are n choices of the intermediate vertex v_m, for each pair of adjacent vertices (v_i, v_j) of H for which there is no loop available. Since it is possible for all h vertices of H to be mapped on to the same vertex of G, we might need k intermediate vertices in the worst case and there are n^k different ways of choosing these intermediate vertices.

Using the linear-time method described in [FL5], we then transform the input graph G into a graph G' so that, finding edge-disjoint paths between the images of vertices of H in G is equivalent to finding vertex disjoint paths between the corresponding vertices in G'. This only increases the problem size by a constant factor.

For each such mapping, we use the cubic time vertex-disjoint paths algorithm of [RS4], resulting in an overall time complexity of $O(n^{h+k+3})$. ∎

Note that this algorithm, besides being slower than the immersion testing algorithm of Theorem 2.2.2 by a factor of n^k, is also based on the disjoint paths algorithm of [RS4] which is highly impractical. This is not surprising for an algorithm that works for any fixed graph H. However, for the obstructions to cutwidth, it may be possible to devise practical weak immersion tests.

6.4 Obstructions to Cutwidth

In this section, we take a closer look at obstructions to cutwidth with a view to developing fast immersion tests for them. The first characterization of cutwidth in

terms of obstructions was found by Makedon and Sudborough, who showed that there are exactly five obstructions to CW(2) in the weak immersion order.

Lemma 6.4.1 [MaS] A graph G has cutwidth at most 2 if and only if G does not contain any of the five graphs of Figure 6.2 in the weak immersion order.

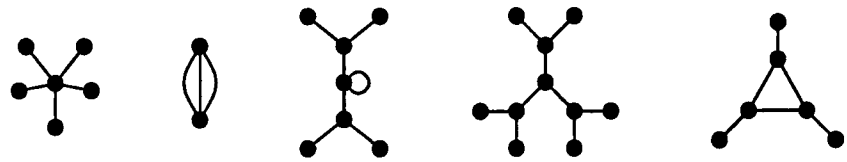


Figure 6.2: The obstruction set for CW(2) in the weak immersion order

Since CW(k) is closed in both the immersion and the weak immersion orders, we shall use $obs(\text{CW}(k), \leq_i)$ to denote the obstruction set for CW(k) in the immersion order and $obs(\text{CW}(k), \leq_{wi})$ for the obstruction set in the weak immersion order.

Lemma 6.4.2 If $G \in obs(\text{CW}(k), \leq_{wi})$, then $G \in obs(\text{CW}(k), \leq_i)$.

Proof Suppose $G \in obs(\text{CW}(k), \leq_{wi})$ and $H <_i G$. Then $H <_{wi} G$ also, and as a result, $cutwidth(H) < cutwidth(G)$. We know that $cutwidth(G) > k$. Therefore, $G \in obs(\text{CW}(k), \leq_i)$. ∎

Thus, the five graphs of Figure 6.2 are obstructions to CW(2) in the immersion order as well. We believe that in general for any fixed k, there are many more obstructions to CW(k) in the immersion order than in the weak immersion order. This is certainly true for CW(2). The number of obstructions to CW(3) in the immersion order is conjectured to be 464 by Kinnersley (see [Ki3]). It would be interesting to see how many obstructions exist for CW(3) in the weak immersion order. If $|obs(\text{CW}(k), \leq_{wi})|$ is significantly less than $|obs(\text{CW}(k), \leq_i)|$, then the additional time taken by the algorithm of Theorem 6.3.2 in comparison to the immersion testing algorithm of Theorem 2.2.2 might be justified. Hence, we shall continue to study the cutwidth problem under both the orders.

The simplest obstruction to CW(k) is the $(k+1)$-bond B_{k+1} which plays a role similar to that of the complete graph for pathwidth and treewidth. Interestingly, the complete graph K_n is also an obstruction for cutwidth.

Lemma 6.4.3 For every $k \geq 2$, the $(k+1)$-bond B_{k+1} is an obstruction to CW(k) in the weak immersion order.

Proof It is clear that $cutwidth(B_{k+1}) = k+1$. Let $H <_{wi} B_{k+1}$. If $H \subset B_{k+1}$, then either $|V(H)| = 1$ or $H \subseteq B_k$ and in either case, $cutwidth(H) \leq k$. Notice that there is no subdivided edge in B_{k+1}. Suppose H is obtained by splitting a vertex of B_{k+1}. Then, $|V(H)| = 3$ and since splitting does not increase the number of edges in the graph, it is easy to obtain a linear layout of cutwidth at most k for H. ∎

The following lemma gives a simple upper bound on the maximum degree of a graph with bounded cutwidth.

Lemma 6.4.4 [MaS] If a graph G has a vertex of degree m, then $cutwidth(G) \geq \lceil \frac{m}{2} \rceil$.

This lemma inspires another simple and general obstruction to CW(k) namely, the $2k+1$-pointed star $K_{1,2k+1}$.

Lemma 6.4.5 For every $k \geq 2$, the $2k+1$-pointed star $K_{1,2k+1}$ is an obstruction to CW(k) in the weak immersion order.

Proof Since $K_{1,2k+1}$ has a vertex of degree $2k+1$, its cutwidth is $k+1$. Let $H <_{wi} K_{1,2k+1}$. If $H \subset K_{1,2k+1}$, then it is simple to obtain a linear layout of cutwidth k for H. There are no subdivided edges in $K_{1,2k+1}$. Suppose H was obtained by splitting a vertex of $K_{1,2k+1}$. Since only the central vertex of the star can be split, H consists of two connected components $K_{1,x}$ and $K_{1,y}$, where $1 \leq x < y < 2k+1$, and $x + y = 2k+1$. Therefore, $cutwidth(H) \leq \lceil \frac{y}{2} \rceil < \lceil \frac{2k+1}{2} \rceil = k+1$. ∎

Therefore, if $K_{1,m} \leq_{wi} G$ then $cutwidth(G) \geq \lceil \frac{m}{2} \rceil$.

Lemma 6.4.6 For every $n \geq 4$, K_n is an obstruction to CW($\lceil \frac{n}{2} \rceil \lfloor \frac{n}{2} \rfloor - 1$) in the weak immersion order.

Proof From the symmetry of K_n, it is easy to see that any linear layout of K_n has cutwidth $\lceil \frac{n}{2} \rceil \lfloor \frac{n}{2} \rfloor$. It remains to show that if $H <_{wi} K_n$ then $cutwidth(H) < cutwidth(K_n)$. Let $v_1, v_2, \ldots, v_n$ be a linear layout of the vertices of K_n. Suppose $H \subset K_n$. If H was obtained by deleting a vertex of K_n, then $H \subseteq K_{n-1}$ and as a result $cutwidth(H) \leq cutwidth(K_{n-1}) < cutwidth(K_n)$. On the other hand if H was obtained by deleting an edge of K_n, then due to the symmetry of H, we can assume without loss of generality that the edge (v_1, v_n) was the one deleted. The resulting layout clearly has cutwidth one less than that of K_n. Since K_n has no subdivided edges, the only other way to obtain $H \subset K_n$ is by splitting a vertex.

Assume without loss of generality that H is obtained by splitting v_n into v_n and v_{n+1}. In H, let degree$(v_n) = x$ and degree$(v_{n+1}) = y$, where $1 \leq x < y < n - 1$ and $x + y = n - 1$. Further, let $N(v_n) = \{v_1, \ldots, v_x\}$ and $N(v_{n+1}) = \{v_{x+1}, \ldots, v_{n-1}\}$. Consider the linear layout $v_n, v_1, v_2, \ldots, v_x, v_{x+1}, \ldots, v_{n-1}, v_{n+1}$ of the vertices of H. Since the subgraph induced by vertices $v_n, v_1, v_2, \ldots, v_{n-1}$ of H is a proper subgraph of K_n, a vertical line between any two vertices to the left of v_x in the linear layout of H will cut less than $\lceil \frac{n}{2} \rceil \lfloor \frac{n}{2} \rfloor$ edges. The same can be said for a vertical line between any two vertices to the right of v_{x+1} because the subgraph induced by vertices $v_1, v_2, \ldots, v_{n-1}, v_{n+1}$ is also a proper subgraph of K_n. The number of edges cut by a vertical line between v_x and v_{x+1} is at most the $cutwidth(K_{n-1})$. Therefore $cutwidth(H) < cutwidth(K_n)$. ∎

As a consequence of Lemma 6.4.2, Lemmas 6.4.3, 6.4.5, and 6.4.6 hold trivially for the immersion order as well.

Recall that multiple edges are allowed in the cutwidth problem. By lifting a pair of edges that share the same two end-points, we can also obtain a loop. Allowing loops in a graph leads to a curious phenomenon in the obstruction set. For odd k, the

graph L_k of order 1 with k loops is in both $obs(\mathrm{CW}(k),\leq_i)$ and $obs(\mathrm{CW}(k-1),\leq_i)$. This is because when k is odd, $cutwidth(L_k) = k+1$, whereas for every $H <_i L_k$, $cutwidth(H) \leq k-1$. In the weak immersion order however, for odd k, L_k is an obstruction only for $\mathrm{CW}(k)$ and not for $\mathrm{CW}(k-1)$.

6.5 Testing for Cutwidth Obstructions

In this section, we shall develop practical algorithms to test for some of the important general obstructions to cutwidth such as the bond, star, and clique, both in the immersion order as well as the weak immersion order. From Theorem 2.2.2 and Theorem 6.3.2 we already know that there exist polynomial-time tests for immersion and weak immersion containment. However, our aim now is to design algorithms that are fast in practice. Towards this end, we also develop some general tools.

Earlier, in Lemma 6.4.2 we showed that the number of obstructions for $\mathrm{CW}(k)$ in the weak immersion order never exceeds the number in the immersion order. In fact, the difference in their sizes is large even for $k = 2$. Therefore, we would like to compare these two orders in terms of the ease of developing practical algorithms to test for obstructions.

Before proceeding further, we clarify the difference between immersion testing and weak immersion testing. If L_k is the graph of order 1 with k loops, then $B_k <_{wi} L_k$, whereas $B_k \not\leq_i L_k$. Similarly, $K_{1,2k+1} <_{wi} B_{2k+1}$ and $K_{1,2k+1} \not\leq_i B_{2k+1}$. Also, $K_n <_{wi} B_{(n-1)^2}$ but $K_n \not\leq_i B_{(n-1)^2}$. Therefore, for each of these three obstructions, testing for immersion containment is not equivalent to testing for weak immersion containment.

In general, for a multigraph G with loops, it is easy to see that $K_{1,m} \leq_{wi} G$ if and only if G has a vertex on which at least m edges are incident. If G is a simple graph, then $K_{1,m} \leq_i G$ if and only if G has a vertex on which at least m non-loop edges are incident. However, if G is a multigraph, then the immersion test for $K_{1,m}$ is not as simple. In this case, all we can say is that $K_{1,m} \leq_i G$ if and only if there exists a

vertex v in G with at least m edge-disjoint paths leading to m other vertices in G. Notice however that if G contains $K_{1,m}$ in either order, then there exists a vertex of degree at least m in G and according to Lemma 6.4.4, $cutwidth(G) \geq \lceil \frac{m}{2} \rceil$. Therefore in practice, we never have to test explicitly for $K_{1,m}$.

Testing for the other obstructions for cutwidth is also more difficult. First we develop tests for the k-bond and then we address the complete graphs.

6.5.1 Bonds

We can characterize a graph that contains a k-bond as follows. $B_k \leq_i G$ if and only if there exists a pair of vertices with at least k edge-disjoint paths between them in G. By computing the number of edge-disjoint paths between each pair of vertices in G, we can determine the largest k for which a $B_k \leq_i G$. In fact, this value of k is a good lower bound on the cutwidth of G.

For each pair of vertices in G, we designate one as the source and the other as the sink and compute the network flow from the source to the sink. The value of the maximum flow between a pair of vertices equals the number of edge-disjoint paths between them. Therefore, by solving $\frac{n(n-1)}{2}$ network flow problems on G, we can determine the size of the largest bond immersed in G. Gomory and Hu [GH] have shown that the flow values between all pairs of vertices can be computed by solving only $n-1$ network flow problems on G. Their algorithm, which has been greatly simplified by Gusfield [Gu], is easily implemented by a minor modification of any maximum flow algorithm that also computes the minimum cut. At present, the fastest algorithm to compute the maximum flow is the one in [KRT] which takes $O(ne + n^{2+\epsilon})$ time, where ϵ is a positive constant.

The following lemma gives us a way to test for weak immersion of bonds.

Lemma 6.5.1 $B_k \leq_{wi} G$ if and only if either $B_k \leq_i G$ or $L_k \leq_i G$.

Proof

($\Rightarrow$): Recall the characterization of weak immersion in terms of edge-disjoint paths used in Theorem 6.3.2. If each of the two vertices of B_k maps to a distinct vertex of G, then $B_k \leq_{wi} G \Rightarrow B_k \leq_i G$. On the other hand if both vertices of B_k map to the same vertex of G, then $B_k \leq_{wi} G \Rightarrow L_k \leq_i G$.

($\Leftarrow$): This implication is obvious. ∎

Testing whether $L_k \leq_i G$ is equivalent to deciding whether there is a vertex in G that lies on k edge-disjoint cycles. Unfortunately, we are not aware of a fast algorithm for this problem. Therefore, we do not have a practical algorithm to decide whether $B_k \leq_{wi} G$.

6.5.2 General Tools for Weak Immersion Testing

We present some general tools for weak immersion testing that can also be used for immersion tests. Graph edge-connectivity is the basis for all of them. Before we can demonstrate this, we need some terminology. Let G be a multigraph. An edge whose removal disconnects G is called a *cut-edge*. A pair of edges, neither of which is a cut-edge, and whose removal disconnects G is called a *cut-edge-pair* of G. For a positive integer k, two vertices are said to be *k-edge-connected* if there exist at least k edge-disjoint paths between them. The vertices of a graph are partitioned by k-edge-connectivity into equivalence classes called *k-edge-connected components*. A graph G is said to be k-edge-connected if and only if every pair of vertices of G is k-edge-connected.

The following lemma shows where a weakly immersed 2-edge-connected graph will be located.

Lemma 6.5.2 A 2-edge-connected graph H is weakly immersed in a graph G if and only if H is weakly immersed in the subgraph induced by a 2-edge-connected component of G.

Proof The proof is by contradiction. Assume that $F \leq_{wi} G$ where $F \cong H$. We

know that if $F \leq_{wi} G$, then there exists a many to one mapping of $V(F)$ to $V(G)$, and $E(F)$ is mapped to edge-disjoint paths in G. Suppose vertices v_1 and v_2 of $V(F)$ are mapped to v_1' and v_2' of $V(G)$ respectively, where v_1' and v_2' are in different 2-edge-connected components of G. Since F is 2-edge-connected, v_1' and v_2' must be 2-edge-connected in G. However, since v_1' and v_2' are in different 2-edge-connected components of G, this is not possible and we have a contradiction. ∎

We can extend this idea of locality to 3-edge-connected graphs. An *augmented component* of G is a graph induced by a 3-edge-connected component $V' \subseteq V(G)$ of G together with a set of virtual edges, one for each pair of vertices in V' that are incident on the same cut-edge pair of G. Figure 6.3 shows a 2-edge-connected graph G and its decomposition into augmented components. Notice the virtual edges (shown as dotted edges) in the augmented components G_1 and G_2.

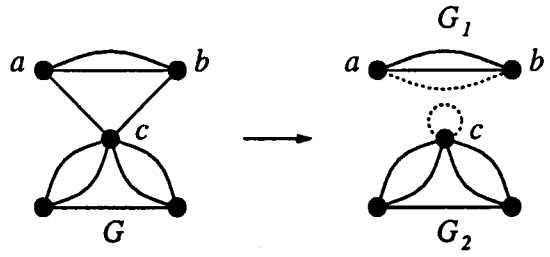


Figure 6.3: A 2-edge-connected graph and its augmented components

Each graph has a unique decomposition into augmented components, which can be computed in linear-time [FRT].

Theorem 6.5.3 A 3-edge-connected graph H is weakly immersed in a graph G if and only if H is weakly immersed in some augmented component of G.

Proof

($\Rightarrow$): The proof is by induction on the number of augmented components of G. If G is itself 3-edge-connected, then the lemma is trivial. Suppose G is not 3-edge-connected. After Lemma 6.5.2, we may safely assume that G is 2-edge-connected.

Let (a, c) and (b, d) be a cut-edge-pair in G. Deleting these two edges splits G into exactly two connected components, say G_1 and G_2. Assume that vertices a and b are in G_1 whereas c and d are in G_2 (note that a and b may be identical, and so can c and d). Since there is exactly one path between a and b that contains c or d, and there is only one path between c and d that contains a or b, by introducing a virtual edge (a, b) in G_1 and (c, d) in G_2, we obtain graphs G_1' and G_2' which are also 2-edge-connected.

Now we show that H is weakly immersed in one of G_1' or G_2'. Suppose $F \leq_i G$ where $F \cong H$. Recall the characterization of weak immersion in terms of vertex-mapping and edge-disjoint paths between the images of the vertices. Since there are at most two edge-disjoint paths between a vertex in G_1 and another in G_2, it is clear that $V(F)$ is mapped either to $V(G_1)$ or to $V(G_2)$. Without loss of generality, we assume that $V(F)$ is mapped to $V(G_1)$. Suppose some edge of F is mapped to a path in G that contains the edge (a, c). Since both end-points of this path are in $V(G_1)$, it is clear that the path must also include the edge (b, d). In G_1', the portion of this path consisting of edges $(a, c), (b, d)$ and any edges in G_2 can be replaced by the virtual edge (a, b). Therefore, $F \leq_{wi} G \Rightarrow F \leq_{wi} G_1'$. Since each of G_1' and G_2' has fewer augmented components than G itself, this completes the induction.
($\Leftarrow$): This implication follows from the fact that each augmented component of a graph G is also weakly immersed in G. ∎

More generally, it seems that if a k-edge-connected graph H is weakly immersed in G, then all the vertices of H map to the same k-edge-connected component of G. Edge-connectivity is particularly useful in practice because of the availability of fast algorithms to determine the edge-connectivity (see [Ga] for instance), and also to find all the minimal edge cuts [KaT]. One use of these tools will be seen in the next section.

6.5.3 An Algorithm for K_4 Immersion Testing

We describe an algorithm to test for immersion containment of K_4, which is the smallest complete graph that is an obstruction to cutwidth. This algorithm, originally presented in [BGLR], is the first truly practical, non-trivial algorithm for immersion testing. Observe that even for a graph as small as K_4, immersion testing is not equivalent to minor testing. This is illustrated in Figure 6.4.

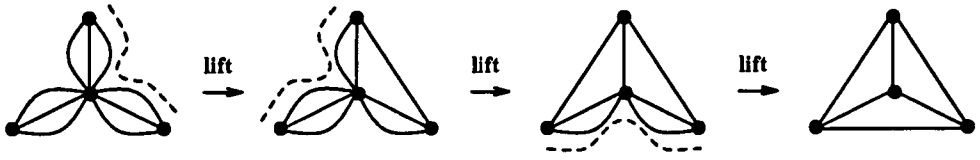


Figure 6.4: A graph with an immersed K_4 but no K_4 minor

It is clear that multiple edges play an important role in immersion testing, whereas they do not matter in the minor order. Thus, immersion testing is quite different from minor testing. Note that loops do not matter for immersion testing of complete graphs.

One consequence of Lemma 6.2.2 is that, we can use the minor test for K_4 as a preliminary test for K_4 immersion. Thus we can restrict attention to graphs that do not contain a K_4 minor, which are just the series-parallel graphs.

Since K_4 is 3-edge-connected, according to Theorem 6.5.3, we need to consider only the augmented components of G. The following lemma enables us to take advantage of Lemma 3.4.1.

Lemma 6.5.4 [BGLR] Each augmented component of a series-parallel graph is also series-parallel.

Algorithm IMMERSION-TEST, shown in Figure 6.5, takes a multigraph G as input and decides whether $K_4 \leq_i G$.

Lemma 6.5.5 [BGLR] Algorithm IMMERSION-TEST correctly decides whether K_4 is immersed in an input multigraph.

Algorithm IMMERSION-TEST

Input : A connected multigraph G
Output: YES, if $K_4 \leq_i G$
NO, otherwise.
begin procedure
1. Read the input graph G as a simple graph with
integer edge weights indicating the multiplicity.
2. **if** $K_4 \leq_m G$ **then** output YES and **stop**;
3. Decompose G into augmented components.
4. **for** (each augmented component C of G) **do**
begin
for (each vertex v in C with exactly one neighbor) **do**
delete all but three copies of edges incident on v;
if any cut point in C has degree ≥ 7 **then** output YES and **stop**;
for (each biconnected component B with four or more vertices) **do**
begin
for (each vertex v with exactly two neighbors u and w) **do**
begin
if multiplicity(uv)=1 **then** multiplicity(vw) $\leftarrow$ 2;
if multiplicity(vw)=1 **then** multiplicity(uv) $\leftarrow$ 2;
end;
if there is a vertex in B with degree ≥ 5 **then**
output YES and **stop**;
end;
end;
5. Output NO and **stop**;
end procedure.

Figure 6.5: Algorithm IMMERSION-TEST

Lemma 6.5.6 [BGLR] Algorithm IMMERSION-TEST runs in $O(n)$ time, where $n = |V(G)|$.

For series-parallel graphs there is a much simpler alternative to the algorithm of [FRT], which also computes the augmented components in $O(n)$ time. Since the rest of the steps are quite simple, the entire algorithm IMMERSION-TEST is easy to implement and is very fast in practice. Both sequential and parallel versions of this algorithm, along with algorithms to find a model of a K_4 when one exists, are

described in detail in [BGLR].

A natural question that arises is whether there is a fast algorithm to test for K_4 containment in the weak immersion order. Even though the minor order is a generalization of the topological order, for graphs such as K_5 it is easier to test for minor containment than topological containment. However, generalization may not always lead to easier tests. For instance, testing for weak immersion of K_4 seems harder than just immersion testing. Figure 6.1 on page 115 is an example of a graph with a weakly immersed K_4 but no immersed K_4. It is conceivable that the weak immersion order, when viewed in an entirely different way from the immersion order, will yield simpler algorithms. At this time, it also seems considerably harder to devise immersion tests for K_5 and larger complete graphs.

6.5.4 Characterizing Graphs that Exclude K_4

Graphs without a K_4 minor have a natural structure, namely, they are series-parallel. In the immersion order however, we were only able to characterize 4-edge-connected graphs that do not contain K_4.

Theorem 6.5.7 For every 4-edge-connected multigraph G, either

i K_4 is immersed in G, or

ii G is a double cycle (i.e. the simple graph underlying G is a cycle and each edge of G has multiplicity exactly two).

Proof It is obvious that any double cycle is 4-edge-connected. It is straightforward to verify that double cycles also exclude K_4 in the immersion order.

The rest of the proof is by induction on the number of vertices n of G. G is a 4-edge-connected graph that does not contain an immersed K_4. It is easy to verify that a double cycle with 4 vertices is the smallest non-trivial 4-edge-connected graph

that excludes K_4. Suppose the lemma is true for all $4 \leq n \leq k$ and now $n = k + 1$. Since $K_4 \not\leq_i G \Rightarrow K_4 \not\leq_m G$, G is series-parallel and has a vertex v with at most two neighbors. If v has only one neighbor w in G, then deleting v gives us a graph G' of order k that is also 4-edge-connected and excludes K_4. Therefore, G' should be a double cycle according to the induction hypothesis. But this implies that the graph shown on the left in Figure 6.6 is immersed in G and as a result $K_4 \leq_i G$, contradicting our initial premise. Therefore, G has no vertices with only one neighbor.

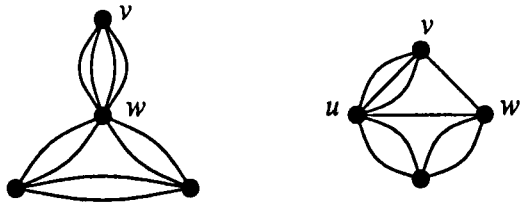


Figure 6.6: K_4 immersion in a 4-edge-connected graph

Suppose v has two neighbors u and w in G with $multiplicity(u,v) \geq multiplicity(v,w)$. Let G' be the graph obtained from G by lifting all possible pairs of edges (u,v) and (v,w) and then deleting the remaining edges (u,v) and the vertex v. Since $G' \leq_i G$, we know $K_4 \not\leq_i G'$. G' is also 4-edge-connected because the number of edge-disjoint paths between any two vertices in G' is exactly the same as in G. Since $|V(G')| = k$, it follows from the induction hypothesis that G' is a double cycle.

We claim that the edge (u,w) does not exist in G. Since G is 4-edge-connected, we know that $multiplicity(u,v) \geq 2$. Let x and u be the neighbors of w in G'. If the edge (u,w) exists in G, then the graph shown on the right in Figure 6.6 is immersed in G and as a result $K_4 \leq_i G$. Hence we conclude that $(u,w) \notin E(G)$. Since $multiplicity(u,w)$ in G' is 2, we know that in G, $multiplicity(v,w) = 2$. Moreover, in G, $multiplicity(u,v)$ is also 2 because, otherwise $K_4 \leq_i G$. Therefore, G is a double cycle and the induction is complete. ∎

We can show that this theorem holds for weak immersion as well.

Corollary 6.5.8 For every 4-edge-connected multigraph G, either

i. K_4 is weakly immersed in G, or

ii. G is a double cycle.

Proof The proof of (i) is based on Theorem 6.5.7 and the fact that $K_4 \leq_i G \Rightarrow K_4 \leq_{wi} G$. It is easy to verify (ii) i.e., double cycles do not contain K_4 in the weak immersion order. ∎

Such simple characterizations for larger classes of graphs that exclude K_4 could lead to simpler algorithms for immersion and weak immersion testing.

6.6 Summary

In this chapter, we surveyed some linear layout problems that arise in VLSI design and saw that the width of such a layout can be modeled in terms of the cutwidth metric of a graph. We also saw that the cutwidth problem is closed under both the immersion and the weak immersion orders. Unlike the minor order, the immersion order has not received much attention in the past. We presented some general obstructions to CW(k) and devised fast tests for them. One of the curious phenomena that we observed in this chapter is the contrast between the immersion order and the weak immersion order for the cutwidth problem. While the number of obstructions to cutwidth k in the weak immersion order seems to be smaller than the number in the immersion order, testing for weak immersion appears to be harder than testing for immersion. Even though we were able to find a polynomial-time algorithm to test for weak immersion of any fixed graph, it is much slower than the best algorithm for immersion testing. Therefore, it would be interesting to further explore problems such as cutwidth and determine which of these two orders is the most useful in practice.

Chapter 7

Conclusions

7.1 Summary

In this dissertation, we developed algorithms for some layout permutation problems in VLSI design using graph width metrics. We began by describing a novel approach to the design of decision algorithms for fixed-parameter problems. This method is based on knowing the set of obstructions for the problem.

We demonstrated our approach for Gate Matrix Layout in 3 or fewer tracks. Linear-time algorithms were developed to test for minor containment of the six smallest and densest obstructions for GML(3). Using these tests we designed a decision algorithm for GML(3). We also designed three different search algorithms all based on a technique called self-reduction. Our layout algorithms are novel in that they only test for a few of the many obstructions and in their use of self-reduction.

All our algorithms have been implemented and are very fast in practice. We tested our algorithms on large numbers of randomly generated inputs. The results reveal that the algorithms are almost always able to find a 3-track layout if one exists. We also presented a family of inputs for which many published heuristics produce egregiously bad results. Our algorithms produce good layouts for these instances. This shows that obstruction based methods and the self-reduction technique may work where other methods fail.

Next we explored ways to estimate the pathwidth metric of a graph quickly. We developed several new lower and upper bounds for treewidth and pathwidth. Fast algorithms were developed to compute many of these bounds.

We introduced a new metric γ of a graph, and showed that it is a lower bound on treewidth and pathwidth. A linear-time algorithm to compute γ was also developed. We then presented a characterization of graphs with bounded treewidth or pathwidth in terms of a collection of interconnected complete bipartite subgraphs in the complement. This characterization enabled us to derive more lower bounds for pathwidth. Fast algorithms to compute some of these bounds were also presented.

We then studied the obstruction sets for pathwidth and treewidth. Some of the lower bounds that we developed were used to characterize and capture many of the obstructions. We presented the first general method to construct explicitly some of the densest obstructions (besides the cliques) for pathwidth and treewidth. These graphs of order $k+3$ are obstructions for both $\mathrm{PW}(k)$ and $\mathrm{TW}(k)$. Moreover, this is the first general method to construct non-trivial obstructions to $\mathrm{TW}(k)$. Even though these new obstructions are very small and dense, we showed that their number is an exponential function dependent on k, unlike the solitary clique. This also provided the first exponential lower bound for the number of obstructions for $\mathrm{TW}(k)$.

We explored the applicability of our general approach to linear layout problems in VLSI design. In particular, we used the cutwidth metric of graphs to develop algorithms for the minimum cut linear arrangement problem. We presented a polynomial-time algorithm to test for weak immersion containment of any fixed graph H in an input graph G. We also showed that the obstructions for cutwidth in the weak immersion order are obstructions in the immersion order as well. We identified several of the small dense obstructions for cutwidth in the weak immersion order and developed fast algorithms to test for them. Some general tools for developing fast containment tests were also presented. As an application of these general tools, a linear-time algorithm

to find an immersed K_4 was presented.

7.2 Directions for Future Research

Our study has shown that graph width metrics and obstruction-based algorithms can be effective in producing good results fairly quickly. More research is required before our general approach can be used to design truly practical algorithms for fixed-parameter problems with large parameter values.

One of the obstacles in developing better VLSI layout tools is the lack of practical decision algorithms for graph width metrics. Therefore, better tools for obstruction testing can be very helpful. Better bounds for graph width metrics such as treewidth, pathwidth, and cutwidth are also important. It appears to be very difficult to derive good upper bounds for pathwidth. For a graph of order n, we were only able to obtain upper bounds of the form $n - c$ for some constant c. It would be worthwhile to develop upper bounds that are independent of n or at least have the form cn for some constant $0 < c < 1$.

Studying the structure and number of obstructions can help in two ways. It can help design better decision algorithms, and it can lead to better bounds. Even though we have completely characterized the obstructions of order $k+3$ for $\mathrm{PW}(k)$, we were unable to devise a fast test for them. It is not known if such simple characterizations can be found for other large families of obstructions. Such characterizations may help us to devise fast tests for entire families of structurally related obstructions. This would be one way to tackle the large number of obstructions that are known to exist.

The relation between the immersion and weak immersion orders is interesting. Algorithms for linear layout problems in VLSI that are based on the cutwidth metric can make use of either order. An important open question is the difference in the number of obstructions for cutwidth under immersion and weak immersion. We

conjecture that there are far fewer obstructions in the weak immersion order. If this were indeed the case, then we would also need faster tests for weak immersion containment.

Bibliography

Bibliography

[ACP] S. Arnborg, D. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM Journal on Algebraic and Discrete Methods* 8 (1987), 277–284.

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[An] G. E. Andrews, The Theory of Partitions, in Gian-Carlo Rota, Editor, Encyclopedia of Mathematics and its Applications, Vol. 2, Addison-Wesley, Reading, MA, 1976.

[AP] S. Arnborg and A. Proskurowski, "Linear time algorithms for $\mathcal{NP}$-hard problems restricted to partial k-trees," *Discrete Applied Mathematics* 23 (1989), 11–24.

[APC] S. Arnborg, A. Proskurowski, and D. Corneil, "Forbidden minors characterization of partial 3-trees," *Discrete Mathematics* 80(1990), 1–19.

[BFL] D. J. Brown, M. R. Fellows, and M. A. Langston, "Polynomial-Time self-reducibility: theoretical motivations and practical results," *International Journal of Computer Mathematics* 31 (1989), 1–9.

[BG] M. Bellare and S. Goldwasser, "The complexity of decision versus search," *SIAM Journal on Computing* 23:1 (1994), 97-119.

[BGHK] H. Bodlaender, J. Gilber, H. Hafsteinsson, and T. Kloks, "Approximating treewidth, pathwidth, and minimum elimination tree height," Technical Report, Utrecth University, The Netherlands, January 1991.

[BGLR] H. Booth, R. Govindan, M. Langston, and S. Ramachandramurthi, "Sequential and parallel algorithms for K_4 immersion testing", Technical Report, Department of Computer Science, University of Tennessee, February 1994.

[BK1] H. L. Bodlaender and T. Kloks, "Better algorithms for the pathwidth and treewidth of graphs," *Proceedings, 18th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science 510 (1991), 544–555.

[BK2] H. L. Bodlaender and T. Kloks, "Efficient and constructive algorithms for the pathwidth and treewidth of graphs," Technical Report, Utrecht University, The Netherlands (1993).

[BL] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms," *Journal of Computer and Systems Science* 13 (1976), 335–379.

[BM] H. L. Bodlaender and R. Möhring, "The pathwidth and treewidth of cographs," *SIAM Journal on Discrete Mathematics* 6:2 (1993), 181–188.

[Bod1] H. L. Bodlaender, "A linear time algorithm for finding tree-decompositions of small treewidth," *Proceedings, 25th Annual ACM Symposium on Theory of Computing* (1993), 226–234.

[Bod2] H. L. Bodlaender, "A tourist guide through treewidth," *Acta Cybernetica* 11 (1993), 1–23.

[Bol] B. Bollobás, "Extremal Graph Theory," Academic Press, New York (1978).

[CH] C. Y. R. Chen and C. Y. Hou, "A new algorithm for CMOS gate matrix layout," *Proceedings, International Conference Computer-Aided Design* (1988), 138–141.

[CNSD] H. Cai, S. Note, P. Six, and H. De Man, "A data path layout assembler for high performance DSP circuits," *Proceedings, 27th Design Automation Conference* (1990), 306–311.

[Co] S. A. Cook, "The complexity of theorem-proving procedures," *Proceed-*

ings, 3rd Annual ACM Symposium on Theory of Computing (1971), 151–158.

[Di] G. A. Dirac, "Homomorphism theorems for graphs," *Mathematische Annalen* 153 (1964), 69–80.

[DKL] N. Deo, M. S. Krishnamoorthy, and M. A. Langston, "Exact and approximate solutions for the Gate Matrix Layout problem," *IEEE Transactions on Computer-Aided Design* 6 (1987), 79–84.

[DN] S. Devadas and R. Newton, "Topological optimization of multiple-level array logic," *IEEE Transactions on Computer-Aided Design* 6 (1987), 915–940.

[Fe] M. Fellows, "The Robertson-Seymour theorems: A survey of applications," *Contemporary Mathematics* 89 (1989), 1–17.

[FG] D. R. Fulkerson and O. A. Gross, "Incidence matrices and interval graphs," *Pacific Journal of Mathematics*, 15:3 (1965), 835–855.

[FHKY] T. Fujii, H. Horikawa, T. Kikuno, and N. Yoshida, "A heuristic algorithm for gate assignment in one-dimensional array approach," *IEEE Transactions on Computer-Aided Design* 6 (1987), 159–164.

[FL1] M. R. Fellows and M. A. Langston, "Nonconstructive advances in polynomial-time complexity," *Information Processing Letters*, 26 (1987/88), 157–162.

[FL2] M. R. Fellows and M. A. Langston, "Nonconstructive tools for proving polynomial-time decidability," *Journal of the ACM*, 35:3 (1988), 727–739.

[FL3] M. R. Fellows and M. A. Langston, "An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations," *Proceedings, 30th Annual Symposium on Foundations of Computer Science* (1989), 520–525.

[FL4] M. R. Fellows and M. A. Langston, "Fast search algorithms for layout permutation problems," *International Journal of Computer Aided VLSI Design* 3 (1991), 325–341.

[FL5] 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* 5:1 (1992), 117–126.

[FL6] M. R. Fellows and M. A. Langston, "On search, decision and the efficiency of polynomial-time algorithms," *Journal of Computer and System Sciences* (to appear). (A preliminary version of this paper can be found in Proceedings, 21st Annual ACM Symposium on Theory of Computing (1989), 501–512.

[FRS] H. Friedman, N. Robertson, and P. Seymour, "The metamathematics of the graph minor theorem," *Contemporary Mathematics* 65 (1987), 229–261.

[FRT] D. Fussell, V. Ramachandran, and R. Thurimella, "Finding triconnected components by local replacement," *SIAM Journal on Computing* 22:3 (1993), 587–616.

[Ga] H. Gabow, "A matroid approach to finding edge connectivity and packing arborescences," *Proceedings, 23rd Annual ACM Symposium on the Theory of Computing* (1991), 112–122.

[GCT] S. Goto, I. Cederbaum, and B. S. Ting, "Suboptimum solution of the backboard ordering with channel capacity constraint," *IEEE Transactions on Circuits and Systems* 24:11 (1977), 645–652.

[GH] R. E. Gomory and T. C. Hu, "Multi-terminal network flows," *SIAM Journal on Applied Mathematics* 9 (1961), 551–570.

[GJ] M. R. Garey and D. S. Johnson, "Computers and Intractability:

A Guide to the theory of NP-Completeness," W. H. Freeman and Company, New York, 1979.

[Go] M. C. Golumbic, "Algorithmic Graph Theory and Perfect Graphs," Academic Press, New York, 1980.

[GRS] R. L. Graham, B. L. Rothschild, and J. H. Spencer, "Ramsey Theory," John Wiley & Sons, New York, 1980.

[Gu] D. Gusfield, "Very simple methods for all pairs network flow analysis," *SIAM Journal on Computing* 19:1 (1990), 143–155.

[HFK] D. K. Hwang, W. K. Fuchs, and S. M. Kang, "An efficient approach to gate matrix layout ," *IEEE Transactions on Computer-Aided Design* 5 (1987), 802–808.

[HPK] Y-S. Hong, K-H. Park, and M. Kim, "Heuristic algorithms for ordering the columns in one-dimensional logic arrays," *IEEE Transactions on Computer-Aided Design* 8 (1989), 547–562.

[HS] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proceedings, 8th Design Automation Workshop* (1971), 155–169.

[HW1] S. Huang and O. Wing, "Gate matrix partitioning," *IEEE Transactions on Computer-Aided Design* 8 (1989), 756–767.

[HW2] S. Huang and O. Wing, "Improved gate matrix layout," *IEEE Transactions on Computer-Aided Design* 8 (1989), 875–889.

[IO] M. J. Irwin and R. M. Owens, "A comparison of four two-dimensional Gate Matrix tools," *Proceedings, Design Automation Conference* (1989), 698–701.

[Ja] I. T. Jakobsen, "On certain homomorphism properties of graphs II," *Mathematica Scandinavica* 52 (1983), 229–261.

[Ka] S. Kang, "Linear ordering and application to placement," *Proceedings, 20th Design Automation Conference* (1983), 457–464.

[KaT] A. V. Karzanov and E. A. Timofeev, "Efficient algorithm for finding all minimal edge cuts of a nonoriented graph," *Cybernetics* 22:2 (1986), 156–162.

[KF] T. Kashiwabara and T. Fujisawa, "NP-Completeness of the problem of finding a minimum-clique-number interval graph containing a given graph as a subgraph," *Proceedings, International Symposium on Circuits and Systems* (1979), 657–660.

[Ki1] N. G. Kinnersley, "Obstruction set isolation for layout permutation problems," Ph. D. Thesis, Department of Computer Science, Washington State University, Pullman, WA 99164, 1989.

[Ki2] N. G. Kinnersley, "The vertex separation number of a graph equals its path-width," *Information Processing Letters* 42 (1992), 345–350.

[Ki3] N. G. Kinnersley, "Immersion Order Obstruction Sets," Technical Report TR-93-3, Department of Computer Science, University of Kansas, Lawrence, KS 66045, 1993.

[KL] N. G. Kinnersley and M. A. Langston, "Obstruction set isolation for the Gate Matrix Layout problem," *Annals of Discrete Mathematics* (to appear).

[KM] A. Kézdy and P. McGuinness, "Sequential and parallel algorithms to find a K_5 minor," *Proceedings,* 3^{rd} *Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), 345–356.

[KoT] A. Kornai and Z. Tuza, "Narrowness, pathwidth, and their application in natural language processing," *Discrete Applied Mathematics* 36 (1992), 87–92.

[KPS] M. Khalifat, T. Politof, and A. Satyanarayana, "On minors of graphs with at least $3n - 4$ edges," *Journal of Graph Theory* 17:4 (1993), 523–529.

[KRT] V. King, S. Rao, and R. Tarjan, "A faster deterministic maximum flow algorithm," *Proceedings, 3^{rd} Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), 157–164.

[Lag] J. Lagergren, "An upper bound on the size of an obstruction," in Graph Structure Theory, N. Robertson and P. Seymour (editors), *Contemporary Mathematics* 147 (1993), 601–621.

[Lan] M. A. Langston, private communication.

[LA] J. Lagergren and S. Arnborg, "Finding minimal forbidden minors using a finite congruence," *Proceedings, 18th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 510 (1991), 533–543.

[Le] T. Lengauer, "Combinatorial algorithms for integrated circuit layout," John Wiley & Sons, New York, 1990.

[LG] P. C. Liu and R. C. Geldmacher, "An O(max(m,n)) algorithm for finding a subgraph homeomorphic to K_4," *Congressus Numerantium* 29 (1980), 597–609.

[Li] J-T. Li, "Algorithms for Gate Matrix Layout", *Proceedings, International Symposium on Circuits and Systems* (1983), 1013–1016.

[LL] A. D. Lopez and H-F. S. Law, "A dense gate matrix layout method for MOS VLSI," *IEEE Transactions on Electron Devices* ED-27 (1980), 1671–1675.

[Mad] W. Mader, "Homomorphiesätze für Graphen," *Mathematische Annalen* 178 (1968), 154–168.

[MaS] F. S. Makedon and I. H. Sudborough, "On minimizing width in linear

layouts," *Discrete Applied Mathematics* 23 (1989), 243–265.

[Mi] E. C. Milner, "Basic WQO- and BQO- theory," in *Graphs and Orders*, I. Rival (ed.), D. Reidel Publishing Co., Amsterdam, 1985.

[Mö] R. H. Möhring, "Graph problems related to Gate Matrix Layout and PLA folding," in Computational Graph Theory, Computing Supplementum 7, 17–51, Springer-Verlag, 1990.

[MoS] B. Monien and I. H. Sudborough, "Min-Cut is NP-Complete for edge weighted trees," *Lecture Notes in Computer Science* 226 (1986), 265–274.

[NFKY] K. Nakatani, T. Fujii, T. Kikuno, and N. Yoshida, "A heuristic algorithm for Gate Matrix Layout," *Proceedings, International Conference on Computer-Aided Design* (1986), 324–327.

[OMKKF] T. Ohtsuki, H. Mori, E. S. Kuh, T. Kashiwabara, and T. Fujisawa, "One-Dimensional logic gate assignment and interval graphs," *IEEE Transactions on Circuits and Systems* 26 (1979), 675–684.

[PS] T. Politof and A. Satyanarayana, "Minors of quasi 4-connected graphs," *Discrete Mathematics* (to appear).

[Rad] S.P. Radziszowski, "Small Ramsey Numbers," Technical Report, Department of Computer Science, Rochester Institute of Technology, Rochester, February 1993.

[Ram] F. P. Ramsey, "On a problem of formal logic," *Proc. London Math. Soc.* 30 (1930), 264–286.

[Re] B. Reed, "Finding approximate separators and computing treewidth quickly," *Proceedings, 24th Annual ACM Symposium on Theory of Computing* (1992), 221–228.

[RS1] N. Robertson and P. D. Seymour, "Graph Minors I. Excluding a forest," *Journal of Combinatorial Theory, Series B* 35 (1983), 39–61.

[RS2] N. Robertson and P. D. Seymour, "Graph Minors II. Algorithmic aspects of treewidth," *Journal of Algorithms* 7(1986), 309–322.

[RS3] N. Robertson and P. D. Seymour, "Graph Minors V. Excluding a planar graph," *Journal of Combinatorial Theory, Series B* 41(1986), 92–114.

[RS4] N. Robertson and P. D. Seymour, "Graph Minors XIII. The disjoint paths problem," manuscript (1986).

[RS5] N. Robertson and P. D. Seymour, "Graph Minors XX. Wagner's Conjecture," manuscript (1988).

[Sa] D. P. Sanders, "On linear recognition of treewidth at most four," manuscript (1992).

[SC] U. Singh and C. Y. R. Chen, "From logic to symbolic layout for gate matrix," *IEEE Transactions on Computer-Aided Design* 11 (1992), 216–227.

[ST] A. Satyanarayana and L. Tung, "A characterization of partial 3-trees," *Networks* 20 (1990), 299–322.

[SWK] W. Shu, M. Y. Wu, and S. M. Kang, "Improved net merging method for gate matrix layout," *IEEE Transactions on Computer-Aided Design* 7 (1988), 947–951.

[UV] T. Uehara and W. M. VanCleemput, "Optimal layout of CMOS functional arrays," *IEEE Transactions on Computers* 30 (1981), 305–312.

[WHW] O. Wing, S. Huang, R. Wang, "Gate Matrix Layout," *IEEE Transactions on Computer-Aided Design* 4 (1985), 220–231 .

[Wi1] O. Wing, "Automated Gate Matrix Layout," *Proceedings, International Symposium on Circuits and Systems* (1982), 681–685.

[Wi2] O. Wing, "Interval-graph-based circuit layout," *Proceedings, International Conference on Computer-Aided Design* (1983), 8 4–85.

[Ya] X. Yan, "A relative approximation algorithm for computing the path-width," Master's Thesis, Department of Computer Science, Washington State University, Pullman, WA 99164, 1989.

[Yk] M. Yannakakis, "A polynomial algorithm for the min-cut linear arrangement of trees," *Journal of the Association for Computing Machinery* 32:4 (1985), 950–988.

Vita

Siddharthan Ramachandramurthi was born on April 1, 1967 and raised in Neyveli, in the state of Tamil Nadu in India. After completing his higher secondary education in 1984 at the St. Joseph of Cluny school at Neyveli, he attended the Birla Institute of Technology and Science at Pilani in India, where he received a Master of Science degree in Computer Science in 1988. He then went to the U.S.A. to obtain a Ph.D. in Computer Science. After enrolling at the Washington State University at Pullman in August 1988, he moved to the University of Tennessee in August 1989.