Exploring the Powers of Stacks and Queues via Graph Layouts

by

Sriram V. Pemmaraju

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

APPROVED:

Dr. Lenwood S. Heath, Chairman

Dr. Donald C. S. Allison

Dr. Monte B. Boisen

Dr. Calvin J. Ribbens

Dr. Clifford A. Shaffer

November, 1992

Blacksburg, Virginia

Exploring the Powers of Stacks and Queues via Graph Layouts

by

Sriram V. Pemmaraju

Committee Chairman: Dr. Lenwood S. Heath

Computer Science

(ABSTRACT)

In this dissertation we employ stack and queue layouts of graphs to explore the relative power of stacks and queues. Stack layout and queue layouts of graphs can be examined from several points of view. A stack or a queue layout of a graph can be thought of as an embedding of the graph in a plane satisfying certain constraints, or as a graph linearization that optimizes certain cost measures, or as a scheme to process the edges of the graph using the fewest number of stacks or queues. All three points of view permeate this research, though the third point of view dominates. Specific problems in stack and queue layouts of graphs have their origin in the areas of VLSI, fault-tolerant computing, scheduling parallel processes, sorting with a network of stacks and queues, and matrix computations.

We first present two tools that are useful in the combinatorial and algorithmic analysis of stack and queue layouts as well as in determining bounds on the *stacknumber* and the *queuenumber* for a variety of graphs. The first tool is a formulation of a queue layout of a graph as a covering of its adjacent matrix with *staircases*. Not only does this formulation serve as a tool for analyzing stack and queue layouts, it also leads to efficient algorithms for several problems related to sequences, graph theory, and computational geometry. The connection between queue layouts and matrix covers also forms the basis of a new scheme for performing matrix computations on a data driven network. Our analysis reveals that this scheme uses less hardware and is faster than existing schemes. The second tool is obtained by considering *separated* and *mingled* layouts of graphs. This tool allows us to obtain lower bounds on the stacknumber and the queuenumber of a graph by partitioning the graph into subgraphs and simply concentrating on the interaction of the subgraphs.

These tools are used to obtain results in three areas. The first area is stack and queue layouts of *directed acyclic graphs* (dags). This area is motivated by problems of scheduling

parallel processes. We establish the stacknumber and the queuenumber of classes of dags such as trees, unicylic graphs, outerplanar graphs, and planar graphs. We then present linear time algorithms to recognize 1-stack dags and *leveled-planar dags*. In contrast, we show that the problem of recognizing 9-stack dags and the problem of recognizing 4-queue dags are both NP-complete. The second area is stack and queue layouts of *partially ordered sets* (posets). We establish upper bounds on the queuenumber of a poset in terms of other measures such as length, width, and jumpnumber. We also present lower bounds on the stacknumber and on the queuenumber of certain classes of posets. We conclude by showing that the problem of recognizing a 4-queue poset is NP-complete. The third area is queue layouts of planar graphs. While it has been shown that the stacknumber of the family of planar graphs is 4, the queuenumber of planar graphs is unknown. We conjecture that a family of planar graphs—the *stellated triangles*—has unbounded queuenumber; using separated and mingled layouts, we demonstrate significant progress towards that result.

To my parents,

Indira and Ramgopal

ACKNOWLEDGEMENTS

Words can scarcely describe how much I have learned from Lenwood Heath, my advisor, in the past four years. It all began with a course in analysis of algorithms that he taught in the spring of 1988. Since then, in addition to analysis of algorithms, I have learned graph theory, computational geometry, complexity theory, and racquet ball from him. He has also taught me how to write and read mathematics and has shown me the importance of patience and perseverance in research. This work is as much his creation as it is mine.

I thank Donald Allison, Monte Boisen, Calvin Ribbens, and Clifford Shaffer for serving on my committee and for encouraging me in many ways. I thank Calvin Ribbens, especially, for two useful references to matrix computations on data driven networks.

I thank Professor Arnold L. Rosenberg for his encouragement two years ago and his subsequent suggestions. I thank Ann Trenk Barrett for sharing her insights into the problem of recognizing 1-queue dags. Professor S. Rao Kosaraju gave me an opportunity to present a portion of my work at the Johns Hopkins University. He also gave me valuable feed back and several useful references to maxdominance problems. I am very grateful to him for all of this. Praveen Paripati has endured several of my talks and I thank him for his patience. This research was partially supported by National Science Foundation Grant CCR-9009953.

With their support and encouragement, my friends have made my work more joyous and I am grateful to all of them. I would especially like to thank my room-mates Babu, Pappu, Seenu, and Vengo who have put up with all the vagaries of my behavior. I would also like to mention Chenn, KC, Nagu, and Siva, though I forget why.

My parents and my two sisters have always supported my education and their blessings have always been with me. None of this would have been possible without Santhi, my inspiration and my best friend.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Stacks and queues are fundamental data structures in computer science that are used singly or multiply to solve many well known problems. Graphs are fundamental mathematical objects used to model problems in diverse areas such as economics, computer science, mechanical engineering, biology, and linguistics. In this dissertation, we explore the relative powers of stacks and queues using graph layouts. In the process, we obtain a variety of graph-theoretic, and algorithmic results, and, furthermore open up an entire new area of applications for laying out graphs in stacks and queues.

This chapter provides an overview of this research area and is structured as follows. Section 1.1 first gives definitions of stack and queue layouts of graphs and then describes some of their basic properties. Section 1.2 presents three distinct points of view from which stack and queue layouts can be examined. In Section 1.3, we discuss previous work in this research area. Section 1.4 describes several applications of stack and queue layouts. In Section 1.5, we present an overview of the main contributions of this research. Finally, Section 1.6 gives the organization of the dissertation.

1.1 Definitions and Basic properties

This section prepares the necessary groundwork for the rest of the dissertation. It begins by defining and reviewing fundamental concepts from discrete mathematics. This is followed by definitions and basic properties of stack and queue layouts of graphs. The notation developed in this section will be used consistently throughout the dissertation.

For any natural number n, let $J_n = \{1, 2, \ldots, n\}$. Let V be a finite set. Denote the

cardinality of V by $|V|$. The set of all subsets of V is the *power set* of V, denoted 2^V. If n is a natural number, a *sequence from* V of *length* n is a function $\sigma : J_n \to V$, usually written as the list $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(n)$. Let $\sigma = v_1, v_2, \ldots, v_n$ be a sequence from V. A *subsequence* of σ is a sequence

$$v_{i_1}, v_{i_2}, \ldots, v_{i_m}$$

where $1 \leq i_1 < i_2 < \ldots < i_m \leq n$.

Example 1.1. $2, 2, 1, 4, 2, 4$ is a sequence of length 6 from the set $\{1, 2, 4\}$, while $4, 2$ is a sequence of length 2 from the same set. $2, 4, 4$ and $2, 2, 2$ are subsequences of $2, 2, 1, 4, 2, 4$. □

A *partial order* $\leq$ on V is a reflexive, transitive, anti-symmetric binary relation on V. If (u, v) is an element of $\leq$, then we write $u \leq v$. If $u \leq v$ and $u \neq v$, then we write $u < v$. A *partially ordered set (poset)* $(V, \leq)$ is a set V with a partial order $\leq$ defined on it.

Example 1.2. Let $S = \{1, 2, 3\}$. Then,

$$2^S = \Big\{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\Big\}.$$

It is easily verified that the subset relation $\subseteq$ is a partial order on 2^S and hence $(2^S, \subseteq)$ is a poset. Figure 1.1 is a minimal representation of $(2^S, \subseteq)$ in the sense that there is an arrow from a to b if $a \subset b$ and b cannot be reached from a by following any other sequence of arrows. □

A *chain* in a poset $(V, \leq)$ is a sequence

$$v_1, v_2, \ldots, v_n$$

from V, such that $v_i < v_{i+1}$ for each i, $1 \leq i \leq n - 1$. A chain is said to *cover* all the elements that it contains. An *antichain* in a poset $(V, \leq)$ is a subset

$$\{u_1, u_2, \ldots, u_m\}$$

of V, such that $u_i \not\leq u_j$ whenever $1 \leq i, j \leq m$.

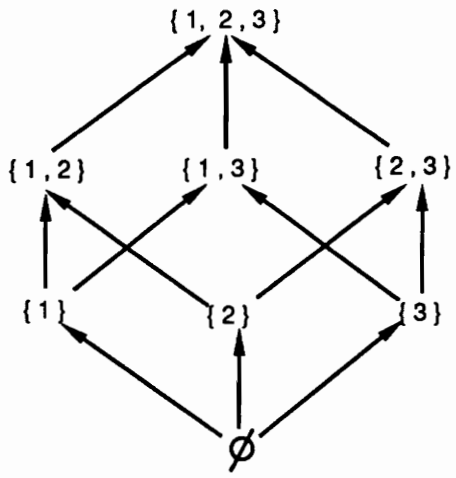


Figure 1.1: The poset $\left(2^S, \subseteq\right)$.

Example 1.3.

$$\emptyset, \{1\}, \{1,2\}, \{1,2,3\}$$

is a chain of length 4 in $(2^S, \subseteq)$. Two examples of antichains in $(2^S, \subseteq)$ are

$$\Big\{\{1,2\}, \{1,3\}, \{2,3\}\Big\}$$

and

$$\Big\{\{1\}, \{2\}, \{3\}\Big\}.$$

□

We now state an important relationship between chains and antichains in a poset, due to Dilworth [20]. This result is invoked in a variety of proofs throughout the dissertation.

Proposition 1.4 (Dilworth [20]) *Let $P = (V, \leq)$ be a poset. The minimum number of chains required to cover all the elements of V equals the size of the largest antichain in P.*

Example 1.5. The following three chains,

$$\emptyset, \{1\}, \{1,3\}, \{1,2,3\},$$

$$\{2\}, \{1,2\},$$

and

$$\{3\}, \{2,3\},$$

cover 2^S. It is easily verified that 2^S cannot be covered with fewer than 3 chains. The size of the largest antichain in $(2^S, \subseteq)$ is also 3 (see Example 1.1), in accord with Dilworth's Theorem. □

A partial order $\leq$ on V is called a *total order* if $u \leq v$ or $v \leq u$ for all $u, v \in V$. Notice that $\subseteq$ is not a total order on 2^S because $\{1,2\} \not\subseteq \{1,3\}$ and $\{1,3\} \not\subseteq \{1,2\}$. Nevertheless, $\subseteq$ is a total order on $\Big\{\emptyset, \{1\}, \{1,2\}, \{1,2,3\}\Big\}$, a subset of 2^S. Often, we will write down a total order $\leq$ on V explicitly, as the sequence $v_1, v_2, \ldots, v_n$, where $V = \{v_1, v_2, \ldots, v_n\}$ and $v_1 < v_2 < \cdots < v_n$. To avoid ambiguity, we give total orders names such as σ, δ, and γ. If σ is a total order on V and $(u, v) \in \sigma$, then we write $u \leq_\sigma v$. If $u \leq_\sigma v$ and $u \neq v$, then we write $u <_\sigma v$. $\sigma^{-1}(v)$ denotes the number of elements $u \in V$, for which $u \leq_\sigma v$. Thus, if σ is viewed as a sequence, then $\sigma^{-1}(v)$ denotes the position of v in σ.

Let $(V, \leq)$ be a poset. A sequence $v_1, v_2, \ldots, v_n$ from V is *monotonically increasing* with respect to $\leq$ if $v_1 \leq v_2 \leq \cdots \leq v_n$. A *monotonically decreasing* sequence is defined analogously.

Example 1.6. The sequence

$$\emptyset, \{1\}, \{1\}, \{1,3\}, \{1,2,3\}$$

is a monotonically increasing sequence from 2^S with respect to $\subseteq$. The sequence

$$\{1,3\}, \{1,3\}, \{3\}, \emptyset$$

is a monotonically decreasing sequence from 2^S with respect to $\subseteq$. □

We now state a fundamental result about sequences chosen from a totally ordered set due to Erdös and Szekeres [21]. This result is invoked in several lower bound proofs in this dissertation.

Proposition 1.7 (Erdös and Szekeres [21]) *Let $\leq$ be a total order defined on a set V. Let $\sigma = v_1, v_2, \ldots, v_n$ be an arbitrary sequence from V. Then, either σ contains a monotonically increasing subsequence of length $\lceil\sqrt{n}\rceil$ or σ contains a monotonically decreasing subsequence of length $\lceil\sqrt{n}\rceil$.*

A *permutation* of V is a bijection $\pi : J_{|V|} \rightarrow V$. Hence, a permutation of V is a sequence from V of length $|V|$ with no element repeated. Each permutation of V corresponds to a different total order on V. The position of an element $u \in V$ in the permutation π is $\pi^{-1}(u)$. If for some $u, v \in V$, $\pi^{-1}(u) < \pi^{-1}(v)$, then we write $u <_\pi v$.

Example 1.8. If $V = \{1, 2, 3, 4, 5\}$, then

$$3, 1, 2, 4, 5$$

and

$$2, 3, 1, 4, 5$$

are 2 of the 5! possible permutations of the elements of V. If $\pi = 3, 1, 2, 4, 5$, then

$$\pi^{-1}(3) = 1, \pi^{-1}(1) = 2, \pi^{-1}(2) = 3, \pi^{-1}(4) = 4, \pi^{-1}(5) = 5.$$

Furthermore,

$$3 <_\pi 1 <_\pi 2 <_\pi 4 <_\pi 5.$$

□

The set

$$\{V_1, V_2, \ldots, V_m\}$$

is a *partition* of V if $V_i \subseteq V$ for each i, $1 \leq i \leq m$, $V = \cup_{i=1}^{m} V_i$, and $V_i \cap V_j = \emptyset$ for each distinct i, j, $1 \leq i, j \leq m$. Each set V_i, $1 \leq i \leq m$ is called a *block* of the partition.

Example 1.9. The following set

$$\Big\{\{1\}, \{2, 5\}, \{3, 4\}\Big\}$$

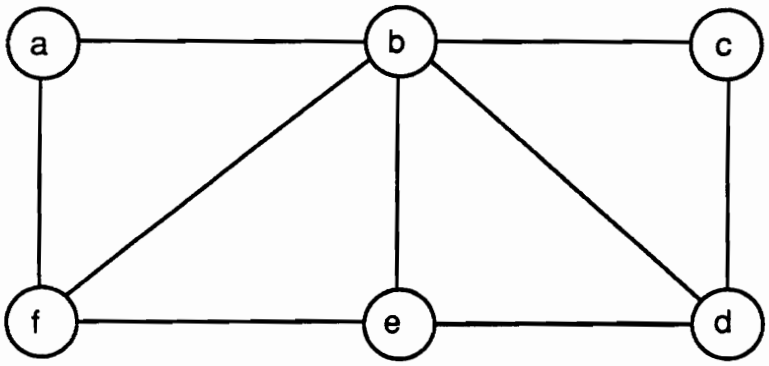


Figure 1.2: An example graph $G_1 = (V_1, E_1)$.

is a partition of the set $\{1,2,3,4,5\}$. $\{1\}$, $\{2,5\}$, and $\{3,4\}$ are the blocks of this partition. □

Let $G = (V, E)$ be a graph. We shall assume throughout this dissertation that a graph is finite and contains no multiple edges or loops. Unless otherwise specified we will assume that a graph is undirected. Following a common distinction, we use vertices and edges for the components of undirected graphs and nodes and arcs for the components of directed graphs. We define the *layout* of G as a total order on its vertex set V.

A *k-stack layout* of a graph $G = (V, E)$ consists of a total order σ on V and an assignment of each edge in E to exactly one of k stacks, $s_1, \ldots, s_k$. Each stack s_j operates as follows. The vertices of V are scanned in left-to-right (ascending) order according to σ. When vertex v is encountered, any edges assigned to s_j that have vertex v as their right endpoint must be on the top of that stack; they are removed (popped). Any edges assigned to s_j that have v as left vertex are placed on the top of the stack (pushed), in descending order of their right vertices. The freedom to choose the total order on V and the assignment of edges in E to stacks so as to optimize some measure of the resulting layout constitutes the essence of the stack layout problem.

Example 1.10. Figure 1.2 shows an undirected graph $G_1 = (V_1, E_1)$ with vertex set $V_1 = \{a, b, c, d, e, f\}$. Figure 1.3 shows the vertices in V_1 laid out on a horizontal line ac-

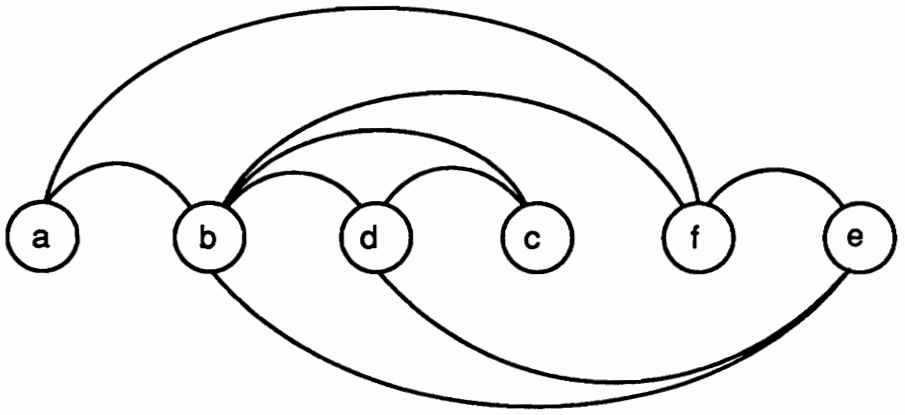


Figure 1.3: A 2-stack layout of G_1.

cording to the total order a, b, d, c, f, e and an assignment of the edges of G_1 to two stacks s_1 and s_2. The edges $\{(a,b),(a,f),(b,d),(b,c),(b,f),(f,e)\}$ are assigned to s_1 and are all drawn above the horizontal line. The edges $\{(b,e),(d,e)\}$ are assigned to s_2 and are drawn below the horizontal line. Figure 1.4 shows the processing of the edges of G_1 using the stacks s_1 and s_2. The contents of s_1 and s_2, after each vertex has been scanned, are shown. Note that edges that share a left vertex and are assigned to the same stack are pushed into the stack in the reverse order of their right vertices. The edges (a,b) and (a,f) share a left vertex, namely a, and are assigned to the same stack, namely s_1. When a is processed, the edges are pushed into the stack in the order $(a,f),(a,b)$. Thus Figure 1.3 shows a 2-stack layout of G_1 because the edges above the line can be assigned to one stack and the edges below the line to a second stack. □

Given a graph $G = (V, E)$ and a total order σ on V, the *stacknumber* $SN(\sigma, G)$ of the layout of G according to σ is the smallest k such that G has a k-stack layout in which the vertices are laid out according to σ. The *stacknumber* $SN(G)$ of G is the smallest k such

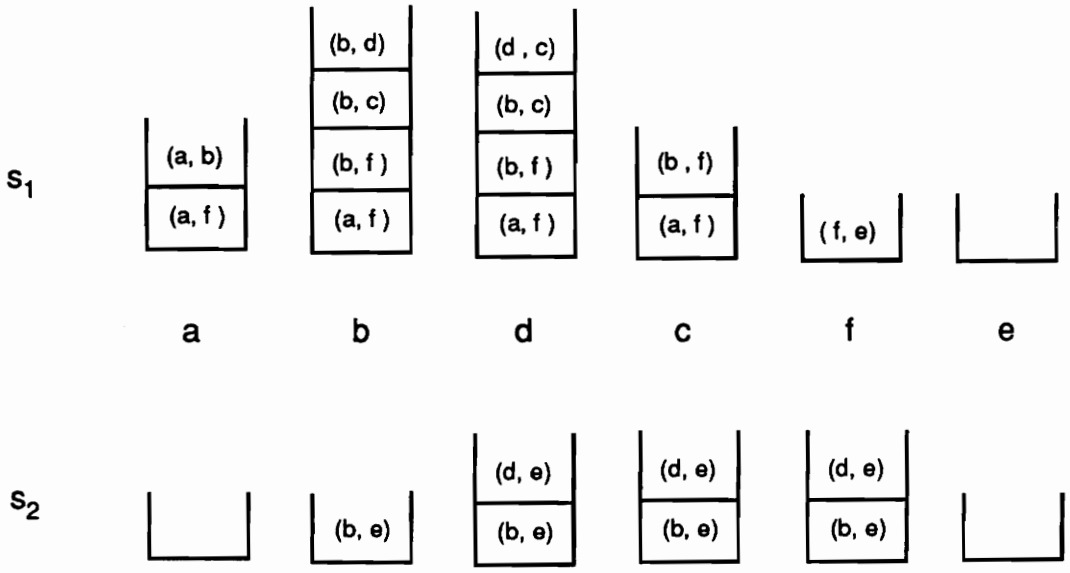


Figure 1.4: Contents of s_1 and s_2.

that G has a k-stack layout. In other words,

$$SN(G) = \min_{\sigma} SN(\sigma, G).$$

G is said to be a *k-stack* graph if $SN(G) = k$.

Example 1.11. Consider the graph $G_1 = (V_1, E_1)$ shown in Figure 1.2. Let $\sigma_1 = a, b, d, c, f, e$ be a total order on V_1 (shown in Figure 1.3). Since, Figure 1.3 shows a 2-stack layout of G_1 in which the vertices are laid out according to σ_1, $SN(\sigma_1, G_1) \leq 2$. It is easily verified that when G_1 is laid out according to σ_1, the edges (b, f) and (d, e) cannot be assigned to the same stack. Thus, $SN(\sigma_1, G_1) = 2$. Figure 1.5 shows a 1-stack layout of the graph G_1 (shown in Figure 1.2). Hence, $SN(G_1) = 1$ and G_1 is a 1-stack graph. Figure 1.3 and Figure 1.5 together show that for the same graph, different total orders on the vertex set may yield different stacknumbers. □

We extend our definition of stacknumber to classes of graphs. Let C be a class of graphs. The *stacknumber of* C, $SN_C(n)$, is a function of the natural numbers that equals the least upper bound of the stacknumber of all the graphs in C with n vertices. We are sometimes

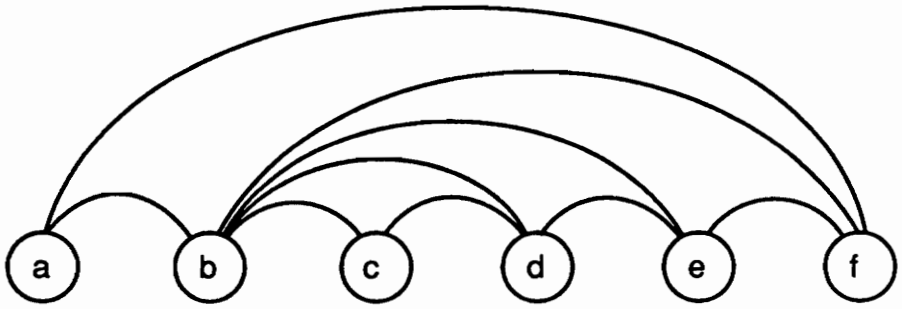


Figure 1.5: A 1-stack layout of G_1.

interested in the asymptotic behavior of $SN_C(n)$ or in whether $SN_C(n)$ is bounded above by a constant.

A *k-queue layout* of an undirected graph $G = (V, E)$ consists of a total order σ on V, and an assignment of each edge in E to exactly one of k queues, $q_1, \ldots, q_k$. Each queue q_j operates as follows. The vertices of V are scanned in left-to-right (ascending) order according to σ. When vertex v is encountered, any edges assigned to q_j that have vertex v as their right endpoint must be at the front of that queue; they are removed (dequeued). Any edges assigned to q_j that have vertex v as left vertex are placed on the back of that queue (enqueued), in ascending order of their right vertices. The freedom to choose a total order on V and an assignment of edges to queues so as to optimize some measure of the queue layout, is the essence of the queue layout problem.

Example 1.12. Figure 1.6 shows the vertices of G_1 (shown in Figure 1.2) laid out according to the total order a, d, c, b, f, e and an assignment of the edges in E_1 to two queues. q_1 and q_2. The edges in the set $\{(a,b),(a,f),(d,e),(b,e)\}$ are assigned to q_1 and are drawn above the horizontal line, while the edges $\{(d,c),(c,b),(d,b),(b,f),(f,e)\}$ are assigned to q_2 and are drawn below the horizontal line. Figure 1.7 shows the processing of the edges of G using queues q_1 and q_2. The contents of q_1 and q_2 after each vertex has been processed, are shown. Note that the edges that share a left vertex v and are assigned to the same queue are enqueued in the order of their right vertices. For example, the edges (a,b) and (a,f)

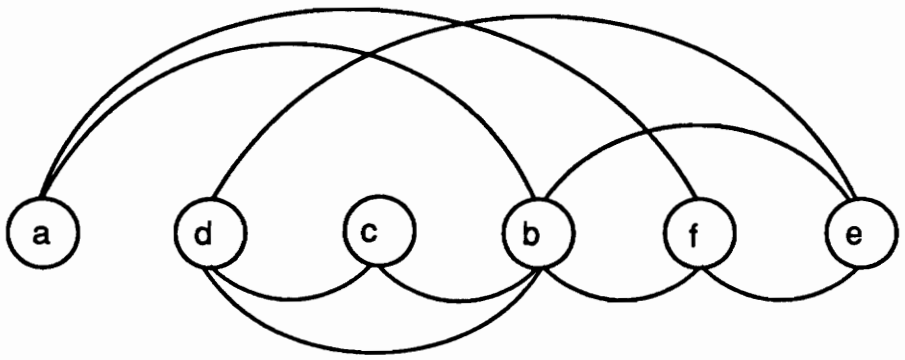


Figure 1.6: A 2-queue layout of G_1.

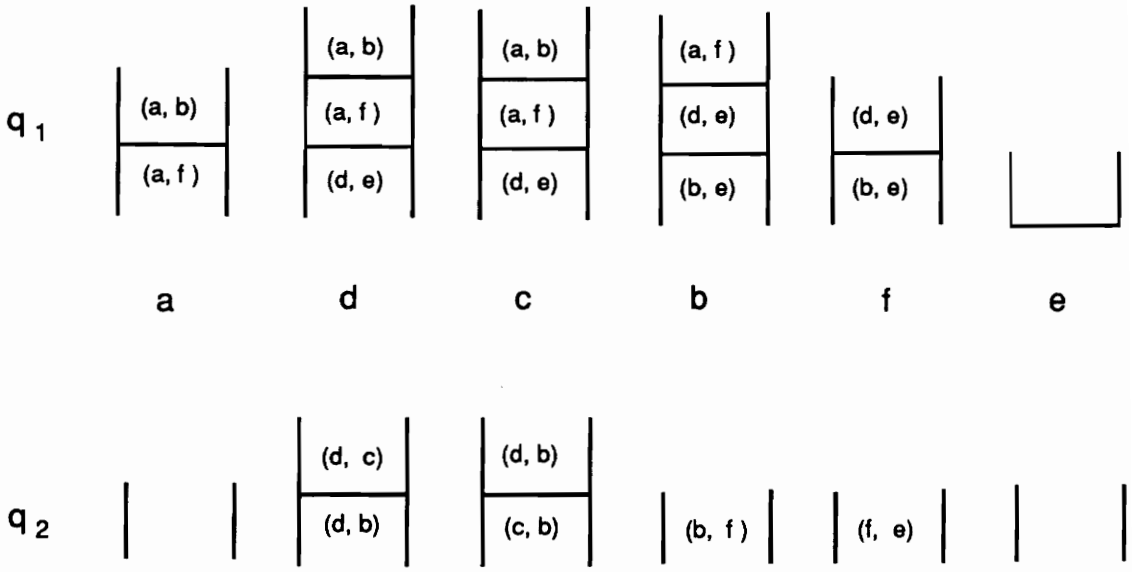


Figure 1.7: Contents of q_1 and q_2.

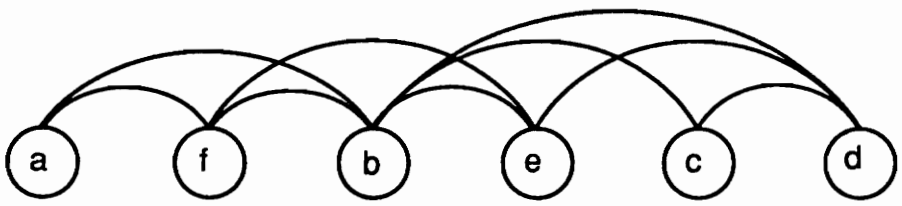


Figure 1.8: A 1-queue layout of G_1.

share a left vertex, namely a, and are assigned to the queue q_1. When a is processed, the edges are enqueued into q_1 in the order (a,b), (a,f). □

Given a graph $G = (V,E)$ and a total order σ of V, the *queuenumber* $QN(\sigma,G)$ of the layout of G according to σ, is the smallest k such that G has a k-queue layout in which the vertices are laid out according to σ. The *queuenumber* $QN(G)$ of G is the smallest k such that G has a *k-queue layout.* In other words,

$$QN(G) = \min_{\sigma} QN(\sigma,G).$$

G is said to be a *k-queue* graph if $QN(G) = k$.

Example 1.13. Consider the graph $G_1 = (V_1,E_1)$ shown in Figure 1.2. Figure 1.6 shows a 2-queue layout of G_1 in which the vertices are laid out according to a total order $\sigma_2 = a,d,c,b,f,e$. Thus $QN(\sigma_2,G_1) \leq 2$. It is easily verified that when G_1 is laid out according to σ_2, the edges (a,b) and (d,c) cannot be assigned to the same queue. Thus, $QN(\sigma_2,G) = 2$. Figure 1.8 shows a 1-queue layout of the graph G_1 (shown in Figure 1.2). Thus $QN(G_1) = 1$ and G_1 is a 1-queue graph. Figure 1.6 and Figure 1.8 together show that for the same graph, different total orders on the vertex set may yield different queuenumbers. □

We extend our definition of queuenumber to classes of graphs. Let C be a class of graphs. The *queuenumber of C*, $QN_C(n)$, is a function of the natural numbers that equals the least upper bound on the queuenumbers of all graphs in C with n vertices. As with

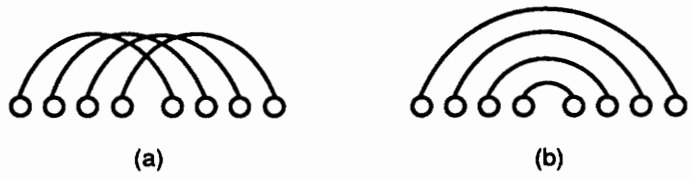


Figure 1.9: (a) A twist. (b) A rainbow.

$SN_C(n)$, we are sometimes interested in the asymptotic behavior of $QN_C(n)$ or in whether $QN_C(n)$ is bounded above by a constant.

For a fixed total order σ on V, we identify sets of edges that are obstacles to minimizing the number of stacks or queues. A *k-twist* is a set of k edges

$$\Big\{(a_i, b_i) \mid 1 \leq i \leq k\Big\}$$

such that

$$a_1 <_\sigma a_2 <_\sigma \cdots <_\sigma a_{k-1} <_\sigma a_k <_\sigma b_1 <_\sigma b_2 <_\sigma \cdots <_\sigma b_{k-1} <_\sigma b_k,$$

i.e., a twist is a *fully intersecting* matching. A *k-rainbow* is a set of k edges

$$\Big\{(a_i, b_i) \mid 1 \leq i \leq k\Big\}$$

such that

$$a_1 <_\sigma a_2 <_\sigma \cdots <_\sigma a_{k-1} <_\sigma a_k <_\sigma b_k <_\sigma b_{k-1} <_\sigma \cdots <_\sigma b_2 <_\sigma b_1,$$

i.e., a rainbow is a *nested* matching. Figure 1.9(a) shows a twist, and Figure 1.9(b) shows a rainbow. A twist is an obstacle to minimizing the number of stacks required to lay out G according to σ because no two edges in a twist can be assigned to the same stack. Similarly, a rainbow is an obstacle to minimizing the number of queues required to lay out G according to σ because no two edges in a rainbow can be assigned to the same queue.

Example 1.14. Figure 1.3 shows a 2-stack layout of $G_1 = (V_1, E_1)$ (shown in Figure 1.2) according to the total order a, b, d, c, f, e. The edges (b, f) and (d, e) form a 2-twist. Assume

that (b, f) and (d, e) are assigned to the same stack. When b is scanned, (b, f) is pushed into a stack. Later, when d is scanned, (d, e) is pushed into the same stack. Therefore, when f is scanned, (b, f) cannot be popped from the stack because it is not on top of the stack.

Figure 1.6 shows a 2-queue layout of $G_1 = (V_1, E_1)$ (shown in Figure 1.2) according to the total order a, d, c, b, f, e. The edges (a, b) and (d, c) form a 2-rainbow. Assume that (a, b) and (d, c) are assigned to the same queue. When a is scanned, (a, b) is enqueued into a queue. Later, when d is scanned, (d, c) is enqueued into the same queue. Therefore, when c is scanned, (d, c) cannot be dequeued from the queue because it is not in the front of the queue. □

Intuitively, we can think of a stack layout or a queue layout of a graph as a drawing of the graph in which the vertices are laid out on a horizontal line and the edges appear as curves above the line. In a stack layout no two edges that intersect can be assigned to the same stack, while in a queue layout no two edges that nest can be assigned to the same queue.

Proposition 1.15 (Heath and Rosenberg [36]) *Suppose $G = (V, E)$ is a graph and σ a total order on V. If σ has a k-rainbow, then $QN(G, \sigma) \geq k$. If σ has a k-twist, then $SN(G, \sigma) \geq k$.*

The largest rainbow in σ precisely determines the minimum queuenumber of a queue layout of σ.

Theorem 1.16 (Heath and Rosenberg [36]) *Suppose $G = (V, E)$ is a graph and σ a total order on V. If the largest rainbow in σ has exactly k edges, then $QN(G, \sigma) = k$. A k-queue layout of G with order σ can be found in time $O(|E| \log\log n)$.*

We might expect the analogous result for stacks to hold; that is, if the largest twist in σ is a k-twist, then the stacknumber of σ is k, and a k-stack layout can be found in polynomial time. However, this is not true.

Proposition 1.17 (Even and Itai [23]) *The problem of minimizing the number of stacks required by a fixed total order on V is NP-complete.*

Alternate characterizations of 1-stack graphs and 1-queue graphs have proved very useful. Bernhart and Kainen [6] characterize 1-stack graphs using the following class of graphs. An *outerplanar graph* $G = (V, E)$ is a planar graph having a planar embedding in which all the vertices of G appear on the circumference of a circle and all edges appear as chords of the circle. Such an embedding is called an *outerplanar embedding* and induces a natural circular order on the vertices of the graph G. Any total order σ that is the counterclockwise or clockwise order of vertices as they appear on the circle, starting from an arbitrary vertex, gives a 1-stack layout of G.

Proposition 1.18 (Bernhart and Kainen [6]) *G is a 1-stack graph if and only if G is an outerplanar graph.*

Heath and Rosenberg [36] characterize 1-queue graphs using a class of graphs that we now define. Consider the two dimensional Cartesian coordinate system. For i an integer, let l_i be the vertical line defined by the equation $x = i$. $G = (V, E)$ has a *leveled-planar embedding* if V can be partitioned into levels $L_1, L_2, \ldots, L_p$ in such a way that

- G has a planar embedding in which all vertices in L_i appear on line l_i;
- each edge in E is embedded as a straight-line segment wholly between l_i and l_{i+1}, for some i.

G is called a *leveled-planar graph*. A leveled-planar embedding induces a partition of E into $E_1, E_2, \ldots, E_{p-1}$, where E_i contains edges between vertices in L_i and L_{i+1}. An edge in E_i is called a *level-i* edge. A leveled-planar embedding of G also induces a natural order σ, called the *leveled planar order*, on V obtained by scanning the lines $l_1, l_2, \ldots, l_p$ in that order from bottom to top and assigning labels $1, 2, \ldots, n$ to the vertices as they are encountered. Figure 1.10 shows a leveled-planar graph. For each i, $1 \leq i \leq p$, let b_i be the *bottom vertex* on line l_i, and let t_i be the *top vertex*. Let s_i be the bottommost vertex on line l_i that is adjacent to a vertex on line l_{i+1}. If there are no edges between L_i and L_{i+1}, then let $s_i = t_i$. Consider augmenting G with new edges called *arches*. A *level-i arch* for G is an

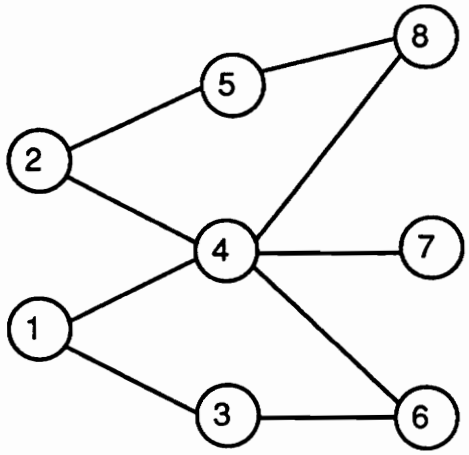


Figure 1.10: A leveled-planar graph.

edge connecting vertex t_i with a vertex $j \in L_i$, where $b_i \leq j \leq \min(t_i - 1, s_i)$. A leveled-planar graph augmented by any number of arches can be drawn in the plane by drawing the arches around level 1; such an embedding is called an *arched leveled-planar embedding.* A graph that has an arched leveled-planar embedding is called an *arched leveled-planar graph.* Figure 1.11 shows an arched leveled-planar graph.

Proposition 1.19 (Heath and Rosenberg [36]) *G is a 1-queue graph if and only if G is an arched leveled-planar graph.*

Let $G = (V, E)$ be an arched leveled-planar graph. By deleting the arches from its arched leveled-planar embedding, we obtain a leveled-planar embedding of a subgraph of G, which in turn induces a leveled-planar order on V. Heath and Rosenberg [36] show that the leveled-planar order of V yields a 1-queue layout of G. They also show how to obtain an arched leveled-planar embedding of a graph from its 1-queue layout.

1.2 Stack and Queue Layouts: Three Points of View

The definitions given in Section 1.1 provide only one of several distinct points of view for examining stack and queue layouts of graph. In this section, we present three points of

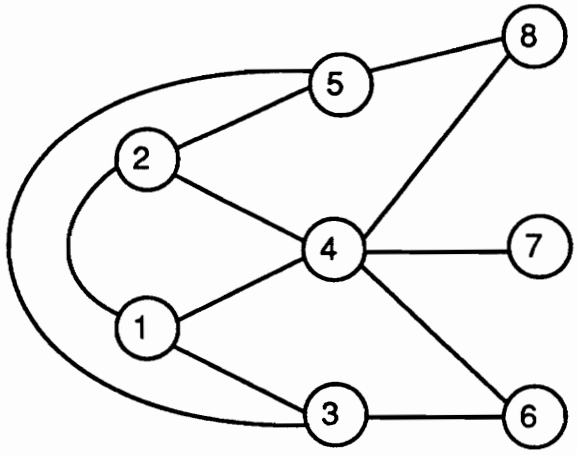


Figure 1.11: An arched leveled-planar graph.

view for examining stack and queue layouts. Each point of view brings with it some insights and tools.

1.2.1 Topological Point of View

A stack or a queue layout of a graph can be thought of as a continuous map of the graph into a particular kind of topological space and satisfying certain constraints. In fact, when first introduced, a stack layout was called a book embedding (Bernhart and Kainen [6]). A *book* is a topological space consisting of a line called its *spine* and a collection of half-planes called its *pages* that have the spine as a common boundary. The vertices of the graph are laid out on the spine and the edges are assigned to pages, such that no two edges assigned to a page intersect, when drawn in that page. The above notion of a book embedding is equivalent to the notion of a stack layout because the edges assigned to a stack can be drawn on a page without intersections. Figure 1.3 shows a 2-page embedding of G_1 in which all the edges drawn above the horizontal line (spine) are assigned to one page and all the edges drawn below the horizontal line are assigned to another page. In a similar, but less natural manner a queue layout can also be viewed in terms of assigning edges to pages. The assignment must satisfy the constraint that no two edges assigned to

a page nest. The above notion is equivalent to the notion of a queue layout because edges assigned to the same queue can be drawn on a page without nesting.

Topological graph theory is a classical area in mathematics whose origins can be traced back to Euler. The notion of embedding a graph in a surface has acquired considerable importance due to applications in areas such as VLSI design. Recognizing and embedding planar graphs is a lively area of research that has led to the development of new algorithmic techniques as well as useful data structures for graph–theoretic problems [8,22,37,46,15]. One such technique is a data structure called PQ-trees, introduced by Booth and Lueker [8], that allows for a simple linear time algorithm for testing the planarity of a graph. Viewing a stack layout or a queue layout of a graph as a topological embedding of the graph satisfying certain geometric and topological constraints lends valuable insights into the area. For example, it is easy to recognize a 1-stack graph as an outerplanar graph and vice versa by examining its embedding in one page.

1.2.2 Graph Linearization Point of View

Stack and queue layout problems clearly belong to the class of graph linearization problems. Graph linearization problems have utility in diverse areas such as matrix computations and information storage and retrieval. Examples of graph linearization problems are bandwidth minimization, optimal linear arrangement, and minimum cut linear arrangement (see Garey and Johnson [27] for the computational complexity of these problems). A classic graph linearization problem is the bandwidth minimization problem. Given a graph $G = (V, E)$ and a total order σ on V, the *length* of an edge $(u, v) \in E$ in σ is given by $|\sigma^{-1}(u) - \sigma^{-1}(v)|$ and the *bandwidth* of σ is

$$\max_{(u,v)\in E} \left|\sigma^{-1}(u) - \sigma^{-1}(v)\right|.$$

Thus the bandwidth of a total order σ on V is the length of the longest edge in σ. The *bandwidth minimization problem* is to find a total order on V with minimum bandwidth.

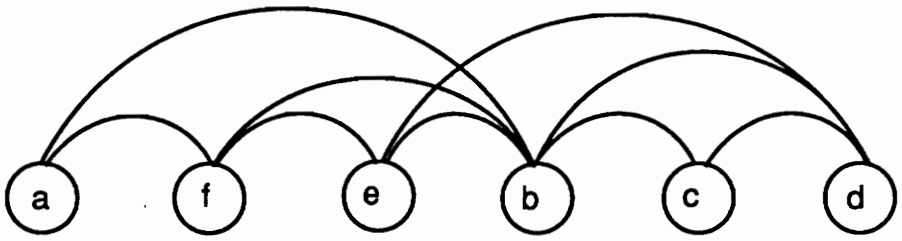


Figure 1.12: Graph linearization

	a	f	e	b	c	d
a	0	1	0	1	0	0
f	1	0	1	1	0	0
e	0	1	0	1	0	1
b	1	1	1	0	1	1
c	0	0	0	1	0	1
d	0	0	1	1	1	0

Figure 1.13: Adjacency matrix of G_1.

Example 1.20. Figure 1.12 shows the vertices of G_1 (shown in Figure 1.2) laid out according to the total order $\sigma = a, f, e, b, c, d$. Since $\sigma^{-1}(e) = 3$ and $\sigma^{-1}(d) = 6$, we have $|\sigma^{-1}(e) - \sigma^{-1}(d)| = 3$. It can be easily verified that the bandwidth of σ is 3. □

The bandwidth minimization problem has direct applications in the area of matrix computations (see Cuthill and McKee [17,18]). Figure 1.13 shows the adjacency matrix of G corresponding to the total order a, f, e, b, c, d, shown in Figure 1.12. Notice that the length of an edge in the total order shown in Figure 1.12 equals the distance between the corresponding non-zero element in the matrix, shown in Figure 1.13, and its main diagonal. Thus, chosing a total order on V that minimizes the bandwidth of G corresponds to chosing a symmetric permutation of the rows and columns of the adjacency matrix so as to minimize the distance between the main diagonal and the non-zero elements farthest away. In matrix computations, in terms of time and space efficiency, it is advantageous to work with matrices in which the non-zero elements are concentrated close to the main diagonal. Thus, techniques for bandwidth minimization are useful for improving on the time and

space efficiency of matrix computations.

The *optimal linear arrangement* problem is defined as follows: Given a graph $G = (V, E)$, find a total order σ on V that minimizes

$$\sum_{(u,v)\in E} \left|\sigma^{-1}(u) - \sigma^{-1}(v)\right|.$$

Thus the optimal linear arrangement problem is to find a total order σ on V that minimizes the sum of the lengths of the edges. The optimal linear arrangement problem has utility in the area of information storage and retrieval (see Chen [13,14]). Information can be abstractly viewed as a graph whose vertices contain pieces of information and whose edges connect related pieces of information. In a typical exercise in information retrieval, a user is likely to browse through related pieces of information, which corresponds to traversing through the graph via the edges. If the information is stored in a sequential access medium such as a CD-ROM, then the delay in seeking new information after browsing through old information is proportional to the distance between the old and new pieces of information. Since, the pieces of information have to be arranged on a CD-ROM in a linear fashion, an arrangement that minimizes the total seek time (the time taken by the head to seek a new piece of information) corresponds to a solution to the optimal linear arrangement problem.

In Chapter 2, we show that viewing queue layouts in terms of graph linearization immediately leads to several combinatorial results in the area. It also leads to a new formulation of queue layouts as matrix covering problems. This formulation, in turn leads to solutions to problems in graph theory and computational geometry and, in a different direction, leads to a new and improved scheme for performing matrix computations on a data driven network.

1.2.3 Data Structures Point of View

From a traditional computer science point of view, a stack or a queue layout of a graph can be thought of as a scheme to process the edges of the graph using stacks or queues. In particular, minimizing stacknumber or queuenumber relates to the general problem of processing data using the least amount of memory. Stacks and queues are ubiquitous data

structures in computer science. In a typical application, either a stack or a queue alone might be used to accomplish a particular function. For example, a single stack is sufficient to implement recursion or to parse a string over a context free grammar and a single queue suffices to schedule tasks in a simple multi-tasking environment. A stack layout (respectively, queue layout) of a graph can be viewed as an example of multiple stacks (respectively, queues) co-operating together to accomplish a particular task—processing the edges of the graph. This view is strengthened by its application to sorting networks that contain stacks or queues and to the DIOGENES approach to fault-tolerant computing [58]. This view also provides insights into the relative abilities of stacks and queues. In simple situations or when used singly, stacks and queues seem to be duals of each other. For example, a stack can be used to implement depth first search, while a queue can be used to implement a breadth first search. Another example arises from the problem of sorting a sequence using stacks or queues in parallel (see Tarjan [62] or Even and Itai [23]). We shall discuss this problem is greater detain in Section 1.4. Even and Itai [23] show that if the largest monotonically decreasing subsequence in a given sequence is of size k, then the sequence can be sorted by k stacks in parallel. On the other hand if the largest monotonically increasing subsequence in a given sequence is of size k, then the sequence can be sorted by k queues in parallel. These and other examples convey a sense of duality between stacks and queues. On the basis of this research, we believe that this duality is somewhat superficial and disappears in more general situations.

1.3 Previous Work

Bernhart and Kainen [6] introduce the notion of embedding a graph in a book. A *book embedding* of a graph has two aspects. First, the vertices are laid out in some total order along the *spine* (a line). Second, each edge of the graph is assigned to a *page* (a half-plane having the spine as boundary), so that no two edges in a page cross. The smallest number of pages required to lay out a graph G is its *pagenumber*.

Bernhart and Kainen conjecture that there are planar graphs with arbitrarily large pagenumber and also conjecture that the class of graphs of genus g, $g \geq 1$, has unbounded pagenumber. Buss and Shor [11] disprove the first conjecture by proving that the pagenumber of any planar graph is at most 9. Heath [32] reduces the upper bound to 7, while Yannakakis [64] obtains the exact bound of 4 pages to embed an arbitrary planar graph in a book. Heath and Istrail [34] disprove the second conjecture of Bernhart and Kainen by showing that genus g graphs can be laid out in $O(g)$ pages with a linear time algorithm. Malitz [47] shows, via a nonconstructive argument, that genus g graphs can be laid out in $O(\sqrt{g})$ pages, which is optimal. Malitz also shows that any graph $G = (V, E)$ can always be laid out in $O(\sqrt{E})$ pages. Chung, Leighton, and Rosenberg [16] show that the book embedding problem is equivalent to the stack layout problem and provide a fundamental study of many aspects of the problem. They present optimal or near-optimal embeddings for numerous families of graphs and establish bounds on the minimum pagenumber of a graph based on various structural properties of the graph.

In keeping with the spirit of Chung, Leighton, and Rosenberg, Heath and Rosenberg [36] study the related problem of laying out a graph using queues. They show that every 1-stack graph is a 2-queue graph and that every 1-queue graph is a 2-stack graph. They characterize the class of 1-queue graphs as arched leveled-planar graphs (see Section 1.1). While 1-stack graphs can be recognized in linear time, they show that the problem of recognizing 1-queue graphs is NP-complete. They present queue layouts for specific classes of graphs and show a tradeoff between queuenumber and stacknumber for a fixed total order of the vertices of a graph. In their conclusion, they conjecture that planar graphs can be laid out in a bounded number of queues.

Heath, Leighton, and Rosenberg [35] show that for a particular class C of graphs (called the ternary n-cubes), the ratio $SN_C(n)/QN_C(n)$ is unbounded; in fact, C has stack requirements that are exponentially greater than its queue requirements. In their conclusion, Heath, Leighton, and Rosenberg conjecture that the reciprocal ratio $QN_C(n)/SN_C(n)$ is bounded for any class of graphs C.

Nowakowski and Parker [53], Sysło [60], and Hung [38] prove results about the stack-number of posets. Their results are discussed in detail in Chapter 5.

1.4 Applications

In this section, we present applications of stack and queue layouts of graphs. We will elaborate on the application of stack and queue layouts to dag in Chapter 4 and the application of queue layouts to parallel matrix computations on a data driven network in Chapter 2.

1.4.1 Sorting with a Network of Stacks or Queues

The problem of sorting a permutation using a network of stacks and queues takes a directed graph with a single source and a single sink such that every other node contains a stack or a queue. The task is to begin with a permutation of $1, 2, \ldots, n$ stored at the source, and route the integers through the network using stacks and queues as intermediate storage, in order to obtain the identity permutation at the sink. Even and Itai [23] distinguish between two cases in the problem of sorting with a network of stacks. If we require that all numbers must leave the source before any number reaches the sink, then we have the problem of sorting with *complete loading*, i.e, the network has to be completely loaded (all numbers have left the sink) before any number can arrive at the sink. Otherwise, we have the less restrictive problem of sorting *without complete loading*, i.e, the sink may receive numbers before all numbers have left the source. The problem of sorting with and without complete loading has been studied before for a highly restricted class of networks [24,23,42,62] called *parallel networks*. Figure 1.14 shows an example of a parallel network, that contains 5 nodes at which stacks or queues are placed.

Chung, Leighton, and Rosenberg [16] show that the problem of sorting with a parallel network of stacks with complete loading can be reformulated as a problem of laying out a class of bipartite graph under certain restrictions. Pemmaraju [57] shows that the problem of

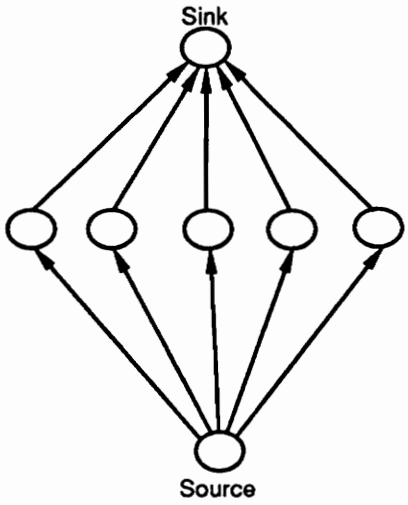


Figure 1.14: Sorting with a parallel network of stacks or queues.

sorting with a parallel network of stacks without complete loading, can also be reformulated as a problem of laying out a class of bipartite graphs under certain other restrictions.

1.4.2 The DIOGENES Design Methodology

Rosenberg [58] proposes the DIOGENES methodology for designing testable, fault-tolerant arrays of processors. In a DIOGENES layout, an array of communicating processors is implemented on a conceptual line, and some number of hardware stacks and/or queues pass over the entire line. The stacks and/or queues implement the communication links among processors in such a way that faulty processors are ignored, and all good processors can be utilized. If the set of processors and their connections is viewed as an undirected graph, then the DIOGENES layout corresponds to laying out an undirected graph in stacks and queues.

1.4.3 Scheduling Parallel Processors

The problem of scheduling parallel computations in an architecture-independent fashion can be modeled using a directed acyclic graph (dag), called the *dependency dag*, in which

each node represents a process and each arc represents a data dependency (see Papadimitriou and Yannakakis [56]). A process cannot begin execution until all processes that it depends on have completed execution. Processes are queued in a first-in/first-out (FIFO) Processor Queue (PQ), and as each process becomes eligible for execution, it is assigned to an idle processor. The flow of data through the computation is managed by a Data Manager (DM) which is a collection of several FIFO queues. When a process terminates, it enqueues its outputs on queues in DM such that when a process P begins execution, all inputs to P are at the heads of some of the queues in DM and are then dequeued. The problem of enqueuing data to and dequeuing data from the queues in DM is equivalent to the problem of laying out the dependency dag in queues.

1.4.4 Parallel Matrix Computations on a Data Driven Network

The size of engineering problems that are solved via computation is increasing rapidly. Multiple processors computing in parallel are being used to solve these problems in a reasonable amount of time. Solving linear systems of equations is an important component of numerical solutions to engineering problems. Many architectures and schemes have been suggested for performing matrix computations in parallel. The systolic arrays of Kung and Leiserson [43,44,45] and the data driven networks of Melhem [51] are two examples of architectures that can perform matrix computations in parallel. Based on covering matrices with staircases, we suggest a new and efficient scheme to perform matrix computations on a data driven network. At no additional hardware or software cost, our scheme provides greater flexibility and efficiency as compared to Melhem's [51] scheme. We elaborate on this scheme in Chapter 2.

1.4.5 Other Applications

Special cases of the book embedding problem have arisen independently in the work of Brady and Brown [9], Even and Itai [23], Stoffers [59], Tarjan [62], and Trotter [63]. Kainen [40] suggests applications to transportation, VLSI, and software. Beyond the above

motivations is the incentive that our research will expose similarities and differences in the computational abilities of stacks and queues.

1.5 Contributions of This Dissertation

In this research, we utilize all three points of view mentioned in Section 1.2, to obtain graph theoretic, combinatorial, and algorithmic results in the area of stack and queue layouts. These results are then used to further develop known applications and propose new applications of stack and queue layouts. While several measures of stack layouts and queue layouts have been defined, we focus on *stacknumber* as a measure of a stack layout and *queuenumber* as a measure of a queue layout. We now describe the results in this dissertation in greater detail.

1.5.1 Tools

We first present two powerful tools that greatly contribute to the results presented in the rest of the dissertation. These tools are also of independent interest because they lead to solutions for problems outside the area of stack and queue layouts of graphs.

The first tool is a formulation of a queue layout of a graph as a *staircase cover* of its adjacency matrix. Several algorithmic and structural results appear in the literature that have been obtained by viewing the problem of minimizing the bandwidth of a matrix as a graph linearization problem in which the bandwidth of a graph is minimized. [13,14,17,18,55,29]. Going in the opposite direction, we take a queue layout from its graph theoretic domain and recast it as a matrix covering problem, and obtain several combinatorial and structural results for queue layouts. Motivated by the connection between a queue layout and covering a matrix with staircases, we propose and solve a general matrix covering problem. This leads to efficient and simple solutions to the *maxdominance problems* proposed by Atallah and Kosaraju [3]. It also leads to efficient solutions to the *minimum independent dominating set problem* for permutation graphs [4,25,3] and to the *largest empty rectangle problem*

[2,1,12,52]. A staircase cover can also be used as the basis of a new and efficient scheme for parallel matrix computations on a data driven network. A data driven network can be thought of as an asynchronous version of a systolic array. In a data driven network, a processing element does not execute a task at each clock beat, instead it waits for data to appear at its inputs and executes the task as soon as it has all the inputs required for the task. Our scheme is an extension of the scheme proposed by Melhem [51].

The second tool is a set of upper bound results for the queuenumber and stacknumber of *separated* and *mingled* layouts of graphs. A separated layout obeys an order on the vertices of a graph in which certain specified subsets of vertices are constrained to appear contiguously, though each subset may occur in any relative order with respect to the other subsets. A mingled layout is a layout in which subsets of vertices are constrained to appear in a particular order, though the subsets may be mingled with each other in any possible manner. Upper bounds on the stacknumber and queuenumber of separated layouts and mingled layouts can be used to simplify existing lower bound arguments and prove new lower bound results. This is demonstrated by the application of separated and mingled layouts to problems in the remainder of the dissertation, in particular in Chapter 6, where we address the queuenumber of planar graphs.

1.5.2 Research Areas

The two tools described above are applied to three research areas.

The first area is stack and queue layouts of *directed acyclic graphs* (dags). This area is motivated by a direct application to managing the flow of data in a parallel processing system [56]. We first establish the stacknumber and queuenumber of the following classes of dags: trees, unicyclic graphs, outerplanar graphs, and planar graphs. We provide examples of families of dags with contrasting stacknumber and queuenumber. We then address the problems of recognizing a 1-stack dag and recognizing a 1-queue dag. We present a linear time algorithm to recognize 1-stack dags and a linear time algorithm recognize leveled planar dags. A 1-queue dag is an arched leveled-planar dag, and we conjecture that our algorithm

to recognize leveled-planar dags can be extended to recognize arched leveled-planar dags efficiently. Our result should be compared with the result of Heath and Rosenberg [36] that shows that the problem of recognizing a leveled planar graph is NP-complete. We then show that the problems of recognizing a 9-stack dag and of recognizing a 4-queue dag are both NP-complete.

The second area is stack and queue layouts of partially ordered sets (posets). Posets are fundamental mathematical objects that have been studied in great depth and have been used to model a variety of problems. Our immediate motivation is the study of Sysło [60] and Nowakowski and Parker [53], who define the stacknumber of a poset as the stacknumber of its Hasse diagram. They also suggest the stacknumber of a poset as a measure of its structure, in the same sense that length, width, jumpnumber, dimension, and thickness are measures of the structure of a poset. We present upper bounds on the queuenumber of a poset in terms of its length, width, and jumpnumber. We present lower bound results that show that the queuenumber and the stacknumber of certain classes of posets are unbounded. We prove that the problem of recognizing a 4-queue poset is NP-complete.

The third area that we address is the queuenumber of undirected planar graphs. As mentioned in Section 1.3, Yannakakis [64] has shown that the stacknumber of planar graphs is 4, but the queuenumber of planar graphs is not known. We conjecture that the queuenumber of planar graphs is unbounded and that the class of stellated triangles (defined in Chapter 6) is a witness to a nonconstant lower bound. Using the tool of separated and mingled layouts, we show that the queuenumber of stellated triangles is unbounded for various natural classes of orders on the vertices. While the problem remains unresolved, we feel that we have presented the tools and insights that will ultimately lead to a resolution of the problem.

1.6 Organization

The organization of the rest of the dissertation is as follows. In Chapter 2, we establish a connection between queue layouts and covering a matrix with staircases. We exploit this connection to obtain combinatorial and lower bound results for queue layouts of graphs and to efficiently solve a general matrix covering problem. We also develop a new and efficient scheme to perform matrix computations on a data driven network, based on covering a matrix with staircases. In Chapter 3, we define and prove upper bound results for separated and mingled stack and queue layouts. These results are tools that are useful in proving lower bounds on the stacknumber and queuenumber of families of graphs. Chapter 4 is a study of stack and queue layouts of dags. In this chapter, we begin by establishing the stacknumber and queuenumber of various families of dags and then address related algorithmic questions. We present a linear time algorithm for recognizing 1-stack dags, a linear time algorithm for recognizing leveled-planar dags, and NP-completeness proofs for the problems of recognizing a 9-stack dag and a 4-queue dag. Chapter 5 is a study of stack and queue layouts of posets. We present bounds on the queuenumber of a poset in terms of other measures of its structure such as length, width, and jumpnumber. We then concentrate on certain classes of posets and show that the queuenumber and stacknumber of these classes is unbounded. Chapter 6 addresses the question of the queuenumber of planar graphs through a succession of bounds on the queuenumber of certain kinds of layouts of stellated triangles. Chapter 7 contains conclusions, open questions, and conjectures.

Chapter 2

Queue Layouts and Matrix Covers

A queue layout of a graph can be characterized in terms of covering its adjacency matrix with *staircases*. A staircase can be thought of as a path consisting of alternating horizontal and vertical line segments descending to the right. This formulation is an entirely new way of examining queue layouts and has several advantages. It simplifies combinatorial analysis of queue layouts considerably and allows us to prove new combinatorial results and simplify existing combinatorial arguments. As examples we present a simple argument for determining the maximum number of edges in a 1-queue graph, give a tight upper bound on the maximum number of edges in a k-queue graph, give an improved lower bound on the queuenumber of arbitrary graphs, and count the exact number of maximal 1-queue layouts. The matrix cover formulation also provides a new tool for constructing lower and upper bound arguments for the queuenumber of graphs. As examples we present proofs to show that $QN(K_n) = \lfloor n/2 \rfloor$ and $QN(K_{m,n}) = min(\lceil n/2 \rceil, \lceil m/2 \rceil)$. These proofs are very different from the proofs of Heath and Rosenberg [36].

The connection between queue layouts and matrix covers has proved useful from an algorithmic point of view also. Inspired by the algorithm of Heath and Rosenberg [36] that computes the queuenumber of a fixed layout of a graph, we provide an efficient algorithm to solve a general class of matrix covering problems. Our solution to the matrix covering problems leads to alternate and possibly simpler solutions to problems related to sequences, graph theory, and computational geometry. Examples are of these problems are:

- Largest Monotonic Subsequence problem (see Dijkstra, Orlowski and Pachter, Bar-Yehuda and Fogel [19,54,5]).

- Maxdominance problems (see Atallah and Kosaraju [3])
- Minimum Independent Dominating Set problem for permutation graphs (see Farber and Keil [25], Atallah, Manacher, and Urrutia [4], or Atallah and Kosaraju [3]).
- Largest Empty Rectangle problem (see Atallah and Frederickson [2], Aggarwal and Suri [1], Chazelle, Drysdale, and Lee [12], or Naamad, Lee, and Hsu [52]).

Finally, we show that a staircase cover of a matrix can be used as the basis for an efficient scheme for performing matrix computations on data driven networks. This scheme, called the Parallel Matrix Computation Network (PMCN), is a generalization of the schemes proposed by Leiserson [45] and Melhem [51]. We show that PMCN typically requires substantially fewer processing elements than Melhem's or Leiserson's scheme. This reduction in hardware requirement does not come at the cost of time; we show that the number of time steps required by PMCN is no more than the number of time steps required by Melhem's scheme. We give a precise characterization of the number of time steps required by PMCN to perform a matrix vector multiplication, in terms of a structural property of the matrix. Based on this characterization, we show an efficient way to compute both the time steps and the number of processing elements required by PMCN. This characterization applies to Melhem's scheme as well, thereby answering a question asked in his paper.

The organization of this chapter is as follows. Section 2.1 provides definitions related to covering a matrix with staircases. In Section 2.2, we establish the connection between queue layouts and covering a matrix with staircases. In Section 2.3, we prove combinatorial results related to queue layouts and determine the queuenumber of the complete graph and the complete bipartite graph. Section 2.4 defines a class of matrix cover problems and presents an algorithm to solve any problem in that class for an $n \times n$ matrix in $O(T(M)\log\log n)$ time, where $T(M)$ is the time required to visit all the non-zeroes in M. We then show how this algorithm leads to efficient solutions to algorithmic questions related to sequences, maxdominance problems proposed by Atallah and Kosaraju, minimum independent dominating set problem for permutation graphs, and the largest empty rectangle problem. In

Section 2.5, we show how covering a matrix with staircases can be used to obtain a new scheme for performing matrix computations in parallel on a data driven network. Analysis of our scheme reveals that it is an improvement over existing schemes in terms of time as well as hardware.

2.1 Staircases

Throughout the chapter, $M = (m_{i,j})_{n\times n}$ is an $n \times n$ matrix. Let $J_n = \{1, 2, \ldots, n\}$ be the set of row and column indices of M. Each ordered pair $(i,j) \in J_n \times J_n = J_n^2$ is a *position index* of M. Define the *southeast partial order* (or poset) $(J_n^2, \leq_{se})$ by $(i_1, j_1) \leq_{se} (i_2, j_2)$ if and only if $i_1 \leq i_2$ and $j_1 \leq j_2$. Thus we may interpret $(i_1, j_1) <_{se} (i_2, j_2)$ as (i_2, j_2) is either to the right (east) of or below (south) (i_1, j_1) or both. A *staircase* $\Psi \subseteq J_n^2$ is a nonempty chain in the poset $(J_n^2, \leq_{se})$ (a set of position indices, every pair of which is related by $\leq_{se}$). The smallest element in Ψ with respect to $\leq_{se}$ is the *source* of Ψ and the largest element is the *destination* of Ψ. A *maximal staircase* Ψ is a staircase that is not a proper subset of any other staircase having the same source and destination. Let Ψ be a maximal staircase given by

$$(i_0, j_0) <_{se} (i_1, j_1) <_{se} \cdots <_{se} (i_r, j_r).$$

For any pair of position indices (i_m, j_m) and (i_{m+1}, j_{m+1}), $0 \leq m \leq r-1$, either

$$(i_{m+1}, j_{m+1}) = (i_m + 1, j_{m+1}) \quad \text{or} \quad (i_{m+1}, j_{m+1}) = (i_{m+1}, j_m + 1);$$

otherwise, there is a position index $(i,j) \notin \Psi$, with $(i_m, j_m) <_{se} (i,j) <_{se} (i_{m+1}, j_{m+1})$, such that $\Psi \cup \{(i,j)\}$ is a staircase having the same source and destination as Ψ, thus violating the maximality of Ψ. Therefore, a maximal staircase Ψ can be visualized as a series of alternating horizontal and vertical line segments descending to the right and forming an unbroken path from the source to the destination.

Example 2.1. Figure 2.1 shows two maximal staircases in a 5×5 matrix. The staircase Ψ_1 has source $(1,1)$ and destination $(5,5)$. The staircase Ψ_2 has source $(2,1)$ and destination

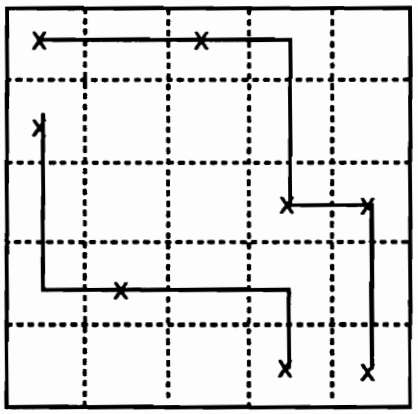


Figure 2.1: Two maximal staircases.

$(5,4)$. The nonzero entries in the matrix are shown by X. They will become relevant when we define the notion of a staircase cover. □

The *size* of a staircase Ψ is its cardinality $|\Psi|$.

Proposition 2.2 *The size of a maximal staircase* Ψ *with source* (i_0, j_0) *and destination* (i_r, j_r) *is*

$$|\Psi| = r + 1 = (i_r - i_0) + (j_r - j_0) + 1.$$

This is seen by noting that the number of elements in Ψ is one more than the length of a path from (i_0, j_0) to (i_r, j_r) consisting of a series of alternating horizontal and vertical line segments. The length of this path is simply the distance between (i_0, j_0) and (i_r, j_r) measured in the L_1 or Manhattan metric. Notice that the size of the staircase depends only on the source and the destination of the staircase and not on its shape.

Example 2.3. The size of the two staircases Ψ_1 and Ψ_2 shown in Figure 2.1 are:

$$\begin{aligned} |\Psi_1| &= (5-1) + (5-1) + 1 = 9 \\ |\Psi_2| &= (5-2) + (4-1) + 1 = 7 \end{aligned}$$

□

The position index (i,j) is said to *cover* the corresponding entry $m_{i,j}$ of M. Let $\Gamma \subseteq J_n^2$ be the subset of all position indices covering nonzero entries of M; that is,

$$\Gamma = \{(i,j) \mid m_{i,j} \neq 0\}.$$

A matrix M is *covered* by k staircases if there exists a set $\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ of k staircases such that

$$\Gamma \subseteq \Psi_1 \cup \Psi_2 \cup \cdots \cup \Psi_k;$$

the set Π is a *staircase cover* of M. If k is the smallest natural number such that M has a k-staircase cover, then k is the *staircase number* of M, denoted $SCN(M)$, and any k-staircase cover of M is an *optimal* or *minimal staircase cover* of M.

Example 2.4. In Figure 2.1, the nonzero entries in the matrix are indicated by X. Since, all the nonzeros in the matrix are covered by $\Psi_1 \cup \Psi_2$, $\{\Psi_1, \Psi_2\}$ is a staircase cover of the matrix. In fact, $\{\Psi_1, \Psi_2\}$ is the minimal staircase cover of M. □

Define the *northeast partial order* $(J_n^2, \leq_{ne})$ by $(i_1,j_1) <_{ne} (i_2,j_2)$ if and only if $i_1 > i_2$ and $j_1 < j_2$. An *antistaircase* AC is a chain in the poset $(J_n^2, \leq_{ne})$. Note that an antistaircase is an antichain in $(J_n^2, \leq_{se})$. Clearly, the size of a largest antistaircase in Γ is a lower bound on the staircase number of M.

Example 2.5. The largest antistaircase in Figure 2.1 is of size 2; the sets

$$\{(2,1),(1,3)\}$$

and

$$\{(4,2),(3,4)\}$$

are two examples of antistaircases in the matrix. We conclude that $\{\Psi_1, \Psi_2\}$ is a minimal staircase cover of the matrix. □

In Example 2.1, the size of the largest antistaircase equals the staircase number of the matrix. This is not a coincidence. Dilworth's Theorem [20] (see Proposition 1.4) asserts that the minimum number of chains required to cover a finite poset equals the size of a

largest antichain in the poset. Applying Dilworth's Theorem to the poset $(\Gamma, \leq_{se})$, we obtain the following Proposition.

Proposition 2.6 *The size of the largest antistaircase in* Γ *equals* $SCN(M)$.

Hence determining $SCN(M)$ is equivalent to finding the size of the largest antistaircase in Γ. But, beyond finding the staircase number of M, our goal is to find a minimal staircase cover Π of M such that the staircases in Π can be ordered in a "natural" way. The natural order on staircases that we require, is defined by extending the partial order $\leq_{ne}$ defined on position indices, to staircases. We first define a partial order $\leq_{\gamma}$ on the set of staircases in J_n^2 and subsequently show that, for any matrix M, it is possible to construct a minimal staircase cover Π of M such that Π is totally ordered by $\leq_{\gamma}$. The binary relation $\leq_{\gamma}$ is defined as follows. For a pair of staircases Ψ_1 and Ψ_2 in J_n^2, $\Psi_2 \leq_{\gamma} \Psi_1$ if

1. There is a position index $(i_1, j_1) \in \Psi_1$ and a position index $(i_2, j_2) \in \Psi_2$ such that $(i_2, j_2) <_{ne} (i_1, j_1)$.

2. For any pair of position indices $(p_1, q_1) \in \Psi_1$ and $(p_2, q_2) \in \Psi_2$, it is not the case that $(p_1, q_1) <_{ne} (p_2, q_2)$.

Example 2.7. Figure 2.1 shows a pair of staircases Ψ_1 and Ψ_2 such that $\Psi_2 \leq_{\gamma} \Psi_1$. Note that $(4,4) \in \Psi_2$, $(3,5) \in \Psi_1$, and $(4,4) \leq_{ne} (3,5)$. On the other hand, there is no pair of position indices $(i,j) \in \Psi_1$ and $(p,q) \in \Psi_2$ such that $(i,j) <_{ne} (p,q)$. □

Theorem 2.8 *Every matrix* M *has a staircase cover consisting of* $SCN(M)$ *staircases that are totally ordered by* $\leq_{\gamma}$.

Proof: The proof uses the sketch of an algorithm called the **Staircase Cover Algorithm (SCA)** (see Figure 2.2). SCA takes as input a matrix M and returns as output a sequence of nonempty sets of position indices:

$$\Psi_1, \Psi_2, \ldots, \Psi_k.$$

$S_0 := \Gamma$;
$\ell := 1$;
while $(S_{\ell-1} \neq \phi)$ **do**
    let Ψ_ℓ be the set of maxima in $(S_{\ell-1}, \leq_{ne})$;
    $S_\ell := S_{\ell-1} - \Psi_\ell$;
    $\ell := \ell + 1$;
endwhile
return $\Psi_1, \Psi_2, \ldots, \Psi_{\ell-1}$

Figure 2.2: A sketch of the **Staircase Cover Algorithm (SCA)**

We show that this sequence of staircases is a staircase cover of M and that $\Psi_r \leq_\gamma \Psi_{r-1}$ for all r, $2 \leq r \leq k$. Assume that SCA terminates after k sets $\Psi_1, \Psi_2, \ldots, \Psi_k$ have been constructed. The requisite properties of these sets are established below.

1. **Π is a staircase cover of M.** If $(i_1, j_1), (i_2, j_2) \in \Psi_\ell$ are distinct position indices, then neither $(i_1, j_1) <_{ne} (i_2, j_2)$ nor $(i_2, j_2) <_{ne} (i_1, j_1)$. Hence either $(i_1, j_1) \leq_{se} (i_2, j_2)$ or $(i_2, j_2) \leq_{se} (i_1, j_1)$. We conclude that Ψ_ℓ is a staircase in M. When SCA terminates,

$$\Gamma = \Psi_1 \cup \Psi_2 \cup \cdots \cup \Psi_k.$$

   Therefore $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ is a staircase cover of M.

2. **Π is a minimal staircase cover.** For an arbitrary element $(i_k, j_k) \in \Psi_k$, there exists at least one element $(i_{k-1}, j_{k-1}) \in \Psi_{k-1}$ such that $(i_k, j_k) <_{ne} (i_{k-1}, j_{k-1})$. Otherwise, (i_k, j_k) would have been included in Ψ_{k-1}. Extending this argument inductively, it is easy to see that there exists a sequence of position indices

$$(i_k, j_k) <_{ne} (i_{k-1}, j_{k-1}) <_{ne} \cdots <_{ne} (i_1, j_1)$$

   such that $(i_\ell, j_\ell) \in \Psi_\ell \subseteq \Gamma$ for all ℓ, $1 \leq \ell \leq k$. This implies that there is an antistaircase of size k in M. By Lemma 2.6, $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ is a minimal staircase cover of M.

3. **$\leq_\gamma$ is a total order on Π.** Consider an arbitrary pair of staircases Ψ_p and Ψ_q in Π such that $p < q$. As shown in the proof of property 2, there is a position

index $(p_1,p_2) \in \Psi_p$ and a position index $(q_1,q_2) \in \Psi_q$ such that $(q_1,q_2) <_{ne} (p_1,p_2)$. Now consider a pair of arbitrary position indices $(i_1,j_1) \in \Psi_p$ and $(i_2,j_2) \in \Psi_q$. If $(i_1,j_1) <_{ne} (i_2,j_2)$ then due the way SCA assigns position indices to staircases, $p > q$. This is a contradiction. Hence, for any pair of position indices $(i_1,j_1) \in \Psi_p$ and $(i_2,j_2) \in \Psi_q$, it is not the case that $(i_1,j_1) <_{ne} (i_2,j_2)$. Hence, $\Psi_q \leq_\gamma \Psi_p$ for all p,q such that $1 \leq p < q \leq k$.

□

Example 2.9. For the matrix shown in Figure 2.1, the sets Ψ_1 and Ψ_2 as constructed by SCA are

$$\begin{aligned} \Psi_1 &= \{(1,1),(1,3),(3,4),(3,5),(5,5)\} \\ \Psi_2 &= \{(2,1),(4,2),(5,4)\}. \end{aligned}$$

□

What is shown in Figure 2.2 is only a sketch and not the algorithm itself because, it contains steps whose details are left unclear. In Section 2.4 we rectify this by presenting SCA in detail and analyzing its performance.

2.2 The Connection

In this section, we establish the connection between a queue layout of a graph and covering its adjacency matrix with staircases. The basic idea behind this connection is expressed in the following observation:

> Two edges in the layout of a graph that can be assigned to the same queue correspond to nonzero entries in the upper triangle of the adjacency matrix of the graph whose position indices can be covered with the same staircase.

In Theorem 2.10, we consider staircase covers of a symmetric matrix with zero entries on the main diagonal. We show that a minimal staircase cover of such a matrix can be obtained

by constructing a minimal staircase cover of the upper triangle and then "reflecting" it to obtain a minimal staircase cover of the lower triangle. The staircase cover of the upper triangle together with its reflection yields the staircase cover of the whole matrix. As the adjacency matrix of a graph is a special case of a symmetric matrix with zero entries on the diagonal, we are able to apply this result to queue layouts of graphs.

Since we need to refer to the upper triangle and the lower triangle of M, we introduce the following notation. Let $U(M) = (u_{i,j})_{n\times n}$ such that $u_{i,j} = m_{i,j}$ if $i < j$; otherwise $u_{i,j} = 0$. Therefore $U(M)$ is the upper triangle of M excluding the main diagonal. Similarly, define $L(M) = (\ell_{i,j})_{n\times n}$ such that $\ell_{i,j} = m_{i,j}$ if $j < i$; otherwise $\ell_{i,j} = 0$. Therefore $L(M)$ is the lower triangle of M excluding the main diagonal. The *reflection* $R(T)$ of a set of position indices $T \subseteq J_n^2$ in M is

$$R(T) = \{(i,j) \mid (j,i) \in T\}.$$

Note that if Γ is the set of position indices covering the nonzero entries in M, then $R(\Gamma)$ is the set of position indices covering the nonzero entries in the transpose of M. Two properties of sets of position indices that remain invariant under reflection are:

1. If Ψ is a staircase, then $R(\Psi)$ is also a staircase.

2. If A is an antistaircase in M such that all of its elements lie in the upper triangle of M, then $R(A)$ is an antistaircase in M such that all of its elements lie in the lower triangle of M. Furthermore, $A \cup R(A)$ is also an antistaircase in M.

Theorem 2.10 *Let M be symmetric with all main diagonal entries 0. Then,*

$$2 \cdot SCN(L(M)) = 2 \cdot SCN(U(M)) = SCN(M).$$

Proof: By Property 1 above, $\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ is a staircase cover of $U(M)$ if and only if

$$R(\Pi) \;=\; \{R(\Psi_1), R(\Psi_2), \ldots, R(\Psi_k)\}$$

is a staircase cover of $L(M)$. Therefore, $SCN(U(M)) = SCN(L(M))$.

If $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ is a staircase cover of $U(M)$, then

$$\{\Psi_1, \Psi_2, \ldots, \Psi_k, R(\Psi_1), R(\Psi_2), \ldots, R(\Psi_k)\}$$

is a staircase cover of M. Therefore, $SCN(M) \leq 2 \cdot SCN(U(M))$. On the other hand, if A is an antistaircase in $U(M)$, then $A \cup R(A)$ is an antistaircase in M by Property 2 above. Therefore, $SCN(M) \geq 2 \cdot SCN(U(M))$.

The theorem follows. □

For the purposes of covering a matrix with staircases, Theorem 2.10 allows us to ignore the lower triangle in a symmetric matrix with zeros in the main diagonal and concentrate on the upper triangle. Now we are ready to establish the connection between covering a matrix with staircases and the queue layout of a graph.

Suppose $G = (V, E)$ is a graph with n vertices and σ is a total order on V. Label the vertices $1, 2, \ldots, n$ according to σ, that is, label a vertex $v \in V$ with $\sigma^{-1}(v)$. Let $AM(G, \sigma) = (g_{i,j})_{n \times n}$ be the adjacency matrix of G for the order σ:

$$g_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E; \\ 0 & \text{otherwise.} \end{cases}$$

Note that $AM(G, \sigma)$ is a symmetric binary matrix with all its main diagonal entries being zeros. Define

$$AM(G) = \{AM(G, \sigma) \mid \sigma \text{ a total order on } V\}.$$

Note that each element of $AM(G)$ can be obtained from any other by a symmetric permutation of its rows and columns. Recall that $P \cdot M \cdot P^T$ is a symmetric permutation of M, where P is a some permutation matrix and P^T is its transpose. Thus for a fixed total order σ on V, it is also true that

$$AM(G) = \{P \cdot AM(G, \sigma) \cdot P^T \mid P \text{ a permutation matrix}\}.$$

Theorem 2.11 *Let $G = (V, E)$ be a graph. If σ is an order on V that yields a k-queue layout of G, then $SCN(AM(G,\sigma)) \leq 2k$. If $SCN(AM(G,\sigma)) = 2k$ for some order σ on V, then σ yields a k-queue layout of G.*

Proof: Suppose σ is an order on V that yields a k-queue layout of G. Then the largest rainbow in L is of size k (see Heath and Rosenberg [36]). Let

$$(i_1, j_1) <_{ne} (i_2, j_2), <_{ne} \cdots <_{ne} (i_t, j_t)$$

be a largest antistaircase in $U(AM(G,\sigma))$. Then, the set of edges

$$\{(i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)\}$$

forms a rainbow in L. Therefore $t \leq k$. By Theorem 2.10, the largest antistaircase in $AM(G,\sigma)$ is of size $2t \leq 2k$. By Lemma 2.6, $SCN(AM(G,\sigma)) \leq 2k$.

Suppose $SCN(AM(G,\sigma)) = 2k$. By Lemma 2.6 a largest antistaircase in $AM(G,\sigma)$ is of size $2k$. By Theorem 2.10, a largest antistaircase in $U(AM(G,\sigma)$ is of size k. Let

$$\{(i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)\}$$

be a set of edges in E that forms a rainbow in L. Then, the set of position indices

$$\{(i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)\}$$

is an antistaircase in $U(AM(G,\sigma))$. Therefore $t = k$, and L is a k-queue layout of G. □

Example 2.12. Figure 2.3 illustrates the connection between a queue layout of a graph and the staircase cover of its adjacency matrix. Figure 2.3 (a) shows a graph $G = (V, E)$ and Figure 2.3 (b) shows a 3-queue layout of G according to $\sigma = (1,2,3,4,5,6,7)$. In Figure 2.3 (b), the plain edges shown above the line of vertices are assigned to a queue q_1, the plain edges shown below the line of vertices are assigned to queue q_2, and the single dashed edge is assigned to queue q_3. Figure 2.3 (c) shows the adjacency matrix of G, $AM(G,\sigma)$.

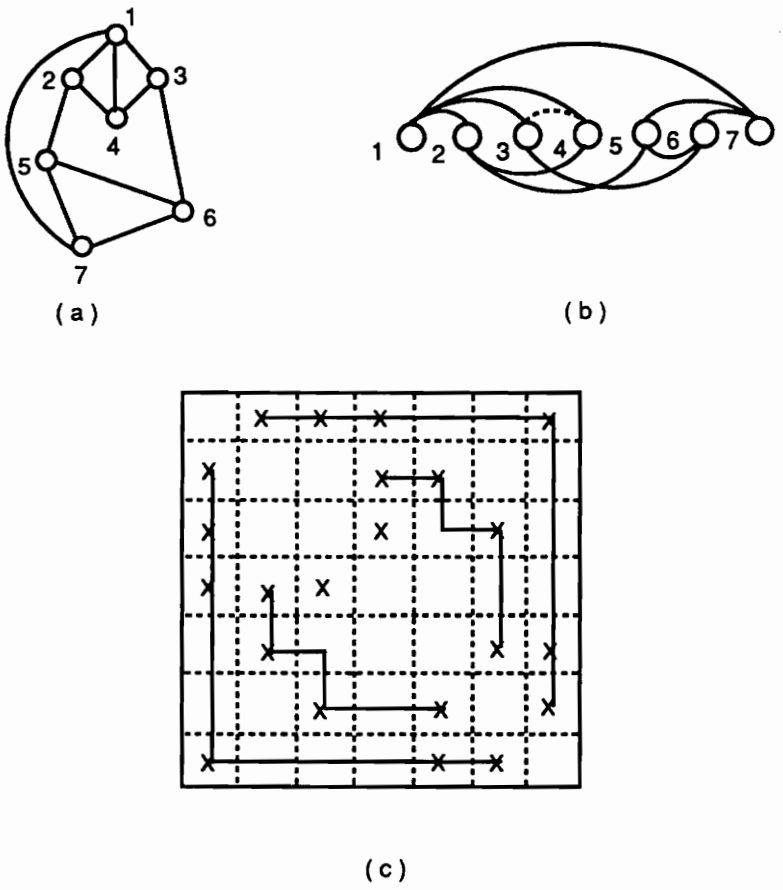


Figure 2.3: The connection between a queue layout of a graph and the staircase cover of its adjacency matrix.

Corresponding to the edges assigned to q_i, there are two staircases Ψ_i and $R(\Psi_i)$, where Ψ_i contains position indices that belong to the upper triangular portion on M. Note that

$$R(\Psi_1) \leq_\gamma R(\Psi_2) \leq_\gamma R(\Psi_3) \leq_\gamma \Psi_3 \leq_\gamma \Psi_2 \leq_\gamma \Psi_1.$$

□

In the subsequent sections, we show how the connection between queue layouts of graphs and covering matrices with staircases, established in Theorem 2.11, allows us to prove new and useful results about queue layouts as well as matrix covers.

2.3 Combinatorial Results

The formulation in Section 2.2 of a queue layout as a staircase cover of a matrix leads to new combinatorial results and simplifies existing ones related to queue layouts. In this section, we present a simple argument for determining the maximum number of edges in a 1-queue graph, give a tight upper bound on the maximum number of edges in a k-queue graph, an improved lower bound on the queuenumber of arbitrary graphs, and count the exact number of maximal 1-queue layouts. The staircase cover formulation can also be used as a tool to establish lower and upper bounds on the queuenumber of specific families of graphs. We demonstrate this by showing that the queuenumber of K_n is $\lfloor n/2 \rfloor$ and that the queuenumber of $K_{m,n}$ is $\min(\lceil m/2 \rceil, \lceil n/2 \rceil)$. Our proofs are very different from the proofs of Heath and Rosenberg [36].

Viewing a queue layout as a staircase cover of a matrix allows for easier combinatorial analysis. For example, Heath and Rosenberg [36] have shown that the maximum number of edges in a 1-queue graph with n vertices is $2n - 3$. The argument was based on their characterization of 1-queue graphs as arched leveled-planar graphs. As shown below, the upper bound on the number of edges in a 1-queue graph immediately follows from the characterization of the adjacency matrix of a 1-queue graph as a symmetric binary matrix whose staircase number is 2.

Theorem 2.13 *The maximum number of edges in a 1-queue graph with n vertices is $2n-3$.*

Proof: Suppose $G = (V, E)$ is a 1-queue graph with n vertices and σ is a total order on V that yields a 1-queue layout. Then, by Theorem 2.11, $SCN(AM(G, \sigma)) = 2$. Let $\Pi = \{\Psi, R(\Psi)\}$ be a staircase cover of $AM(G, \sigma)$ such that Ψ only contains position indices belonging to the upper triangle of $AM(G, \sigma)$. Therefore, $|\Psi|$ is an upper bound on the number of edges in G. $|\Psi|$ is maximized when it has source $(1, 2)$ and destination $(n-1, n)$. By Proposition 2.2,

$$|\Psi| \leq (n - 1 - 1) + (n - 2) + 1 = 2n - 3.$$

It remains to show that this upper bound is tight. Consider the graph $G = (V, E)$ with

$$\begin{aligned} V &= \{v_1, v_2, \ldots, v_n\} \\ E &= \{(v_1, v_i) \mid 2 \leq i \leq n\} \cup \{(v_n, v_i) \mid 2 \leq i \leq n-1\}. \end{aligned}$$

G has $2n - 3$ edges, and a 1-queue layout of G is obtained by laying out the vertices in V in the order $v_1, v_2, \ldots, v_n$. The theorem follows. □

Theorem 2.13 implies that a k-queue graph $G = (V, E)$ has no more than $k(2|V| - 3)$ edges. Heath and Rosenberg [36] use this observation to derive a lower bound of $|E|/(2|V| - 3)$ on the queuenumber of G. For $k \geq 2$, this bound is not tight. We show a tight upper bound in the following theorem.

Theorem 2.14 *The maximum number of edges in a k-queue graph with n vertices is*

$$k(2n - 3) - 2k(k - 1).$$

For each k, $k \geq 1$, there exists a k-queue graph with $n = 2k$ vertices and with $k(2n - 3) - 2k(k - 1)$ edges.

Proof: Suppose $G = (V, E)$ is a k-queue graph and σ is a total order on V that yields a k-queue layout of G. Let $M = U(AM(G, \sigma))$. Let $\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ be the staircase

cover of M produced by the algorithm SCA. We claim that for all ℓ, $1 \leq \ell \leq k$,

$$|\Psi_\ell| \leq (2n-3) - 4(\ell-1).$$

Let the source of Ψ_ℓ be (i_ℓ, j_ℓ) and the destination be (i'_ℓ, j'_ℓ). There exists an element $(i_{\ell-1}, j_{\ell-1}) \in \Psi_{\ell-1}$ such that $(i_\ell, j_\ell) <_{ne} (i_{\ell-1}, j_{\ell-1})$. Otherwise, (i_ℓ, j_ℓ) would have been included in $\Psi_{\ell-1}$. This can be inductively extended to show that there exists an antistaircase

$$(i_\ell, j_\ell) <_{ne} (i_{\ell-1}, j_{\ell-1}) <_{ne} \cdots <_{ne} (i_1, j_1) \tag{2.1}$$

such that $(i_p, j_p) \in \Psi_p$ for all p, $1 \leq p \leq \ell$. Similarly, there exists a second antistaircase of size ℓ involving destinations:

$$(i'_\ell, j'_\ell) <_{ne} (i'_{\ell-1}, j'_{\ell-1}) <_{ne} \cdots <_{ne} (i'_1, j'_1). \tag{2.2}$$

From the relationship in (2.1), we obtain

$$i_\ell > i_{\ell-1} > \cdots > i_2 > i_1 \geq 1$$

and hence $i_\ell \geq \ell$. From the relationship in (2.2), we obtain

$$j'_\ell < j'_{\ell-1} < \cdots < j'_2 < j'_1 \leq n$$

and hence $j'_\ell \leq n-\ell+1$. These conditions along with the fact that (i_ℓ, j_ℓ) and (i'_ℓ, j'_ℓ) belong to the upper triangle of $AM(G, \sigma)$ imply that

$$\ell \leq i_\ell < j_\ell$$

and

$$i'_\ell < j'_\ell \leq n - \ell + 1.$$

Recall that $|\Psi_\ell| \leq (i'_\ell - i_\ell) + (j'_\ell - j_\ell) + 1$. Thus, $|\Psi_\ell|$ is maximized when

$$(i_\ell, j_\ell) = (\ell, \ell+1)$$

and

$$(i'_\ell, j'_\ell) = (n-\ell, n-\ell+1).$$

Therefore,

$$\begin{aligned} |\Psi_\ell| &\leq [(n-\ell)-\ell]+[(n-\ell+1)-(\ell+1)]+1 \\ &= (2n-3)-4(\ell-1). \end{aligned}$$

Since, each edge in G corresponds to a distinct element in Ψ_ℓ, $1 \leq \ell \leq k$, the number of edges in a k-queue graph can be at most

$$\begin{aligned} \sum_{\ell=1}^{k} |\Psi_\ell| &\leq \sum_{\ell=1}^{k} ((2n-3)-4(\ell-1)) \\ &= k(2n-3)-2k(k-1). \end{aligned}$$

Now we show that the upper bound proved above is tight. For each $k \geq 1$, choose $n = 2k$ and consider the complete graph on n vertices, K_n. Heath and Rosenberg [36] have shown that $QN(K_n) = \lceil \frac{n}{2} \rceil = k$ and hence K_n is a k-queue graph. The upper bound of $k(2n-3)-2k(k-1)$ dictates that K_n have at most

$$\begin{aligned} & \frac{n}{2}(2n-3)-n\left(\frac{n}{2}-1\right) \\ = \; & \frac{n(n-1)}{2} \end{aligned}$$

edges. But, this is precisely the number of edges that K_n has. The theorem follows. □

In the next Theorem, we use the upper bound proved in Theorem 2.14 to derive bounds on the queuenumber of a graph as a function of its number of vertices and its number of edges.

Theorem 2.15 *Suppose that $G = (V, E)$ be a graph with n vertices and m edges. Then the queuenumber of G satisfies the following bounds:*

$$\frac{(2n-1)-\sqrt{(2n-1)^2-8m}}{4} \leq QN(G) \leq \frac{(2n-1)+\sqrt{(2n-1)^2-8m}}{4}.$$

Proof: Suppose $G = (V, E)$ is a k-queue graph with n vertices. Theorem 2.14 gives the following upper bound on its number of edges:

$$m \leq k(2n-3)-2k(k-1).$$

Simplification leads to

$$2k^2 + (1 - 2n)k + m \leq 0.$$

Factoring this quadratic in k, we obtain

$$(k - k_1)(k - k_2) \leq 0, \tag{2.3}$$

where

$$\begin{aligned} k_1 &= \frac{1}{4}\left((2n-1) + \sqrt{(2n-1)^2 - 8m}\right) \\ k_2 &= \frac{1}{4}\left((2n-1) - \sqrt{(2n-1)^2 - 8m}\right) \end{aligned}$$

Since, $m \leq n(n-1)/2$,

$$(2n-1)^2 - 8m \geq 1$$

and hence k_1 and k_2 are real and $k_1 > k_2$. From inequality 2.3, it follows that

$$k \leq k_1 \quad \text{and} \quad k \geq k_2$$

or

$$k \geq k_1 \quad \text{and} \quad k \leq k_2.$$

Since, $k_1 > k_2$, we can only have

$$k \leq k_1 \quad \text{and} \quad k \geq k_2.$$

The theorem follows. □

When $m = n(n-1)/2$, the bounds proved in Theorem 2.15 simplify to

$$\frac{n-1}{2} \leq QN(G) \leq \frac{n}{2}.$$

This leads to the following corollary.

Corollary 2.16

$$QN(K_n) = \left\lfloor \frac{n}{2} \right\rfloor.$$

Heath and Rosenberg [36] have shown that $QN(K_n) = \lfloor n/2 \rfloor$ by establishing a lower bound of $\lfloor n/2 \rfloor$ on $QN(K_n)$ and then explicitly constructing a queue layout of K_n with $\lfloor n/2 \rfloor$ queues. In contrast with that proof, the above corollary is based simply on the number of edges that K_n has.

When $m < n(n-1)/2$, the upper bound proved in Theorem 2.15 is greater than $n/2$ and is therefore useless. The lower bound is more interesting, and it is a slight improvement on the lower bound proved by Heath and Rosenberg [36]. This can be seen as follows. Theorem 2.15 shows that

$$\frac{1}{4}\left((2n-1) - \sqrt{(2n-1)^2 - 8m}\right) \leq QN(G)$$

for a graph G that has n vertices and m edges. Therefore

$$\begin{aligned} QN(G) &\geq \frac{2n-1}{4}\left(1 - \sqrt{1 - \frac{8m}{(2n-1)^2}}\right) \\ &\geq \frac{2n-1}{4}\left(1 - \left(1 - \frac{8m}{2(2n-1)^2}\right)\right) \\ &= \frac{m}{(2n-1)} \end{aligned}$$

Thus Theorem 2.15 leads to the following corollary.

Corollary 2.17 *Let G be a graph with n vertices and n edges. Then*

$$QN(G) \geq \frac{m}{(2n-1)}.$$

A *maximal n-vertex 1-queue layout* is a 1-queue layout of a graph with n vertices and $2n-3$ edges. In the next theorem, we determine the number of maximal 1-queue layouts.

Theorem 2.18 *The number of maximal n-vertex 1-queue layouts is the Catalan number*

$$C_{n-2} = \binom{2n-4}{n-2}\frac{1}{n-1}.$$

Proof: Suppose $G = (V, E)$ is a 1-queue graph with n vertices and $(2n-3)$ edges and σ is a total order on V that yields a 1-queue layout of G. By Theorem 2.11, we know that $SCN(AM(G,\sigma)) = 2$. Let $\Pi = \{\Psi, R(\Psi)\}$ be a staircase cover of $AM(G,\sigma)$, where $\{\Psi\}$ is

the staircase cover of $U(AM(G,\sigma))$. Since, G has $2n-3$ edges, Ψ is a maximal staircase with $|\Psi| = 2n-3$. Therefore, the source of Ψ is $(1,2)$ and the destination is $(n-1,n)$. Counting the number of maximal n-vertex 1-queue layouts is therefore equivalent to counting the number of distinct maximal staircases with source $(1,2)$ and destination $(n-1,n)$. Let the elements in Ψ be

$$(i_1,j_1) = (1,2) \leq_{se} (i_2,j_2) \leq_{se} \cdots \leq_{se} (i_{2n-3},j_{2n-3}) = (n-1,n).$$

We transform this sequence of elements into a balanced parentheses expression. As observed earlier (see Section 2.1), for all ℓ, $1 \leq \ell \leq 2n-4$, either

1. $i_{\ell+1} + 1 = i_\ell$ and $j_{\ell+1} = j_\ell$, or
2. $i_{\ell+1} = i_\ell$ and $j_{\ell+1} + 1 = j_\ell$.

Scan the elements of Ψ in the ascending order of $\leq_{se}$. On moving from one position index to the next, if (1) holds then call the move a *horizontal move*. Write down a left parenthesis corresponding to a horizontal move. If (2) holds then the move is called a *vertical move*. Write down a right parenthesis corresponding to a vertical move. The sequence of parentheses so obtained has the following properties.

1. It contains $n-2$ left parentheses and $n-2$ right parentheses. This is because moving from $(1,2)$ to $(n-1,n)$ involves $n-2$ horizontal moves and $n-2$ vertical moves.
2. Any prefix of this sequence of parentheses contains at least as many left parentheses as right parentheses. This is because Ψ contains only position indices belonging to the upper triangular portion of $AM(G,\sigma)$. This implies that the number of vertical moves can be no more than the number of horizontal moves.

These two properties imply that the sequence of parentheses is a balanced sequence of parentheses with $n-2$ left parentheses and $n-2$ right parentheses. Thus each distinct maximal staircase with source $(1,2)$ and destination $(n-1,n)$ corresponds to a distinct balanced parentheses expression, with $n-2$ left parentheses and $n-2$ right parentheses. We

know that the number of balanced parentheses expressions with $n-2$ left parentheses and $n-2$ right parentheses is the Catalan number $C_{n-2} = \binom{2n-4}{n-2}/n-1$ (see Graham, Knuth, and Patashnik [28] for a proof of this). Therefore the number of maximal 1-queue layouts is the Catalan number C_{n-2}. □

We now demonstrate, through an example, how the connection between queue layouts and matrix covers can be used as a tool to derive upper and lower bounds on the queuenumber of specific families of graphs. Heath and Rosenberg [36] derive the exact value of the queuenumber of a complete bipartite graph. In the proof of Theorem 2.19 below, we present a new technique, based on minimally covering a matrix with staircases, for obtaining the same value.

Theorem 2.19 *The queuenumber of the complete bipartite graph $K_{m,n}$ is*

$$QN(K_{m,n}) \;=\; \min\left(\left\lceil\frac{m}{2}\right\rceil, \left\lceil\frac{n}{2}\right\rceil\right).$$

Proof: We establish the lower bound first. Let $K_{m,n} = (V_1, V_2, E)$ be a complete bipartite graph, where $|V_1| = m$ and $|V_2| = n$. Without loss of generality, assume that $m \leq n$. Let $k = \lfloor m + n/2 \rfloor$. Let σ be an arbitrary total order on the vertices in $V_1 \cup V_2$. Let M' be the $k \times k$ submatrix of $AM(G,\sigma)$ consisting of the first k rows and the last k columns. Let $A \subseteq V_1$ be the set of vertices belonging to V_1 that occur among he first k positions in σ. Let $B \subseteq V_2$ be the set of vertices belonging to V_2 that occur among the first k positions in σ. Thus, $|A| + |B| = k$. M' contains an $|A| \times |V_2 - B|$ submatrix all of whose entries are 1 and a $|V_1 - A| \times |B|$ submatrix all of whose entries are 1. The largest antistaircase in either of these two submatrices has size

$$\max\Big(\min\left(|A|, n - |B|\right), \min\left(|B|, m - |A|\right)\Big) =$$

$$\max\left(\min\left(|A|, n - \left\lfloor\frac{m+n}{2}\right\rfloor + |A|\right), \min\left(\left\lfloor\frac{m+n}{2}\right\rfloor - |A|, m - |A|\right)\right)$$

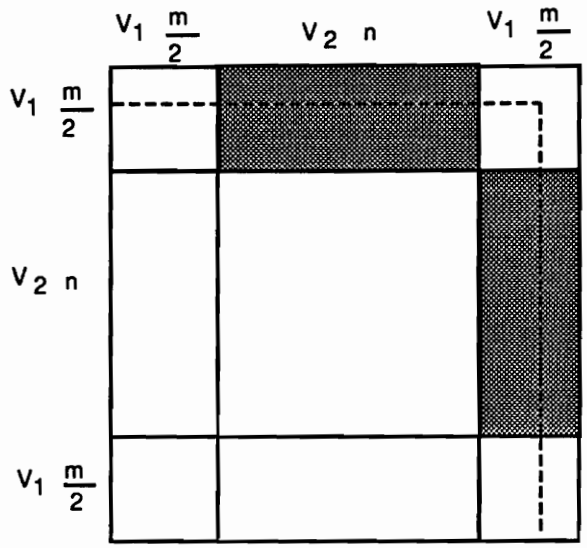


Figure 2.4: Queuenumber of $K_{m,n}$ is $\min(\lceil m/2 \rceil, \lceil n/2 \rceil)$.

This expression is minimized when $|A| = \lfloor m/2 \rfloor$, and the minimum value it takes is $\lceil m/2 \rceil$. The lower bound follows.

Consider $AM(K_{m,n}, \sigma)$ shown in Figure 2.4, where σ contains $\lceil m/2 \rceil$ elements of V_1 followed by n elements of V_2, followed by the remaining elements of V_1. As can be seen from the figure, $U(AM(K_{m,n}, \sigma))$ can be covered with $\lceil m/2 \rceil$ staircases. Thus, $QN(K_{m,n}) \leq \lceil m/2 \rceil$. □

2.4 Algorithmic Results

In this section, we generalize the notion of a staircase cover to that of an (h, w)-staircase cover, where each staircase is restricted to covering at most h nonzeros in any row and at most w nonzeros in any column (see Subsection 2.4.1). We present an algorithm called (h, w)-*Staircase Cover Algorithm* that finds a minimal (h, w)-staircase cover of M in $O(T(M) \log \log n)$ time (see Subsection 2.4.2), where $T(M)$ is the time required to visit all the nonzeros in M. We conclude the section with several applications of the (h, w)-Staircase

Cover Algorithm to problems related to sequences, graph theory, and computational geometry (see Subsection 2.4.3).

2.4.1 A Generalized Matrix Covering Problem

The problem solved by algorithm SCA (see Section 2.1, Theorem 2.8) may be stated formally as follows.

> **Staircase Cover Problem (SCCOVER)**
> Given a matrix $M = (m_{i,j})_{n \times n}$, find a minimal staircase cover Π of M, such that the staircases in Π are totally ordered by $\leq_\gamma$.

Recall that the proof of Theorem 2.8 sketches SCA. Its time complexity was left unanalyzed. In this section, we are interested in solving a general form of SCCOVER, called (h,w)-SCCOVER (the (h,w)-Staircase Cover Problem). Several problems including SCCOVER and the stripe cover problem (see Melhem [51]) are special cases of (h,w)-SCCOVER. Our algorithm to solve (h,w)-SCCOVER can also be used to provide alternate solutions to the maxdominance problems proposed by Atallah and Kosaraju [3]. Atallah and Kosaraju point out that a solution to the maxdominance problems leads to a solution to the minimum independent dominating set problem for permutation graphs (MIDS) and also to the largest empty rectangle problem (LER). Therefore our algorithm to solve (h,w)-SCCOVER can provide alternate and often improved solutions to MIDS and LER. This point is elaborated upon in Section 2.4.3.

Define an (h,w)-*staircase* as a staircase that contains at most w nonzero elements in each row and at most h nonzero elements in each column. An (h,w)-*staircase cover* of M is a set of (h,w)-staircases that covers M. The size of a smallest (h,w)-staircase cover of M is the (h,w)-*staircase number* of M, and is denoted by $SCN(M,h,w)$. The (h,w)-SCCOVER can be stated as follows.

> **(h,w)-Staircase Cover Problem ((h,w)-SCCOVER)** Given a matrix $M = (m_{i,j})_{n \times n}$, find a minimal (h,w)-staircase cover, Π, of M such that the staircases

in Π are totally ordered by $\leq_\gamma$.

Note that an (n,n)-staircase cover of M is nothing but a staircase cover of M. Therefore a solution to (h,w)-SCCOVER is also a solution to SCCOVER. A *stripe* is a $(1,1)$-staircase. Thus a stripe can have at most one element in each row and in each column. Melhem [51] uses a minimal cover of a matrix with stripes that are totally ordered by $\leq_\gamma$ as the basis of a scheme to perform matrix vector multiplications on a data driven network. An algorithm for (h,w)-SCCOVER can also solve the problem of finding a smallest stripe cover that consists of stripes that are totally ordered by $\leq_\gamma$.

Like staircases, (h,w)-staircases are also intimately related to queue layouts. Let $G = (V,E)$ be an undirected graph and let σ be a total order on the vertices in V. Define a *left edge* of a vertex $v \in V$ as an edge from v to a vertex that is earlier in σ and define a *right edge* of v as an edge from v to a vertex that is later in σ. Let

$$\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_k\}$$

be a smallest (h,w)-staircase cover of $U(AM(G,\sigma))$. A k-queue layout of G can be constructed from Π by assigning an edge (i,j) to queue q_ℓ if position index $(i,j) \in \Psi_\ell$. The k-queue layout so constructed has the property that each queue contains at most h left edges of each vertex and at most w right edges of each vertex. It can be easily checked that from a k-queue layout of G in which each queue contains at most h left edges of each vertex and at most w right edges of each vertex, a (h,w)-staircase cover of size k for $U(AM(G,\sigma))$ can be constructed. These observations are formalized in the following theorem.

Theorem 2.20 *Let $G = (V,E)$ be a graph. If σ is an order on V that yields a k-queue layout of G such that each queue contains at most h left edges of each vertex and at most w right edges of each vertex, then $SCN(AM(G,\sigma),h,w) \leq 2k$. If $SCN(AM(G,\sigma),h,w) = 2k$ for some order σ on V, then σ yields a k-queue layout of G in which each queue contains at most h left edges of each vertex and at most w right edges of each vertex.*

Notice that Theorem 2.20 is just a generalization of Theorem 2.11 and the proof is similar.

There is a motivation for looking at queue layouts in which there is a bound on the number of left edges and the number of right edges of each vertex. Buss, Rosenberg, and Knott [10] define the *type* of a vertex in a k-stack layout as a $p \times 2$ matrix of non-negative integers

$$\tau(v) = \begin{pmatrix} L_1 & R_1 \\ L_2 & R_2 \\ . & . \\ . & . \\ L_k & R_k \end{pmatrix}$$

such that each L_i (respectively, R_i) is the number of left edges (respectively, right edges) of v assigned to stack s_i. Buss, Knott, and Rosenberg observe that the number of vertex types in a stack layout relates to the amount of logic necessary to realize fault-tolerant arrays of processors using the DIOGENES design methodology [58]. In a similar manner, a vertex type can be defined for each vertex in a queue layout and this also has utility from the point of view of the DIOGENES design methodology. For a queue layout in which the number of left edges of each vertex assigned to each queue is at most h and the number of right edges of each vertex assigned to each queue is at most w, there is a bound on the number of vertex types in terms of h and w. The number of vertex types relates to the amount of logic to be implemented in hardware in the DIOGENES approach to fault tolerant processing. Thus keeping the number of vertex types small is important and given an upper bound on the number of vertex types that are desirable for a queue layout, we may be able to achieve that upper bound by choosing appropriate values for h and w and computing the corresponding queue layout by solving the (h, w)-SCCOVER.

2.4.2 An Algorithm for (h, w)-SCCOVER

Recall that the staircase number of M, $SCN(M)$, is the size of the largest antistaircase in M. Our first goal in this subsection is to extend the notion of an antistaircase, to that of an (h, w)-*antistaircase* such that $SCN(M, h, w)$ is the size of the largest (h, w)-antistaircase

in M. The first step in defining an (h,w)-antistaircase is to define a partial order $\leq_{ne,h,w}$ such that an (h,w)-antistaircase is a chain in the poset $(\Gamma, \leq_{ne,h,w})$. This is analogous to the fact that an antistaircase is a chain in $(\Gamma, \leq_{ne})$.

For $1 \leq i \leq n$ and $1 \leq j_1 \leq j_2 \leq n$, define

$$R_i(j_1,j_2) = \Gamma \cap \{(i,j) \mid j_1 \leq j \leq j_2\}.$$

Thus, $R_i(j_1,j_2)$ is the set of position indices that cover nonzero entries in row i between column j_1 and column j_2 (inclusive of j_1 and j_2). Similarly, for $1 \leq j \leq n$ and $1 \leq i_1 \leq i_2 \leq n$, define

$$C_j(i_1,i_2) = \Gamma \cap \{(i,j) \mid i_1 \leq i \leq i_2\}.$$

Thus $C_j(i_1,i_2)$ is the set of position indices that cover nonzero entries in column j between row i_1 and row i_2 (inclusive of i_1 and i_2). Define a binary relation $\leq_{ne,h,w}$ on the set of position indices such that for any two position indices $(i_1,j_1),(i_2,j_2) \in \Gamma$ for which $i_1 \geq i_2$ and $j_1 \leq j_2$,

$$(i_1,j_1) \leq_{ne,h,w} (i_2,j_2) \text{ if } \begin{cases} i_1 = i_2 = i & \text{and } |R_i(j_1,j_2)| > w \text{ or} \\ j_1 = j_2 = j & \text{and } |C_j(i_2,i_1)| > h \text{ or} \\ (i_1,j_1) \leq_{ne} (i_2,j_2). \end{cases}$$

The first task is to show that the binary relation $\leq_{ne,h,w}$ is a partial order.

Lemma 2.21 *$(\Gamma, \leq_{ne,h,w})$ is a poset.*

Proof: First it is shown that $\leq_{ne,h,w}$ is antisymmetric. Then it is shown that $\leq_{ne,h,w}$ is transitive.

Assume that $(i_1,j_1) \leq_{ne,h,w} (i_2,j_2)$ and $(i_1,j_1) \neq (i_2,j_2)$. If $(i_1,j_1) \leq_{ne} (i_2,j_2)$, then $(i_2,j_2) \not\leq_{ne} (i_1,j_1)$ and $i_1 > i_2$ and $j_1 < j_2$. Therefore

$$(i_2,j_2) \not\leq_{ne,h,w} (i_1,j_1).$$

On the other hand, if $(i_1,j_1) \leq_{ne,h,w} (i_2,j_2)$ but $(i_1,j_1) \not\leq_{ne,h,w} (i_2,j_2)$, then either $i_1 = i_2 = i$ and $|R_i(j_1,j_2)| > w$ or $j_1 = j_2 = j$ and $|C_j(i_2,i_1)| > h$. In this case also, $(i_2,j_2) \not\leq_{ne,h,w} (i_1,j_1)$. Therefore $\leq_{ne,h,w}$ is anti-symmetric.

To show that $\leq_{ne,h,w}$ is transitive assume that

$$(i_1,j_1) \leq_{ne,h,w} (i_2,j_2)$$
$$(i_2,j_2) \leq_{ne,h,w} (i_3,j_3).$$

If either $(i_1,j_1) = (i_2,j_2)$ or $(i_2,j_2) = (i_3,j_3)$, then $(i_1,j_1) \leq_{ne,h,w} (i_2,j_2)$. Therefore, we assume that $(i_1,j_1) <_{ne,h,w} (i_2,j_2)$ and $(i_2,j_2) <_{ne,h,w} (i_3,j_3)$. This immediately implies that $i_3 \leq i_2 \leq i_1$ and $j_1 \leq j_2 \leq j_3$. Therefore, if either $(i_1,j_1) <_{ne} (i_2,j_2)$ or $(i_2,j_2) <_{ne} (i_3,j_3)$, then $(i_1,j_1) <_{ne} (i_3,j_3)$ and therefore $(i_1,j_1) <_{ne,h,w} (i_3,j_3)$. Hence, assume that $(i_1,j_1) \not<_{ne} (i_2,j_2)$ and $(i_2,j_2) \not<_{ne} (i_3,j_3)$. Then there are two possible relationships between (i_1,j_1) and (i_2,j_2) and they are treated below separately as case (1) and case (2).

Case 1: $i_1 = i_2 = i$ and $|R_i(j_1,j_2)| > w$.

Notice that $j_2 > j_1 + w$ since $|R_i(j_1,j_2)| > w$. There are two possible relationships between (i_2,j_2) and (i_3,j_3). If $i_2 = i_3 = i$ and $|R_i(j_2,j_3)| > w$, then $i_1 = i_3 = i$ and $|R_i(j_1,j_3)| > w$. Therefore $(i_1,j_1) <_{ne,h,w} (i_3,j_3)$. The other possibility is that $j_1 = j_2 = j_3$ and $|C_j(i_1,i_3)| > h$. In this case, $i_1 = i_2 > i_3 + h$ and $j_3 = j_2 > j_1 + w$. Therefore $i_1 > i_3$ and $j_1 < j_3$ implying that $(i_1,j_1) <_{ne} (i_3,j_3)$ and hence $(i_1,j_1) <_{ne,h,w} (i_3,j_3)$.

Case 2: $j_1 = j_2 = j$ and $|C_j(i_2,i_1)| > h$.

This case is similar to case 1.

This shows that $<_{ne,h,w}$ is transitive. The theorem follows. □

An (h,w)-*antistaircase* is a chain in the poset $(\Gamma, \leq_{ne,h,w})$. In the next lemma, we show that an (h,w)-antistaircase is indeed an obstacle to minimizing the (h,w)-staircase number of M. More precisely, we show that $SCN(M,h,w)$ is bounded below by the size of the largest (h,w)-antistaircase in M. Recall the situation when we were only concerned about staircases and antistaircases. We showed that no pair of position indices in an antistaircase

can be elements in the same staircase. This observation immediately implied that the length of a largest antistaircase in M is a lower bound on $SCN(M)$. A similar result, showing that no two position indices in an (h,w)-antistaircase in M can be elements in the same (h,w)-staircase would be useful. Unfortunately, as illustrated in the following example, this is not true. Consider the position indices (i,j_1) and (i,j_1+w) and assume that $\{(i,j) \mid j_1 \leq j \leq j_2\} \subseteq \Gamma$. In other words, there is a block of nonzero entries in row i, from column j_1 to column j_2. Therefore $(i,j_1) <_{ne,h,w} (i,j_1+w)$. But, the position indices (i,j_1) and (i,j_1+w) could be elements in the same staircase, provided some of the position indices in the set $\{(i,j) \mid j_1 < j < j_2\}$ belong to other staircases. This example does *not* deal a fatal blow to our plans. We are still able to show, in a weaker form, that an (h,w)-antistaircase is indeed an obstacle to minimizing the (h,w)-staircase number of M.

Lemma 2.22 *If M contains an (h,w)-antistaircase of size k, then*

$$SCN(M,h,w) \geq k.$$

Proof: Let Q denote an (h,w)-antistaircase in M of size k, written as an increasing sequence with respect to $\leq_{ne,h,w}$. The sequence Q can be broken into contiguous subsequences

$$Q_1, Q_2, \ldots, Q_r$$

such that

- The first element of Q_1 is the first element of Q.
- The last element of Q_r is the last element of Q.
- The first element of Q_ℓ, $2 \leq \ell \leq r$, is the last element of $Q_{\ell-1}$.
- Each subsequence Q_ℓ, $1 \leq \ell \leq r$, is of one of three types:

  Type 1:

  $$Q_\ell = (i_1,j_1) <_{ne} (i_2,j_2) <_{ne} \cdots <_{ne} (i_m,j_m).$$

Type 2:

$$Q_\ell = (i, j_1) <_{ne,h,w} (i, j_2) <_{ne,h,w} \cdots <_{ne,h,w} (i, j_m)$$

such that

$$|R_i(j_p, j_{p+1})| > w$$

for all p, $1 \leq p < m$.

Type 3:

$$Q_\ell = (i_1, j) <_{ne,h,w} (i_2, j) <_{ne,h,w} \cdots <_{ne,h,w} (i_m, j)$$

such that

$$|C_j(i_{p+1}, i_p)| > h$$

for all p, $1 \leq p < m$.

- If Q_ℓ is of type i, then $Q_{\ell+1}$ is of type j, where $i \neq j$.

Thus, $|Q| = 1 - r + \sum_{\ell=1}^{r} |Q_\ell|$. Let Π be an arbitrary (h, w)-staircase cover of M. If any $\Psi \in \Pi$ contains an element from two distinct subsequences, then the element is the last element in subsequence Q_ℓ and is the first element in subsequence $Q_{\ell+1}$ for some ℓ, $1 \leq \ell \leq r - 1$.

Let λ_ℓ be the number of (h, w)-staircases in Π required to cover subsequence Q_ℓ. From the above observations, a lower bound on the cardinality of Π is

$$|\Pi| \geq 1 - r + \sum_{\ell=1}^{r} \lambda_\ell.$$

We now show that $\lambda_\ell \geq |Q_\ell|$ for all ℓ, $1 \leq \ell \leq r$. If Q_ℓ is of type 1, then it is clear that $\lambda_\ell \geq |Q_\ell|$.

So we concentrate on subsequences Q_ℓ of types 2 and 3. Consider the following subsequence Q_ℓ of type 2:

$$(i, j_1) <_{ne,h,w} (i, j_2) <_{ne,h,w} \cdots <_{ne,h,w} (i, j_m).$$

$S_0 := \Gamma$;
$\ell := 1$;
while $(S_{\ell-1} \neq \phi)$ **do**
    Let Ψ_ℓ be the set of maxima in $(S_{\ell-1}, \leq_{ne,h,w})$;
    $S_\ell := S_{\ell-1} - \Psi_\ell$;
    $\ell := \ell + 1$;
endwhile
return $\Psi_1, \Psi_2, \ldots, \Psi_{\ell-1}$

Figure 2.5: A sketch of the (h,w)-**Staircase Cover Algorithm**

We know that $|R_i(j_p, j_{p+1})| > w$ for all p, $1 \leq p < m = |Q_\ell|$. Therefore,

$$|R_i(j_1, j_m)| > w(|Q_\ell| - 1).$$

Thus, at least $|Q_\ell|$ (h,w)-staircases are required to cover all the elements in Q_ℓ. A similar argument can be used to show that for a subsequence Q_ℓ of type 3, at least $|Q_\ell|$ (h,w)-staircases are required to cover all the elements. Therefore

$$|\Pi| \geq 1 - r + \sum_{\ell=1}^{r} |Q_\ell| = |Q| = k.$$

The theorem follows. □

Now we have all the tools necessary to construct an algorithm to solve (h,w)-SCCOVER and prove its correctness. In principle, (h,w)-SCCOVER can be solved by the slightly modified version of SCA shown in Figure 2.5. Based on the sketch provided in Figure 2.5, we construct an algorithm called (h,w)-Staircase Cover Algorithm ((h,w)-SCA) to solve (h,w)-SCCOVER in $O(T(M)\log\log n)$ time.

The algorithm (h,w)-SCA processes M row by row starting at row 1 and proceeding to row n. Within each row, (h,w)-SCA processes the entries one by one starting at the entry in column n and proceeding to the entry in column 1. Let $M(i,j)$ be the submatrix of M that is to the northeast of (i,j) (inclusive of row i and column j). When a nonzero entry $m_{i,j}$ is encountered, (h,w)-SCA inserts (i,j) into Ψ_ℓ if the size of a largest (h,w)-antistaircase in $M(i,j)$ is of size ℓ. Clearly, this criterion for inserting (i,j) into Ψ_ℓ will result in (h,w)-SCA

producing an (h, w)-staircase cover

$$\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_k\}$$

only if the size of the largest (h, w)-antistaircase in M is k. By Lemma 2.22, we know that Π is a minimal (h, w)-staircase cover of M.

Finding ℓ, the size of a largest (h, w)-antistaircase in M, is rendered easy by the fact that (h, w)-SCA has already processed the submatrix of M that is to the northeast of (i, j). In order to find ℓ, (h, w)-SCA does work that can be thought of as consisting of two separate phases. In phase 1, (h, w)-SCA determines ℓ_1, the size of the largest chain in $(\Gamma_1, \leq_{ne,h,w})$ where

$$\Gamma_1 = \Big(\Gamma \cap \{(p, q) \mid p \leq i \text{ and } q > j\}\Big) \cup \{(i, j)\}.$$

Note that Γ_1 consists of position indices that are to the north or strictly to the northeast of (i, j). Thus ℓ_1 is the size of a largest (h, w)-antistaircase in in $M\prime(i, j)$ which is the same as $M(i, j)$, except that the only nonzero entry in row i is $m_{i,j}$. In phase 2, (h, w)-SCA determines ℓ_2, the size of a largest (h, w)-antistaircase in $M(i, j)$ that contains a position index in row i different from (i, j). Note that ℓ_2 may be 0 if there are no nonzero position indices in row i other than (i, j). Clearly, the size of a largest (h, w)-antistaircase in $M(i, j)$ is given by

$$\ell = \max\{\ell_1, \ell_2\}.$$

What follows is a precise description of phase 1 and phase 2.

Phase 1: In order to determine ℓ_1, (h, w)-SCA makes use of two arrays

$$R[0..2n]$$

and

$$Y[1..2n - 1].$$

$R[t]$ contains the largest column index of any position index in Ψ_t, excluding the position indices in the current row (row i). R satisfies the following invariant:

- $R[t] \geq R[t+1]$ for all t, $0 \leq t \leq 2n-1$.
- There exists a t, $1 \leq t \leq 2n$ such that $R[t] = 0$ and $R[t-1] > R[t]$.

R is initialized as follows:

$$\begin{aligned} R[0] &= n+1 \\ R[t] &= 0 \text{ for all } t,\ 1 \leq t \leq 2n \end{aligned}$$

Clearly, R satisfies the invariant to begin with. For the sake of convenience, we assume that Ψ_0 and Ψ_{2n} are fictitious staircases that contain the position indices $(0, n+1)$ and $(n+1, 0)$ respectively. (h,w)-SCA first finds a t such that

$$R[t] \geq j > R[t+1].$$

The fact that $j > R[t+1]$ implies that the size of a largest chain in $(\Gamma_1 - \{(i,j)\}, \leq_{ne,h,w})$ is at most t. The fact that $R[t] \geq j$ implies that the size of a largest chain in $(\Gamma_1 - \{(i,j)\}, \leq_{ne,h,w})$ is at least t. Therefore, the size of a largest chain in $(\Gamma_1 - \{(i,j)\}, \leq_{ne,h,w})$ is t. Whether the size of a largest chain in $(\Gamma_1, \leq_{ne,h,w})$ is t or $t+1$ depends upon whether $R[t] > j$ or $R[t] = j$. These two cases are dealt with separately, below.

1. $R[t] > j$. In this case, there exists $(p,q) \in \Gamma_1$ such that

   $$(i,j) \leq_{ne,h,w} (p,q)$$

   and therefore the largest chain in $(\Gamma_1, \leq_{ne,h,w})$ is of size $t+1$. This implies that $\ell_1 = t+1$.

2. $R[t] = j$. In this case there exists a $(p,j) \in \Gamma_1 - \{(i,j)\}$ such that $(p,q) \in \Psi_t$. But, to determine whether $\ell_1 = t$ or $\ell_1 = t+1$ (h,w)-SCA makes use of the array $Y[1..2n-1]$. The elements of Y are all initialized to 0. $Y[t]$ contains the number of position indices most recently inserted into Ψ_t, excluding the position indices in the current row (row i), that belong to the same column. To determine whether $\ell_1 = t$ or $\ell_1 = t+1$, (h,w)-SCA simply checks $Y[t]$. If $Y[t] < h$, then $\ell_1 = t$, otherwise $\ell_1 = t+1$. This is the end of phase 1.

Phase 2: In this phase, (h,w)-SCA determines ℓ_2, the size of the largest (h,w)-antistaircase in $M(i,j)$ that contains at least on position index belonging to row i, other than (i,j). To make this determination, (h,w)-SCA uses two variables c and s; c is the index of the staircase that the most recently processed position index in row i was inserted into and s contains the number of most recently processed position indices in row i that have been inserted into the same (h,w)-staircase. Clearly, $\ell_2 = c$ or $\ell_2 = c+1$. More precisely, if $s < w$, then $\ell_2 = c$; otherwise if $s = w$, then $\ell_2 = c+1$. This ends phase 2.

Figure 2.6 shows (h,w)-SCA. The algorithm in the Figure follows closely, the above description. But it leaves out the implementation details of three steps marked (1), (2), and (3). We elaborate on these steps now.

Step (1): The first step that needs elaboration is how (h,w)-SCA finds an index t such that $R[t] \geq j > R[t+1]$. Instead of using a simple binary search, which would cause this step to take $O(\log n)$ time, we make use of a data structure introduced by Johnson [39], that allows (h,w)-SCA to perform this step in $O(\log\log n)$ time. This data structure, that we shall refer to as *Johnson's priority queue* from now onwards, allows for initialization, insertion, and deletion in $O(\log\log n)$ time each if the elements come from the restricted domain $[1..n]$. Karlsson and Overmars [41] use Johnson's priority queue in a manner similar to ours in scanline algorithms for computational geometry on a grid.

Step (2): The second step that needs elaboration is how the position index (i,j) is inserted into Ψ_ℓ, once ℓ has been determined. We want each staircase Ψ_ℓ to be maintained in the increasing order of $\leq_{se}$. The reason for this requirement will become clear when we discuss the applications of (h,w)-SCA. Unfortunately, the position indices inserted into Ψ_ℓ are not processed in the increasing order of $\leq_{se}$ and hence maintaining Ψ_ℓ in the increasing order of $\leq_{se}$ involves some work. Each staircase Ψ_ℓ is maintained as two lists L_ℓ and L'_ℓ. L_ℓ contains all elements of Ψ_ℓ, excluding those in row i, in the increasing order of $\leq_{se}$. L'_ℓ contains all elments of Ψ_ℓ that belong to row i, maintained in the decreasing order of $\leq_{se}$. Since the position indices in each row are processed in the decreasing order of $\leq_{se}$, (i,j) is inserted into Ψ_ℓ by appending it to the back of L'_ℓ. When all the position indices in row i have been

```
(* Initialization *)
R[0] := n + 1; R[1..2n − 1] := 0; Y[1..2n − 1] := 0;
for i := 1 to n do
(* Initialization *)
    c := 0;      s := 0;
    for j := n downto 1 do
        if (m_{i,j} ≠ 0) then
            Find t such that R[t] ≥ j > R[t + 1];                    (1)
(* Finding ℓ_1 *)
            if (R[t] > j) then
                ℓ_1 := t + 1
            else if(R[t] = j) and (Y[ℓ] = h) then
                ℓ_1 := t + 1
            else (R[t] = j) and (Y[ℓ] < h) then
                ℓ_1 := t;
(* Finding ℓ_2 *)
            if (c = 0) then
                ℓ_2 := 1
            else if (c > 0) and (s < w) then
                ℓ_2 := c
            else if (c > 0) and (s = w) then
                ℓ_2 := c + 1;
(* Finding ℓ and inserting (i, j) into Ψ_ℓ *)
            ℓ := max(ℓ_1, ℓ_2); Ψ_ℓ := Ψ_ℓ ∪ {(i, j)};              (2)
(* Updating Y *)
            if (R[t] > j) then
                Y[ℓ] := 1;
            else if (R[t] = j) and (ℓ = t) then
                Y[ℓ] := Y[ℓ] + 1;
(* Updating c and s *)
            if (ℓ = c) then
                s := s + 1
            else if (ℓ > c) then
                s := 1; c := ℓ;
            endif (* m_{i,j} ≠ 0 *)
        endfor (* j loop *)
    Update array R to accommodate elements in row i;                 (3)
    endfor (* i loop *)
```

Figure 2.6: The (h, w)-**Staircase Cover Algorithm** ((h, w)-**SCA**)

processed, L'_ℓ is reversed and is appended at the back of L_ℓ.

Step (3): The third step that needs elaboration is how the array R is updated to reflect the insertion of the position indices in row i into (h,w)-staircases. There are several simple and efficient ways in which this can be done and we describe one of these, for the sake of completeness and specificity. Maintain a temporary list *Rtemp* such that if a position index (i,j) is inserted into a staircase Ψ_ℓ, then the tuple $(\ell,(i,j))$ is appended to the back of *Rtemp*. After the entire row i in M has been processed, update R by scanning *Rtemp* from front to back and for each element $(\ell,(i,j))$ in *Rtemp* assign j to $R[\ell]$ if $j > R[\ell]$. *Rtemp* is initialized to an empty list after all the elements in it have been processed.

The time complexity of (h,w)-SCA is easy to establish. For each $m_{i,j} \neq 0$, (h,w)-SCA requires, in the worst case, $O(\log\log n)$ time to insert (i,j) into the appropriate (h,w)-staircase Ψ_ℓ. The number of nonzero entries in M is at most $O(n^2)$. It follows that, in the worst case, (h,w)-SCA requires $O(n^2 \log\log n)$ time to compute a minimal (h,w)-staircase of M. But, in all applications that we use (h,w)-SCA for, the matrix M is relatively sparse and therefore $O(n^2 \log\log n)$ is a pessimistic measure of the performance of (h,w)-SCA. A more accurate measure is expressed in the following theorem.

Theorem 2.23 *A minimal (h,w)-staircase cover of M is constructed by (h,w)-SCA in $O(T(M)\log\log n)$ time, where $T(M)$ is the number of time steps required to visit all the nonzeros in M.*

Three observations about (h,w)-SCA are in order.

1. (h,w)-SCA computes a minimal staircase cover of M when $h = w = n$. Therefore, in $O(T(M)\log\log n)$ time, a minimal staircase cover of M can be computed.

2. (h,w)-SCA computes a minimal stripe cover of M (see Melhem [51]) when $h = w = 1$. Therefore, a minimal stripe cover of M can be computed in $O(T(M)\log\log n)$ time.

3. Given a graph $G = (V,E)$ and a fixed total order σ on V, the queuenumber of a layout in which each vertex has at most h left edges and w right edges assigned to a queue, can be determined in $O(n\log\log n)$ time, using (h,w)-SCA.

2.4.3 Applications of (h, w)-SCA

This section is devoted to showing how (h, w)-SCA can be used to provide efficient solutions to problems in different areas. We begin by showing in the following theorem that (h, w)-SCA can be used to provide efficient algorithms to solve problems related to sequences.

Theorem 2.24 *Given a sequence σ of length n, consisting of numbers drawn from the restricted domain $[1..n]$, (h, w)-SCA can solve the following problems in $O(n \log\log n)$ time.*

1. *Find a smallest partition of σ into strictly increasing (decreasing) subsequences.*
2. *Find a smallest partition of σ into increasing (decreasing) subsequences.*
3. *Find a largest strictly increasing (decreasing) subsequence of σ.*
4. *Find a largest increasing (decreasing) subsequence of σ.*

Proof: We first focus on items (1) and (2) in the above list. Corresponding to σ, a matrix $M(\sigma) = m_{i,j}$ can be defined as follows:

$$m_{i,j} = \begin{cases} 1 & \text{if } \sigma(j) = i \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that if the input to (h, w)-SCA is $M(\sigma)$, then it computes the smallest partition of σ into increasing subsequences if $h = w = n$ and the smallest partition of σ into strictly increasing subsequences if $h = w = 1$.

Suppose that $M'(\sigma) = m'_{i,j}$ is obtained from σ by flipping each row in $M(\sigma)$. In other words, $m'_{i,j} = m_{i,n-j+1}$, for all i, j, $1 \leq i, j \leq n$. It is easy to see that if the input to (h, w)-SCA is $M'(\sigma)$, then it computes the smallest partition of σ into decreasing subsequences if $h = w = n$ and the smallest partition of σ into strictly decreasing subsequences if $h = w = 1$.

It is easy to see that a largest (h, w)-antistaircase can be extracted from a minimal (h, w)-staircase cover of M in time proportional to the number of nonzero entries in M. A largest

	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	0
3	0	0	0	0	1
4	0	0	1	0	0
5	0	0	0	0	0

Figure 2.7: The matrix $M(\sigma)$ corresponding to the sequence σ

(n,n)-antistaircase in $M(\sigma)$ corresponds to a largest strictly decreasing subsequence of σ and a largest $(1,1)$-antistaircase in $M(\sigma)$ corresponds to a largest decreasing subsequence of σ. Similarly, a largest (n,n)-antistaircase in $M'(\sigma)$ corresponds to a largest strictly increasing subsequence of σ and a largest $(1,1)$-antistaircase in $M(\sigma)$ corresponds to a largest increasing subsequence of σ. These observations imply that items (3) and (4) in the above list can be solved in $O(n \log\log n)$ time if items (1) and (2) can.

The following observations suffice to show that items (1) and (2) in the above list can be solved in $O(n \log\log n)$ time using (h,w)-SCA. There is no reason to construct $M(\sigma)$ and $M'(\sigma)$ explicitly. Simply by scanning σ once, it is possible to construct a list of position indices that cover nonzero entries in $M(\sigma)$, such that the list is in the order that (h,w)-SCA needs it in. A similar list corresponding to $M'(\sigma)$ can be constructed. These lists are of size n. We appeal to Theorem 2.23, and conclude that (h,w)-SCA can solve the four problems in the above list in $O(n \log\log n)$ time each. □

The proof of Theorem 2.24 is illustrated in the following example.

Example 2.25. Let $n = 5$ and let $\sigma = 2,1,4,1,3$. $M(\sigma)$ is shown in Figure 2.7. A minimal $(1,1)$-staircase cover of $M(\sigma)$ is

$$\begin{aligned} \Psi_1 &= \{(1,4),(3,5)\} \\ \Psi_2 &= \{(1,2),(4,3)\} \\ \Psi_3 &= \{(2,1)\}. \end{aligned}$$

This corresponds to the following minimal partition of σ into strictly increasing subsequences 1,3; 1,4; and 2. A minimal $(5,5)$-staircase cover of $M(\sigma)$ is

$$\begin{aligned} \Psi_1 &= \{(1,2),(1,4),(3,5)\} \\ \Psi_2 &= \{(2,1),(4,3)\}. \end{aligned}$$

This corresponds to the following minimal partition of σ into increasing subsequences 1,1,3; and 2,4. □

The algorithm (h,w)-SCA can be used to provide alternate solutions to the maxdominance problems suggested and solved by Atallah and Kosaraju [3]. Atallah and Kosaraju show how solutions to the maxdominance problems lead to solutions to MIDS (Minimum Independent Dominating Set problem for permutation graphs) and LER (Largest Empty Rectangle problem). Therefore, indirectly we can think of (h,w)-SCA contributing to the solutions of MIDS and LER also.

For a point p in the Cartesian plane, we use $p.x$ and $p.y$ to denote the x-coordinate and the y-coordinate of p respectively. Given a pair of points p and q in the plane, p is *dominates* q if $p.x > q.x$ and $p.y > q.y$. The domination relation is a partial order on points in the plane. Let P be a set of points in a plane. MAX(P) is the subset of points in P not dominated by any other point in P; that is, MAX(P) is the set of maxima in S with respect to the domination partial order. For an arbitrary point p, $\mathrm{DOM}_P(p)$ is the subset of points in P that are dominated by p. The first maxdominance problem proposed by Atallah and Kosaraju is MD1:

MD1

Compute MAX($\mathrm{DOM}_P(p)$) for all $p \in P$.

The second maxdominance problem proposed by Atallah and Kosaraju is MD2:

MD2

Each point $p \in P$ has a non-negative integer weight, $w(p)$, associated with it. For each $p \in P$, compute the functions LABEL and PRED defined inductively as follows: If $\mathrm{DOM}_P(p) = \phi$, then

$$\mathrm{LABEL}(p) = w(p)$$

and

$$\mathrm{PRED}(p) = \textbf{null}.$$

Otherwise, let q be a point in $\mathrm{MAX}(\mathrm{DOM}_P(p))$ with the smallest LABEL; then,

$$\mathrm{LABEL}(p) = w(p) + \mathrm{LABEL}(q)$$

and

$$\mathrm{PRED}(p) = q.$$

Let $|P| = n$, and let $t = \sum_{p \in P} |\mathrm{MAX}(\mathrm{DOM}_P(p))|$. Atallah and Kosaraju provide an $O(t + n \log n)$ time algorithm for MD1 and an $O(n \log n)$ time algorithm for MD2. We show how MD1 and MD2 can be solved, with the same time complexities, using (h, w)-SCA for $h = w = n$. Since, (h, w)-SCA for $h = w = n$ is nothing but SCA, for the rest of this section we use SCA rather than (h, w)-SCA.

From the set of points P, a matrix $M(P)$ can be constructed as follows. Obtain a sequence σ, by sorting the points in P, in the increasing order of their x-coordinates. Obtain another sequence δ, by sorting the points in P, in the decreasing order of their y-coordinates. Thus, each point $p \in P$, is identified uniquely, by its position in the two sequences. More precisely, with each point $p \in P$, we associate the pair $(\delta^{-1}(p), \sigma^{-1}(p)) \in J_n^2$. The matrix $M(P) = (m_{i,j})_{n \times n}$ is defined as

$$m_{i,j} = \begin{cases} 1 & \text{if } (\delta^{-1}(p), \sigma^{-1}(p)) = (i, j) \\ 0 & \text{otherwise} \end{cases}$$

Notice that p dominates q if and only if

$$(\delta^{-1}(q), \sigma^{-1}(q)) \leq_{ne} (\delta^{-1}(p), \sigma^{-1}(p)).$$

In the following discussion we say (i,j) dominates (i',j') if $(i',j') \leq (i,j)$. As before, there is no reason to explicitly construct $M(P)$ in order to obtain a minimal staircase cover of $M(P)$. All we need is a list of position indices covering nonzero entries of $M(P)$ in an order that SCA expects them in. Such a list can be constructed in $O(n \log n)$ time by sorting P twice and this is the step that dominates the time complexity of our algorithms that solve **MD1** and **MD2**. We now show how SCA can be used to solve MD1 in $O(t + n \log\log n)$ time and MD2 in $O(n \log n)$ time.

For an arbitrary $p \in P$, we show how to compute $\text{MAX}(\text{DOM}_P(p))$ from

$$\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_k\},$$

a minimal staircase cover of M produced by SCA. Let $(\delta^{-1}(p), \sigma^{-1}(p)) = (i,j)$ and let $(i,j) \in \Psi_\ell$, for some ℓ, $1 \leq \ell \leq k$. For each t, $\ell + 1 \leq t \leq k$, define

$$S_t = \{(i',j') \in \Psi_m \mid (i',j') \leq_{ne} (i,j)\}$$

and let

$$S = \cup_{t=\ell+1}^{k} S_t.$$

Clearly, $q \in \text{DOM}_P(p)$ if and only if $(\delta^{-1}(q), \sigma^{-1}(q)) \in S$. Thus, by computing S, we have essentially computed $\text{DOM}_P(p)$. Recall that each staircase produced by SCA is stored as a sequence in the increasing order of $\leq_{se}$. Therefore, S_t, for each t, $\ell + 1 \leq t \leq k$, is a contiguous subsequence of Ψ_t, whose end points can be found by doing two binary searches on Ψ_t, in $O(\log |\Psi_t|)$ time each. Thus, for each $p \in P$, $\text{DOM}_P(p)$ can be found in

$$O\left(\sum_{t=\ell+1}^{m} |\log \Psi_t|\right) = O(\log n)$$

time. Therefore, $\text{DOM}_P(p)$ for all $p \in P$ can be found in $O(n \log n)$ time.

We know that

$$\text{MAX}(\text{DOM}_P(p)) \subseteq \text{DOM}_P(p).$$

Assume that

$$(i', j') = (\delta^{-1}(q), \sigma^{-1}(q)) \in S_r$$

for some r, $\ell + 1 \leq r \leq k$. Then, $q \in \text{MAX}(\text{DOM}_P(p))$ if there is no element in $\cup_{t=\ell+1}^{r-1} S_t$ that dominates (i', j'). Let I_r denote the subset of S_r that consists of position indices that are not dominated by any position index in $\cup_{t=\ell+1}^{r-1} S_t$. Therefore,

$$\bigcup_{t=\ell+1}^{k} I_t = \text{MAX}(\text{DOM}_P(p)).$$

The key to solving **MD1** efficiently lies in finding each set I_t efficiently. Let

$$m = \min\{t \mid \ell + 1 \leq t \leq k \text{ and } S_m \neq \emptyset\}.$$

Clearly,

$$I_{\ell+1}, I_{\ell+2}, \ldots, I_{m-1} = \emptyset$$

and $I_m = S_m$. Thus having computed S_t, for all t, $\ell+1 \leq t \leq k$, it takes no additional work to determine $I_{\ell+1}, I_{\ell+2}, \ldots, I_m$. The question that we now address is: how do we compute the sets

$$I_{m+1}, I_{m+2}, \ldots, I_k.$$

The answer to this question is suggested by Figure 2.8. The figure shows three elements in I_m (shown by X). The elements in I_{m+1} are shown as circles. To find elements in I_{m+1}, we simply need to find the smallest element $(i_m, j_m) \in I_m$ and the largest element $(i'_m, j'_m) \in I_m$ with respect to $\leq_{se}$. The elements in I_{m+1} have row coordinates less than or equal to i_m and column coordinates greater than or equal to j'_m. Thus

$$I_{m+1} = \{(i', j') \in S_{m+1} \mid i' \leq i_m\} \cup \{(i', j') \in S_{m+1} \mid j' \geq j'_m\}.$$

Notice that I_{m+1} can be determined in $O(\log |S_{m+1}|)$ time by searching in S_m for (i_m, j_m) using i_m as a key and for (i'_m, j'_m) using j'_m as a key. The searches can be performed in

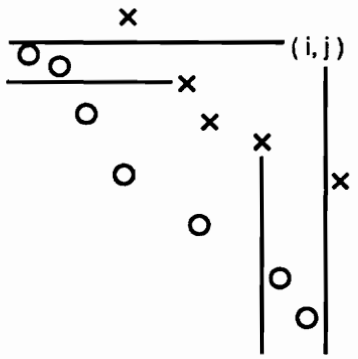


Figure 2.8: Solving **MD1** using a (h, w)-SCA.

Let $m := \min\{t \mid \ell + 1 \leq t \leq k \text{ and } S_m \neq \emptyset\}$;
$D_m := S_m$;
for $t := m + 1$ **to** k **do**
    Let (i_t, j_t) be the smallest element in D_{t-1} with respect to $\leq_{se}$;
    Let (i'_t, j'_t) be the largest element in D_{t-1} with respect to $\leq_{se}$;
    $I_t = \{(i', j') \in S_t \mid i' \leq i_t\} \cup \{(i', j') \in S_t \mid j' \geq j_t\}$;
    $D_t := D_{t-1} \cup I_t$;
return D_k;

Figure 2.9: Algorithm to solve **MD1**

$O(\log |S_m|)$ time because S_m is stored in increasing order with respect to $\leq_{se}$. The above observations suggest the algorithm shown in Figure 2.9 to compute the sets I_t, for all t, $m + 1 \leq t \leq k$. The time complexity of finding the sets I_t is

$$O\left(\sum_{t=m+1}^{k} \log |S_t|\right) = O(\log n).$$

Thus for each $p \in P$, $\text{MAX}(\text{DOM}_P(p))$ can be computed in $O(\log n)$ time and therefore $\text{MAX}(\text{DOM}_P(p))$ for all $p \in P$ can be computed in $O(n \log n)$ time. To output $\text{MAX}(\text{DOM}_P(p))$ for all $p \in P$, requires an additional $O(t)$ time where

$$t = \sum_{p \in P} |\text{MAX}(\text{DOM}_P(p))|.$$

The above discussion is summarized in the following theorem.

Theorem 2.26 *Given a set P of n points in a plane, SCA can be used to compute and output $MAX(DOM_P(p))$ for all $p \in P$ in $O(t+n\log n)$ time, where $t = \sum_{p\in P} MAX(DOM_P(p))$.*

MD2 can be solved in $O(n\log n)$ time by slightly modifying our algorithm for solving **MD1**. The position indices covering nonzero entries in $M(P)$ are processed in the order: elements in Ψ_1, followed by elements in $\Psi_2,\ldots$, and finally followed by elements in Ψ_k. Within each staircase, elements are processed in the increasing order of $\leq_{se}$. Clearly, if $(i,j) \in \Psi_1$ and $(\delta^{-1}(p),\sigma^{-1}(p)) = (i,j)$ then

$$\text{LABEL}(p) = w(p)$$

and

$$\text{PRED}(p) = \textbf{null}.$$

Assume that the position index $(i,j) = (\delta^{-1}(p),\sigma^{-1}(p)) \in \Psi_\ell$ is currently being processed. Notice that the position indices corresponding to all points that are in $\text{MAX}(\text{DOM}_P(p))$ have already been processed and therefore the functions LABEL and PRED have been computed for all $q \in \text{MAX}(\text{DOM}_P(p))$. If there were a data structure that contained all points in $\text{MAX}(\text{DOM}_P(p))$ sorted in increasing order of their LABELs, then in $O(1)$ time, we could determine LABEL(p) and PRED(p). But, such a data structure is easy to maintain. We simply maintain a subset of P in a heap sorted in the increasing order of LABEL such that when $p \in P$ is being processed, the heap contains $\text{MAX}(\text{DOM}_P(p))$. Assume that $r \in P$ was the point processed just before point p. After LABEL(r) and PRED(r) have been computed, the points in

$$\text{MAX}(\text{DOM}_P(r)) - \text{MAX}(\text{DOM}_P(p))$$

are deleted from the heap and the points in

$$\text{MAX}(\text{DOM}_P(p)) - \text{MAX}(\text{DOM}_P(r))$$

are added to the heap. By observing that no point in P is inserted into the heap more than once, we conclude that at most n insertions and deletions are performed. Since each

insertion and deletion requires $O(\log n)$ time, the total amount of work required to maintain the heap is $O(n \log n)$. Therefore we have the following theorem.

Theorem 2.27 *Given a set of n points, SCA can be used to compute $LABEL(p)$ and $PRED(p)$ for all $p \in P$ in $O(n \log n)$ time.*

2.5 Parallel Matrix Computations

In this section we develop a new scheme for performing parallel matrix computations on a data driven network, based on covering a matrix with staircases. In Subsection 2.5.1, we motivate and provide the background for parallel matrix computations. In Subsection 2.5.2, we present our new scheme for performing parallel matrix computations on a data driven network, based on covering a matrix with staircases. In Subsection 2.5.3, we analyze the performance of our scheme and show that it is an improvement over Melhem's scheme in terms of hardware cost as well as time.

2.5.1 Introduction

Matrix computations involving large, sparse, matrices in which the non-zeroes are distributed in a "regular" fashion arise frequently in numerical solutions to engineering problems. An example of a matrix in which the nonzeroes are distributed in a regular fashion is a *banded matrix*. A matrix $M = (m_{i,j})_{n \times n}$ is a *banded matrix* with *bandwidth* $p + q + 1$ if

$$m_{i,j} = 0 \text{ if } j - i > q \text{ or if } i - j > p.$$

All the nonzero elements in M are concentrated in a "band" of width $p + q + 1$ around the main diagonal. Many algorithms and high speed architectures that take advantage of the sparsity of a matrix and the regularity of the distribution of nonzeros in a matrix have been proposed. These algorithms and architectures lead to improvements in space efficiency and time efficiency of matrix computations and therefore to solutions for larger engineering problems.

One such scheme is the *systolic array* proposed and developed by Kung and Leiserson [43, 44,45]. A systolic array is a network of simple processors, called *processing elements (PEs)*, connected in a regular manner such that all processors performs the same computation at each clock beat. Systolic arrays can be used to perform various elementary matrix operations such as matrix vector multiplication, matrix matrix multiplication, and solving triangular systems of linear equations. Consider the problem of multiplying an $n \times n$ matrix M of bandwidth b and an n-vector x. Leiserson [45] proposes a systolic array to solve this problem that uses b PEs connected as a linear array and performs the multiplication in $2n + b$ clock beats. Note that the smaller the bandwidth of M, the fewer the number of PEs and the fewer the number of clock beats required to compute the product Mx.

As an extension of Leiserson's scheme, Melhem [50] suggests a data driven network that contains PEs having the capability of recognizing and skipping trivial operations. In a data driven network, each PE performs a computation as soon as it has all the inputs. Such a network is asynchronous and therefore less amenable to time complexity analysis than systolic arrays. Nevertheless, we show (Theorem 2.32) how to analyze such a network, answering an open question on Melhem's scheme. In some engineering applications such as finite element analysis, the matrix M may have large bandwidth but may still be quite sparse (see Melhem [49]). Leiserson's scheme cannot adjust to such a situation and requires b PEs and $2n + b$ clock beats independent of the sparsity of the band. As compared to this, the data driven network suggested by Melhem exhibits a large speedup when the band of the matrix is sparse. However, it should be kept in mind that this speedup is achieved through additional cost for the hardware that recognizes and skips trivial operations.

Motivated by similar considerations, Melhem [51] suggests another data driven network that is also based on performing non-trivial operations only but is primarily aimed at reducing the number of PEs in the network by taking advantage of the sparsity of the band of the matrix. This network, called MAT/VEC, does not improve on Leiserson's scheme in terms of time but can substantially improve on the number of PEs required, depending on the distribution of the nonzeros in the band. In the next section, we describe a scheme

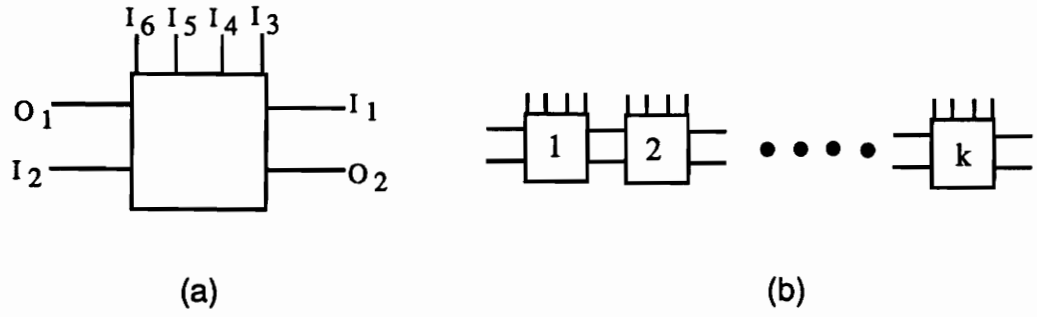


Figure 2.10: Parallel Matrix Computation Network.

called the Parallel Matrix Computation Network (PMCN), which is a generalization of and an improvement on MAT/VEC. PMCN has the flexibility to take advantage of various distributions of nonzeros in the band and is based on a staircase cover of M.

2.5.2 The Scheme

We focus on the problem of multiplying an $n \times n$ matrix $M = (m_{i,j})_{n \times n}$ and an n-vector $x = (x_i)_{n \times 1}$ to produce an n-vector $y = (y_i)_{n \times 1}$. Let $\{\Psi_1, \Psi_2, \ldots, \Psi_k\}$ be a staircase cover of M such that $\Psi_\ell <_\gamma \Psi_{\ell-1}$ for all ℓ, $2 \leq \ell \leq k$. Assume that each staircase Ψ_ℓ is stored as a list in increasing order with respect to $\leq_{se}$. PMCN uses k PEs, each PE containing six inputs, labeled I_1, I_2, I_3, I_4, I_5 and I_6 and two outputs, labeled O_1 and O_2. A PE in PMCN is shown in Figure 2.10(a). The k PEs are labeled 1 through k and are connected as a linear array shown in Figure 2.10(b). The components of the vector x are fed from the right of PMCN through input I_1 of the PE labeled 1, in the order $x_1, x_2, \ldots, x_n$. The components of the vector y, initialized to 0, are fed from the left of PMCN, through input I_2 of the PE labeled k, in the order $y_1, y_2, \ldots, y_n$. The product Mx is accumulated in y. Thus, when y_i exits PE k, it contains the value $\sum_{j=1}^{n} m_{i,j} x_j$. Now assume that

$$(i_1, j_1), (i_2, j_2), \ldots (i_m, j_m)$$

are the elements of Ψ_ℓ, for some ℓ, $1 \leq \ell \leq k$, written down in increasing order of $\leq_{se}$. Initially, m_{i_1,j_1}, i_1, and j_1 are fed into the inputs I_3, I_4 and I_5 respectively of PE ℓ. For

```
Step1:  R_a := [I_3]; R_j := [I_4]; R_i := [I_5];
Step2:  Fork into 2.1 and 2.2;
        2.1 (* Wait until appropriate x-component is available on I_1 *)
            R_x := [I_1]; CX := CX + 1;
            if (CX < R_j) then {O_1 := R_x; goto 2.1}
            else Join step 2.2
        2.2 (* Wait until appropriate y-component is available on I_2 *)
            R_y := [I_2]; CY := CY + 1;
            if (CY < R_i) then {O_2 := R_y; goto 2.2}
            else Join step 2.1
Step3:  R_y := R_y + R_a · R_x
Step4:  if ([I_6] = h) then {O_1 := R_x}
        else if ([I_6] = v) then {O_2 := R_y}
        else {O_1 := R_x; O_2 := R_y}
```

Figure 2.11: Program for each PE

each $p \geq 2$, m_{i_p,j_p}, i_p, and j_p are fed into inputs I_3, I_4, and I_5 respectively of PE ℓ, after $m_{i_{p-1},j_{p-1}}$, i_{p-1}, and j_{p-1} are fed. Thus each PE not only obtains an entry in the matrix but also obtains the corresponding position index and this allows the PE to determine when it has the appropriate component of x and the appropriate component of y in order to perform the computation. Suppose the PE labeled ℓ obtains the element $m_{i,j}$ on input I_3. It then waits for x_j and y_i to appear on inputs I_1 and I_2 respectively and when they do appear, it updates y_i according to $y_i := y_i + m_{i,j} \cdot x_j$. The value fed into a PE via I_6 depends on the relationship between the current input on I_3 and the next input on I_3. If the next input on I_3 is in the same row as the current input, then I_6 has the value h (horizontal). If the next input on I_3 is in the same column as the current input, then I_6 has the value v (vertical). Otherwise I_6 has the value d (diagonal).

Each PE in the network executes the program shown in Figure 2.11 repeatedly until Mx has been computed. Each PE has five internal registers, R_a, R_i, R_j, R_x, and R_y and two counters CX and CY. After receiving $m_{i,j}$, i, and j into registers R_a, R_i, and R_j

respectively (step 1), each PE checks in parallel the contents of the inputs I_1 and I_2 (steps 2.1 and 2.2) for the appropriate components of x and y. After finding the appropriate components of x and y, the PE updates the component of y (step 3). In step 4, the PE decides whether to retain either the component of x or the component of y for future use. If the next element in the staircase is in the same row as the current element ($[I_6] = h$), then y_i is retained and x_j is passed on. If the next element in the staircase is in the same column as the current element ($[I_6] = v$), then x_j is retained and y_i is passed on. If neither is true ($[I_6] = d$), then both x_j and y_i are passed on.

From the point of view of analyzing PMCN, it is useful to force some synchronization on it. Synchronization may slow the network down because each PE may have to wait for a synchronization signal, even when all the required data is available at its inputs. Melhem suggests the following synchronization scheme that we make use of. Replace step 3 in the program that each PE executes by the following step:

Step 3: Wait for a synchronization signal SYNC;
$R_y := R_y + R_a \cdot R_x$;

The purpose of the synchronization signal SYNC is to force the execution of PMCN into two alternate phases: a *communications phase* and a *processing phase*. During the communications phase data flows through the network until each PE is either blocked waiting for data (in steps 2.1 or 2.2), or waiting for SYNC (in step 3). We assume that all the PEs are connected to a hypothetical controller that issues the signal SYNC after detecting the termination of the communications phase. At the instant SYNC is received, all PEs that are waiting in step 3 perform the computation $R_y = R_y + R_a \cdot R_x$, while the other PEs remain idle. This is the processing phase. A communications phase followed by a processing phase is called a *global cycle* of the network. We measure the time taken by PMCN to compute Mx by the number of global cycles required. We denote by CP_t and

	1	1	1	
2	2	2	1	
	3		1	
	3	2		1

Figure 2.12: A matrix with staircase number 3.

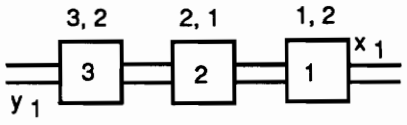


Figure 2.13: Initial status of PMCN.

PP_t, the communications phase and the processing phase, respectively, of global cycle t. Note that the synchronization described above is only a tool for analysis and we are not actually proposing to synchronize the execution of PMCN.

Example 2.28. Figures 2.12, 2.13, and 2.14 shows the working of PMCN. Figure 2.12 shows a 5×5 matrix whose staircase number is 3. The nonzero elements covered by Ψ_1 are labeled 1, those covered by Ψ_2 are labeled 2, and those covered by Ψ_3 are labeled 3. Note that $\Psi_3 \leq_\gamma \Psi_2 \leq_\gamma \Psi_1$. Figure 2.13 shows the status of PMCN initially and Figure 2.14 shows the 6 global cycles that PMCN goes through in order to perform the matrix vector multiplication. For each global cycle, the status of PMCN after the communications phase and the computations performed during the processing phase are shown. □

We now prove the correctness of PMCN by demonstrating that for each i, $1 \leq i \leq n$, y_i exits the network with the value $\sum_{j=1}^{n} m_{i,j} \cdot x_j$. For this proof as well as for other proofs in the remainder of the section, we make use of the following property of the staircase cover

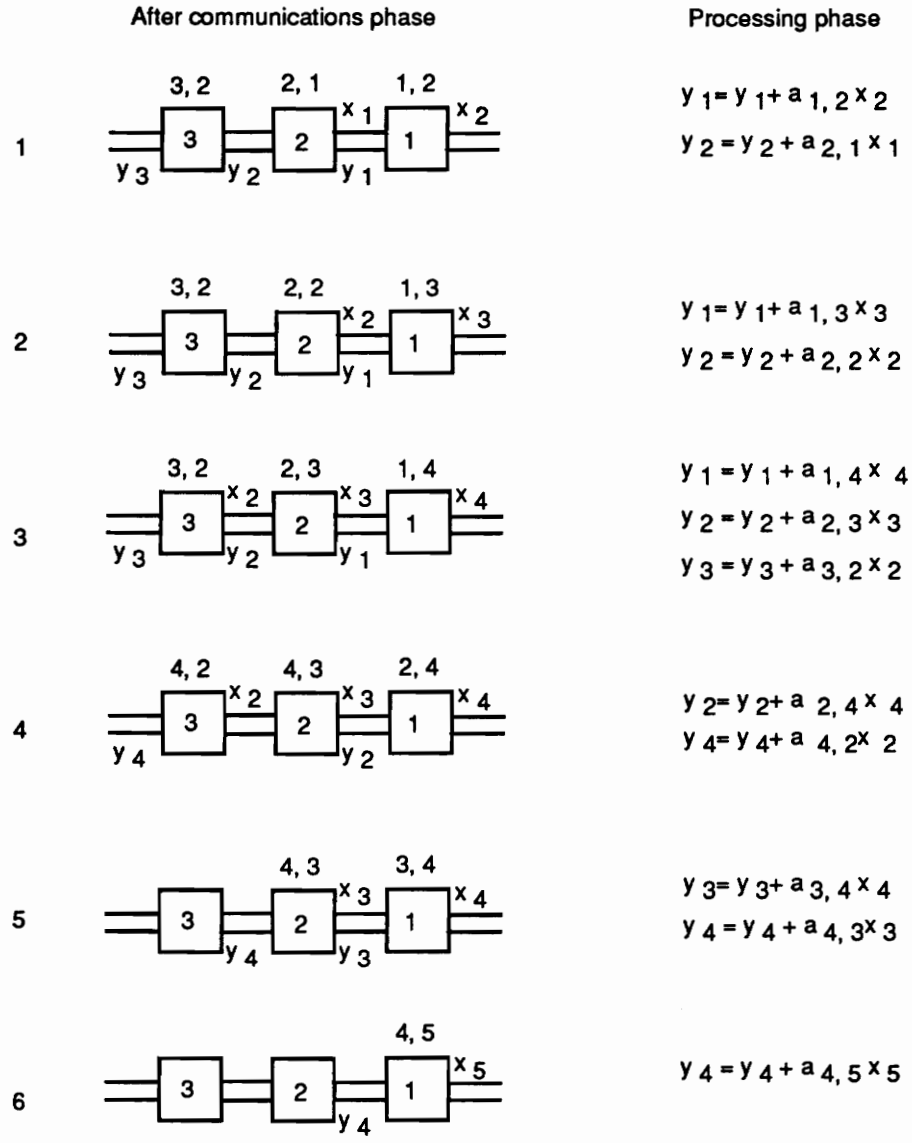


Figure 2.14: The computations performed by PMCN.

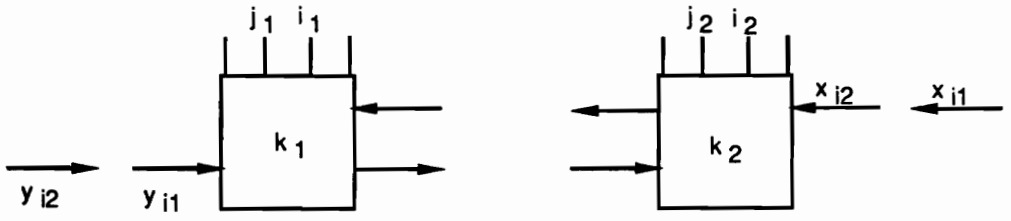


Figure 2.15: A deadlock in PMCN.

of M that (h, w)-SCA constructs.

> **Property (A):** Suppose that $(i_1, j_1), (i_2, j_2) \in \Gamma$ are position indices such that $(i_2, j_2) \leq_{ne} (i_1, j_1)$, $(i_1, j_1) \in \Psi_p$, and $(i_2, j_2) \in \Psi_q$. Then $p \leq q$.

Theorem 2.29 *For all i, $1 \leq i \leq n$, when y_i exits* PMCN, *its value is $y_i = \sum_{i=1}^{n} m_{i,j} \cdot x_j$.*

Proof: We first show that PMCN does not reach a deadlock state and for each i, $1 \leq i \leq n$, y_i exits PMCN. We assume that PMCN deadlocks and show a contradiction. A deadlock state is shown in Figure 2.15. PE k_1 is waiting for x_{j_2} which is blocked by PE k_2, which in turn is waiting for y_{i_2} which is blocked by PE k_1. Clearly, $i_2 \geq i_1$, $j_1 \geq j_2$, but $k_1 < k_2$. This contradicts Property (A) and hence a deadlock can never occur in PMCN.

Now we show that for each i, $1 \leq i \leq n$, when y_i exits PMCN, it has the value $\sum_{i=1}^{n} m_{i,j} \cdot x_j$. Assume that at some point of time during the computation y_i has reached PE ℓ and the element of M currently available to PE ℓ is $a_{p,q}$. Thus

$$[I_2] = y_i, \quad [I_3] = p, \quad [I_4] = q.$$

Three cases are possible: (i) $p < i$, (ii) $p = i$, (iii) $p > i$.

We now provide an argument that rules out Case (i). Since $[I_2] = y_i$ for PE ℓ currently, the component y_{i-1} must have passed through PE ℓ at some earlier time. Assume that when $[I_2] = y_{i-1}$, $[I_3] = r$ for PE ℓ. Since elements of M are fed through each PE in the increasing order with respect to $\leq_{se}$, $r \leq p$. We consider the possibility $r < p$ and $r = p$ separately. The inequality $r < p$ together with $p < i$ implies that $r < i - 1$. This implies

that y_{i-1} is blocked by PE ℓ and hence cannot pass through PE ℓ. The other possibility is that $r = p = i - 1$. In this case, PE ℓ waits for the appropriate component of x to appear on I_1 (PE ℓ may already have the appropriate component of x on its input I_1 in which case there is no wait) and updates y_{i-1}. But, after updating y_{i-1}, the PE retains it for future computations because the next element of the matrix to be fed into it is in the same row as the current element. In fact, y_{i-1} is retained as long as the input on $[I_3]$ is an element in M belonging to row $i - 1$. Thus $[I_2] = y_{i-1}$ even when the element $a_{p,q}$ is fed into I_3. This contradicts our assumption that $[I_2] = y_i$ when $a_{p,q}$ is fed into I_3 of PE ℓ. Hence Case (i) cannot hold.

This leaves the possibility that $p \geq i$. If $p > i$, then y_i is passed on to O_2 and it is the case that Ψ_ℓ contains no element in row i. If $p = i$, then PE ℓ updates y_i and retains it if the next element in Ψ_ℓ is in the same row (row i) as the current element; otherwise PE ℓ allows y_i to pass on. Thus all elements in row i belonging to each staircase are used in updating y_i and since the position index of every nonzero entry belongs to some staircase, we conclude that when y_i exits PMCN, it contains the value $\sum_{i=1}^{n} m_{i,j} \cdot x_j$. □

2.5.3 Analysis of PMCN

Having shown the correctness of PMCN, we analyze its performance. The analysis reveals that PMCN is an improvement on MAT/VEC, and it also provides a method to determine the number of global cycles PMCN will take to compute Mx, based on the structure of M. The analysis identifies the sets of position indices that participate in computations in each global cycle. Compute a sequence of sets $CF_1, CF_2, CF_3, \ldots$, each called a *computational front*, by the algorithm shown in Figure 2.5.3.

Example 2.30. For the matrix in Figure 2.12,

$$\begin{aligned} CF_1 &= \{(1,2),(2,1)\} & CF_4 &= \{(4,2),(2,4)\} \\ CF_2 &= \{(1,3),(2,2)\} & CF_5 &= \{(3,4),(4,3)\} \\ CF_3 &= \{(1,4),(2,3),(3,2)\} & CF_6 &= \{(4,5)\}. \end{aligned}$$

$S_0 := \Gamma$;
$\ell := 1$;
while $(S_{\ell-1} \neq \emptyset)$ **do**
  let CF_ℓ be the set of minima in $(S_{\ell-1}, \leq_{se})$;
  $S_\ell := S_{\ell-1} - CF_\ell$;
  $\ell := \ell + 1$;
endwhile
return $CF_1, CF_2, \ldots, CF_{\ell-1}$

Figure 2.16: Algorithm to the construct computational fronts of a matrix.

□

In Example 2.5.3, notice that for each $(i,j) \in CF_t$, for $1 \leq t \leq 6$, the computation $y_i = y_i + m_{i,j} \cdot x_j$ is performed in processing phase PP_t. That this is true in general is shown in the following theorem.

Theorem 2.31 *For all t, $(i,j) \in CF_t$ if and only if the computation $y_i = y_i + m_{i,j} \cdot x_j$ is performed in PP_t.*

Proof: Let

$$CF_1 = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$$

where

$$(i_1, j_1) <_{ne} (i_2, j_2) <_{ne} \cdots <_{ne} (i_m, j_m).$$

Assume that $(i_p, j_p) \in \Psi_{k_p}$ for all p, $1 \leq p \leq m$. Using Property (A) and the fact that no two elements of CF_1 can be assigned to the same staircase, we obtain

$$k_1 > k_2 > \cdots > k_m.$$

Since each element $(i_p, j_p) \in CF_1$ is the smallest element in staircase Ψ_p, initially, for each PE labeled k_p, $[I_3] = i_p$ and $[I_4] = j_p$. The initial status of the PEs in PMCN labeled $k_1, k_2, \ldots, k_m$ is shown in Figure 2.17. We now show that after the first communications phase, CP_1, $[I_1] = x_{j_p}$ and $[I_2] = y_{i_p}$ for PE k_p. This implies that during the first processing phase, PP_1, the PE k_p performs the update $y_{i_p} := y_{i_p} + m_{i_p,j_p} \cdot x_{j_p}$. During CP_1, x_{j_p}

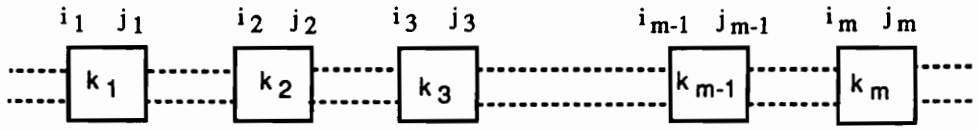


Figure 2.17: The initial status of PEs $k_1, k_2, \ldots, k_n$ in PMCN.

travels past PEs $k_m, k_{m-1}, \ldots, k_{p+1}$ because $j_q > j_p$ for all q, $p+1 \leq q \leq m$. Thus if x_{j_p} is blocked, it is due to a PE between k_{p+1} and k_p. Assume that there is a PE ℓ, $k_{p+1} < \ell < k_p$ with $[I_3] = q$ and $[I_4] = r$ blocking x_{j_p}. Thus $r \leq j_p$. If $q \leq i_p$, then $(q,r) \leq_{se} (i_p.j_p)$ and (i_p, j_p) is not minimal in $(\Gamma, \leq_{se})$ – a contradiction. The other possibility is that $q > i_p$. Therefore we have

$$q > i_p \quad \text{and} \quad r \leq j_p.$$

The position index $(q,r) \in \Psi_\ell$ and the position index $(i_p, j_p) \in \Psi_{k_p}$. Property (A) implies that $\ell \geq k_p$. This contradicts our assumption that $k_{p+1} < \ell < k_p$. We have thus shown that x_{j_p} reaches the PE labeled k_p at the end of CP_1. Similarly, it can be shown that y_{i_p} reaches the PE labeled k_p at the end of CP_1. Hence, the PE k_p performs the update on y_{i_p} in PP_1. Since, (i_p, j_p) is an arbitrary element in CF_1, we have shown that for any $(i,j) \in CF_1$, the computation $y_i := y_i + a_{i,j} \cdot x_j$ is performed in PP_1.

We now show that if $(i,j) \notin CF_1$, then the computation $y_i := y_i + a_{i,j} \cdot x_j$ is not performed in PP_1. If (i,j) is not the smallest element in some staircase, then it will not be fed into any PE in the first global cycle and hence the computation $y_i := y_i + a_{i,j} \cdot x_j$ cannot take place in PP_1. Hence, we assume that (i,j) is the smallest element in some staircase Ψ_ℓ, $1 \leq \ell \leq k$. Therefore (i,j) is fed into PE ℓ in the first global cycle. Three cases are possible: (i) $\ell > k_1$, (ii) $\ell < k_m$, and (iii) there exists a p, $2 \leq p \leq m$ such that $k_p < \ell < k_{p-1}$. If $\ell > k_1$, it can be verified that $(i_1, j_1) \leq_{se} (i,j)$. If $\ell < k_m$, it can be verified that $(i_m, j_m) \leq_{se} (i,j)$. If there exists a p, $2 \leq p \leq m$, such that $k_p < \ell < k_{p-1}$, then either $(i_{p-1}, j_{p-1}) \leq_{se} (i,j)$ or $(i_p, j_p) \leq_{se} (i,j)$ or both. We focus on Case (iii), because the proof for Case (iii) applies to Cases (i) and (ii) also. Assume that $(i_{p-1}, j_{p-1}) \leq_{se} (i,j)$. This implies that $i \geq i_{p-1}$. We know that at the end of CP_1, $[I_2] = y_{i_{p-1}}$ for PE k_{p-1}. Hence,

y_i cannot reach PE ℓ at the end of CP_1 and hence PE ℓ cannot perform the computation $y_i := y_i + a_{i,j} \cdot x_j$ in PP_1. Similarly, it can be shown that if $(i_p, j_p) \leq_{se} (i,j)$, then PE k_p blocks x_{j_p} from reaching PE ℓ in CP_1 and hence PE ℓ cannot update y_i in PP_1. Thus, we have shown that if $(i,j) \notin CF_1$, then the computation $y_i := y_i + a_{i,j} \cdot x_j$ cannot be performed in PP_1.

We have shown that $(i,j) \in CF_1$ if and only if $y_1 := y_i + a_{i,j} \cdot x_j$ is performed in PP_1. After the completion of PP_1 we are dealing with a matrix, whose set of position indices corresponding to nonzero entries is $\Gamma - CF_1$. CF_2 plays the same role now as CF_1 did in the first global cycle. Thus, the argument presented above can be extended inductively to show that $(i,j) \in CF_t$ if and only if the computation $y_i := y_i + a_{i,j} \cdot x_j$ is performed in PP_t. □

We are now in a position to compare the performance of PMCN with the performance of Melhem's scheme, MAT/VEC. The number of PEs required by PMCN to compute Mx is equal to $SCN(M)$. The number of PEs required by MAT/VEC to compute Mx is equal to the stripenumber of M. Since, the staircase number of a matrix is never greater than its stripe number and is often substantially less, PMCN clearly has a smaller hardware requirement than MAT/VEC. As an example, consider the matrix shown in Figure 2.18. The bandwidth of the matrix is b and the nonzeros in the band are those that are covered by the staircase shown in the figure. Though one staircase suffices to cover the nonzeros, the stripenumber of the matrix is b.

Now we compare the time, measured in global cycles, taken by MAT/VEC and PMCN to compute Mx. Theorem 2.31 shows that for all t, a position index belongs to CF_t, if and only if $y_i := y_i + m_{i,j} \cdot x_j$ is performed in PP_t. This result is important because, as shown by Melhem, even for MAT/VEC it is true that $(i,j) \in CF_t$ if and only if $y_i := y_i + m_{i,j} \cdot x_j$ is computed in PP_t. In other words, PMCN as well as MAT/VEC take the same number of global cycles to compute Mx and that is equal to the number of computational fronts in M. This shows that PMCN, despite using lesser hardware, does *not* need more time steps

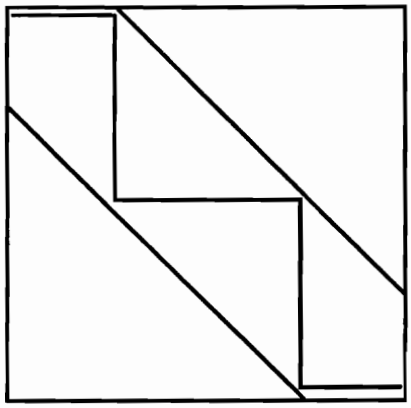


Figure 2.18: A matrix whose band is sparse.

	3	2	1	
5	4	3	2	
	5		3	
	6	4		1

Figure 2.19: A matrix with stripenumber 6.

than MAT/VEC. The comparison between MAT/VEC and PMCN is summarized in the following theorem.

Theorem 2.32 *Let M be an $n \times n$ and x an n-vector. Suppose* PMCN *takes k_1 PEs and t_1 global cycles to compute Mx. Suppose* MAT/VEC *takes k_2 PEs and t_2 global cycles to compute Mx. Then, $k_1 \leq k_2$ and $t_1 = t_2$.*

Example 2.33. Figures 2.19, 2.20, and 2.21 show the working of MAT/VEC as it preforms a matrix vector multiplication for the matrix shown in Figure 2.12. The matrix is minimally covered with 6 stripes and elements assigned to stripe j are labeled j. Notice

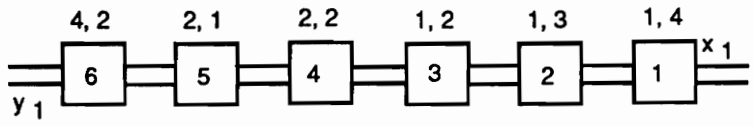


Figure 2.20: The initial status of MAT/VEC.

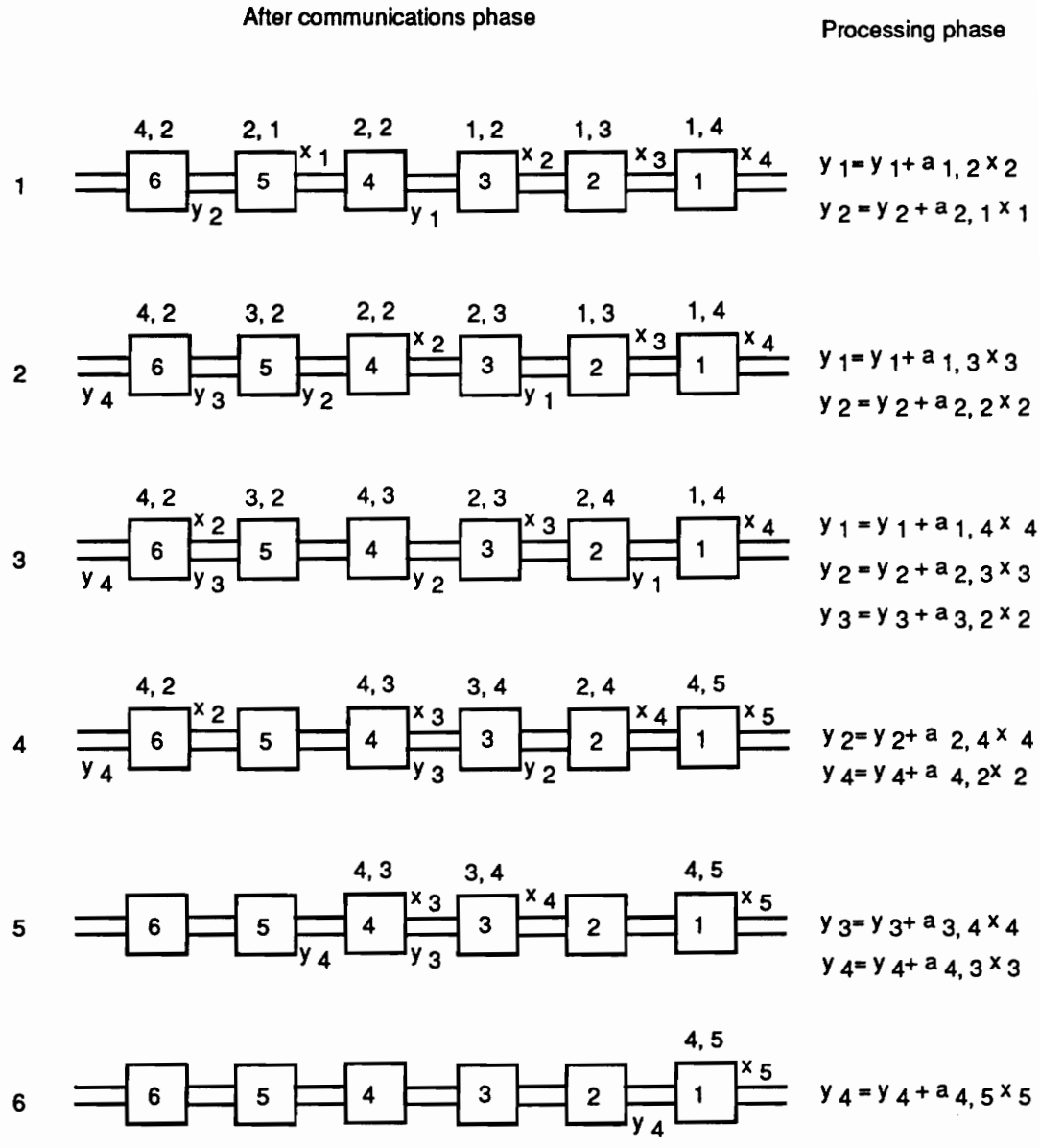


Figure 2.21: The computations performed by MAT/VEC.

that the stripes are non-crossing. Figure 2.20 shows the status of MAT/VEC initially, and Figure 2.21 shows the network after each communications phase and the computations performed in each processing phase. Notice that the computations performed in each processing phase are the same for PMCN and MAT/VEC, though PMCN requires three PEs while MAT/VEC requires six PEs. □

Melhem provides a lower bound (see Proposition 3.1 in [51]) and an upper bound (see Proposition 3.2 in [51]) on the number of global cycles that MAT/VEC requires to complete Mx. Neither of the bounds are tight. In the next theorem we determine exactly the number of global cycles required by PMCN (as well as by MAT/VEC) to compute Mx, as a function of the structure of matrix M.

Theorem 2.34 *Let $\Psi \subseteq \Gamma$ be a largest staircase in the $n \times n$ matrix M. Then, for any n-vector x, the number of global cycles required by* PMCN *or* MAT/VEC *to compute Mx is $|\Psi|$.*

Proof: The number of computational fronts in M is equal to the minimum number of chains required to cover the poset $(\Gamma, \leq_{ne})$. Using Dilworth's Theorem we conclude that the number of computational fronts in M is equal to the size of the largest antichain in $(\Gamma, \leq_{ne})$. The theorem follows from the fact that a largest antichain in $(\Gamma, \leq_{ne})$ is a largest staircase in M entirely consisting of position indices of nonzero entries of M. □

Example 2.35. For the matrix shown in Figures 2.12 and 2.19 (a), a largest staircase consisting entirely of position indices of nonzero entries of M is

$$\{(2,1),(2,2),(2,3),(2,4),(3,4),(4,5)\}.$$

□

In $O(T(M)\log\log n)$ time, a minimal (h,w)-staircase cover of $M = (m_{i,j})_{n\times n}$ can be computed, where $T(M)$ is the time required to visit all the nonzeros in the matrix (see

Theorem 2.23). Since the number of PEs required by PMCN is equal to the size of a minimal staircase cover of M and the number of PEs required by MAT/VEC is equal to the size of a minimal stripe cover of M, we have that the number of PEs required either by PMCN or by MAT/VEC can be computed in $O(T(M)\log\log n)$ time.

Computing the size of the largest staircase in M that contains only position indices of nonzeros corresponds to computing the minimum number of chains in $(\Gamma, \leq_{ne})$ required to cover Γ. Relabel the columns of M by changing the column label i to $n - i + 1$ to produce a new matrix M'. Finding the minimum number of chains in $(\Gamma, \leq_{ne})$ that cover Γ corresponds to finding a minimal $(1,1)$-staircase cover in M'. But, this problem can be solved in $O(T(M)\log\log n)$ time. This leads to the following theorem.

Theorem 2.36 *Given an $n \times n$ matrix M, the number of PEs and the number of global cycles required by PMCN or MAT/VEC to compute Mx, for any n-vector x, can be determined in $O(T(M)\log\log n)$ time, where $T(M)$ is the time required to visit all the nonzeros in M.*

Melhem modifies MAT/VEC slightly and uses it to solve linear systems of equations using an efficient iterative technique called the *preconditioned conjugate gradient method* (PCCG). The bulk of the work in each iteration of the PCCG method is a matrix vector multiplication and the solution of two triangular linear systems of equations. Melhem shows that a slightly modified version of MAT/VEC can be used to solve triangular linear systems of equations. In exactly the same way PMCN can be modified to solve triangular linear systems of equations and hence can be used to iteratively solve linear systems of equations using the PCCG method.

2.6 Conclusions

The research presented in this chapter was the result of recognizing a simple and elegant connection between queue layouts and covering a matrix with staircases. This connection has led to a harvest of results that can be broadly classified as follows:

1. **Results for queue layouts:** We have exploited the connection between queue layouts and covering a matrix with staircases to obtain combinatorial, lower bound, and upper bound results for queue layouts. In some cases we have significantly simplified existing proofs.

2. **Efficient algorithms to solve a class of matrix covering problems:** Inspired by an algorithm of Heath and Rosenberg that computes the queuenumber of a fixed layout of a graphs, we have constructed an efficient algorithm to solve a large class of matrix covering problems. This algorithm, in turn, leads to alternate and possibly simpler solutions to problems related to sequences, graph theory, and computational geometry.

3. **An efficient scheme for matrix computations on a data driven network:** We have presented a scheme for performing matrix computations in parallel on a data driven network. Analysis of our scheme reveals that it is an improvement in terms of hardware as well as in terms of time, over existing schemes. Our analysis also answers questions related to Melhem's scheme.

Chapter 3

Separated and Mingled Layouts

Separated layouts are layouts in which the vertex set of the graph is partitioned and vertices in each block of the partition are constrained to appear contiguously and therefore separate from vertices in other blocks. Mingled layouts are layouts in which the vertex set of the graph is partitioned and the vertices belonging to each block are constrained to appear in a particular order, though the blocks themselves may be mingled in any manner. Separated and mingled layouts provide a systematic approach to proving lower bounds on the stacknumber and queuenumber of various families of graphs. In this chapter, we develop the theory of separated and mingled layouts by presenting various upper and lower bound results.

We begin by making precise the notion of separated and mingled layouts in Section 3.1. In Section 3.2, we develop the theory of separated and mingled queue layouts of graphs with respect to partitions of size 2. In Section 3.3, we develop the theory of separated stack layouts for partitions of size 2. Section 3.4 examines separated stack and queue layouts for partitions of arbitrary size. Section 3.5 presents open questions and conjectures in the area of separated and mingled stack and queue layouts.

3.1 Definitions and Notation

Suppose $G = (V, E)$ is a graph and $\gamma = \{V_1, V_2, \ldots, V_m\}$ is a partition of V. A *separated order on V with respect to γ* is a total order σ of V such that for some permutation π of J_m, if $u \in V_i$ and $v \in V_j$, then $u <_\sigma v$ if $\pi^{-1}(i) < \pi^{-1}(j)$. In other words, σ is an order of V in which for each i, $1 \leq i \leq m$ the elements of V_i appear consecutively and the blocks

themselves appear in the order specified by π. It is easy to see that the number of separated orders on V with respect to γ is

$$m! \cdot \prod_{i=1}^{m} |V_i|!$$

Example 3.1. Let $V = \{1,2,3,4\}$ and let $\{V_1, V_2\}$ be the partition of V given by $V_1 = \{1,3\}$ and $V_2 = \{2,4\}$. The separated orders on V with respect to $\{V_1, V_2\}$ are enumerated below.

$$1,3,2,4$$

$$2,4,1,3$$

$$1,3,4,2$$

$$2,4,3,1$$

$$3,1,4,2$$

$$4,2,3,1$$

$$3,1,2,4$$

$$4,2,1,3$$

The formula $m! \cdot \prod_{i=1}^{m} |V_i|!$ predicts that the number of separated orders on V with respect to $\{V_1, V_2\}$ is $2!(2! \cdot 2!) = 8$, precisely the number of total orders on V enumerated above. □

A *separated layout of G with respect to γ* is a layout of G according to a separated order on V with respect to γ.

Suppose σ is a total order of V. A *separated order δ with respect to σ and γ* is a separated order of V with respect to γ in which if $u, v \in V_i$ for some i, $1 \leq i \leq m$ and $u <_\sigma v$, then $u <_\delta v$. In other words, δ is a separated order of V in which for each i, $1 \leq i \leq m$, the elements of block V_i occur consecutively and in the order specified by σ. It is easy to see that the number of separated orders on V with respect to σ and γ is simply $m!$

Example 3.2. Let V, V_1, and V_2 be as in Example 3.1. Let $\sigma = 2,4,3,1$ be a total order on V. Then, the separated orders on V with respect to σ and $\{V_1, V_2\}$ are

$$2,4,3,1$$

and

$$3, 1, 2, 4.$$

Note that these are the separated orders enumerated in Example 3.1 in which elements in V_1 and V_2 occur in the order specified by $\sigma = 2, 4, 3, 1$. □

A *separated layout of G with respect to σ and γ* is a layout of G according to a separated order on V with respect to σ and γ.

A *mingled order δ with respect to σ and γ* is defined as a total order of V such that if $u, v \in V_i$ for some i, $1 \leq i \leq m$ and $u <_\sigma v$, then $u <_\delta v$. In other words, elements belonging to each block appear in an order specified by σ, but the blocks themselves can be mingled in any manner possible. It is easy to see that the number of mingled orders with respect to σ and γ is

$$\frac{n!}{\prod_{i=1}^{m} |V_i|!}.$$

Example 3.3. Let V, V_1, V_2, and σ be as in Example 3.1. The mingled orders on V with respect to σ and $\{V_1, V_2\}$ are enumerated below.

$$2, 4, 3, 1$$

$$3, 2, 4, 1$$

$$2, 3, 4, 1$$

$$3, 2, 1, 4$$

$$2, 3, 1, 4$$

$$3, 1, 2, 4$$

The formula $n!/\prod_{i=1}^{m} |V_i|!$ predicts that the number of mingled orders on V with respect to $\{V_1, V_2\}$ is $4!/2!2! = 6$, precisely the number of total orders enumerated above. □

A *mingled layout of G with respect to σ and γ* is a layout of G according to a mingled order on V with respect to σ and γ.

Suppose $G = (V_1, V_2, \ldots, V_m, E)$ is an m-partite graph. The partition $\{V_1, V_2, \ldots, V_m\}$ is called the *natural partition* of $V = \cup V_i$.

3.2 Separated and Mingled Queue Layouts

We now prove a fundamental result of this chapter. The following theorem shows that any 1-queue bipartite graph has a 2-queue separated layout with respect to the natural partition of its vertex set. This result is first extended to mingled layouts of 1-queue bipartite graphs, then to separated and mingled layouts of arbitrary bipartite graphs, and finally to separated and mingled layouts of arbitrary graphs. This result also leads to a connection between queue layouts and separated stack layouts for bipartite graphs.

Theorem 3.4 *Suppose $G = (V_1, V_2, E)$ is a 1-queue bipartite graph. Then there exists a 2-queue separated layout of G with respect to the natural partition $\gamma = \{V_1, V_2\}$. Furthermore, if σ is a total order on $V_1 \cup V_2$ that yields a 1-queue layout of G, then any separated order on $V_1 \cup V_2$ with respect to σ and γ yields a 2-queue layout of G.*

Proof: Let σ be an arbitrary total order on $V_1 \cup V_2$ that yields a 1-queue layout of G and let δ be a separated order on $V_1 \cup V_2$ with respect to σ and γ. Since σ yields a 1-queue layout of G, it induces an arched leveled-planar embedding of G, which in turn induces a partition of $V_1 \cup V_2$ into levels $L_1, L_2, \ldots, L_p$. Lay out the vertices of G according to δ. We derive an assignment of the edges of G to two queues q_1 and q_2 from the arched leveled-planar embedding of G induced by σ. Assign to queue q_1 those level-i edges which connect a vertex $u \in V_1 \cap L_i$ to a vertex $v \in V_2 \cap L_{i+1}$, for all i, $1 \leq i \leq p-1$, and arches (u, t_i) for $u \in V_1$, and all i $1 \leq i \leq p$. Similarly, assign to queue q_2 those level-i edges that connect a vertex $u \in V_2 \cap L_i$ to a vertex $v \in V_1 \cap L_{i+1}$, for all i, $1 \leq i \leq p-1$, and arches (u, t_i) for $u \in V_2$, and for all i, $1 \leq i \leq p$.

We now show that no two edges assigned to q_1 nest. Let (u_1, v_1) and (u_2, v_2) be two edges assigned to q_1. Without loss of generality assume that $u_1 <_\sigma v_1$, $u_2 <_\sigma v_2$, $u_1 <_\sigma u_2$, $u_1 \in L_i$, and $u_2 \in L_j$. Note that these assumptions imply that $1 \leq i \leq j \leq p$. Since δ is a separated order on $V_1 \cup V_2$ with respect to σ and γ and $u_1, u_2 \in V_1$, $u_1 <_\sigma u_2$ implies that $u_1 <_\delta u_2$. There are four cases based on whether the edges (u_1, v_1) and (u_2, v_2) are leveled edges or arches.

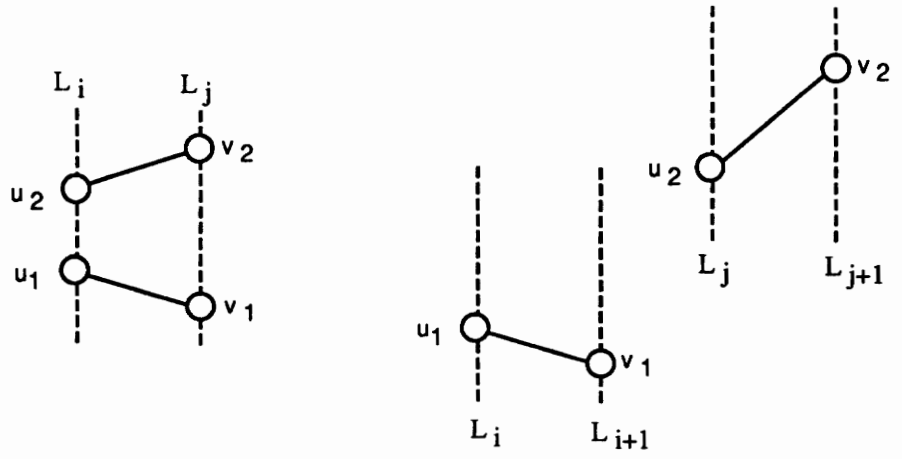


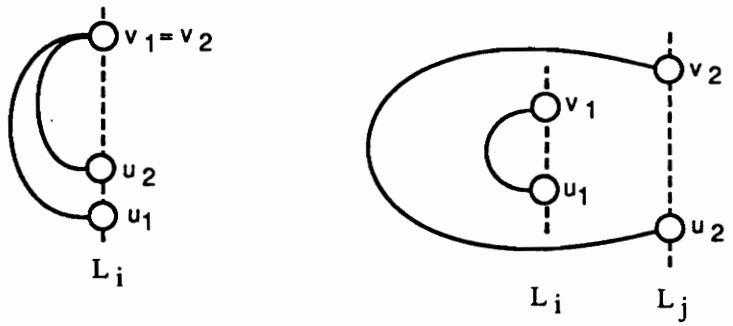
Figure 3.1: (u_1, v_1) is a level-i edge and (u_2, v_2) is a level-j edge. (a) $i = j$. (b) $i < j$.

Case 1 (u_1, v_1) and (u_2, v_2) are leveled edges. Then $v_1 \in L_{i+1}$ and $v_2 \in L_{j+1}$. If $i = j$, then by planarity $v_1 <_\sigma v_2$. See Figure 3.1 (a). If $i < j$ then $i + 1 < j + 1$ and hence $v_1 <_\sigma v_2$. See Figure 3.1 (b). In either case, $v_1 <_\delta v_2$ and hence the edges (u_1, v_1) and (u_2, v_2) do not nest.

Case 2 (u_1, v_1) and (u_2, v_2) are arches. Then $v_1 \in L_i$ and $v_2 \in L_j$. If $i = j$, then $v_1 = v_2 = t_i$ and (u_1, v_1) and (u_2, v_2) do not nest. See Figure 3.2 (a). If $i < j$ then $v_1 <_\sigma v_2$. See Figure 3.2 (b). Therefore $v_1 <_\delta v_2$ and hence the edges (u_1, v_1) and (u_2, v_2) do not nest.

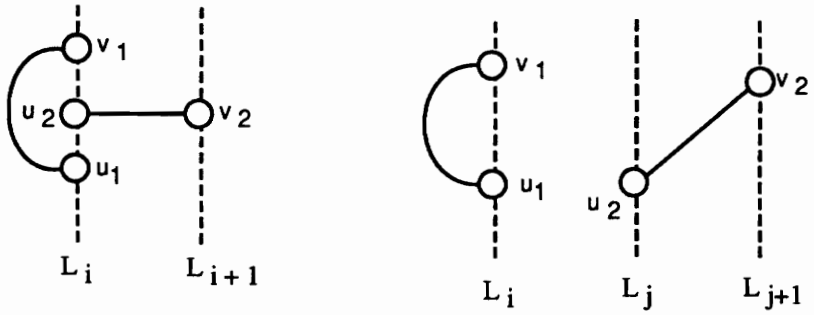
Case 3 (u_1, v_1) is an arch and (u_2, v_2) is a leveled edge. Then $v_1 \in L_i$ and $v_2 \in L_{j+1}$ Since $i \leq j$ it follows that $v_1 <_\sigma v_2$. See Figures 3.3 (a) and 3.3 (b). Therefore $v_1 <_\delta v_2$ and hence the edges (u_1, v_1) and (u_2, v_2) do not nest.

Case 4 (u_1, v_1) is a leveled edge and (u_2, v_2) is an arch. Then $v_1 \in L_{i+1}$ and $v_2 \in L_j$. If $i = j$ then $u_1 \not<_\sigma u_2$ by the definition of an arched, leveled-planar graph. If $i < j$ then $i + 1 \leq j$. If $i + 1 = j$, then $v_1 <_\sigma v_2$ because v_2 is the topmost vertex in its level. See



(a) (b)

Figure 3.2: (u_1, v_1) is an arch in level i and (u_2, v_2) is an arch in level j. (a) $i = j$. (b) $i < j$.



(a) (b)

Figure 3.3: (u_1, v_1) is an arch in level i and (u_2, v_2) is a level-j edge. (a) $i = j$. (b) $i < j$.

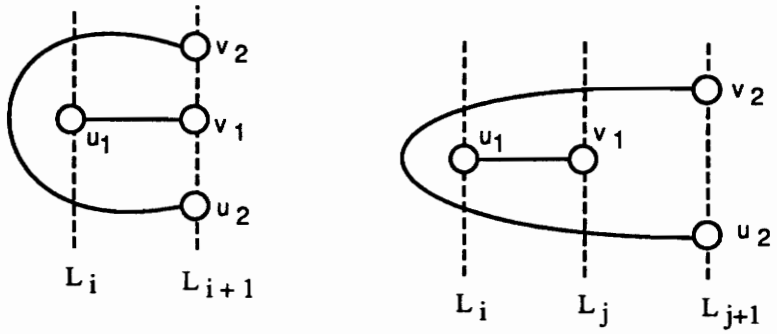


Figure 3.4: (u_1, v_1) is a level-i edge and (u_2, v_2) is an arch in level j. (a) $i + 1 = j$. (b) $i + 1 < j$.

Figure 3.4 (a). If $i+1 < j$, then $v_1 <_\sigma v_2$. See Figure 3.4 (b). In either case, $v_1 <_\delta v_2$ and hence the edges (u_1, v_1) and (u_2, v_2) do not nest.

This exhausts all possibilities and shows that the edges (u_1, v_1) and (u_2, v_2) do not nest. Since the choice of the edges (u_1, v_1) and (u_2, v_2) was arbitrary, we have shown that no two edges assigned to q_1 nest. A similar proof shows that no two edges assigned to q_2 nest. The theorem follows. □

Example 3.5. Figure 3.5 (a) shows an arched leveled-planar embedding of a 1-queue bipartite graph $G = (V_1, V_2, E)$ where $V_1 = \{1, 5, 6\}$ and $V_2 = \{2, 3, 4, 7\}$. The vertices in V_2 are shaded to distinguish them from the vertices in V_1. Figure 3.5 (b) shows a 1-queue layout of G according to the total order $\sigma = 1, 2, 3, 4, 5, 6, 7$. Note that this total order is derived from the arched leveled-planar embedding shown in Figure 3.5 (a). Figure 3.5 (c) shows a 2-queue separated layout of G with respect to σ and the natural partition. The assignment of edges to the two queues is as described in the proof of Theorem 3.4. □

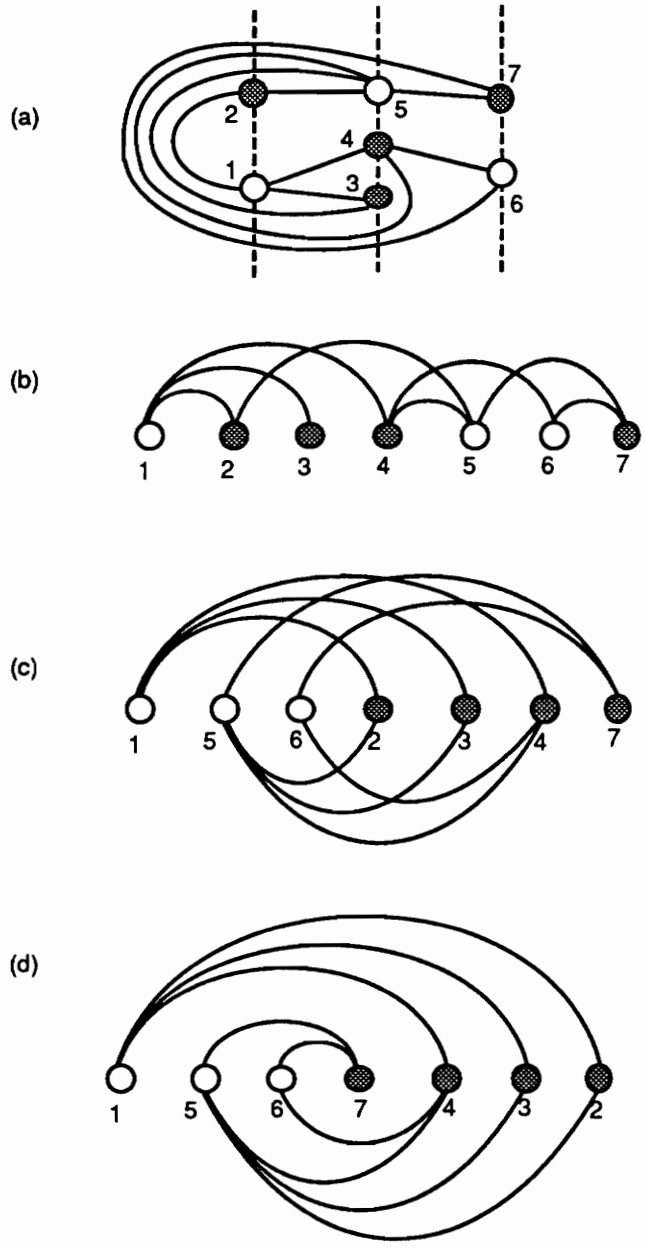


Figure 3.5: (a) An arched leveled-planar embedding of a 1-queue bipartite graph $G = (V_1, V_2, E)$. (b) 1-queue layout of G. (c) 2-queue separated layout of G. (d) 2-stack separated layout of G.

Theorem 3.4 shows that there are ways of separating the two blocks of the natural partition of a 1-queue bipartite graph so that the queue requirement of the new layout is at most 2. It is easy to see that the upper bound in Theorem 3.4 is tight. Consider the complete bipartite graph $K_{2,2}$. $K_{2,2}$ is a 1-queue bipartite graph such that any separated layout of $K_{2,2}$ with respect to the natural partition of its vertex set requires 2 queues.

Theorem 3.4 can be used to obtain a connection between queue layouts and separated stack layouts of bipartite graphs. Suppose $G = (V_1, V_2, E)$ is a 1-queue bipartite graph and σ is a total order on $V_1 \cup V_2$ that yields a 1-queue layout of G. Theorem 3.4 shows that if δ is a separated order on $V_1 \cup V_2$ with respect to σ and the natural partition $\gamma = \{V_1, V_2\}$ of V, then δ yields a 2-queue layout of G. Lay out G according to δ and assign the edges to two queues q_1 and q_2. From δ obtain δ' by reversing the order of vertices in V_2, that is if $u, v \in V_2$ and $u <_\delta v$, then $v <_{\delta'} u$. Note that δ' is a separated order on $V_1 \cup V_2$ with respect to γ. Lay out the vertices of $V_1 \cup V_2$ according to δ' and for $i = 1, 2$, assign to stack s_i those edges that were previously assigned to queue q_i. Since every edge in G is between a vertex in V_1 and a vertex in V_2 and no two edges assigned to a queue nest when the vertices are laid out according to δ, it follows that no two edges assigned to the same stack intersect when the vertices are laid out according to δ'. This leads to the following corollary.

Corollary 3.6 *If $G = (V_1, V_2, E)$ is a 1-queue bipartite graph, then there exists a 2-stack separated layout of G with respect to the natural partition of V.*

Example 3.7. Figure 3.5 (d) shows a 2-stack separated layout of the graph G in Figure 3.5 (a) with respect to the natural partition of the vertex set of G. Note that the 2-stack separated layout is simply obtained by reversing the order on the vertices in V_2 in the 2-queue separated layout of G shown in Figure 3.5 (c). □

In the proof of Theorem 3.4, the fact that the vertices in V_1 are separated from the vertices in V_2 is not used. What is used is the fact that the vertices within each block occur in the order prescribed by σ. This observation immediately leads to the following

generalization of Theorem 3.4 to mingled layouts.

Corollary 3.8 *Suppose $G = (V_1, V_2, E)$ is a 1-queue bipartite graph, γ is the natural partition of its vertex set, and σ is a total order on $V_1 \cup V_2$ that yields a 1-queue layout of G. Then any mingled order on $V_1 \cup V_2$ with respect to σ and γ yields a 2-queue layout of G.*

The next step is a generalization of Theorem 3.4 to k-queue bipartite graphs.

Theorem 3.9 *Suppose $G = (V_1, V_2, E)$ is a k-queue bipartite graph and γ is the natural partition of its vertex set. Then there exists a $2k$-queue separated layout of G with respect to γ. Furthermore, if σ is a total order on $V_1 \cup V_2$ that yields a k-queue layout of G, then any separated order on $V_1 \cup V_2$ with respect to σ and γ yields a $2k$-queue layout of G.*

Proof: Let σ be an arbitrary total order on $V_1 \cup V_2$ that yields a k-queue layout of G. Consider a k-queue layout of G according to σ in which the edges are assigned to queues $q_1, q_2, \ldots, q_k$. For each i, $1 \leq i \leq k$, let $E_i \subseteq E$ be the set of edges assigned to q_i and let $G_i = (V_1, V_2, E_i)$ be the subgraph of G induced by E_i. It follows that for each i, $1 \leq i \leq k$, σ yields a 1-queue layout of G_i. Let δ be a separated order on $V_1 \cup V_2$ with respect to σ and γ. From Theorem 3.4, it follows that δ yields a 2-queue layout of G_i, for each i, $1 \leq i \leq k$. Hence, we can lay out the vertices of G according to δ and partition E_i into two sets each of which is assigned to a queue. Thus δ yields a $2k$-queue layout of G. □

Theorem 3.9 shows that there are ways of separating the two blocks of the natural partition of an arbitrary bipartite graph so that the queue requirements of the new layout are at most doubled. Heath and Rosenberg [36] show that $QN(K_{m,n}) = \min(\lceil m/2 \rceil, \lceil n/2 \rceil)$ by establishing matching lower and upper bounds on $QN(K_{m,n})$. Theorem 3.9 yields a much easier lower bound proof for $QN(K_{m,n})$. (see Chapter 2, Corollary 2.19 for a rather different proof of this result). Any separated layout of $K_{m,n}$ with respect to the natural partition requires at least $\min(m, n)$ queues. By combining this observation with the result proved in Theorem 3.9 we obtain the following corollary.

Corollary 3.10 $QN(K_{m,n}) \geq \min(\lceil m/2 \rceil, \lceil n/2 \rceil)$.

Note that $K_{m,n}$ is also an example to show that the upper bound of $2k$ on the queuenumber of a separated layout of a k-queue bipartite graph with respect to its natural partition is tight. Just as Theorem 3.4 leads to a connection between queue layouts and separated stack layouts of 1-queue bipartite graphs, Theorem 3.9 leads to a connection between queue layouts and separated stack layouts of arbitrary bipartite graphs.

Corollary 3.11 *If $G = (V_1, V_2, E)$ is a k-queue bipartite graph, then G has a $2k$-stack separated layout with respect to its natural partition.*

Theorem 3.9 was obtained by extending Theorem 3.4. In the same way, we extend Corollary 3.8 to obtain the following result for mingled layouts of arbitrary bipartite graphs.

Corollary 3.12 *Suppose $G = (V_1, V_2, E)$ is a k-queue bipartite graph and σ is a total order on $V_1 \cup V_2$ that yields a k-queue layout of G. Then any mingled order with respect to σ and the natural partition of G yields a $2k$-queue layout of G.*

The result in Theorem 3.9 can be further generalized to arbitrary graphs to obtain the following Separation Theorem.

Theorem 3.13 (Separation Theorem.) *Suppose $G = (V, E)$ is a k-queue graph and $\gamma = \{V_1, V_2\}$ is an arbitrary partition of V. Then there is a separated layout with respect to γ that requires at most $3k$ queues. Furthermore, if σ is a total order on V that yields a k-queue layout of G, then any separated order on V with respect to σ and γ yields a $3k$-queue layout of G.*

Proof: Let σ be an order of V, that yields a k-queue layout of G. Let

$$E' = E - \{(u, v) \mid u, v \in V_1 \text{ or } u, v \in V_2\}$$

and let $G' = (V_1, V_2, E')$. G' is a bipartite graph and σ yields a k-queue layout of G'. From Theorem 3.9, it follows that if δ is a separated order on V with respect to σ and γ, then G' can be laid out according to δ in $2k$ queues. Furthermore, the edges between the vertices in V_1 and those between vertices in V_2 can be assigned to an additional k queues. The theorem

follows. □

The following Mingling Theorem is an extension of the Separation Theorem to mingled layouts.

Theorem 3.14 (Mingling Theorem.) *Suppose $G = (V, E)$ is a graph that can be laid out in k queues according to some total order σ on V. Let $\gamma = \{V_1, V_2\}$ be an arbitrary partition of V. Then any mingled order on V with respect to σ and δ yields a $4k$-queue layout of G.*

Proof: As in the proof of the Separation Theorem, let

$$E' = E - \{(u, v) \mid u, v \in V_1 \text{ or } u, v \in V_2\}$$

and let $G' = (V_1, V_2, E')$. Then G' is a bipartite graph and σ yields a k-queue layout of G'. From Corollary 3.12, it follows that if δ is a mingled order on V with respect to σ and γ, then G' can be laid out according to δ in $2k$ queues. Unlike in the proof of the Separation Theorem, edges between vertices in V_1 and those between vertices in V_2 cannot be assigned to the same k queues. But, we can assign them to an additional $2k$ queues. The theorem follows. □

3.3 Separated and Mingled Stack Layouts

There is an asymmetry between the results we have for separated stack layouts and the results that we have for separated queue layouts. For queues we showed that if σ yields a 1-queue layout of a bipartite graph $G = (V_1, V_2, E)$, then any separated order on $V_1 \cup V_2$ with respect to σ and the natural partition of the vertex set of G, yields a 2-queue layout of G. This is not true for stacks. In the following example, we construct a class of 1-stack bipartite graphs. For any graph $G = (V_1, V_2, E)$ in this class and an arbitrary total order σ on $V_1 \cup V_2$ that yields a 1-stack layout of G, we show that a separated order δ on $V_1 \cup V_2$

with respect to σ and the natural partition of the vertex set of G does *not* yield a 2-stack layout of G. In fact, we show that the stacknumber of the layout of G according to δ can be arbitrarily bad.

Example 3.15. Let $G(n) = (V_1, V_2, E)$ be a bipartite graph with $2n$ vertices such that

$$\begin{aligned} V_1 &= \{v_0, v_2, \ldots, v_{2n-2}\}, \\ V_2 &= \{v_1, v_3, \ldots, v_{2n-1}\}, \\ E &= \{(v_i, v_{i+1 \bmod 2n}) \mid 0 \leq i \leq 2n-1\}. \end{aligned}$$

Notice that $G(n)$ is simply a cycle of length $2n$. Let σ be a total order on $V_1 \cup V_2$ that yields a 1-stack layout of $G(n)$. Since $G(n)$ is a biconnected 1-stack graph, it has a unique outerplanar embedding and σ is obtained by traversing the face of the outerplanar embedding in clockwise or counter clockwise direction starting at some vertex. Without loss of generality, assume that $v_i <_\sigma v_j$ if $i < j$. Let δ be a separated order on $V_1 \cup V_2$ with respect to σ and the natural partition of the vertex set of $G(n)$. It is either the case that all vertices in V_1 occur before the vertices in V_2 in δ, that is $u <_\delta v$ for all $u \in V_1$ and $v \in V_2$ or it is the case that all vertices in V_2 occur before the vertices in V_1, that is $u <_\delta v$ for all $u \in V_2$ and $v \in V_1$. In either case, the set of edges

$$\left\{(v_i, v_{i+1}) \mid i = 0, 2, 4, \ldots, 2n-2\right\}$$

form an n-twist in δ. Thus $SN(G(n), \delta) \geq n$. In fact, $SN(G(n), \delta) = n$. Figure 3.6 shows 1-stack layout of $G(3)$ according to $\sigma = 0, 1, 2, 3, 4, 5$ and Figure 3.7 shows the separated layout of $G(3)$ with respect to σ and the natural partition of $G(3)$. Notice the twist of size 3 in the layout shown in Figure 3.7. □

By comparing the above example to Theorem 3.4, we notice A contrast between separated stack layouts and separated queue layouts. Though we can no longer entertain hopes of proving a result analogous to Theorem 3.4 for stacks, it is possible to prove the following weaker result.

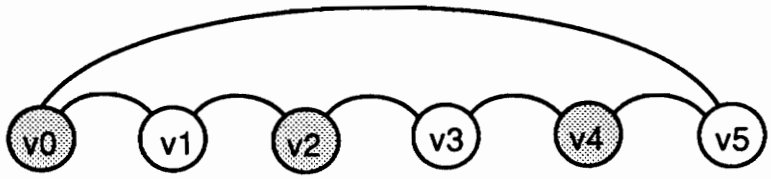


Figure 3.6: A 1-stack layout of the graph in Example 3.3.

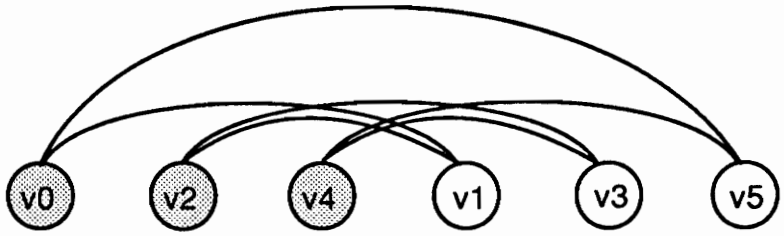


Figure 3.7: A separated layout of the graph in Example 3.3.

Theorem 3.16 *If a bipartite graph $G = (V_1, V_2, E)$ can be laid out in 1 stack, then it has a 2-stack separated layout with respect to $\gamma = \{V_1, V_2\}$.*

Proof: Without loss of generality, we may assume that G is connected. Since G is a 1-stack graph, it is outerplanar. Consider an arbitrary outerplanar embedding of G and let σ be a total order on $V_1 \cup V_2$ obtained by traversing the outer face of the embedding of G in clockwise or counter clockwise direction. Partition $V_1 \cup V_2$ into levels $L_1, L_2, \ldots, L_p$ in the following manner. Choose an arbitrary vertex $r \in V_1$ and let $L_1 = \{r\}$. For all i, $1 \leq i \leq p-1$,

$$L_{i+1} = \{u \,|\, (u,v) \in E, v \in L_i, \text{ and } u \notin L_j \text{ for any } j \leq i\}.$$

In other words, the partitioning of the vertices of G into levels is achieved by doing a breadth first search of G starting at r. The vertices in odd-indexed levels belong entirely to V_1 while the vertices in even-indexed levels belong entirely to V_2. Based on the partition of $V_1 \cup V_2$ into levels, we construct a leveled-planar embedding of G as follows. Place the vertices in L_i on a vertical line in the Cartesian plane defined by the equation $x = i$. Vertices placed on

each line are placed from bottom to top according to σ. The embedding of G so obtained is a leveled embedding because all edges in G are between vertices in L_i and vertices in L_{i+1} for some i. It is easy to see that it is also a planar embedding because any two edges in the leveled embedding that intersect have to intersect in the outerplanar embedding also. Therefore G is a leveled-planar graph. But, a leveled-planar graph is a 1-queue graph and by Corollary 3.6, any bipartite 1-queue graph has a 2-stack separated layout with respect to the natural partition of its vertex set. Therefore G has a 2-stack separated layout with respect to its natural partition γ. □

Example 3.17. Figure 3.8 (a) shows an outerplanar embedding of a 1-stack bipartite graph $G = (V_1, V_2, E)$ where $V_1 = \{1,3,5,7\}$ and $V_2 = \{2,4,6,8\}$. The vertices in V_2 are shaded to distinguish them from the vertices in V_1. Figure 3.8 (b) shows a leveled planar embedding of G obtained by doing a breadth first search starting at vertex 2. Figure 3.8 (c) shows a 2-queue separated layout of G with respect to the natural partition of the vertex set of G, derived from the leveled-planar enbedding. Figure 3.8 (d) shows a 2-stack separated layout of G derived from the 2-queue separated layout of G shown in Figure 3.8 (c). □

Theorem 3.4, which shows the existence of a 2-queue separated layout for any 1-queue graph, easily extends to Theorem 3.9, which shows the existence of a $2k$-queue separated layout for any k-queue graph. This extension was possible because a 2-queue separated layout of a graph is obtained from a 1-queue layout by simply shifting the vertices in V_2 to the right of the rest of the vertices. Hence, if we take a k-queue layout and shift all the vertices in V_2 to the right of the rest of the vertices, we have essentially shifted to the right, the vertices in V_2, in all the constituent 1-queue layouts and hence each of those layouts is transformed into a 2-queue layout. In other words the demands made by each 1-queue layout are consistent and hence can be met. On the other hand the demands made by each 1-stack layout on the order of a k-stack graph need not be consistent with each other. Hence, we do not have a Separation Theorem for stack layouts. Instead, we have

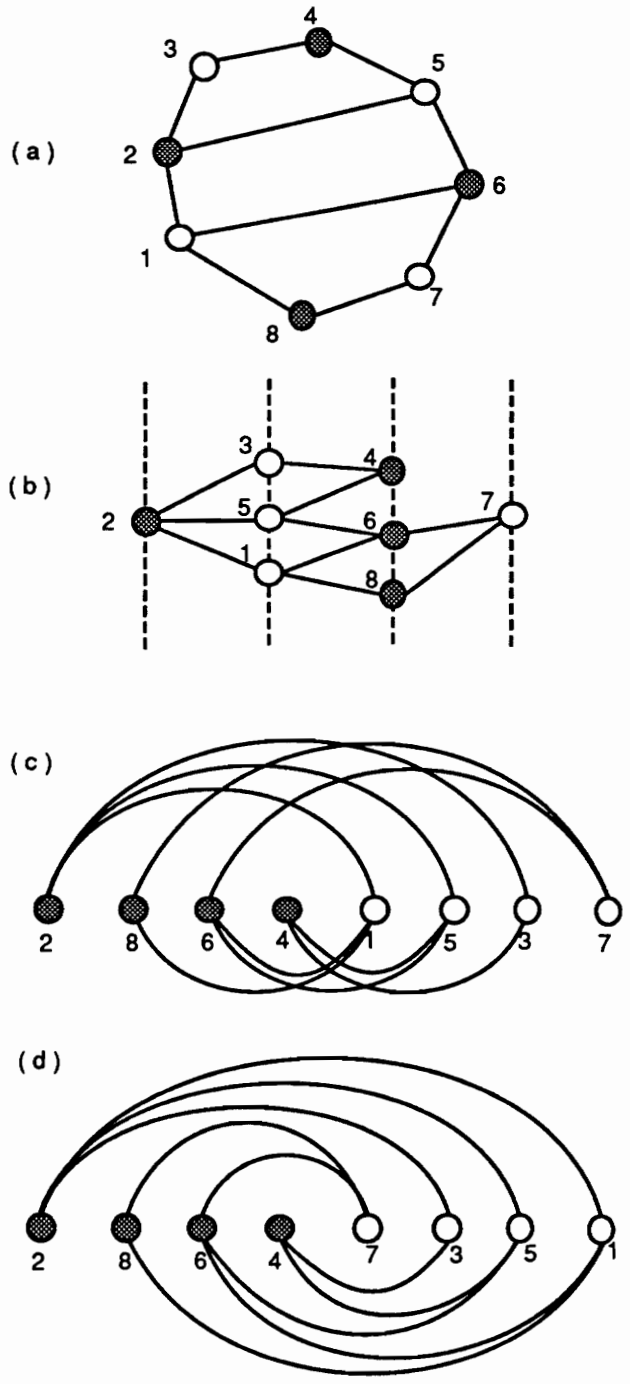


Figure 3.8: (a) An outerplanar embedding of a 1-stack bipartite graph $G = (V_1, V_2, E)$. (b) Leveled planar embedding of G. (c) 2-queue separated layout of G. (d) 2-stack separated layout of G.

a smaller result, the proof of which is similar to the proof of the Separation Theorem for queue layouts.

Theorem 3.18 *Suppose $G = (V, E)$ is a 1-stack graph and $\gamma = \{V_1, V_2\}$ is an arbitrary partition of V. Then G has a separated layout with respect to γ in 3 stacks.*

Thus the following question is open:

> Is there some constant c such that for every $k \geq 2$, every k-stack bipartite graph has a ck-stack separated layout with respect to the natural partition of its vertex set?

We do not know the answer to this question even for $k = 2$. The answer to this question has significant implications for an open question of Heath and Rosenberg [36]. If every k-stack bipartite graph has, for some constant c, a ck-stack separated layout with respect to the natural partition of its vertex set, then every k-stack graph has a $(c+1)k$-stack layout with respect to an arbitrary partition of size two. This would mean that for partitions of size two, separated stack layouts and separated queue layouts behave in a similar fashion. On the other hand, if for some k, the stacknumber of the family of separated layouts of k-stack bipartite graphs with respect to their natural partition, is $f(n)$ where f is an increasing function of n, then the queuenumber of the family of k-stack bipartite graphs with respect to the natural partition of their vertex sets is $\Omega(f(n))$. This follows immediately from Corollary 3.11. Thus the family of k-stack graphs would be an example of a family of graphs with arbitrarily large queuenumber as compared to the stacknumber. This result, in turn, would resolve the open question posed by Heath and Rosenberg [36] as to whether there is a family of graphs whose queue requirement is arbitrarily greater than its stack requirements.

3.4 Partitions of Arbitrary Size

So far in this chapter, we have concentrated on partitions of size 2. A whole new class of questions arises when we consider partitions of arbitrary size. We answer some of those

questions in this section. We show that the results in Section 3.2 for separated queue layouts can be extended to partitions of larger size in a natural manner. In contrast, we show that, for a partition of size 3 or greater, the stacknumber of any separated layout of a graph with respect to that partition may be arbitrarily large as compared to its stacknumber.

3.4.1 Separated Queue Layouts

In this section, we consider separated queue layouts for arbitrary partitions. Our first theorem provides an upper bound on the queuenumber of a separated layout of an m-partite graph with respect to the natural partition of its vertex set.

Theorem 3.19 *Suppose $G = (V_1, V_2, \ldots, V_m, E)$ is a k-queue m-partite graph and γ is the natural partition of its vertex set. Then G has a $2(m-1)k$-queue separated layout with respect to γ. Furthermore, if σ is a total order on $\cup_{i=1}^{m} V_i$ that yields a k-queue layout, then any separated order on $\cup_{i=1}^{m} V_i$ with respect to σ and γ yields a $2(m-1)k$-queue layout of G.*

Proof: Let σ be a total order on $\cup_{i=1}^{m} V_i$ that yields a k-queue layout of G. Consider any k-queue layout of G according to σ in which the edges are assigned to k queues $q_1, q_2, \ldots, q_k$. Let δ be any separated order on $\cup_{i=1}^{m} V_i$ with respect to σ and γ in which the blocks of the natural partition, namely $V_1, V_2, \ldots, V_m$, occur according to a permutation π of J_m. Lay out the vertices of G according to δ.

We obtain an assignment of the edges of G to $2(m-1)k$ queues from the assignment of the edges to $q_1, q_2, \ldots, q_k$ as follows. According to Theorem 3.4, the edges between vertices in V_i and V_j, $1 \leq i, j \leq m$, previously assigned to a queue q_p, $1 \leq p \leq k$, can be assigned to two queues when V_i and V_j are separated. Furthermore, edges between a pair of blocks V_{i_1} and V_{j_1} and another pair of blocks V_{i_2} and V_{j_2} do not nest if

$$|\pi^{-1}(i_1) - \pi^{-1}(j_1)| = |\pi^{-1}(i_2) - \pi^{-1}(j_2)|.$$

These observations suggest the following scheme for assigning edges to queues. Assign the edges previously assigned to queue q_p, $1 \leq p \leq k$, between a pair of sets V_i and V_j,

$1 \leq i,j \leq m$, to two queues, $q_{l,p}$ and $q'_{l,p}$ if $|\pi^{-1}(i) - \pi^{-1}(j)| = l$. Since, $1 \leq |i-j| \leq m-1$ and $1 \leq p \leq k$, we have an assignment of the edges of G to $2(m-1)k$ queues. □

Two observations are in order.

1. As shown in Section 3.2, the bound in Theorem 3.19 is tight for bipartite graphs ($m = 2$). Whether the bound is tight for $m > 2$ is unknown.

2. The proof of Theorem 3.19 uses the fact that the blocks are separated from each other in addition to the fact that vertices in each block are ordered according to σ. Thus Theorem 3.19 cannot be immediately extended to mingled layouts. Whether there exists an upper bound on the queuenumber of mingled layouts, similar to the one proved for separated layouts in Theorem 3.19, is unknown.

Our second theorem provides an upper bound on the queuenumber of a separated layout of an *arbitrary* graph with respect to an *arbitrary* partition of size m of its vertex set.

Theorem 3.20 *Suppose $G = (V,E)$ is a k-queue graph and $\gamma = \{V_1, V_2, \ldots, V_m\}$ is an arbitrary partition of its vertex set. Then G has a $(2m-1)k$-queue separated layout with respect to γ. Furthermore, if σ is a total order on V that yields a k-queue layout of G, then any separated order on V with respect to σ and γ yields a $(2m-1)k$-queue layout of G.*

Proof: Delete the edges between pairs of vertices in each set V_i, $1 \leq i \leq m$, to obtain an m-partite graph $G' = (V_1, V_2, \ldots, V_m, E')$. Let σ be a total order on V that yields a k-queue layout of G. Therefore, σ yields a k-queue layout of G'. Using Theorem 3.19, we conclude that any separated order on V with respect to σ and γ yields a $2(m-1)k$-queue layout of G'. Let δ be a separated order on V with respect to σ and γ. Lay out G' according to δ in $2(m-1)k$ queues. Add the edges deleted from G back to the layout of G'. These edges can be assigned to an additional k queues since σ yields a k-queue layout of G. Thus δ yields a $(2m-1)k$-queue layout of G. □

This result is a useful one and we use it to obtain results related to the queuenumber of a family of planar graphs in Chapter 6.

3.4.2 Separated Stack Layouts

In this section, we re-interpret a result of Chung, Leighton and Rosenberg [16] in the context of separated stack layouts. The *depth-n sum of triangles graph* $T(n) = (V, E)$ has vertex set

$$V = \{a_i, b_i, c_i \mid 1 \leq i \leq n\}$$

and edges connecting each triple a_i, b_i, and c_i into a triangle, that is,

$$E = \Big\{(a_i, b_i), (b_i, c_i), (c_i, a_i) \mid 1 \leq i \leq n\Big\}.$$

They consider separated layouts of $T(n)$ with respect to the partition

$$\gamma = \Big\{\{a_i \mid 1 \leq i \leq n\}, \{b_i \mid 1 \leq i \leq n\}, \{c_i \mid 1 \leq i \leq n\}\Big\}$$

and prove bounds on the stacknumber of these layouts.

Theorem 3.21 (Chung, Leighton, and Rosenberg [16]) *There is a separated layout of $T(n)$ with respect to γ that requires no more than $3n^{1/3}$ stacks and no separated layout of $T(n)$ with respect to γ requires fewer than $n^{1/3}$ stacks.*

The lower bound result stated in the above theorem has proved to be extremely useful in establishing a lower bound on the stack number of a family of graphs [35] called *n-dimensional ternary hypercubes.* This lower bound has revealed that n-dimensional ternary hypercubes is a family of graphs whose stacknumber grows arbitrarily large as compared to its queuenumber. Theorem 3.21 shows the existence of a 1-stack, tripartite graph whose separated layout with respect to the natural partition has unbounded stacknumber. This shows that for a partition of size 3 or greater, even for 1-stack graphs, it is not possible to prove constant upper bounds on the stacknumber of separated layouts with respect to that partition. This is in marked contrast with the fact that we have proved upper bounds on the queuenumber of separated layouts with respect to arbitrary partitions.

3.5 Conclusions

We have proved tight upper bounds on the queuenumber of separated and mingled layouts of graphs with respect to partitions of size 2. These bounds have helped us prove lower bound results for the queuenumber of families of dags (see Chapter 4), posets (see Chapter 5), and planar graphs (see Chapter 6). We have proved a tight upper bound on the stacknumber of separated layouts of a 1-stack graph with respect to partitions of size 2.

The following more general question remains open.

> Does there exist a constant upper bound on the stacknumber of separated layouts of k-stack graphs with respect to partitions of size two?

The answer has significant implications for contrasting the abilities of stacks and queues. Based on a result of Chung, Leighton, and Rosenberg [16], we conclude that for a partition of size 3 or greater, no constant upper bound is possible on the stacknumber of separated layouts with respect to that partition. This is in marked contrast with the constant upper bounds that we have proved on the queuenumber of separated and mingled layouts of graphs.

Chapter 4

Stack and Queue Layouts of Dags

Stack layouts and queue layouts of undirected graphs have appeared in a variety of contexts such as VLSI, fault-tolerant processing, process scheduling in parallel processing systems, and sorting networks. In Chapter 2, we show how the connection between queue layouts and covering a matrix with staircases forms the basis of an efficient scheme to perform matrix computations in parallel. In some applications of stack and queue layouts, it is more realistic to model the application domain with directed acyclic graphs (dags), rather than with undirected graphs. For example, Heath, Leighton, and Rosenberg [35] remark that queue layouts of undirected graphs idealize the problem of process scheduling in a parallel processing system. On the other hand, queue layouts and stack layouts of dags form a more realistic model of the problem. Various questions that have been asked about stack and queue layouts of undirected graphs acquire a new flavor due to the presence of directed edges (arcs). This is because the direction of the arcs imposes restrictions on the possible total orders on the node set of the graph that yield a valid layout.

The organization of this chapter is as follows. Section 4.1 contains definitions and notation related to stack and queue layouts of dags. Section 4.2 contains specific applications of stack and queue layouts of dags. In Section 4.3, we examine the stack and queue layouts of dags whose underlying undirected graphs are trees, unicyclic graphs, outerplanar graphs, and planar graphs. Section 4.4 presents a linear time algorithm for recognizing 1-stack dags, and Section 4.5 presents a linear time algorithm for recognizing leveled-planar dags. In Section 4.6, we show that the problem of recognizing a 4-queue dag and the problem of recognizing a 9-stack dag are both NP-complete. Section 4.7 contains open questions, conjectures, and conclusions.

4.1 Definitions and Notation

Let $G = (V, E)$ be an undirected graph. A *directing* of E is a function

$$\tau : E \rightarrow V \times V$$

such that $\tau(\{u, v\}) = (u, v)$ or $\tau(\{u, v\}) = (v, u)$. A *directing* of G is a directed graph $(V, \tau(E))$ such that τ is a directing of E. Hence a directing of G is any directed graph obtained by choosing an arbitrary direction for each edge of G. Since there are $2^{|E|}$ distinct functions τ, there are $2^{|E|}$ directings of G. If $(V, \tau(E))$ is a dag, then it is called an *acyclic directing* of G. An arbitrary or particular acyclic directing of G is denoted $\vec{G} = (V, \vec{E})$, and G is called the *covering graph* of $\vec{G}$. A *topological order* on $\vec{G}$ is a total order σ on V such that if $(u, v) \in \vec{E}$, then $u <_\sigma v$. A *k-stack layout* of $\vec{G}$ is a k-stack layout of G, according to a topological order on $\vec{G}$. The *stacknumber* of $\vec{G}$, denoted by $SN(\vec{G})$, is the smallest k such that $\vec{G}$ has a k-stack layout. $\vec{G}$ is a *k-stack dag* if $SN(\vec{G}) = k$. A *k-queue layout* of $\vec{G}$ is a k-queue layout of G, according to a topological order of $\vec{G}$. The *queuenumber* of $\vec{G}$, denoted by $QN(\vec{G})$, is the smallest k such that $\vec{G}$ has a k-queue layout. $\vec{G}$ is a *k-queue dag* if $QN(\vec{G}) = k$. The following example illustrates the above definitions.

Example 4.1. Figure 4.1 shows a dag $\vec{G}$ with 6 nodes. The total order $\sigma = 1, 2, 6, 3, 4, 5$ is a topological order on $\vec{G}$. Figure 4.2 shows a 2-stack layout of $\vec{G}$ according to σ. Figure 4.3 shows a 2-queue layout of $\vec{G}$ according to σ. It is easily seen that $\vec{G}$ does not have a 1-stack layout because every topological order on $\vec{G}$ contains a 2-twist. Neither does it have a 1-queue layout because every topological order on $\vec{G}$ contains a 2-rainbow. Thus $\vec{G}$ is a 2-stack, 2-queue dag. Note that a stack layout or a queue layout of a dag has the visual characteristic that all arcs of the dag proceed from left to right in the layout. □

Let $\mathcal{C}$ be a class of dags. The *stacknumber of* $\mathcal{C}$, denoted by $SN_{\mathcal{C}}(n)$, is a function of the natural numbers that equals the least upper bound of the stacknumber of all the dags in $\mathcal{C}$ with n nodes. Similarly, the *queuenumber of* $\mathcal{C}$, denoted by $QN_{\mathcal{C}}(n)$, is a function of the natural numbers that equals the least upper bound of the queuenumber of all the dags in

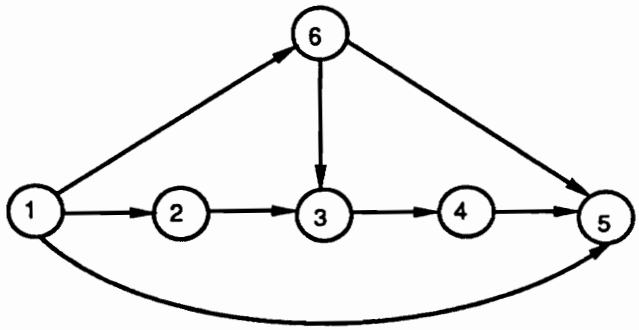


Figure 4.1: A 2-stack, 2-queue dag.

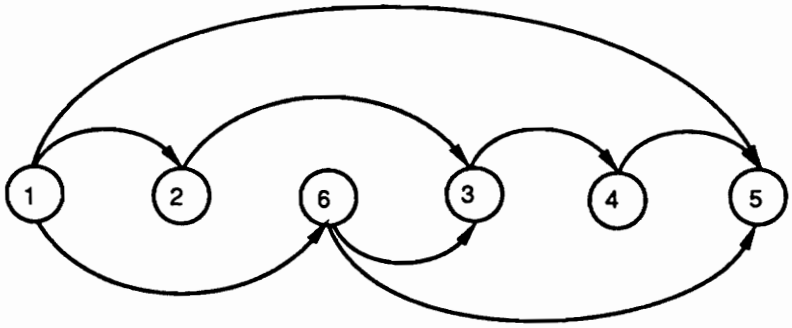


Figure 4.2: A 2-stack layout of the dag in Figure 4.1.

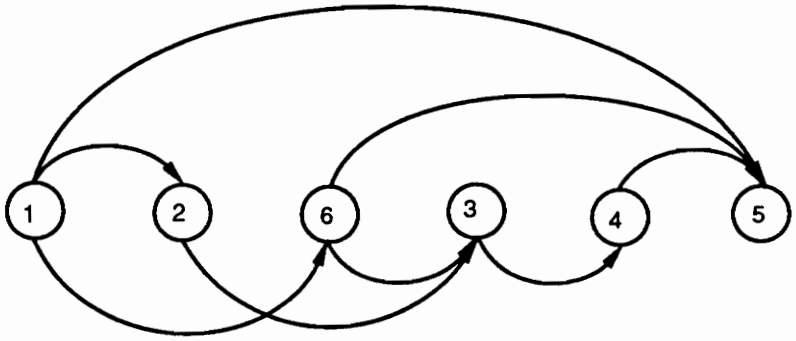


Figure 4.3: A 2-queue layout of the dag in Figure 4.1.

$\mathcal{C}$ with n nodes. We are interested in the asymptotic behavior of $SN_{\mathcal{C}}(n)$ and $QN_{\mathcal{C}}(n)$ for classes of dags classified based on the structure of their covering graphs.

4.2 Motivation and Applications

In this section, we elaborate on two specific applications of stack and queue layouts of dags. The first is process scheduling in a parallel processing system and the second is sorting with stacks and queues in parallel. In addition to these applications, we are also motivated by the fact that stack and queue layouts of dags form a basis for studying stack and queue layouts of posets (see Chapter 5).

4.2.1 Process Scheduling on a System of Parallel Processors

One of the motivations for investigating stack and queue layouts of dags is the problem of process scheduling on a system of parallel processors [56]. A process can be viewed as a set of computations that have certain interdependencies. Therefore, a process can be modeled by a dag, called the *dependency dag*, with nodes in this dag representing computations and arcs representing dependencies. Typically, in a system of parallel processors, computations are executed in some topological order of the dependency dag and each available processor grabs the waiting computation, next in queue. The fundamental advantage of a parallel processing system is that independent computations can be executed in parallel by different processors, thus reducing the total computation time required to execute the task. As the execution of the task progresses, the processors in the system need to exchange data as dictated by the interdependencies among the computations. This flow of data can be managed by a set of queues (or stacks) as explained below. Each arc (dependency) is assigned to a queue, with the following meaning. If arc (A, B) is assigned to queue q_j, then, when its execution is complete, computation A enqueues its outputs intended for computation B into q_j. When computation B is ready to begin execution, it dequeues from queue q_j all the inputs that it expects from A. For this scheme to be successful, it should

be the case that when B attempts to dequeue its inputs from q_j, the outputs of A are at the front of the queue. But, this is just another way of saying that the arcs (dependencies) assigned to each queue obey a first-in/first-out discipline. Thus the problem of determining an assignment of arcs (dependencies) to queues is simply the problem of finding a queue layout of the dependency dag. In a similar manner, the flow of data in a system of parallel processors can be managed by a set of stacks and determining an assignment of the arcs (dependencies) to stacks corresponds to finding a stack layout of the dependency dag.

4.2.2 Sorting with Stacks and Queues in Parallel

In general, layouts of dags are useful in situations in which we want to restrict our attention to only a subset of the possible total orders on the vertex set of the graph. For example, consider the problem of sorting with parallel stacks without complete loading [62,42,24,23]. The problem can be described as follows. We are given a system with k stacks and a permutation π of the integers $1, 2, \ldots, n$. The numbers $1, 2, \ldots, n$ enter the system, in that order. Each number that enters the system is immediately pushed onto some stack. There is no restriction on when numbers can be popped from stacks, but once a number is popped from a stack, it has to leave the system. The goal is to push and pop numbers from stacks so as to ensure that the numbers leave the system in the order prescribed by π. The problem is to determine the minimum k, for which this is possible. Note that some numbers may leave the system before all numbers have entered the system. In other words, the system need not be completely loaded. In Section 1.4, we introduce the problem of sorting with a network of stacks (queues) in parallel with and without complete loading. The problem of sorting with parallel stacks without complete loading is equivalent to the stack layout problem for the following dag. Define a dag $\vec{G} = (V, \vec{E})$ with nodes

$$V = \{a_1, a_2, \ldots, a_n\} \cup \{b_1, b_2, \ldots, b_n\}$$

and arcs

$$E = \{(a_i, a_{i+1}) \mid 1 \leq i \leq n-1\} \cup$$

$$\{(b_{\pi(i)}, b_{\pi(i+1)}) \mid 1 \leq i \leq n-1\} \cup$$
$$\{(a_i, b_i) \mid 1 \leq i \leq n\}.$$

Each node a_i signifies the entry of a number i into the system, while each node b_i signifies the departure of the number i from the system. Arcs between nodes signify temporal relationships. More precisely, arcs of the type (a_i, a_{i+1}) signify the order in which the numbers enter the system, $(b_{\pi(i)}, b_{\pi(i+1)})$ signify the order in which numbers should leave the system, and (a_i, b_i) signify that a number can leave the system only after it has entered the system. Any layout of $\vec{G}$ has to obey the temporal restrictions imposed by the arcs and if two arcs (a_i, b_i) and (a_j, b_j) cross in the layout, it implies that the numbers i and j cannot be pushed into the same stack. Hence, the minimum number of stacks to which arcs of the form (a_i, b_i) can be assigned, in any stack layout of $\vec{G}$, is equal to the minimum number of stacks required to sort π without complete loading.

Example 4.2. Let $n = 10$ and let

$$\pi = (4, 8, 9, 10, 2, 6, 7, 3, 5, 1).$$

To determine how many stacks are required to sort the above permutation in parallel, without complete loading we construct the dag $\vec{G} = (V, \vec{E})$ where

$$\begin{aligned} V &= \{a_i \mid 1 \leq i \leq 10\} \cup \{b_i \mid 1 \leq i \leq 10\}, \\ \vec{E} &= \{(a_i, b_i) \mid 1 \leq i \leq 10\} \cup \\ &\quad \{(a_i, a_{i+1}) \mid 1 \leq i \leq 9\} \cup \\ &\quad \{(b_{\pi(i)}, b_{\pi(i+1)}) \mid 1 \leq i \leq 9\} \end{aligned}$$

Figure 4.4 shows a 3-stack layout of $\vec{G}$ in which the arcs between the nodes in the set $\{a_i \mid 1 \leq i \leq 10\}$ and those between the nodes in the set $\{b_{\pi(i)} \mid 1 \leq i \leq 10\}$ are excluded. The plain arcs shown above the line of vertices are all assigned to one stack; the dashed arcs shown above the line of vertices are all assigned to a second arc; the arcs

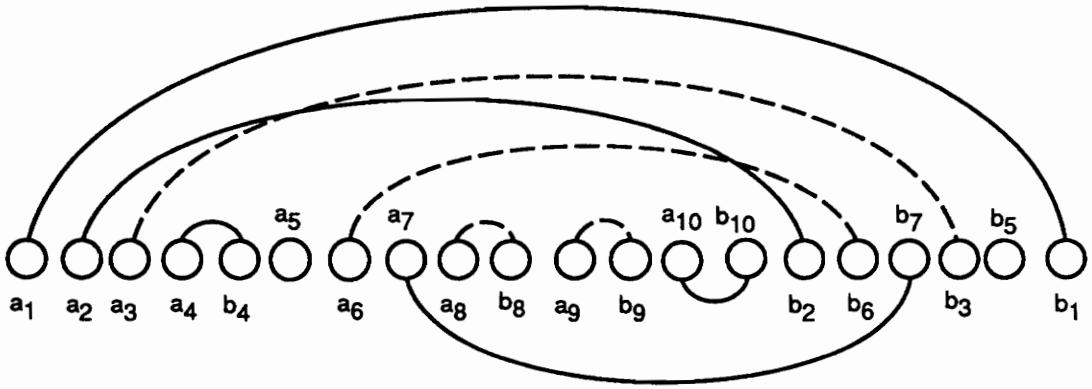


Figure 4.4: The 3-stack layout of $\vec{G}$ showing that π can be sorted with three stacks in parallel without complete loading.

drawn below the line of vertices are all assigned to a third arc. The 3-stack layout shown in Figure 4.4 corresponds to sorting π with 3 stacks in parallel without complete loading. □

4.3 Layouts of Specific Families of Dags

In this section, we focus on four classes of dags classified according to the structure of their covering graphs and study the asymptotic behavior of their stacknumber and queuenumber. We first consider the class of dags whose covering graphs are trees and show that its stacknumber is 1 and its queuenumber is 2. We next consider the class of dags whose covering graphs are unicylic and show that its stacknumber is 2 and its queuenumber is also 2. We then provide an example of a class of dags, with outerplanar covering graphs, whose stacknumber is 1, but whose queuenumber is $\Omega(n)$. We conclude the section, with an example of a family of dags, with planar covering graphs, whose stacknumber is $\Omega(n)$, but whose queuenumber is 2.

4.3.1 Directed Trees

Suppose that $T = (V, E)$ is a tree. An acyclic directing of T, $\vec{T} = (V, \vec{E})$ is called a *directed tree*. In this section, we show that $SN(\vec{T}) \leq 1$ and $QN(\vec{T}) \leq 2$. We then provide an example of a 2-queue directed tree with 7 nodes, and conclude that if $\mathcal{T}$ is the class of directed trees, then $SN_{\mathcal{T}}(n) = 1$ for all $n \geq 2$ and $QN_{\mathcal{T}}(n) = 2$ for all $n \geq 7$.

Let $P = (V, E)$ be a graph with $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{(v_i, v_{i+1}) \,|\, 1 \leq i \leq n-1\}$. Thus, P is a path of length $n - 1$. An acyclic directing of P, $\vec{P} = (V, \vec{E})$, is called a *directed path*. Let τ be the directing of E such that $\vec{E} = \tau(E)$. We first show that $SN(\vec{P}) = QN(\vec{P}) = 1$. Our proof illustrates techniques that are common to the proofs in this section. Designate one of the two ends of $\vec{P}$, say v_1, as the *origin* of $\vec{P}$. Define the *polarity* of the arcs of $\vec{P}$ by the function $p : \vec{E} \rightarrow \{-1, +1\}$:

$$p(e) = \begin{cases} -1 & \text{if } e = \tau(\{v_{i-1}, v_i\}) = (v_i, v_{i-1}), \\ 1 & \text{if } e = \tau(\{v_{i-1}, v_i\}) = (v_{i-1}, v_i). \end{cases}$$

Thus, arcs directed away from the origin have a positive polarity, while those directed towards the origin have a negative polarity. Let J be the set of integers. Define *distance* from the origin of each node in V by the function $d : V \rightarrow J$:

$$d(v_k) = \sum_{i=2}^{k} p\Big(\tau(\{v_{i-1}, v_i\})\Big).$$

If $d(v) = i$, then v is called a *distance-i* node. Let D_i denote the set of distance-i nodes. The following theorem establishes the stacknumber and queuenumber of the class of directed paths.

Lemma 4.3 *Let $\mathcal{P}$ be the class of directed paths. Then*

$$SN_{\mathcal{P}}(n) = QN_{\mathcal{P}}(n) = 1$$

for all $n \geq 2$.

Proof: Let $\vec{P} = (V, \vec{E})$ be a directed path with covering graph $P = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{(v_i, v_{i+1}) \,|\, 1 \leq i < n\}$. We first prove that $SN(\vec{P}) \leq 1$. By

induction on n, the number of nodes in V, we show that $\vec{P}$ has a 1-stack layout with its origin occurring either as the first node or as the last node in the layout.

Base Case: For $n = 1$, the claim is trivially true.

Induction Case: For $n > 1$ assume that any directed path on $n-1$ nodes has a 1-stack layout with its origin occurring either as the first node or as the last node in the layout. Without loss of generality, assume that v_1 is the origin of $\vec{P}$. Let $\vec{P}_1$ be the directed subgraph of $\vec{P}$ induced by $\{v_2, v_3, \ldots, v_n\}$. Then, $\vec{P}_1$ is a directed path on $n-1$ nodes. Let v_2 be the origin of $\vec{P}_1$. By the induction hypothesis, $\vec{P}_1$ has a 1-stack layout with v_2 appearing either as the first or as the last node in the layout. Consider such a layout and add v_1 to it in the following manner. If $(v_1, v_2) \in \vec{E}$, then place v_1 to the left of all other nodes and if $(v_2, v_1) \in \vec{E}$, then place v_1 to the right of all other nodes in the layout. In either case, the new arc does not intersect any of the other arcs in $\vec{P}_1$. Thus we have a 1-stack layout of $\vec{P}$ in which the origin, v_1, occurs either as the first node or as the last node in the layout.

We now show that $QN(\vec{P}) = 1$. Let δ be the total order $v_1, v_2, \ldots, v_n$ on V. Let $\gamma = \{D_1, D_2, \ldots,\}$ be a partition of V into blocks such that each block contains nodes that are equidistant from the origin. Let σ be a separated order on V with respect to δ and γ such that nodes in block D_i occur before nodes in block D_j if $i < j$. σ is a topological order on $\vec{P}$ because every arc in $\vec{P}$ is from a distance-$(k-1)$ node to a distance-k node for some k. To check that σ yields a 1-queue layout of $\vec{P}$, we only have to check that no two arcs (v_i, v_j) and (v_p, v_q) where $v_i, v_p \in D_{k-1}$ and $v_j, v_q \in D_k$, nest. If $v_i <_\sigma v_p$, then $i < p$. This is because nodes in each block D_k occur according to δ. But, $i < p$ implies that $j \leq q$ which in turn implies that either $v_j = v_q$ or $v_j <_\sigma v_q$. A similar argument shows that if $v_p <_\sigma v_i$ then, either $v_q = v_j$ or $v_q <_\sigma v_j$. In either case (v_i, v_j) and (v_p, v_q) do not nest. □

The next result establishes the stacknumber and the queuenumber of the class of directed trees. If $T = (V, E)$ is a tree, let τ be an arbitrary directing of E, and let $\vec{T} = (V, \tau(E))$ be the corresponding directing of T. Choose an arbitrary node r to be the root of T and of $\vec{T}$. The *polarity* $p(e)$ of an arc e in $\vec{T}$ is defined as: $p(e) = +1$ if e is directed away from

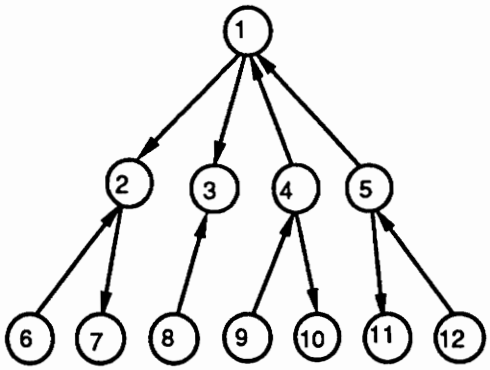


Figure 4.5: A directed tree.

r, otherwise $p(e) = -1$. For any node $v \in V$ in T, there is a unique path in T from r to v, consisting, say, of edges $e_1, e_2, \ldots, e_k$. The *distance* of v from the root r is

$$d(v) = \sum_{i=1}^{k} p(e_i).$$

If $d(v) = i$ then, v is called a distance-i node and the set of all distance-i nodes is denoted by D_i. We use $\vec{T}[v]$ to denote the unique directed subtree rooted at v. Note that $\vec{T}[r] = \vec{T}$. Denote the nodes in $\vec{T}[v]$ by $V[v]$. If $(u, v) \in \vec{E}$, then u is the *in-neighbor* of v, and v is the *out-neighbor* of u.

Theorem 4.4 *Let $\mathcal{T}$ be the class of directed trees. Then*

$$SN_{\mathcal{T}}(n) = 1$$

for all $n \geq 2$ and

$$QN_{\mathcal{T}}(n) = 2$$

for all $n \geq 7$.

Proof: Let $\vec{T} = (V, \vec{E})$ be a directed tree with root r and $|V| = n$. Let $u_1, u_2, \ldots, u_l$ be in-neighbors of r and let $v_1, v_2, \ldots, v_m$ be the out-neighbors of r. We first prove by induction on n that $\vec{T}$ can be laid out in 1 stack with r *exposed.* A node $v \in V$ is exposed in a layout,

if there is no arc in the layout whose tail is to the left of v and whose head is to the right of v.

Base case: When $n = 1$, the claim is trivially true.

Induction Case: Suppose $n > 1$ and every directed tree with $n - 1$ or fewer nodes can be laid out in 1 stack with its root exposed. The induction hypothesis implies that each of the directed subtrees

$$\vec{T}[u_1], \vec{T}[u_2], \ldots, \vec{T}[u_l], \vec{T}[v_1], \vec{T}[v_2], \ldots, \vec{T}[v_m]$$

has a 1-stack layout with its root exposed. Concatenate these layouts in the order suggested above:

$$\vec{T}[u_1], \vec{T}[u_2], \ldots, \vec{T}[v_m].$$

Place r between the layouts of $\vec{T}[u_l]$ and $\vec{T}[v_1]$. Since all the neighbors of r are exposed, none of the arcs incident to r intersect any of the other arcs. Furthermore, there is no arc in the layout whose tail is to the left of r and whose head is to the right of r and thus r is exposed in the layout of $\vec{T}$. This shows that if $\mathcal{T}$ is class of directed trees, then $SN_{\mathcal{T}}(n) = 1$ for all $n \geq 2$.

We now show that $QN(\vec{T}) \leq 2$. Let δ be a total order on V obtained by doing a breadth-first search on the vertices of T starting at r. Let γ be the partition of V defined by

$$\gamma = \{\{r\}, V[u_1], V[u_2], \ldots, V[u_\ell], V[v_1], V[v_2], \ldots, V[v_m]\}.$$

Let σ be a total order on V with respect to δ and γ such that the nodes in D_i occur to the left of the nodes in D_j if $i < j$. Lay out the nodes in V according to σ and assign arcs with a positive polarity to queue q_1 and arcs with a negative polarity to queue q_2. We show that no two arcs assigned to q_1 nest. The argument to show that no two arcs assigned to q_2 nest, is similar. Each arc assigned to q_1 is from a distance-$(k-1)$ node to a distance-k node for some integer k. Let $e_1 = (v_i, v_j)$ and $e_2 = (v_p, v_q)$ be a pair of arcs assigned to q_1 such that $v_i, v_j \in D_{k-1}$ and $v_p, v_q \in D_k$. In any breadth-first search of $\vec{T}$, the nodes v_i and v_p are traversed in the same order as the nodes v_j and v_q. This implies that the arcs (v_i, v_j)

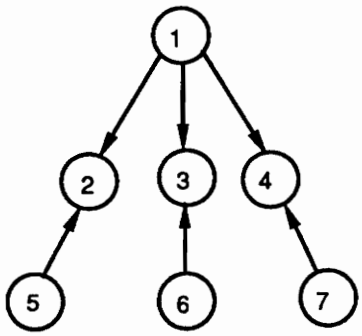


Figure 4.6: A 2-queue directed tree.

and (v_p, v_q) do not nest. Hence, $QN(\vec{T}) \leq 2$. Figure 4.6 gives an example of a directed tree with 7 nodes that requires 2 queues. This shows that if $\mathcal{T}$ is the class of directed tree, then $QN_{\mathcal{T}}(n) = 2$ for all $n \geq 7$. □

We illustrate the stack layout and the queue layout described in the above proof with an example.

Example 4.5. Let $\vec{T}$ be the directed tree shown in Figure 4.5 and let node 1 be its root. The in-neighbors of 1 are 4 and 5 and the out-neighbors of 1 are 2 and 3. The directed subtrees $\vec{T}[5]$, $\vec{T}[4]$, $\vec{T}[2]$, and $\vec{T}[3]$ are laid out in 1-stack each in a separated fashion as shown in Figure 4.7. The root is placed between the layouts of subtrees $\vec{T}[4]$ and $\vec{T}[2]$ to give a 1-stack layout of $\vec{T}$. Note that 1 is exposed in the layout.

The sets of nodes equidistant from the root are:

$$D_{-2} = \{9, 12\} \qquad D_{-1} = \{4, 5\}$$
$$D_0 = \{4, 5\} \qquad D_2 = \{7\}$$
$$D_0 = \{1, 6, 8, 10, 11\}$$

The total order $\sigma = 1, 2, \ldots, 12$ can be obtained by doing a breadth-first search on $\vec{T}$ starting at the root. The total order shown in Figure 4.8 is a separated order with respect to σ and the partition $\{D_{-2}, D_{-1}, D_0, D_1, D_2\}$ such that the blocks of the partition occur in the order D_{-2}, D_{-1}, D_0, D_1, and D_2. The arcs with positive polarity are assigned to q_1 and are drawn above the layout, while the arcs with negative polarity are assigned to q_2 and are

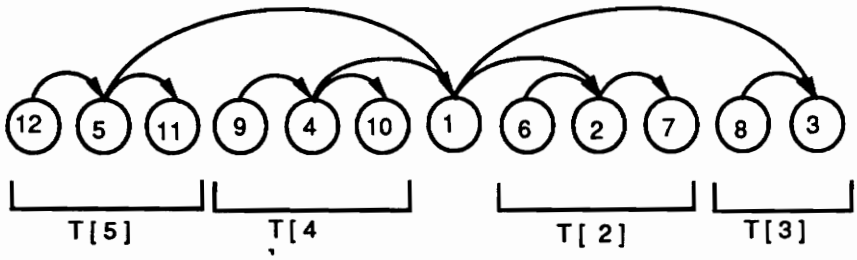


Figure 4.7: A 1-stack layout of the directed tree shown in the Figure 4.5.

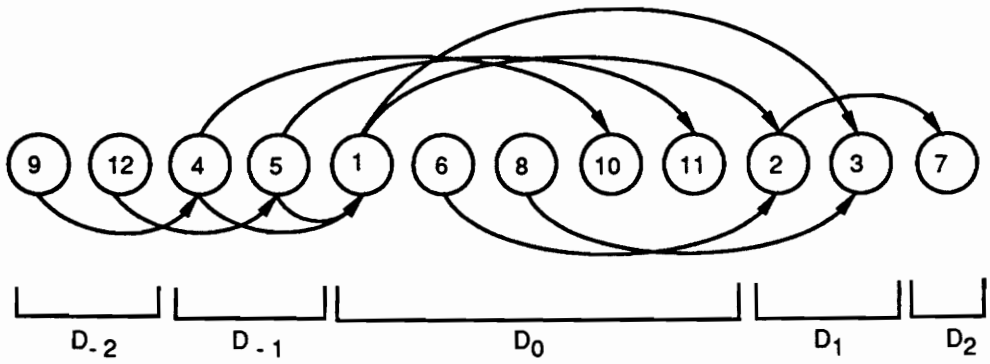


Figure 4.8: A 2-queue layout of the directed tree shown in Figure 4.5.

drawn below the nodes. □

An algorithmic question motivated by Theorem 4.4 is whether 1-queue directed trees can be efficiently recognized. In Section 4.5, we provide an algorithm that takes as input a directed tree $\vec{T} = (V, \vec{E})$ and determines whether $\vec{T}$ is a 1-queue dag in $O(|V|)$ time.

4.3.2 Unicyclic Dags

A *unicyclic graph* is a graph that contains exactly one cycle. If $U = (V, E)$ is a unicyclic graph, then any acyclic directing of U, $\vec{U} = (V, \vec{E})$ is called a *unicyclic dag*. In this subsection, we show that if $\mathcal{U}$ is the class of unicyclic dags, then

$$SN_{\mathcal{U}}(n) = QN_{\mathcal{U}}(n) = 2$$

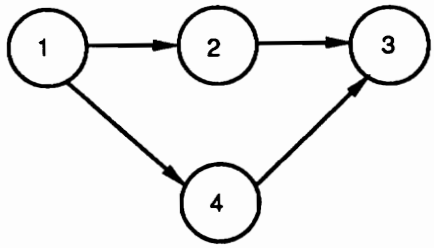


Figure 4.9: A directed cycle that requires 2 stacks.

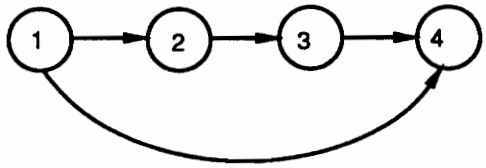


Figure 4.10: A directed cycle that requires 2 queues.

for all $n \geq 4$. First we address the stacknumber and queuenumber of a simpler class of dags. If $C = (V, E)$ is a cycle, then any acyclic directing of C, $\vec{C} = (V, \vec{E})$ is called a *directed cycle*. The following simple argument suffices to show that $SN(\vec{C}) \leq 2$ and $QN(\vec{C}) \leq 2$. Let v_0 be a source in $\vec{C}$. Delete v_0 from $\vec{C}$ to obtain a directed path $\vec{P}$ that can be laid out in 1 stack (1 queue) as shown in Lemma 4.3. Place v_0 to the left of the layout of $\vec{P}$ and assign the arcs incident on v_0 to a second stack (or queue) to obtain a 2-stack (2-queue) layout of $\vec{C}$. Figure 4.9 shows a directed cycle that cannot be laid out in 1 stack and Figure 4.10 shows a directed cycle that cannot be laid out in 1 queue. So we have the following proposition.

Proposition 4.6 *Let $\mathcal{C}$ be the class of directed cycles. Then*

$$SN_{\mathcal{C}}(n) = QN_{\mathcal{C}}(n) = 2$$

for all $n \geq 4$.

The above proposition motivates the algorithmic questions, whether 1-stack directed cycles and 1-queue directed cycles can be recognized efficiently. In Section 4.4, we provide an

algorithm that determines whether $\vec{C} = (V, \vec{E})$ is a 1-stack dag in $O(|V|)$ time. In Section 4.5, we provide an algorithm that determines whether $\vec{C} = (V, \vec{E})$ is a 1-queue dag in $O(|V|)$ time.

We now extend Proposition 4.6 to the class of unicylic dags.

Theorem 4.7 *Let $\mathcal{U}$ be the class of unicyclic dags. Then*

$$SN_{\mathcal{U}}(n) = QN_{\mathcal{U}}(n) = 2$$

for all $n \geq 4$.

Proof: Suppose $\vec{U} = (V, \vec{E})$ is a unicyclic dag. Let $\vec{C}$ be the subgraph of $\vec{U}$ that is a directed cycle. Let the node set of the corresponding undirected cycle C be $V_1 = \{v_0, v_1, \ldots, v_{n-1}\}$, where v_0 is a source, and let its arc set be

$$\{(v_i, v_{i+1}) \mid 1 \leq i \leq n-2\} \cup \{(v_{n-1}, v_0)\}.$$

We first describe a 2-stack layout of $\vec{U}$. Delete arcs (v_0, v_1) and (v_0, v_{n-1}) from $\vec{U}$ to obtain two disjoint directed trees called $\vec{T_1}$ and $\vec{T_2}$ such that v_0 is a node in $\vec{T_1}$. By Theorem 4.4, $\vec{T_1}$ and $\vec{T_2}$ can be laid out in 1 stack each. Place the 1-stack layout of $\vec{T_1}$ to the left of a 1-stack layout of $\vec{T_2}$ and add the arcs (v_0, v_1) and (v_0, v_{n-1}) to the layout, assigning them to a second stack. This gives a 2-stack layout of $\vec{U}$.

The above strategy for obtaining a 2-stack layout of $\vec{U}$, can be used to obtain a 3-queue layout for $\vec{U}$ because any directed tree can be laid out in 2-queues. But, a more careful choice of the topological order on $\vec{U}$ leads to a 2-queue layout.

A 2-queue layout of $\vec{U}$ is constructed in two stages. In the first stage, $\vec{C}$ is laid out in 2 queues such that exactly one arc is assigned to queue q_2 and the rest of the arcs are assigned to queue q_1. In the second stage, the directed trees rooted at the nodes in $\vec{C}$ are added to the layout in a manner so as to produce a 2-queue layout of $\vec{U}$.

Stage 1: Let $\vec{P_1}$ be the directed path obtained by deleting (v_0, v_1) and (v_0, v_{n-1}) from $\vec{C}$. Designate v_1 as the origin of $\vec{P_1}$ and assign distances to all the nodes in $\vec{P_1}$ (the terminology

of directed paths is defined in Section 4.3.1). Let $v_{n-1} \in D_{t'}$ for some integer t'. Depending on the sign of t' there are two cases.

1. $t' \geq 0$. In this case, let $\vec{P}$ be the directed path obtained by deleting the arc (v_0, v_{n-1}) from $\vec{C}$. Using v_0 as the origin, assign distances to the nodes in $\vec{P}$ and lay out $\vec{P}$ in 1-queue as described in the proof of Lemma 4.3. Let $d(v_{n-1}) = t$. Note that $t \geq d(v_1) = d(v_0) + 1$ and thus v_{n-1} occurs to the right of v_0. The arc (v_0, v_{n-1}) is added back to the layout and assigned to queue q_2.

2. $t' < 0$. In this case, let $\vec{P}$ be the directed path obtained by deleting the arc (v_0, v_1) from $\vec{C}$. Using v_0 as the origin, assign distances to the nodes in $\vec{P}$ and lay out $\vec{P}$ in 1-queue as described in the proof of Lemma 4.3. Let $d(v_1) = t$. Note that $t > d(v_{n-1}) = d(v_0) + 1$ and thus v_1 occurs to the right of v_0. The arc (v_0, v_{n-1}) is added back to the layout and assigned to queue q_2.

For each integer i, such that $D_i \neq \emptyset$, let *block-i* $= D_i$. Thus we have a 2-queue layout of $\vec{C}$ in which nodes belonging to block-i appear contiguously and arcs assigned to q_1 are from nodes in block-k to nodes in block-$(k+1)$ and the lone arc assigned to q_2 is from a node in block-0 to a node in block-t. This completes the first stage of our construction.

Stage 2: Having laid out $\vec{C}$ in 2 queues, we now focus on the rest of the dag. Consider the directed subgraph of $\vec{U}$ induced by the set of nodes in $V - (V_1 - \{v_i\})$ for some node $v_i \in V_1$. This directed subgraph may have several connected components, each of which is a directed tree. Denote the directed tree that v_i belongs to by $\vec{T}[v_i]$. Designate v_i as its root and assign a polarity to each of its arcs and a distance to each of its nodes. Denote the node set of $\vec{T}[v_i]$ by $V[v_i]$. Let σ_i be a total order on $V[v_i]$ obtained by doing a breadth-first search of $\vec{T}[v_i]$ starting at v_i. Let σ be a separated order on V with respect to the partition

$$\{V[v_0], V[v_1], \ldots, V[v_{n-1}]\}$$

in which nodes in $V[v_i]$ occur according to the total order σ_i. The nodes in $V - V_1$ are added to the layout of $\vec{C}$ one by one according to σ, in a manner so as to satisfy the following inductive hypothesis.

Inductive hypothesis: Let $V_2 \subseteq V - V_1$ be the be the set of nodes that have been added to the layout of $\vec{C}$. Then we have a two queue layout of the directed subgraph of $\vec{U}$ induced by $V_1 \cup V_2$ such that

1. Nodes belonging to each set block-i, appear contiguously in the layout.
2. Arcs assigned to queue q_1 are from nodes in block-k to nodes in block-$(k+1)$ for some integer k.
3. Arcs assigned to q_2 have one of the following three forms:
   (1) From nodes in block-0 to nodes in block-j for all j, $1 \leq j \leq t$.
   (2) From nodes in block-j to nodes in block-$(t+1)$ for all j, $1 \leq j \leq t$.
   (3) From nodes in block-k to nodes in block-$(k+1)$ for all $k \notin \{0, 1, \ldots, t-1, , t\}$.

Base case: $V_2 = \emptyset$. Clearly, the 2-queue layout of $\vec{C}$ constructed in stage 1 satisfies the induction hypothesis.

Inductive case: Assuming that the induction hypothesis is true for some $V_2 \subseteq V - V_1$, we show that it is also true for $V_2 \cup \{v\} \subseteq V - V_1$ where v is the next node to be added, according to σ. Assume that v belongs to $\vec{T}[v_i]$ for some i, $0 \leq i \leq n-1$ and let u be the parent of v in $\vec{T}[v_i]$. Since u occurs before v in σ, u has already been laid out. There are two cases depending on whether $(u, v) \in \vec{E}$ or $(v, u) \in \vec{E}$.

Case 1: If $(u, v) \in \vec{E}$ and u belongs to block-k, then add v to block-$(k+1)$. Place v in the layout so that the nodes in block-$(k+1)$ remain contiguous and (u, v) does not nest with any of the arcs assigned to q_1 that go from block-k to block-$(k+1)$. Assign (u, v) to queue q_1. Note that the induction hypothesis is satisfied.

Case 2: Suppose that $(v, u) \in \vec{E}$ and u belongs to block-k for some integer k. If $k = t+1$, then place v in the layout so that is the neighbor of some node in block-j for some $j \in \{1, 2, \ldots, t\}$ and (v, u) does not nest with any arc assigned to q_2. Add v to block-j. If $k \in \{1, 2, \ldots, t\}$, then add u to block-0 and place v in the layout so that (v, u)

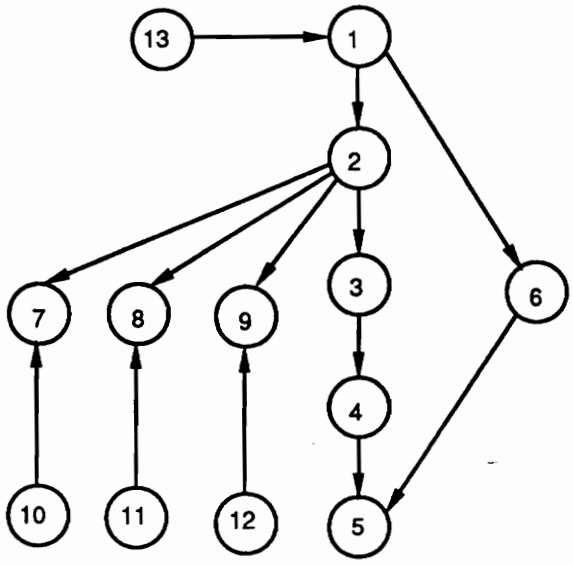


Figure 4.11: A directed unicyclic graph.

does not nest with any arc assigned to q_2 and the block-0 nodes appear contiguously. Otherwise, add u to block-$(k-1)$ and place v in the layout such that (v,u) does not nest with any arc in q_2 and the block-$(k-1)$ nodes appear contiguously in the layout. In all the three cases, assign (v,u) to queue q_2. Note that the inductive hypothesis is satisfied by the new layout and thus $\vec{U}$ has a 2-queue layout.

□

We illustrate the construction of the 2-stack layout and the 2-queue layout described in Theorem 4.7 with an example.

Example 4.8. Figure 4.11 shows directed unicyclic graph $\vec{U}$ with 13 nodes. $\vec{U}$ contains a directed cycle $\vec{C}$ induced by the nodes in $\{1,2,3,4,5,6\}$. Node 1 is the only source in $\vec{C}$ and to obtain a 2-stack layout of $\vec{U}$, we delete the arcs $(1,6)$ and $(1,2)$ from $\vec{U}$. The result is two directed trees, $\vec{T}_1$ and $\vec{T}_2$ such that the nodes in $\vec{T}_1$ are $\{1,13\}$ and the rest of the nodes in $\vec{U}$ belong to $\vec{T}_2$. Figure 4.12 shows a 2-stack layout of $\vec{U}$, which consists of a 1-stack layout of $\vec{T}_1$ followed by a 1-stack layout of $\vec{T}_2$. The two arcs $(1,2)$ and $(1,6)$ are then added back to the layout and assigned to a second stack. Figure 4.13 shows a 2-queue layout of $\vec{U}$.

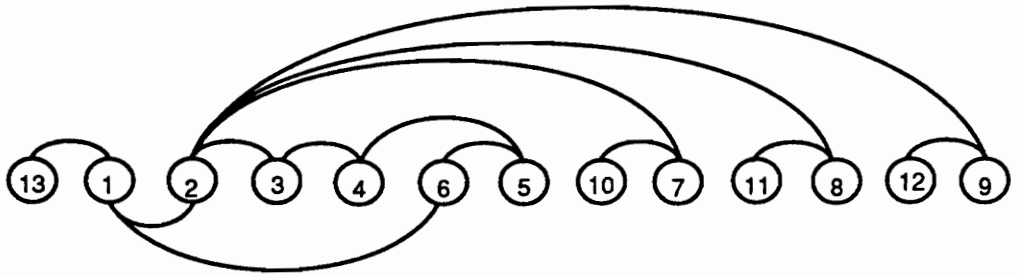


Figure 4.12: A 2-stack layout of the directed unicyclic graph shown in Figure 4.11.

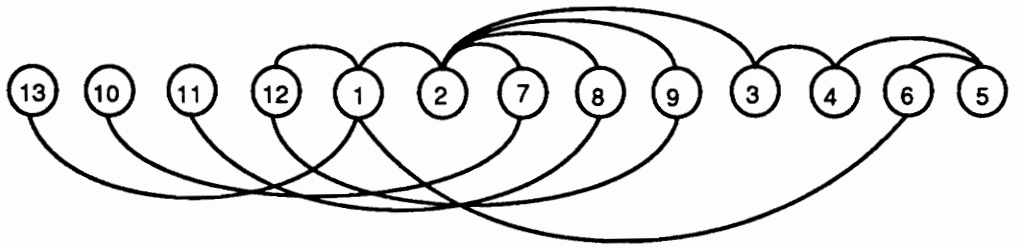


Figure 4.13: A 2-queue layout of the directed unicyclic graph shown in Figure 4.11.

The directed cycle $\vec{C}$ is laid out in 2 queues, with one arc $(1,6)$ assigned to queue q_2 and the rest of the arcs are assigned to queue q_1. The arcs in q_1 are shown above the layout while the arcs in q_2 are shown below. Note that arc $(1,6)$ is from block-0 to block-3, while the rest of the arcs are from block-k to block-$(k+1)$ for some k. The two directed trees $\vec{T}[1]$ and $\vec{T}[2]$ are added to the layout of $\vec{C}$, as shown in Figure 4.13, without adding to the queue requirements of the dag. □

4.3.3 Outerplanar Dags

An *outerplanar dag* is any acyclic directing of an outerplanar graph. Outerplanar graphs can be laid out in 1 stack (see Bernhart and Kainen [6]) and in 2 queues (see Heath, Leighton, and Rosenberg [35]). In contrast, it is easy to construct an example of an outerplanar dag

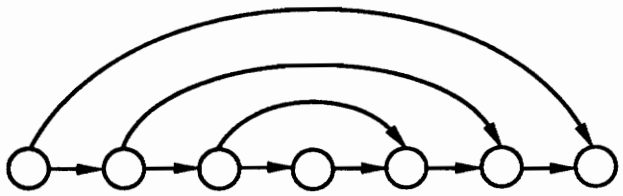


Figure 4.14: The outerplanar dag $\vec{A}(7)$.

whose queuenumber is arbitrarily large as compared to its stacknumber.

Let $\mathcal{A} = \{\vec{A}(n) \mid n \geq 1\}$ be a class of outerplanar dags, where each dag $\vec{A}(n)$ has node set $\{v_1, v_2, \ldots, v_n\}$ and arc set

$$\{(v_i, v_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(v_i, v_{n-i+1}) \mid 1 \leq i \leq \lfloor n/2 \rfloor\}.$$

Figure 4.14 shows $\vec{A}(7)$. $\vec{A}(n)$ has a unique topological order given by the directed hamiltonian path $v_1, v_2, \ldots, v_n$. When laid out according to this topological order, it requires 1 stack and $\lfloor n/2 \rfloor$ queues. Thus we have the following proposition.

Proposition 4.9 *If $\mathcal{D}$ is the class of outerplanar dags, then*

$$QN_{\mathcal{D}}(n) = \theta(n).$$

Since we know that the queuenumber of outerplanar graphs is 2 (see Heath, Leighton, and Rosenberg [35] for results that show that every 1-stack graph is a 2-queue graph and that every 1-queue graph is a 2-stack graph), we obtain the contrast expressed in the following corollary.

Corollary 4.10 *Let $\mathcal{O}$ be the class of outerplanar graphs and let $\mathcal{D}$ be the class of outerplanar dags. Then,*

$$\frac{QN_{\mathcal{D}}(n)}{QN_{\mathcal{O}}(n)} = \theta(n).$$

We do not know the stacknumber of outerplanar dags and leave it as an open question. Nevertheless, we present a class of outerplanar dags

$$\mathcal{B} = \{\vec{B}(n) \mid n = 2^m + 1, m \geq 0\}$$

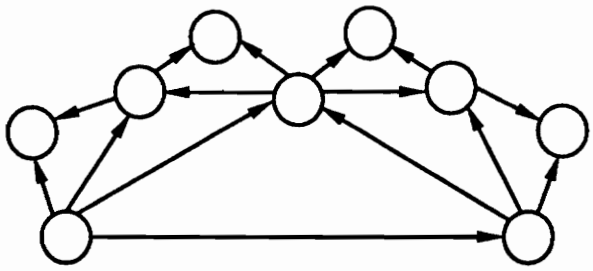


Figure 4.15: The outerplanar dag $\vec{B}(9)$.

and show that $SN_{\mathcal{B}}(n) \leq 3$ for all $n = 2^m + 1$ where $m \geq 0$. We conjecture that the inductive proof used to establish the above result can be extended to show that the stacknumber of outerplanar dags is also bounded above by a constant. Each outerplanar dag, $\vec{B}(n) \in \mathcal{B}$ is described as follows. $\vec{B}(2)$ contains two nodes and a single arc, called its *base arc*. $\vec{B}(2n-1)$ is constructed from $\vec{B}(n)$ as follows. Let $\vec{B}_1$ and $\vec{B}_2$ be two copies of $\vec{B}(n)$. Let (u_1, v_1) be the base arc in $\vec{B}_1$ and let (u_2, v_2) be the base arc in $\vec{B}_2$. $\vec{B}(2n-1)$ is constructed from $\vec{B}_1$ and $\vec{B}_2$ by identifying v_1 and v_2, and adding the arc (u_1, u_2), which is the base arc of $\vec{B}(2n-1)$. The node $v_1 = v_2$ is called the *apex* of $\vec{B}(2n-1)$; $\vec{B}_1$ is called the *left dag* of $\vec{B}(2n-1)$; $\vec{B}_2$ is called the *right dag* of $\vec{B}(2n-1)$. Figure 4.15 shows $\vec{B}(9)$ in which arc $(1,2)$ is the base arc and 3 is the apex.

Theorem 4.11 *$SN(\vec{B}(n)) \leq 3$ for all $n = 2^m + 1$ where $m \geq 0$.*

Proof: We show that $SN(\vec{B}(n)) \leq 3$ by inductively constructing a 3-stack layout of $\vec{B}(n)$. The inductive hypothesis is as follows.

Induction hypothesis: Let the base arc of $\vec{B}(n)$ be (u, v), let its apex be w, let its left dag be $\vec{A}$ and let its right dag be $\vec{B}$. Then $\vec{B}(n)$ has a 3-stack layout in which

1. u, v, and w are the first 3 nodes of the layout, in that order followed by the remaining nodes of the right dag, followed by the remaining nodes of the left dag.

2. The arcs with tail u are assigned to stack s_1, the arcs with tail v are assigned to stack s_2, and the arcs with tail w are assigned to stack s_3. Each of the remaining arcs is assigned to one of stacks s_1, s_2, or s_3.

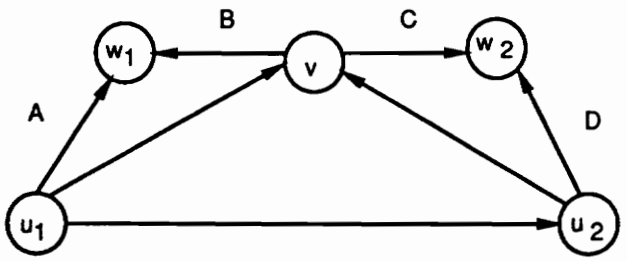


Figure 4.16: The labeling of $\vec{B}(n)$.

The base case is when $n = 0$. Since $\vec{B}(2)$ is simply an arc, the inductive hypothesis is trivially satisfied. Now we assume that $\vec{B}(n)$ for some $n = 2^m + 1$ where $m \geq 0$ has a 3-stack layout satisfying the inductive hypothesis. Using this we show that $\vec{B}(2n-1)$ can also be laid out in 3 stacks such that the inductive hypothesis is satisfied. Let (u_1, u_2) be the base arc of $\vec{B}(2n-1)$ and let v be its apex. Let $\vec{B}_1$ be its left dag and $\vec{B}_2$ be its right dag. Let w_1 be the apex of $\vec{B}_1$ and let $\vec{A}$ and $\vec{B}$ be the left and the right dags respectively of $\vec{B}_1$. Let w_2 be the apex of $\vec{B}_2$ and let $\vec{D}$ and $\vec{C}$ be the left and the right dags respectively of $\vec{B}_2$. This labeling is shown in Figure 4.16. $\vec{B}_1$ and $\vec{B}_2$ have 3-stack layouts that satisfy the conditions specified in the induction hypothesis. These are shown in Figure 4.17 (a) and (b) respectively. The arcs assigned to s_1 are drawn in plain lines above the layout; the arcs assigned to s_2 are drawn in broken lines above the layout; and the arcs assigned to s_3 are drawn below the layout. The layouts of $\vec{B}_1$ and $\vec{B}_2$ are mingled together as shown in Figure 4.17 (c) to give a 3-stack layout of $\vec{B}(2n-1)$. Arc (u_1, u_2) is assigned to stack s_1 and the mingling of the nodes is accompanied by a suitable reassignment of arcs to stacks. The contents of stacks s_1 and s_2 are exchanged. The contents of s_3 are assigned to s_1, the contents of s_1 are assigned to s_2, and the contents of s_2 are assigned to s_3. It is easy to see from Figure 4.17 (c) that the proposed mingling does produce a 3-stack layout of $\vec{B}(2n-1)$ that satisfies the induction hypothesis. □

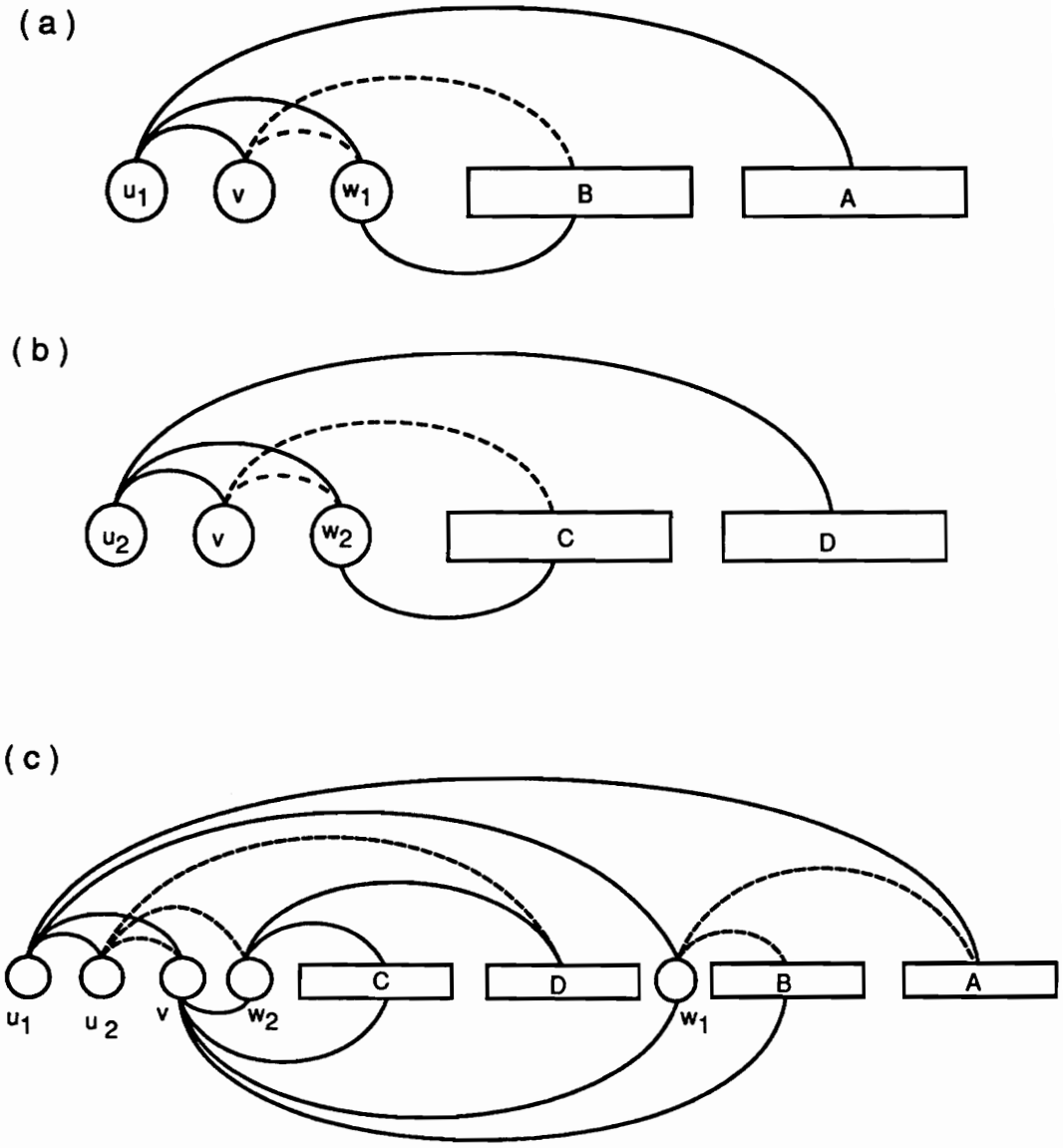


Figure 4.17: The inductive step to show that $SN(\vec{B}(n)) \leq 3$.

4.3.4 Planar Dags

A *planar dag* is any acyclic directing of a planar graph. We now present a class of planar dags

$$\mathcal{L} = \{\vec{L}(n) \mid n \text{ is even}\}$$

such that

$$QN(\vec{L}(n)) = 2$$

and

$$SN(\vec{L}(n)) = \frac{n}{2}$$

for all even n. For the rest of the description we assume that n is even and let $m = n/2$. The node set of $\vec{L}(n)$ is:

$$V = \Big\{u_i : 1 \leq i \leq m\Big\} \cup \Big\{v_i : 1 \leq i \leq m\Big\}.$$

The arc set $\vec{R} \cup \vec{S} \cup \vec{T}$ is:

$$\vec{R} = \Big\{(u_i, v_i) : 1 \leq i \leq m\Big\},$$

$$\vec{S} = \Big\{(u_i, u_{i+1}) : 1 \leq i \leq (m-1)\Big\} \cup \Big\{(v_i, v_{i+1}) : 1 \leq i \leq (m-1)\Big\},$$

$$\vec{T} = \Big\{(u_m, v_1)\Big\}.$$

Figure 4.18 shows $\vec{L}(8)$. Note that all $\vec{L}(n) \in \mathcal{L}$ are planar dags.

Theorem 4.12 *$SN_{\mathcal{L}}(n) = n/2$ and $QN_{\mathcal{L}}(n) = 2$ for all $n = 2m$ where $m \geq 2$.*

Proof: The order of the nodes in any layout of $\vec{L}(n)$ is given by the unique directed hamiltonian path

$$u_1, u_2, \ldots, u_m, v_1, v_2, \ldots, v_m$$

in $\vec{L}(n)$. This implies that every pair of arcs in $\vec{R}$ mutually cross, thus giving a twist of size $m = n/2$. Hence $SN(\vec{L}(n)) \geq n/2$. To show that $SN(\vec{L}(n)) \leq n/2$, we provide an assignment of the arcs in $\vec{E}$ to $n/2$ stacks such that no two arcs assigned to the same stack cross. The arcs in $\vec{S}$, the sole arc in $\vec{T}$ and an arbitrary arc in $\vec{R}$ can all be assigned to 1

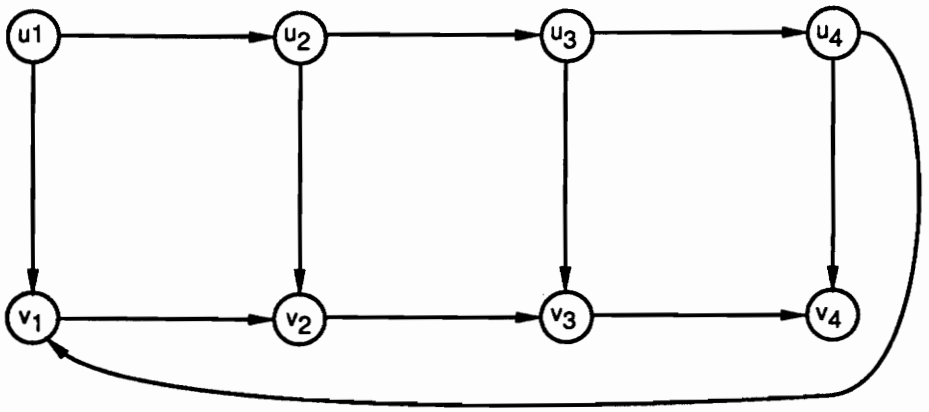


Figure 4.18: The planar dag $\vec{L}(8)$.

stack. Each of the remaining $n/2-1$ arcs in $\vec{R}$ can be assigned to a stack all by itself. This shows that $SN(\vec{L}(n)) = n/2$ and thus $SN_{\mathcal{L}}(n) = n/2$ for all $n = 2m$ where $m \geq 2$.

To see that $QN(\vec{L}(n)) \geq 2$ notice that in the unique layout of $\vec{L}(n)$ each of the arcs (u_i, v_i), $2 \leq i \leq n-1$ nests over the arc (u_n, v_1). To see that $QN(\vec{L}(n)) \leq 2$ notice that in the unique layout of $\vec{L}(n)$ the arcs in $\vec{S} \cup \vec{T}$ can be assigned to 1 queue while all the arcs in $\vec{R}$ can be assigned to another queue. Thus $QN_{\mathcal{L}}(n) = 2$. □

The above result reveals two contrasts. First, it shows a contrast between the stacknumber of the class of planar graphs and the stacknumber of the class of planar dags. The result of Yannakakis [64], showing that the stacknumber of the class of planar graphs is 4, along with Theorem 4.12, lead to the following corollary.

Corollary 4.13 *Let $\mathcal{G}$ be the class of planar graphs and let $\mathcal{H}$ be the class of planar dags. Then*

$$SN_{\mathcal{H}}(n) = \theta(n)$$

and

$$\frac{SN_{\mathcal{H}}(n)}{SN_{\mathcal{G}}(n)} = \theta(n).$$

Second, it shows the existence of a class of dags whose stacknumber is arbitrarily large as compared to the queuenumber. Proposition 4.9 and Theorem 4.12 lead to the following

corollary.

Corollary 4.14 *There exists a class of planar dags $\mathcal{P}$ such that*

$$\frac{SN_{\mathcal{P}}(n)}{QN_{\mathcal{P}}(n)} = \theta(n)$$

and there exists a class of planar dags $\mathcal{Q}$ such that

$$\frac{QN_{\mathcal{Q}}(n)}{SN_{\mathcal{Q}}(n)} = \theta(n).$$

This corollary should be viewed in light of the fact that there is no known class of undirected graphs whose queuenumber is arbitrarily large as compared to its stacknumber. In fact, Heath, Leighton, and Rosenberg [35] conjecture that no such family exists.

4.4 Algorithm for Recognizing 1-Stack Dags

The main result of this section is an $O(|V|)$ time algorithm that determines whether a dag $\vec{G} = (V, \vec{E})$ is a 1-stack dag. If $\vec{G}$ is a 1-stack dag, then the algorithm explicitly constructs a 1-stack layout of $\vec{G}$. Recall that an undirected graph is a 1-stack graph if and only if it is an outerplanar graph (see Proposition 1.18). Sysło and Iri [61] give an $O(|V|)$ time algorithm that determines whether a graph $G = (V, E)$ is an outerplanar graph. Thus, whether a graph is a 1-stack graph can be tested in linear time, in the size of the vertex set of the graph. The result that we prove in this section, parallels the above result.

The algorithm has 3 stages and we describe them one by one.

Stage 1: In this stage, the algorithm verifies that $|\vec{E}| \leq 2|V| - 3$. This is because an outerplanar graph can have at most $2|V| - 3$ edges. Then the algorithm decomposes $\vec{G}$ into its connected components and passes on each connected component separately to stage 2. Note that stage 1 can be completed in $O(|V|)$ time.

In stage 2, our algorithm decomposes $\vec{G}$ into biconnected components and verifies that each biconnected component is a 1-stack dag. Bernhart and Kainen [6] show that the stacknumber of a graph is the maximum of the stacknumber of its biconnected components.

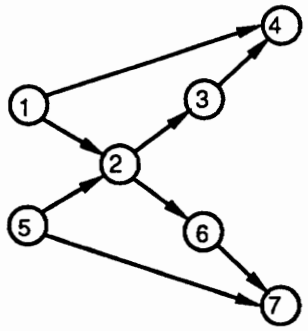


Figure 4.19: A 2-stack dag each of whose biconnected components are 1-stack dags

The analogous result does not hold for dags. Figure 4.19 shows a dag $\vec{G}$ that contains two biconnected components, $\vec{A}$ containing the arcs $\{(1,2),(2,3),(3,4),(1,4)\}$ and $\vec{B}$ containing the arcs $\{(5,2),(2,6),(6,7),(5,7)\}$. Node 2 is the cut point. The dags $\vec{A}$ and $\vec{B}$ are 1-stack dags, but it can be easily seen that $\vec{G}$ is a 2-stack dag. Therefore, it is a necessary, but not sufficient condition that all biconnected components of a 1-stack dag $\vec{G}$, be 1-stack dags. The following lemma shows that a biconnected dag can be tested for being a 1-stack dag in $O(|V|)$ time.

Lemma 4.15 *Whether a biconnected dag $\vec{G} = (V, \vec{E})$ is a 1-stack dag can be tested in $O(|V| + |E|)$ time.*

Proof: A biconnected dag $\vec{G} = (V, \vec{E})$ is a 1-stack dag if and only if

1. $\vec{G}$ is an outerplanar dag.

2. There is a directed hamiltonian path obtained by traversing the external face of an outerplanar embedding of $\vec{G}$ in clockwise or counter-clockwise order. The 1-stack layout of $\vec{G}$ is obtained by laying out the nodes of $\vec{G}$ according to the directed hamiltonian path.

Condition (i) can be verified in $O(|V| + |E|)$ time using the algorithm of Sysło and Iri [61]. Assuming that $\vec{G}$ has an outerplanar embedding, condition (ii) can be verified in $O(|V|)$ time by traversing the outer face of $\vec{G}$ in clockwise and in anti-clockwise direction. □

Now we are ready to precisely describe stage 2 of our algorithm. Recall that the input to stage 2 is a connected dag with no more than $2|V| - 3$ arcs.

Stage 2: Decompose $\vec{G}$ into biconnected components. This can be done in $O(|V|)$ time (see Manber [48]). Verify that each biconnected component is a 1-stack dag. This can also be done, as shown in Lemma 4.15, in $O(|V|)$ time.

In stage 3, the algorithm takes the 1-stack layouts of the biconnected components of $\vec{G}$ constructed in stage 2 and determines whether these biconnected components can be combined to produce a 1-stack layout of $\vec{G}$. To describe stage 3, we need some definitions and notation. The *block-cutpoint tree* $BCPT(\vec{G})$ of a dag $\vec{G}$ is the undirected graph whose vertex set is

$$\left\{\vec{B} \mid \vec{B} \text{ is a biconnected component of } \vec{G}\right\} \cup \left\{u \mid u \text{ is a cutpoint in } \vec{G}\right\}$$

and whose edge set is

$$\left\{(\vec{B}, u) \mid u \text{ is a cutpoint in } \vec{B}\right\}.$$

Harary and Prins [30] show that $BCPT(\vec{G})$ is a tree if $\vec{G}$ is connected. An *intermediate node* in a biconnected component $\vec{B}$ is any cut-point of $\vec{G}$ that is neither a source nor a sink in $\vec{B}$. As we shall see, intermediate nodes can be obstacles to obtaining a 1-stack layout of $\vec{G}$.

If $\vec{B}$ is a biconnected component of G, define $BCPT(\vec{G}, \vec{B})$ to be $BCPT(\vec{G})$ rooted at $\vec{B}$. If x is a cutpoint of G contained in $\vec{B}$, define $N(\vec{B}, x)$ to be the set of nodes in the subtree of $BCPT(\vec{G}, \vec{B})$ with root x. Consider the following property that $\vec{G} = (V, \vec{E})$ may possess:

> *Property (B):* There do not exist cutpoints $u, v \in V$ such that u is an intermediate node in a biconnected component $\vec{B}_i$, v is an intermediate node in a biconnected component $\vec{B}_j$, $\vec{B}_i \in N(\vec{B}_j, v)$, and $\vec{B}_j \in N(\vec{B}_i, u)$.

The following lemma establishes Property (B) as a condition that $\vec{G}$ has to satisfy in order to be a 1-stack dag.

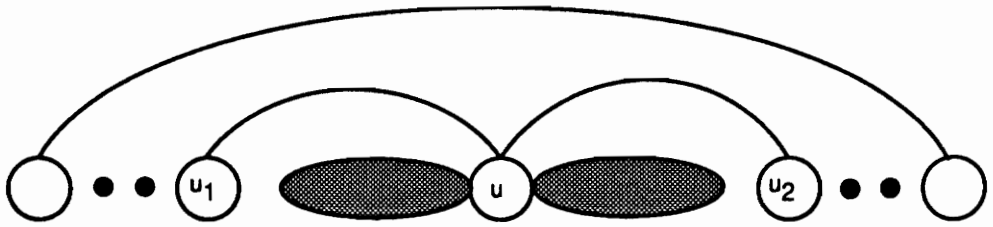


Figure 4.20: Violation of Property (B).

Lemma 4.16 *If $\vec{G}$ is a 1-stack dag, then it satisfies Property (B).*

Proof: Assume that $\vec{G}$ violates Property (B) and there exist cutpoints $u, v \in V$ such that u is an intermediate node in biconnected component $\vec{B}_i$, v is an intermediate node in biconnected component $\vec{B}_j$, $\vec{B}_i \in N(\vec{B}_j, v)$ and $\vec{B}_j \in N(\vec{B}_i, u)$. Let u_1 and u_2 be nodes in $\vec{B}_i$ such that u_1 occurs immediately before and u_2 occurs immediately after u in the 1-stack layout of $\vec{B}_i$. (We know that u_1 and u_2 exist if u is an intermediate node in $\vec{B}_i$.) Then, the directed subgraph of $\vec{G}$ induced by the elements in $N(\vec{B}_i, u) - \{u\}$ must be placed completely between u_1 and u_2, in any 1-stack layout of $\vec{G}$. In particular, $\vec{B}_j$ must be placed completely between u_1 and u_2. See Figure 4.20 for an illustration of this requirement. Similarly, if v_1 and v_2 are nodes in $\vec{B}_j$ such that v_1 occurs immediately before and v_2 occurs immediately after v in the 1-stack layout of $\vec{B}_j$, then the directed subgraph of $\vec{G}$ induced by the elements in $N(\vec{B}_j, v) - \{v\}$ must be placed completely between v_1 and v_2, in any 1-stack layout of $\vec{G}$. But, both these conditions cannot be simultaneously met without causing an intersection. □

Now we describe stage 3 of our algorithm in which a 1-stack layout of $\vec{G}$ is constructed if $\vec{G}$ satisfies Property (B).

Stage 3: Let $\vec{B}_1, \vec{B}_2, \ldots, \vec{B}_\ell$ be a total order on the biconnected components of $\vec{G}$, obtained by doing a breadth first search on $BCPT(\vec{G})$ starting at $\vec{B}_1$. We begin with a 1-stack layout of $\vec{B}_\ell$ and add the rest of the biconnected components in the order $\vec{B}_{\ell-1}, \vec{B}_{\ell-2}, \ldots, \vec{B}_2, \vec{B}_1$. Each biconnected component is added so as to maintain an inductive hypothesis. To be

able to describe the inductive hypothesis, we need the following definitions. Let $\vec{H}$ be a directed subgraph of $\vec{G}$ induced by a subset of the set of all biconnected components of $\vec{G}$, $\vec{B}_1, \vec{B}_2, \ldots, \vec{B}_\ell$. A cutpoint u of $\vec{G}$ is said to be *restricted* in $\vec{H}$ if $u \in N(\vec{B}_i, v)$ for an intermediate node v in some biconnected component $\vec{B}_i$ in $\vec{H}$. A node v of $\vec{G}$ is said to be *exposed in* a layout if there is no arc in the layout with its tail to the left of v and its head to the right of v.

Inductive hypothesis: Let $\vec{G}_i$ be the directed subgraph of $\vec{G}$ induced by the biconnected components $\vec{B}_\ell, \vec{B}_{\ell-1}, \ldots, \vec{B}_i$. If $\vec{G}_i$ satisfies property (A) then $\vec{G}_i$ has a 1-stack layout in which all cutpoints that are not restricted are exposed.

Base case: $\vec{B}_\ell$ has a unique 1-stack layout in which its source and sink are exposed and the rest of the nodes are not. Let u be a node in $\vec{B}_\ell$ that is neither a source nor a sink. If u is a cutpoint in $\vec{G}$, then u is an intermediate node and since $u \in N(\vec{B}_\ell, u)$, u is restricted. Thus the only nodes in $\vec{B}_\ell$ that may be cutpoints that are not restricted are the source and the sink. Since these are exposed the inductive hypothesis is satisfied.

Inductive case: Assume that the inductive hypothesis is satisfied for some i, $\ell \geq i > 1$. We show how to add $\vec{B}_{i-1}$ to the layout of $\vec{G}_i$ such that the inductive hypothesis is maintained. $\vec{G}_i$ may have several connected components and they occur one after the other in some arbitrary order in the 1-stack layout of $\vec{G}_i$. $\vec{B}_{i-1}$ is connected to $\vec{G}_i$ via cutpoints, some of which are intermediate nodes and some of which are not. We examine the two kinds of cutpoints in separate cases.

Case 1: Let u be a cutpoint in $\vec{G}$ that is an intermediate node in $\vec{B}_{i-1}$. If u is restricted in $\vec{G}_i$, then $\vec{G}_{i-1}$ violates Property (B) and does not have a 1-stack layout. Thus, if $\vec{G}_{i-1}$ has a 1-stack layout, then none of the intermediate nodes in $\vec{B}_{i-1}$ are restricted.

Case 2: If both the source s and the sink t of $\vec{B}_{i-1}$ are restricted cutpoints in $\vec{G}_i$, then $\vec{G}_{i-1}$ violates Property (B) and does not have a 1-stack layout. Thus, if $\vec{G}_{i-1}$ has a 1-stack layout, then at most one of s and t is restricted in $\vec{G}_i$.

Assuming that $\vec{G}_{i-1}$ has a 1-stack layout, $\vec{B}_{i-1}$ can be added to $\vec{G}_i$ as follows. Let

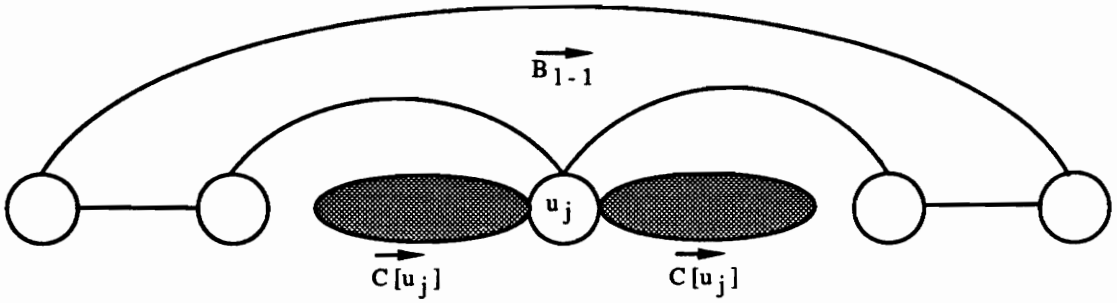


Figure 4.21: Merging $\vec{C}[u_j]$ with $\vec{B}_{i-1}$ when u_j is an intermediate node.

$u_1, u_2, \ldots, u_m$ be the cutpoints of $\vec{G}$ that belong to $\vec{B}_{i-1}$. Place the 1-stack layout of $\vec{B}_{i-1}$ to the right of the layout of $\vec{G}_i$. Note that for each j, $1 \leq j \leq m$, u_j occurs twice in the layout, once in $\vec{B}_{i-1}$ and once in $\vec{G}_i$. For each j, $1 \leq j \leq m$, let $\vec{C}[u_j]$ be the connected component in $\vec{G}_i$ that u_j belongs to. For each u_j that is not restricted in $\vec{G}_i$, shift the layout of $\vec{C}[u_j]$ to the right, keeping the nodes contiguous and their relative order unchanged, so that the two copies of u_j coincide. See Figure 4.21 for an illustration of how $\vec{C}[u]$ is merged with $\vec{B}_{i-1}$. As stated in case 2, at most one of the u_js may be restricted in $\vec{G}_i$. Let u_k, be restricted in $\vec{G}_i$ for some k, $1 \leq k \leq m$. From the discussion in case 2, u_k can either be the source or the sink in $\vec{B}_{i-1}$. If it is the source, then shift the layout of $\vec{C}[u_k]$ to the right, without changing the relative order of the nodes, such that the two occurrences of u_k coincide and

- All nodes that occur to the left of u_k in the layout of $\vec{C}[u_k]$ still occur immediately to the left of u_k as a contiguous sequence.
- All nodes that occur to the right of u_k in the layout of $\vec{C}[u_k]$ now occur immediately to the right of the layout of $\vec{B}_{i-1}$, as a contiguous sequence.

See Figure 4.22 for an illustration of how $\vec{C}[u_k]$ is merged with $\vec{B}_{i-1}$. If u_k is the sink, the addition of $\vec{B}_{i-1}$ to $\vec{G}_i$ is similar and in shown in Figure 4.23. If u_k is the sink, the addition

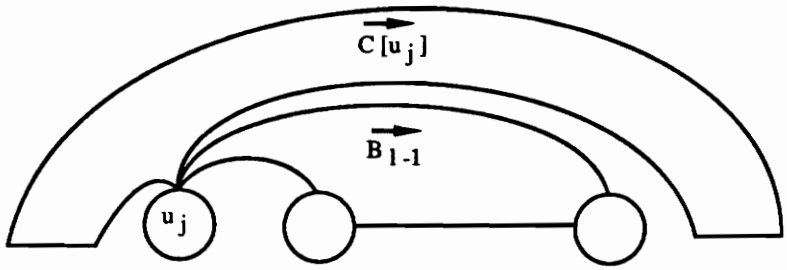


Figure 4.22: Merging $\vec{C}[u_j]$ with $\vec{B}_{i-1}$ when u_j is a source.

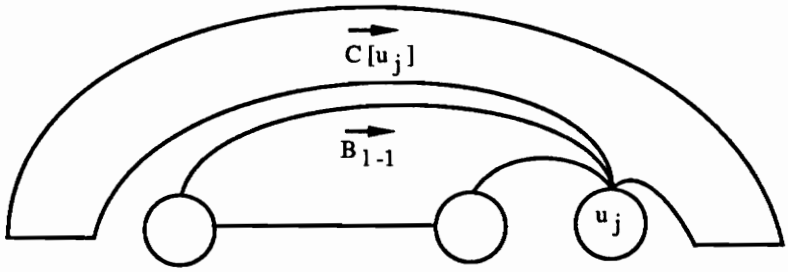


Figure 4.23: Merging $\vec{C}[u_j]$ with $\vec{B}_{i-1}$ when u is a sink.

of $\vec{B}_{i-1}$ to $\vec{G}_i$ is similar It is easily seen that the induction hypothesis is satisfied.

Each addition of a biconnected component $\vec{B}_{i-1}$ can be done in $O(1)$ time and hence stage 3 can be completed in $O(\ell)$ time. Since, $\ell \leq |V|$ we conclude that a 1-stack layout of a dag can be constructed in $O(|V|)$ time, if it exists. The algorithm described above produces a 1-stack layout of $\vec{G}$ if all its biconnected components are 1-stack dags and it satisfies Property (B). Since we know that if $\vec{G}$ violates Property (B), then it is not a 1-stack dag (see Lemma 4.16) we have the following characterization of 1-stack dags.

Theorem 4.17 *A dag $\vec{G} = (V, \vec{E})$ is a 1-stack dag if and only if every biconnected component in $\vec{G}$ is a 1-stack dag and $\vec{G}$ satisfies Property (B).*

The above theorem is easily extended to obtain the following sufficient conditions for a dag to be a k-stack dag.

Theorem 4.18 *Let $\vec{G}$ be a dag with biconnected components $\vec{B}_1, \vec{B}_2, \ldots, \vec{B}_\ell$ such that*

$$\max_{1 \leq i \leq \ell} SN(\vec{B}_i) = k.$$

If $\vec{G}$ satisfies Property (B), then $SN(\vec{G}) = k$.

It is easily seen that the condition in Theorem 4.18 is sufficient, but not necessary.

Finally, since directed cycles and unicyclic dags are outerplanar dags, given a directed cycle or a unicyclic dag, we can determine whether the given dag is a 1-stack dag in time linear in the size of the node set.

4.5 Algorithm for Recognizing Leveled Planar Dags

Heath and Rosenberg [36] characterize 1-queue graphs as arched leveled-planar graphs (see Proposition 1.19). Using this characterization, they show that the problem of recognizing a 1-queue graphs is NP-complete, by showing that the problem of recognizing arched leveled-planar graphs is NP-complete. They also show that the problem of recognizing a leveled-planar graph is NP-complete. In this section, we characterize 1-queue dags in a

similar manner and attempt to recognize then efficiently. A dag $\vec{G} = (V, \vec{E})$ is called a *leveled dag* if there is a function $level : V \rightarrow J$, where J is the set of integers, such that if $(u, v) \in \vec{E}$, then $level(v) = level(u) + 1$. $\vec{G}$ is a called *leveled planar dag* if its covering graph, G, has a leveled planar embedding such that if $(u, v) \in \vec{E}$ and u is on the vertical line L_i, then v is on the vertical line L_{i+1}. $\vec{G}$ is an *arched leveled-planar dag* if its covering graph, G, has an arched leveled-planar embedding such that each arc $(u, v) \in \vec{E}$ is either

- a *leveled arc*; that is, u is on the vertical line L_i and v is on the vertical line L_{i+1} for some i or
- a *directed arch*; that is, u and v are on the same vertical line L_i, for some i, and v is the topmost node on L_i.

The following proposition is immediate.

Proposition 4.19 *A dag is a 1-queue dag if and only if it is an arched leveled-planar dag.*

The main result in this section is an algorithm to recognize leveled-planar dags efficiently. We conjecture that this algorithm can be extended to recognize arched leveled-planar dags efficiently. As a first step to describing our algorithm, we show that a 1-queue dag that is leveled has a leveled-planar embedding. Throughout this section, we restrict our attention to connected graphs. We have the luxury of doing this because if a dag has more than one connected component, then its connected components can be dealt with separately.

Theorem 4.20 *Suppose that $\vec{G}$ is a leveled dag. Then, $\vec{G}$ is a 1-queue dag if and only if it is a leveled-planar dag.*

Proof: Clearly, if $\vec{G}$ is a leveled-planar dag, then it is a 1-queue dag.

Assume that $\vec{G}$ is a leveled dag. We now show that if $\vec{G}$ is a 1-queue dag, then $\vec{G}$ is a leveled-planar dag. Let σ be a total order on V that yields a 1-queue layout of $\vec{G}$. Since, $\vec{G}$ is leveled, there exists a function $level : V \rightarrow J$ such that if $(u, v) \in \vec{E}$, then $level(v) = level(u) + 1$. Place a node $u \in V$, with $level(u) = i$, on the vertical line L_i

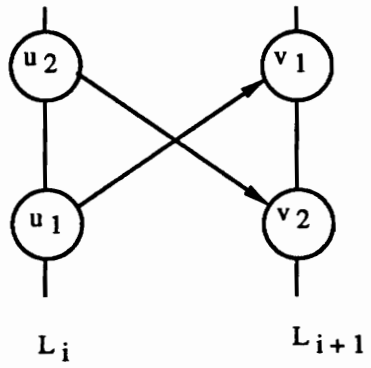


Figure 4.24: Intersection of (u_1, v_1) and (u_2, v_2).

described by the equation $x = i$ in the Cartesian plane. On each vertical line, place nodes bottom to top in the order prescribed by σ. Each arc of $\vec{G}$ can be drawn as an arrow whose tail is on line L_i and whose head is on line L_{i+1} for some integer i. We now show that the embedding described above is planar. It suffices to show that no two arcs (u_1, v_1) and (u_2, v_2) with distinct end-points and $level(u_1) = level(u_2)$ intersect. Assume without loss of generality that $u_1 <_\sigma u_2$ and and $level(u_1) = level(u_2) = i$ for some integer i. Since $u_1 <_\sigma u_2$, u_1 appears below u_2 on L_i. We assume that on line L_{i+1}, v_2 appears below v_1, implying that $v_2 <_\sigma v_1$ and (u_1, v_1) and (u_2, v_2) intersect (see Figure 4.24). Since, $(u_2, v_2) \in \vec{E}$ and σ is a topological order on $\vec{G}$, $u_2 <_\sigma v_2$. Thus we have

$$u_1 <_\sigma u_2 <_\sigma v_2 <_\sigma v_1.$$

This implies that (u_1, v_1) and (u_2, v_2) nest in σ which is absurd because σ is a 1-queue layout of $\vec{G}$. Therefore our assumption that v_2 appears below v_1 on line L_{i+1} is incorrect. We have thus shown that the arcs (u_1, v_1) and (u_2, v_2) do not intersect and $\vec{G}$ has a leveled-planar embedding.

Clearly, if $\vec{G}$ is a leveled-planar dag, it is a 1-queue, leveled dag. □

An example of a leveled dag is a directed tree. The above theorem implies that any 1-queue directed tree has a leveled-planar embedding. This is a very useful result because it allows us to ignore the possibility that there might be arches in the embedding.

We now present the main result of this section: an algorithm, called Leveled Planar Dag Recognition (LPDR), that takes as input a dag $\vec{G} = (V, \vec{E})$ and determines whether $\vec{G}$ is a leveled-planar dag in $O(|V|)$ time. There are 3 stages to LPDR. In stage 1, LPDR determines if $|\vec{E}| \leq 2|V| - 4$, because a leveled-planar dag can have at most $2|V| - 4$ arcs. This can be seen as follows. Let a *maximal leveled-planar dag* be a leveled-planar dag that has the maximum possible number of arcs. Any maximal leveled-planar dag, can be thought of as being obtained by deleting directed arches from some maximal arched leveled-planar dag. A maximal arched leveled-planar dag $\vec{G} = (V, \vec{E})$ has at most $2|V| - 3$ arcs (see Theorem 2.13 for a simple proof of this fact) and at least one directed arch. This implies that a maximal leveled-planar dag $\vec{G} = (V, \vec{E})$ can have at most $2|V| - 4$ arcs. In stage 2, LPDR determines if $\vec{G}$ is a leveled dag, and if it is, then LPDR explicitly calculates the function *level*. Thus the input to stage 3 is a leveled dag along with the function *level*. Using PQ-trees, an elegant and powerful data structure introduced by Booth and Lueker [8], LPDR determines whether a leveled dag $\vec{G} = (V, E)$ is leveled-planar, in $O(|V|)$ time. We now describe in detail stages 2 and 3 of LPDR.

Stage 2: In $O(|V|)$ time, a depth-first search tree T of $\vec{G} = (V, \vec{E})$ is constructed. Let u be the root of T. Let $level(u) = 0$. Let $v \in V$ be a vertex in T with parent $v' \in V$. Then, $level(v) = level(v') + 1$ if $(v', v) \in \vec{E}$ and $level(v) = level(v') - 1$ if $(v, v') \in \vec{E}$. In this manner, values of the function *level* are computed for all nodes in V. Determining these values take $O(|V|)$ time. To check that for each arc, $(u, v) \in \vec{E}$, $level(v) = level(u) + 1$, requires an additional $O(|V|)$ time. If for each arc (u, v), $level(v) = level(u) + 1$, then $\vec{G}$ is a leveled dag, otherwise it is not a leveled dag and therefore not a 1-queue dag.

Stage 3: In this stage, PQ-trees are utilized to determine whether $\vec{G}$ has a leveled-planar embedding. Our usage of PQ-trees is similar to the usage of PQ-trees by Chiba, Nishizeki, Abe, and Ozawa in determining the planarity of arbitrary graphs [15]. Label the nodes with distinct labels from the set $\{1, 2, \ldots, |V|\}$ such that if $level(u) < level(v)$, then u has a smaller label than v. For the rest of this algorithm, we shall refer to the nodes in V by their labels. LPDR processes the nodes in V in increasing order of their labels. For each

k, $1 \leq k \leq |V|$, let $V_k = \{1, 2, \ldots, k\}$ and let $\vec{G}_k$ be the directed subgraph of $\vec{G}$ induced by V_k. For each k, $1 \leq k \leq |V|$ there is at least one node in V_k that is adjacent to some node in $V - V_k$. Based on this observation, we construct a dag $\vec{H}_k$ from $\vec{G}_k$ by adding certain nodes and arcs to $\vec{G}_k$ as follows: For each node $i \in V_k$ and for each node $j \in V - V_k$, such that $(i, j) \in \vec{E}$ add a node $v_{i,j}$ to $\vec{G}_k$ and add the arc $(i, v_{i,j})$. Each node $v_{i,j}$ is assigned the label j. Note that there may be several nodes in V_k that are adjacent to a node j in $V - V_k$, and each of these adjacencies is responsible for a node labeled j in $\vec{H}_k$ Thus several nodes in $\vec{H}_k$ may have the same label. Therefore, to avoid confusion, we take care to differentiate between the labels and the nodes in $\vec{H}_k$. Each node in $\vec{H}_k$ that is not in $\vec{G}_k$ is called a *virtual node* and each arc in $\vec{H}_k$ that is not in $\vec{G}_k$ is called a *virtual arc.* Note that each virtual node in $\vec{H}_k$ is labeled j for some $j > k$ and the rest of the nodes are nodes that were originally in $\vec{G}_k$ and therefore are labeled $1, 2, \ldots, k$. Each virtual arc in $\vec{H}_k$ is from a node labeled i, for some $i \leq k$ (a node that is not virtual) to a node labeled j for some $j > k$ (a virtual node).

Example 4.21. Figure 4.25 (a) shows a leveled-planar embedding of a dag $\vec{G}$. Figure 4.25 (b) shows its directed subgraph $\vec{G}_4$ induced by the $V_4 = \{1, 2, 3, 4\}$. Figure 4.25 (c) shows $\vec{H}_4$. The virtual nodes are labeled 5, 6, 6, and 7 and are shown as squares. The virtual arcs in $\vec{H}_4$ are (3,6), (4,6), (4,7), and (2,5). □

Define an *almost leveled-planar embedding* of $\vec{H}_k$ as a planar embedding in which the subgraph of $\vec{H}_k$ induced by V_k, namely $\vec{G}_k$, has a leveled planar embedding, all the virtual nodes in $\vec{H}_k$ appear on a vertical line L_{k+1} described by the equation $x = k + 1$, and each virtual arc is drawn as a straight line segment. Thus an almost leveled-planar embedding of $\vec{H}_k$ may contain virtual arcs that are stretched across several levels, but the remainder of the arcs go from one level to the next. Clearly, $\vec{H}_k$ has an almost leveled-planar embedding if and only if $\vec{G}_k$ has a leveled planar embedding. We use PQ-trees to represent *all* the almost leveled-planar embeddings of $\vec{H}_k$ and construct all the almost leveled-planar embedding of $\vec{H}_{k+1}$, that exist. Any almost leveled-planar embeddings of $\vec{H}_{|V|}$, that exist, correspond to

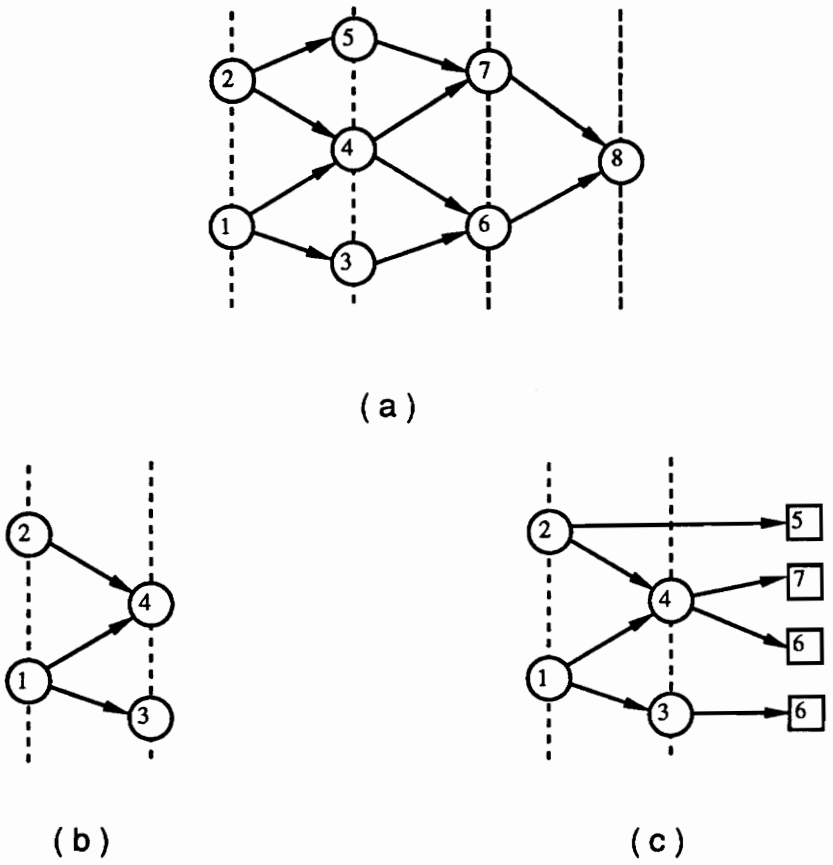


Figure 4.25: (a) A leveled planar embedding of a dag $\vec{G}$. (b) Its directed subgraph $\vec{G}_4$ induced by V_4. (c) $\vec{H}_4$.

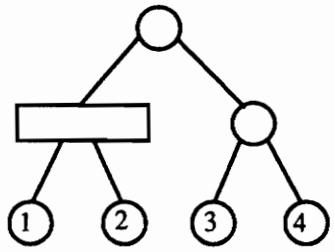


Figure 4.26: A PQ-tree.

leveled-planar embeddings of $\vec{G}$

Now we introduce the PQ-tree data structure. A PQ-tree is a rooted tree that contains three types of nodes: P-nodes, Q-nodes, and leaves. A PQ-tree represents a subset of the set of all permutations of its leaves. This subset of permutations is described using the following rules:

- The children of a P-node can be permuted arbitrarily.
- The children of a Q-node can only be reversed.

Example 4.22. The notion of a PQ-tree is further illustrated in the following example. The PQ-tree shown in Figure 4.26 represents the subset

$$1,2,3,4 \quad 1,2,4,3 \quad 3,4,1,2 \quad 4,3,1,2$$
$$2,1,3,4 \quad 2,1,4,3 \quad 3,4,1,2 \quad 4,3,2,1$$

of the set of 24 permutations of 1,2,3,4. □

$\vec{H}_k$ may have several almost leveled-planar embeddings, each of which corresponds to a different permutation of its virtual nodes on L_{k+1}. Therefore, a succinct way of representing all the almost leveled-planar embeddings of $\vec{H}_k$ is by representing a subset of the set of all permutations of the virtual nodes of vH_k. This subset of the set of all permutations, as we have seen, can be represented by a PQ-tree, called T_k, whose leaves are the virtual nodes of $\vec{H}_k$. More precisely, each virtual node v in $\vec{H}_k$ has a corresponding leaf in T_k that has the same label as v. For notational convenience, let T_0 denote the PQ-tree with a single node, a P-node. Note that T_0 corresponds to the empty set of permutations. Having constructed

T_k, LPDR constructs T_{k+1}, the PQ-tree corresponding to all the almost leveled-planar embeddings of $\vec{H}_{k+1}$. If T_{k+1} represents the empty set of permutations, then $\vec{H}_{k+1}$ does not have an almost leveled-planar embedding, which in turn implies that $\vec{G}_{k+1}$, a directed subgraph of $\vec{G}$, does not have a leveled-planar embedding. This implies that $\vec{G}$ does not have a leveled-planar embedding.

Define a *full node* in T_k as any node labeled $k+1$ or any node all of whose descendents that are leaves, are labeled $k+1$. Depending on whether $k+1$ is a leaf in T_k, there are two cases.

Case 1: $\vec{H}_k$ does not contain a virtual node labeled $k+1$. This implies that T_k does nt have a leaf labeled $k+1$. T_{k+1} is constructed from T_k by performing the following two steps:

1. Add a node labeled $k+1$ to T_k by making it a child of the root of T_k.
2. There may be several virtual nodes in $\vec{H}_{k+1}$ that are adjacent to $k+1$. Add all these nodes to T_k, by making them the children of the node labeled $k+1$ that was added to T_k in step (1).

The two steps described above take $O(d)$ time, in the worst case, where d is the out-degree of node $k+1$. Thus, on the whole, they contribute a total of at most $O(|E|)$ to the time complexity of LPDR.

Case 2: $k+1$ is the label of one or more virtual nodes in $\vec{H}_k$. Hence, $k+1$ is the label of one or more leaves in T_k. To construct T_{k+1} from T_k, LPDR goes through two steps.

Reduction step: The first step called the *reduction step* consists of applying the nine template matchings described by Booth and Lueker [8] to T_k in order to bring all the leaves labeled $k+1$ together as a contiguous sequence. If the template matchings fail to bring all the nodes labeled $k+1$ together, then the reduction step is said to have *failed* and T_{k+1} cannot be constructed. Otherwise,

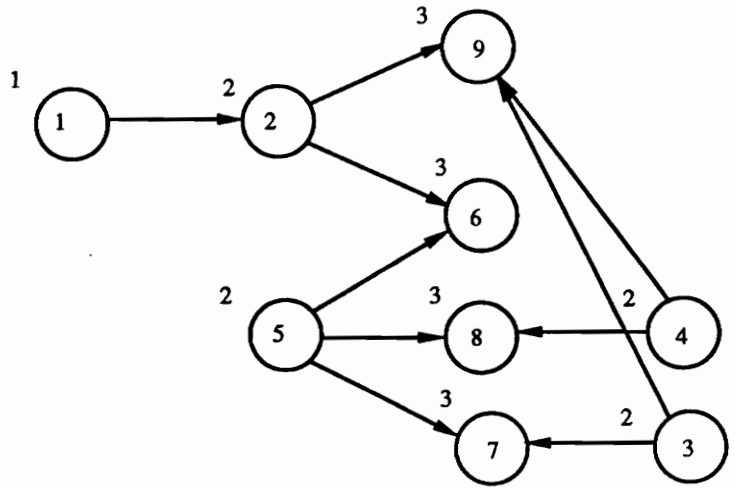


Figure 4.27: A leveled dag $\vec{G} = (V, \vec{E})$

the reduction step is said to have *succeeded* and LPDR proceeds to the second step called the *vertex addition* step.

Vertex Addition step: In the vertex addition step, all full nodes of T_k are replaced by a single node labeled $k+1$. As before, there may be several virtual nodes in $\vec{H}_{k+1}$ that are adjacent to $k+1$. These nodes are added to T_k by making them children of the node $k+1$, that is the node that was just created by merging several nodes labeled $k+1$.

It is easy to see that the vertex addition step takes a total of $O(|E|)$ (for all nodes) in the worst case. Booth and Lueker [8] show that the reduction step takes a total of $O(|V|+|E|)$ (for all nodes) in the worst case.

Since, $|E| = O(|V|)$ for stages 2 and 3, we conclude that LPDR takes a total of $O(|V|)$ time. The algorithm described above is illustrated in the following example.

Example 4.23. A leveled dag $\vec{G} = (V, \vec{E})$ is shown in Figure 4.27. The numbers inside the circles that represent the nodes, are the labels of the nodes, while the numbers written adjacent to the circles are values of the function *level* corresponding to each node. Figure

4.28 shows $\vec{H}_i$ and the corresponding PQ-tree T_i for each i, $1 \leq i \leq 5$. The virtual nodes in $\vec{H}_i$ are shown as squares and remaining nodes are shown as circles. Figure 4.29 shows $\vec{H}_6$, $\vec{H}_7$, and the corresponding PQ-trees. The reduction step and the vertex addition step are shown separately for T_6 and T_7. $\vec{H}_8$ and the corresponding PQ-tree, T_8, are not shown because the reduction step when applied on T_7 fails. This is because all the leaves labeled 8 cannot be brought together as a contiguous sequence. This means $\vec{H}_8$ does not have an almost leveled-planar embedding and hence $\vec{G}_8$ does not have a leveled-planar embedding. From this we conclude that $\vec{G}$ does not have a leveled planar embedding, though $\vec{G}_7$ does. □

The algorithm described above leads to the following results.

Theorem 4.24 *Suppose that $\vec{G} = (V, \vec{E})$ is a dag. Whether $\vec{G}$ is a leveled-planar dag can be determined in $O(|V|)$ time.*

Corollary 4.25 *Suppose that $\vec{T} = (V, \vec{E})$ is a directed tree. Whether $\vec{T}$ is a 1-queue dag can be determined in $O(|V|)$ time.*

Now we address the question of recognizing 1-queue directed cycles. Note that a 1-queue directed cycle may not have a leveled-planar embedding. This implies that LPDR cannot, as it is, be used to recognize a 1-queue directed cycle. Nevertheless, we utilize other properties of a directed cycle to reduce the problem of recognizing 1-queue directed cycles to a problem that LPDR can solve. The property that we need is expressed in the following Lemma.

Lemma 4.26 *A 1-queue directed cycle has an arched leveled-planar embedding in which at most 1 arc is a directed arch.*

Theorem 4.27 *Suppose that $\vec{C} = (V, \vec{E})$ is a directed cycle. Whether $\vec{C}$ is a 1-queue dag can be recognized in $O(|V|)$ time.*

Proof: Let $\vec{C} = (V, \vec{E})$ be a directed cycle such that $V = \{v_0, v_1, \ldots, v_{n-1}\}$, where v_0 is a source, and $E = \{(v_i, v_{i+1}) \mid 0 \leq i \leq n-2\} \cup \{(v_{n-1}, v_0)\}$. Delete (v_0, v_1) from $\vec{C}$ to obtain

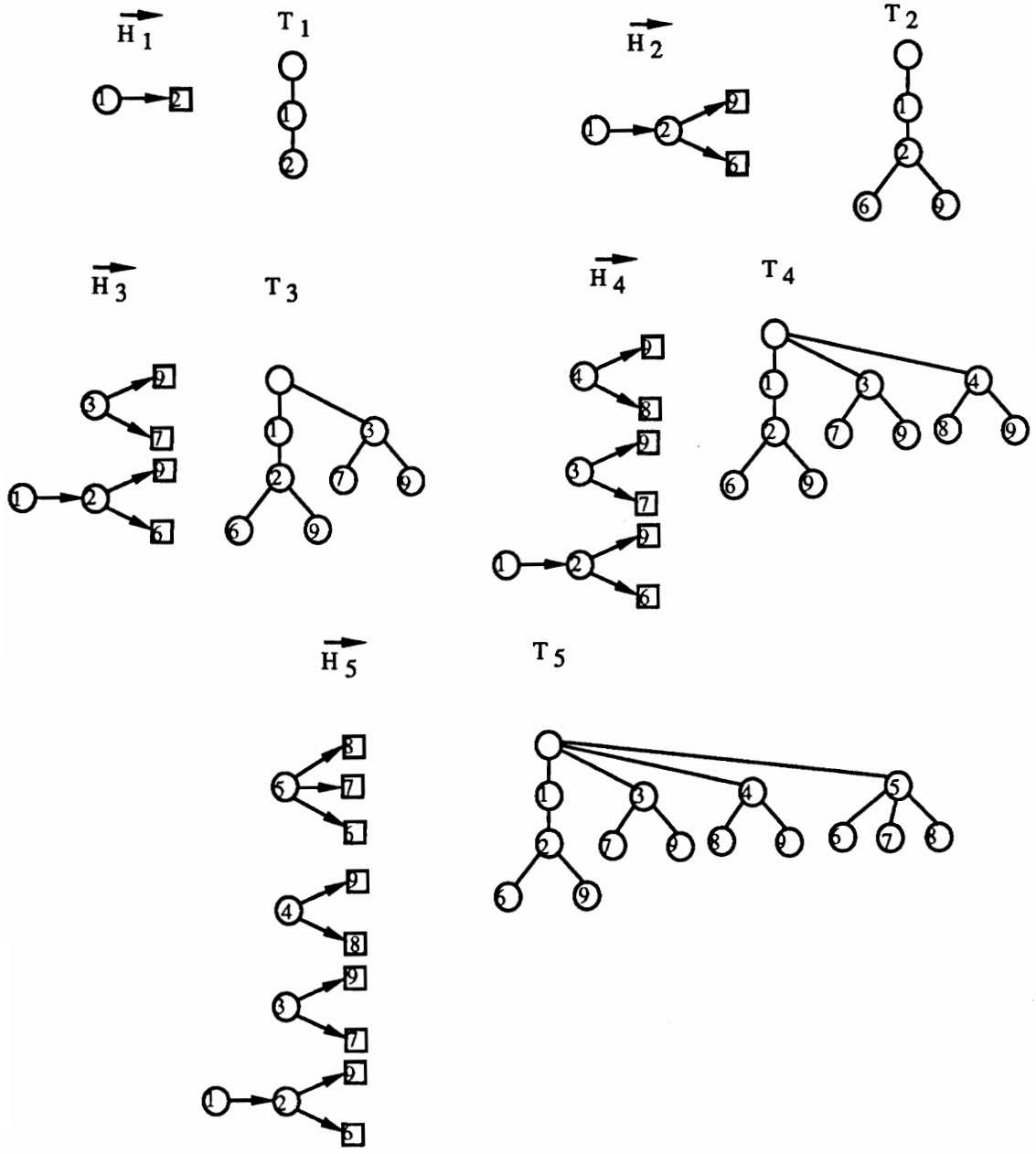


Figure 4.28: An illustration of how LPDR works.

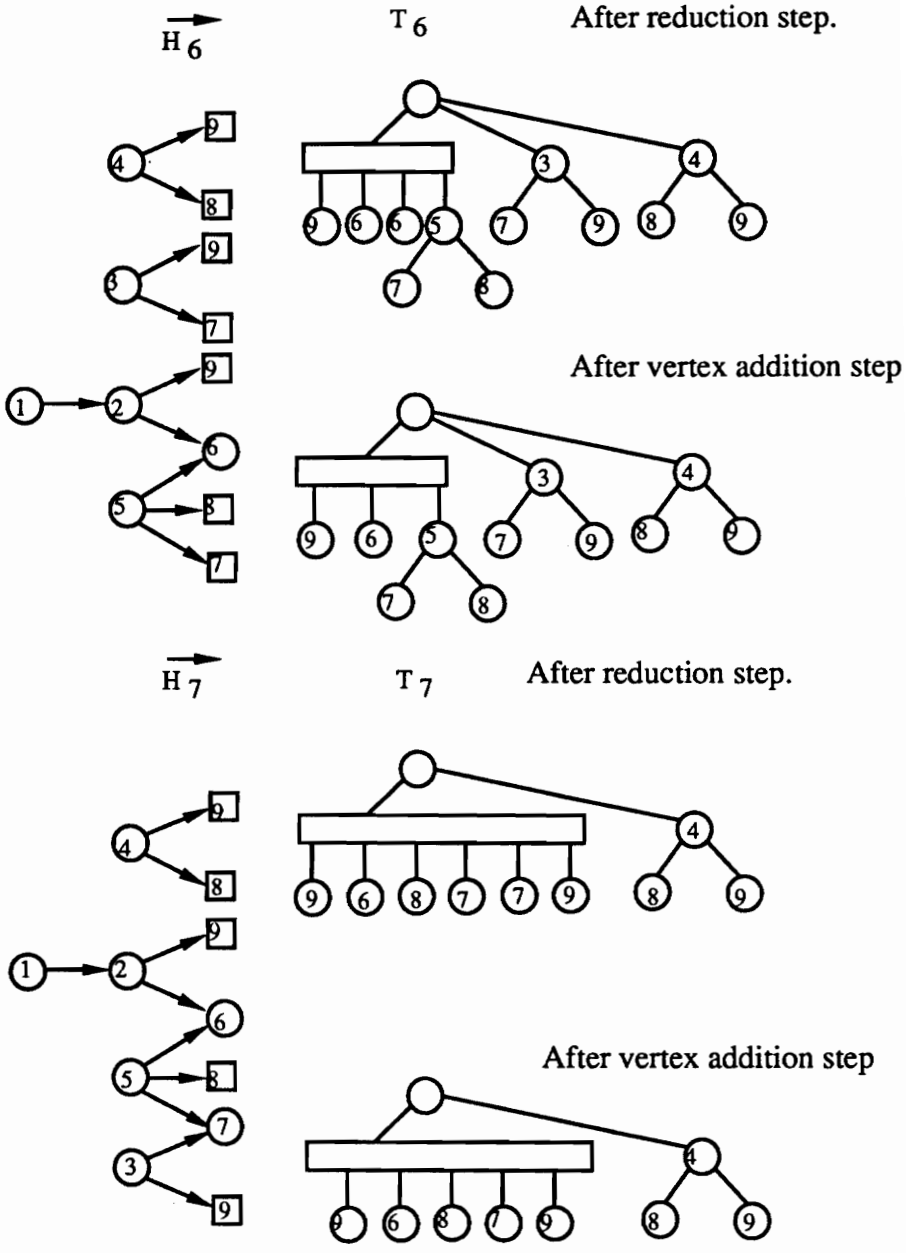


Figure 4.29: An illustration of how LPDR works.

a directed path, $\vec{P}$. Designate v_0 as the origin of $\vec{P}$ and assign distances to all the nodes in $\vec{P}$. There are four cases, depending on the values of $d(v_1)$.

Case 1: $d(v_1) = 0$. The directed path $\vec{P}$ has a leveled-planar embedding in which v_0 and v_1 occur in the same level; v_0 is the bottom most node in the level; and v_1 is the topmost node in the level. To this embedding add the arc (v_0, v_1), to obtain an arched leveled-planar embedding of $\vec{C}$ in which (v_0, v_1) is the sole directed arch.

Case 2: $d(v_1) = 1$. Then, $\vec{C}$ is a leveled dag and by Theorem 4.20, it has a leveled-planar embedding if it is a 1-queue dag. Therefore, we can use LPDR to determine whether $\vec{C}$ is a 1-queue dag.

Case 3: $d(v_1) = 2$. In this case, delete the arc (v_0, v_{n-1}) from $\vec{C}$ to obtain a directed path $\vec{P}_1$. As before, using v_0 as the origin assign distances to each of the nodes. Notice that $d(v_{n-1}) = 0$. Hence, $\vec{P}_1$ has a leveled-planar embedding in which v_0 and v_{n-1} appear on the same level; v_0 is the bottom most node in the level; and v_{n-1} is the top most node in the level. To this embedding add the arc (v_0, v_{n-1}) to obtain an arched leveled-planar embedding of $\vec{C}$ in which (v_0, v_{n-1}) is the sole directed arch.

Case 4: $d(v_1) \notin \{0, 1, 2\}$. Then, $\vec{C}$ does not have an arched leveled-planar embedding with at most one directed arch. Hence, by Lemma 4.26, it is not a 1-queue dag.

□

The important characteristic of the above proof is that we have reduced the problem of determining whether $\vec{C}$ has an arched leveled-planar embedding into the problem of determining whether it has a leveled-planar embedding. We conjecture that this approach extends to an efficient algorithm for recognizing 1-queue dags.

4.6 NP-completeness Results

For undirected graphs, the problems of finding an optimal stack layout and an optimal queue layout are known to be NP-complete. In fact, the problem of determining the number

of stacks required for a fixed layout of an undirected graph is also NP-complete. In this section, we examine the computational complexity of the problems of finding optimal stack and queue layouts of dags. In the next theorem, Theorem 4.28, we show that the problem of determining whether a dag has a 4-queue layout is NP-complete. In Theorem 4.29, we show that the problem of determining whether a dag has a 9-stack layout is NP-complete. In fact we prove a stronger result for queue layouts. Define a *transitive arc* in a dag $\vec{G} = (V, \vec{E})$ as an arc (u, v) such that there is a directed path from u to v not containing the arc (u, v). We show that even when a dag belongs to a restricted class of dags, namely the class of dags without transitive arcs, the problem of determining whether the dag has a 4-queue layout is NP-complete. The motivation for restricting the class of dags to those without transitive arcs comes from the study of stack and queue layouts of posets that we consider in Chapter 5.

DAGQN

INSTANCE: A dag $\vec{G} = (V, \vec{E})$ that does not contain any transitive arcs.

QUESTION: Can $\vec{G}$ be laid out in 4 queues?

Define the decision problem, DAGSN as follows

DAGSN

INSTANCE: A dag $\vec{G} = (V, \vec{E})$.

QUESTION: Can $\vec{G}$ be laid out in 9 stacks?

In the following theorem, DAGQN is shown to be NP-complete. Subsequently, DAGSN will be shown to be NP-complete. The reduction in both proofs is from the known NP-complete problem 3-SATISFIABILITY (3-SAT) (see Garey and Johnson [27], which is defined below.

3-SATISFIABILITY

INSTANCE: Collection $C = \{c_1, c_2, \ldots, c_m\}$ of clauses on a finite set $X = \{x_1, x_2, \ldots, x_n\}$ of variables such that $|c_i| = 3$ for all i, $1 \leq i \leq m$.

QUESTION: Is there a truth assignment of X that satisfies all the clauses in C?

Theorem 4.28 DAGQN *is NP-complete.*

Proof: The queuenumber of a fixed layout of a graph $G = (V, E)$ can be determined in $O(|E| \log\log |V|)$ time [36]. DAGQN is in NP because a nondeterministic Turing machine can guess an ordering of the nodes of $\vec{G}$, check if the ordering is topological, and determine the queuenumber of the layout corresponding to that ordering in polynomial time.

DAGQN is shown to be NP-hard by reduction from 3-SAT. Let $C = \{c_1, c_2, \ldots, c_m\}$ and $X = \{x_1, x_2, \ldots, x_n\}$ be an instance of 3-SAT. A dag $\vec{G} = (V, \vec{E})$ that contains no transitive arcs is constructed such that $\vec{G}$ has a 4-queue layout if and only if there exists a truth assignment for the variables such that all clauses are satisfied. Corresponding to each clause c_i, $1 \leq i \leq m$, $\vec{G}$ contains a subgraph called the *truth-setting dag* TS_i. First, the construction of TS_i is described, followed by a description of how the truth-setting dags are connected together to form $\vec{G}$. The truth-setting dag TS_i can itself be thought of as containing four distinct subgraphs connected together. These are

- A *literal dag* X_i.
- A *clause dag* C_i.
- A *small enforcer dag* F_i.
- A *big enforcer dag* F_i'.

We describe the construction of TS_i by first explaining the construction of X_i, C_i, F_i, and F_i' and by then showing how they connect together to form TS_i. We use $N(\vec{G})$ to denote the set of nodes of the dag $\vec{G}$ and $A(\vec{G})$ to denote the set of arcs of the dag $\vec{G}$.

Literal dag X_i:

$$\begin{aligned} N(X_i) &= \{x_{i,j} \mid 0 \leq j \leq n\} \cup \\ &\quad \{\overline{x_{i,j}} \mid 0 \leq j \leq n\} \cup \\ &\quad \{y_{i,j} \mid 1 \leq j \leq n\} \\ A(X_i) &= \{(y_{i,j}, x_{i,j}),\ (y_{i,j}, \overline{x_{i,j}}) \mid 1 \leq j \leq n\} \cup \\ &\quad \{(x_{i,j}, y_{i,j+1}),\ (\overline{x_{i,j}}, y_{i,j+1}) \mid 0 \leq j \leq n-1\} \end{aligned}$$

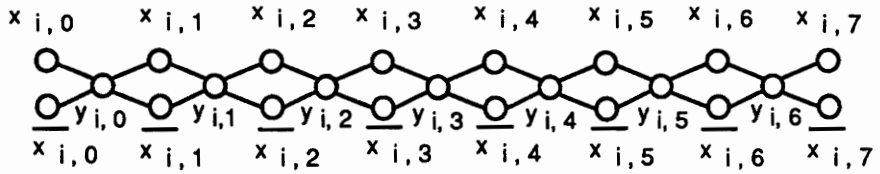


Figure 4.30: The literal dag X_i.

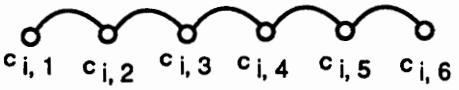


Figure 4.31: The clause dag C_i.

Figure 4.30 shows a literal dag X_i. Note that there are no transitive arcs in X_i. Corresponding to each positive literal x_j, $1 \leq j \leq n$, X_i contains a node $x_{i,j}$ and corresponding to each negative literal $\overline{x_j}$ X_i contains a node $\overline{x_{i,j}}$. Any topological ordering of X_i contains the nodes $x_{i,j}$ and $\overline{x_{i,j}}$ followed by $y_{i,j+1}$ followed by $x_{i,j+1}$ and $\overline{x_{i,j+1}}$ for each j, $0 \leq j \leq n-1$. But there is a choice in the order of pairs of nodes $(x_{i,j}, \overline{x_{i,j}})$ for each $j, 0 \leq j \leq n$. The choice of a particular order of a pair of nodes $(x_{i,j}, \overline{x_{i,j}})$ in the topological ordering of $\vec{G}$ will be interpreted as a particular truth assignment to the variable x_j depending on the literals in c_i.

Clause dag C_i:

$$\begin{aligned} N(C_i) &= \{c_{i,j} \mid 1 \leq j \leq 6\} \\ A(C_i) &= \{(c_{i,j}, c_{i,j+1}) \mid 1 \leq j \leq 5\} \end{aligned}$$

The clause dag C_i is simply a directed path of length 5 and has a unique topological ordering. Later we will show how nodes in X_i are connected to nodes in C_i such that a smaller nesting is caused when there is at least one true literal in c_i, then when there is no true literal in c_i. Figure 4.31 shows a clause dag C_i.

Enforcer dags F_i and F_i':

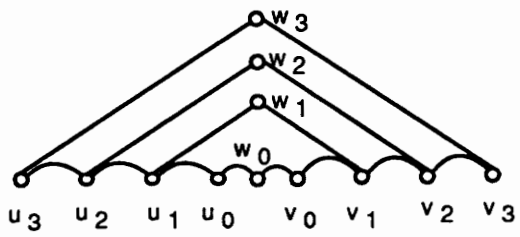


Figure 4.32: The dag $F(3)$.

Define a dag $F(\ell)$ as follows.

$$\begin{aligned}
N(F(\ell)) &= U(\ell) \cup V(\ell) \cup W(\ell) \\
U(\ell) &= \{u_j \mid 0 \leq j \leq \ell\} \\
V(\ell) &= \{v_j \mid 0 \leq j \leq \ell\} \\
W(\ell) &= \{w_j \mid 0 \leq j \leq \ell\} \\
A(F(\ell)) &= \{(u_j, u_{j-1}) \mid 1 \leq j \leq \ell\} \cup \\
&\quad \{(v_j, v_{j+1}) \mid 0 \leq j \leq \ell - 1\} \cup \\
&\quad \{(u_j, w_j) \mid 0 \leq j \leq \ell\} \cup \\
&\quad \{(w_j, v_j) \mid 0 \leq j \leq \ell\}
\end{aligned}$$

Figure 4.32 shows $F(3)$. The small enforcer dag F_i is isomorphic to $F(s)$ for some integer $s \geq 1$, whose value will be determined later. Each node u_j in $F(s)$, $0 \leq j \leq s$ is mapped into a node $u_{i,j}$ in F_i; each node v_j in $F(s)$, $0 \leq j \leq s$ is mapped into a node $v_{i,j}$ in F_i; and each node w_j, $0 \leq j \leq s$ is mapped into a node $w_{i,j}$ in F_i. The large enforcer dag F_i' is isomorphic to $F(s+t+5)$ for some integer $t \geq 1$, whose value will also be determined later. Each node u_j in $F(s+t+5)$, $0 \leq j \leq s+t+5$ is mapped into a node $u'_{i,j}$ in F_i'; each node v_j in $F(s+t+5)$, $0 \leq j \leq s+t+5$ is mapped into a node $v'_{i,j}$ in F_i'; and each node w_j in $F(s+t+5)$, $0 \leq j \leq s+t$ is mapped into a node $w'_{i,j}$ in F_i'.

Based on their role, the arcs of $F(\ell)$ can be thought of as being of two types. A *base*

arc is an arc that belongs to the set

$$\{(u_j, u_{j-1}) \mid 1 \leq j \leq \ell\} \cup \{(v_j, v_{j+1}) \mid 0 \leq j \leq \ell - 1\} \cup \{(u_0, w_0), (w_0, v_0)\}.$$

Base arcs simply serve the purpose of forcing an ordering on the nodes in $U \cup V$. A *two path arc* is an arc that belongs to the set

$$\{(u_j, w_j), (w_j, v_j) \mid 1 \leq j \leq \ell\}.$$

Two path arcs utilize the ordering of the nodes in $U \cup V$ that is forced by the base arcs, to form a rainbow. No arcs between nodes in $U \cup V$ can be added to the base arcs because any other arc will either be a transitive arc or will cause a directed cycle. Therefore, we have chosen two paths as the means to force a rainbow. A pair of two paths (a_1, b_1, c_1) and (a_2, b_2, c_2) *nest* in a layout if the nodes in $\{a_1, b_1, a_2, b_2\}$ occur in the order

$$a_1, a_2, b_2, b_1.$$

Notice that in any layout of $F(\ell)$ the 2-paths (u_j, w_j, v_j), $0 \leq j \leq \ell$ nest. Therefore, the size of the largest 2-path nesting in any layout of $F(\ell)$ is $\ell + 1$. The size of the largest rainbow in a layout of $F(\ell)$ is a function of the size of the largest 2-path nesting in that layout. In Chapter 5 we determine nearly exact bounds on the queuenumber of $F(\ell)$:

$$\left\lfloor \sqrt{\ell} \right\rfloor + 1 \leq QN(F(\ell)) \leq \left\lfloor \sqrt{\ell} + 2 \right\rfloor$$

In particular, we know that

$$\begin{aligned} QN(F(6)) &= 3 \\ QN(F(9)) &= 4 \\ QN(F(12)) &= 4 \\ QN(F(16)) &= 5. \end{aligned}$$

These results provide the queuenumber of the small enforcer dag F_i in terms of s and the queuenumber of the large enforcer dag F_i' in terms of s and t. This is precisely the

purpose of these dags—to enforce a particular queuenumber of the whole layout through an appropriate choice of the constants s and t. For notational convenience denote $QN(F(\ell))$ by $f(\ell)$. The constants s and t are chosen such that

$$\begin{aligned} f(s+5)+1 &= f(s+5+1) \\ &= f(s+5+t) \\ &= f(s+5+t+1) \\ &= f(s+5+t+2)-1. \end{aligned} \tag{4.1}$$

From the four values of f given above it is easy to see that there exist s and t such that Equation 4.1 is satisfied and

$$\begin{aligned} f(s+5) &= 3 \\ f(s+5+1) &= f(s+5+1) \\ &= f(s+5+t) \\ &= f(s+5+t+1) \\ &= 4 \\ f(s+5+t+2) &= 5. \end{aligned}$$

Having described the four major components of a truth-setting dag TS_i, we now describe the connections between them. The first set of connections simply ensure that the four dags appear in the order X_i, F_i, C_i, F_i' in any topological ordering of TS_i. These connections are

- $(x_{i,n}, u_{i,s}), (\overline{x_{i,n}}, u_{i,s})$ from X_i to F_i.
- $(v_{i,s}, c_{i,1})$ from F_i to C_i.
- $(c_{i,6}, u'_{i,s+t+5})$ from C_i to F_i'.

Figure 4.33 shows the connections described above.

The second set of connection depends upon the literals that the clause c_i contains and these are the connections which will cause a nesting of varying size depending on truth

values of the literals in the clause. For these connections we need additional nodes

$$Z_i = \{z_{i,j} \mid 1 \leq j \leq 6\}.$$

These nodes are connected to the clause dag C_i through the arcs

$$\{(z_{i,j}, c_{i,j}) \mid 1 \leq j \leq 6\}.$$

The connections between nodes in X_i and the nodes in Z_i are best explained with an example. Let $c_i = \{x_2, \overline{x_4}, x_7\}$. Then TS_i contains the arcs

$$(x_{i,2}, z_{i,5}) \quad (\overline{x_{i,2}}, z_{i,6})$$

corresponding to the positive literal x_2,

$$(x_{i,4}, z_{i,4}) \quad (\overline{x_{i,4}}, z_{i,3})$$

corresponding to the negative literal $\overline{x_4}$, and the arcs

$$(x_{i,7}, z_{i,1}) \quad (\overline{x_{i,7}}, z_{i,2})$$

corresponding to the positive literal x_7. Figure 4.33 shows the connections between the literal dag X_i and the clause dag C_i via the nodes in Z_i. Note that the relative order of the pairs of nodes $(x_{i,2}, \overline{x_{i,2}})$, $(x_{i,4}, \overline{x_{i,4}})$ and $(x_{i,7}, \overline{x_{i,7}})$ determines the size of the 2-path nesting between X_i and C_i. This size could be as small as 3 if the pairs of nodes occurred in the order $(x_{i,2}, \overline{x_{i,2}})$; $(\overline{x_{i,4}}, x_{i,4})$; and $(x_{i,7}, \overline{x_{i,7}})$ and as large as 6 if the pairs of nodes occur in reverse order.

This completes the description of a truth-setting dag TS_i. TS_i is shown in Figure 4.33.

Now we describe how the truth-setting dags are connected together to form $\vec{G}$. The arcs

$$(v_{i,s+t+5}, x_{i+1,0}) \qquad (v_{i,s+t+5}, \overline{x_{i+1,0}})$$

ensure that X_{i+1} occurs to the right of X_i for all i, $1 \leq i \leq m-1$ in any topological ordering of $\vec{G}$. Each truth-setting dag, TS_i is connected to the truth-setting dag, TS_{i+1} via 2-paths from nodes in X_i to nodes in X_{i+1}. For these 2-paths we need the additional nodes

$$R_i = \{r_{i,j}, \overline{r_{i,j}} \mid 1 \leq j \leq n\}.$$

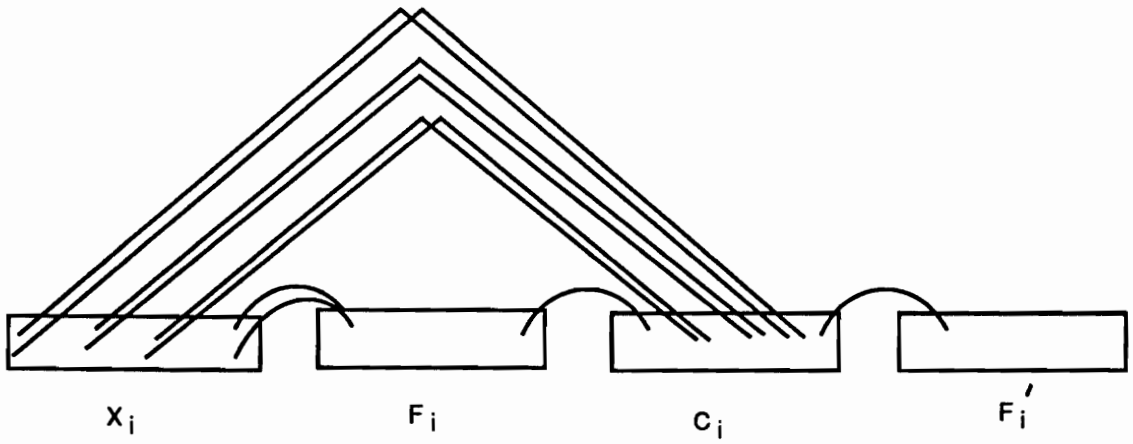


Figure 4.33: The truth-setting dag TS_i.

The 2-paths connecting X_i and X_{i+1} are:

$$(x_{i,j}, r_{i,j}) \quad (r_{i,j}, x_{i+1,j})$$
$$(\overline{x_{i,j}}, \overline{r_{i,j}}) \quad (\overline{r_{i,j}}, \overline{x_{i+1,j}}).$$

These 2-paths from TS_i to TS_{i+1} serve the purpose of causing a 2-path nesting whose size depends on the relative order of the pair of nodes $(x_{i,j}, \overline{x_{i,j}})$ in the literal dag X_i and the pair of nodes $(x_{i+1,j}, \overline{x_{i+1,j}})$ in the literal dag X_{i+1}. If all the pairs of nodes occur in the same relative order, then the size of the nesting is 1, otherwise it is 2. As we shall establish later, this connection between the relative order of pairs of nodes in the literal dags X_i and X_{i+1} and the size of the 2-path nesting between X_i and X_{i+1} is responsible for truth-values "flowing" consistently from clause to clause.

This completes the description of $\vec{G}$.

Given a topological ordering σ of the nodes of $\vec{G}$, a clause c_i and a variable x_j, c_i is said to contain a variable that appears in *true order* with respect to σ if c_i contains the positive literal x_j and $x_{i,j}$ appears prior to $\overline{x_{i,j}}$ in σ or c_i contains the negative literal $\overline{x_i}$ and $\overline{x_{i,j}}$ appears prior to $x_{i,j}$ in σ.

Consider the following two properties that a topological ordering σ of the nodes of $\vec{G}$ may possess:

1. Every clause c_i contains at least one variable that occurs in true order in σ.

2. For each j, $1 \leq j \leq n$, either $x_{i,j}$ occurs before $\overline{x_{i,j}}$ for all i, $1 \leq i \leq m$ or $x_{i,j}$ occurs after $\overline{x_{i,j}}$ for all i, $1 \leq i \leq m$ in σ. In other words, the pair of nodes $x_{i,j}$ and $\overline{x_{i,j}}$ occur in the same relative order in all literal dags X_i, $1 \leq i \leq m$ in σ.

Clearly, a topological ordering of $\vec{G}$, σ, satisfies properties (1) and (2), if and only if the given instance of 3-SAT is satisfiable.

We now show that if σ is a topological ordering of the nodes of $\vec{G}$ that yields a queue layout of $\vec{G}$ in $f(s+t+6)$ queues, then σ satisfies properties (1) and (2). If σ does not satisfy property (2), then it must be the case that there exist some $k,p, 1 \leq k \leq m, 1 \leq p \leq n$ such that the pairs of nodes $(x_{k,p}, \overline{x_{k,p}})$ and $(x_{k+1,p}, \overline{x_{k+1,p}})$ occur in reverse order relative to each other. In other words, if $x_{k,p}$ precedes $\overline{x_{k,p}}$ in σ, then $x_{k+1,p}$ follows $\overline{x_{k+1,p}}$ in σ and vice versa. This implies that between TS_k and TS_{k+1} there is a there is a 2-path nesting of size at least 2 caused by the nodes in X_k and X_{k+1}. This 2-path nesting nests over the big enforcer dag F'_k to give a total nesting of size $s+t+7$. Therefore the subgraph of $\vec{G}$ induced by the nodes in TS_k and TS_{k+1} requires at least $f(s+t+7)$ queues to be laid out. But, by our choice of s and t, $f(s+t+7) = f(s+t+6)+1$. Therefore, if σ yields a layout of $\vec{G}$ in $f(s+t+6)$ queues, then σ satisfies condition (2).

We now show that if σ yields a queue layout of $\vec{G}$ in $f(s+t+6)$ queues, then σ satisfies condition (1).

But, prior to showing that we show an additional property that σ has if it yields a layout of $\vec{G}$ in $f(s+t+6)$ queues. If σ yields a $f(s+t+6)$-queue layout of $\vec{G}$, then all nodes in R_i for all $i, 1 \leq i \leq m$ have to appear between $u_{i,s+t+5}$ (the first node in the layout of F'_i) and $w_{i,s+t+5}$ (the last node in the layout of F'_i). This is because, if a node $r \in R_i$ appears to the left of $u'_{i,s+t}$, then there is an arc from r to a node in TS_{i+1} that nests over F'_i. Similarly, if a node $r \in R$ appears to the right of $w_{i,s+t}$, then there is an arc from a node in TS_i to r that nests over F'_i. Since, the queuenumber of F'_i is $f(s+t+5)$, the arc incident on r that nests over F'_i increases the nesting size to $f(s+t+5)+1 = f(s+t+6)+1$. Therefore, if σ

yields a $f(s+t+6)$-queue layout of $\vec{G}$ then all nodes in R occur between $u_{i,s+t}$ and $v_{i,s+t}$ in σ.

If σ does not satisfy property (1), then it must be the case that there exists a clause c_k, such that none of the variables that it contains occur in the true order in σ. Then the 2-paths between the literal dag X_k and the clause dag C_k yield a 2-path nesting of size 6. This nesting of 2-paths of size 6 nests over the small enforcer dag F_i to yield a 2-path nesting of total size $s+6$. The subgraph induced by the nodes in X_k, the nodes in C_k, and the nodes in F_k requires at least $f(s+6)$ queues to be laid out. In addition, the arc $(x_{i,0}, r_{i,0})$ nests over any layout of the subgraph described above, to yield a total nesting of size $f(s+6)+1$. Since $f(s+6) = f(s+t+6)$, it is the case that if σ yields a $f(s+t+6)$ queue layout of $\vec{G}$, then σ satisfies property (1).

We now show that if σ is an ordering that satisfies properties (1) and (2), then σ yields a $f(s+t+6)$-queue layout. If σ satisfies properties (1) and (2), then the largest nesting of 2-paths between X_i and C_i for any $i, 1 \leq i \leq n$ is of size 5 and the largest nesting of 2-paths between X_i and X_{i+1} for any $i, 1 \leq i \leq n-1$ is of size 1. The following is an assignment of arcs of $\vec{G}$ to $f(s+t+6)$ queues such that if $\vec{G}$ is laid out according to σ, then no two arcs assigned to the same queue nest. For some $i, 1 \leq i \leq n$, consider the subgraph of $\vec{G}$ induced by the nodes in X_i, C_i, Z_i, and F_i. The largest 2-path nesting in any layout of this subgraph is of size $s+5$ and hence this subgraph can be laid out in $f(s+5)$ queues. Now consider the subgraph induced by the the 2-paths from X_i to X_{i+1} and the nodes in F_i'. The largest 2-path nesting in any layout of this subgraph in of size $s+t+6$. Therefore, this subgraph can be laid out in $f(s+t+6)$ queues. Queues can be reused for the assignment of arcs in the two subgraphs to yield a layout for the whole dag in $f(s+t+6)$ queues.

Since, satisfying properties (1) and (2) corresponds to the existence of a truth satisfying assignment for all the clauses, we have that there exists a truth assignment that satisfies all clauses if and only if there exists an ordering of the nodes in $\vec{G}$ that yields a yields a $f(s+t+6)$ queue layout. This completes the reduction. Clearly, the reduction can be achieved in polynomial time, thereby showing that DAGQN is NP-complete. □

We now show that DAGSN is also NP-complete.

Theorem 4.29 *DAGSN is NP-complete.*

Proof: The construction of a dag $\vec{G}$ from an instance of 3-SAT is simpler in this case than the corresponding construction in Theorem 4.28 because we no longer need to avoid transitive arcs. Corresponding to each clause c_i, $\vec{G}$ contains a truth-setting dag TS_i. The truth-setting dag can be thought of as containing three dags connected together; a literal dag X_i, a clause dag C_i, and an enforcer dag E_i. The clause dag C_i is as defined in the proof of Theorem 4.28, but the literal dag X_i is more complicated. It contains two subgraphs A_i and B_i that are isomorphic to each other. We describe the construction of A_i and then how the nodes of A_i and B_i connect together to form X_i. To begin with, A_i contains the nodes

$$\{x_{i,j} \mid 1 \leq j \leq n\} \cup \{x_{i,j} \mid 1 \leq j \leq n\}$$

connected by the arcs

$$\{(x_{i,j}, x_{i,j+1}), (x_{i,j}, \overline{x_{i,j+1}}), (\overline{x_{i,j}}, x_{i,j+1}), (\overline{x_{i,j}}, \overline{x_{i,j+1}}) \mid 1 \leq j \leq n-1\}.$$

Additionally, it contains the 6 nodes $\{a_{i,j} \mid 1 \leq j \leq 6\}$. The nodes $a_{i,1}$ and $a_{i,2}$ are connected to the rest of the nodes by virtue of the arcs

$$(a_{i,1}, a_{i,2}), (a_{i,2}, x_{i,1}), (a_{i,2}, \overline{x_{i,1}}),$$

while $a_{i,3}, a_{i,4}, a_{i,5}$, and $a_{i,6}$ are connected to the rest of the nodes by virtue of the arcs

$$(x_{i,n}, a_{i,3}), (\overline{x_{i,n}}, a_{i,3}), (a_{i,3}, a_{i,4}), (a_{i,4}, a_{i,5}), (a_{i,5}, a_{i,5}).$$

Finally, A_i also contains the arcs $(a_{i,1}, a_{i,3})$ and $(a_{i,3}, a_{i,4})$. Any layout of A_i will contain node $a_{i,1}$ followed by node $a_{i,2}$, followed by the nodes $x_{i,j}$ and $\overline{x_{i,j}}$, for all j, $1 \leq j \leq n$, followed by the nodes $a_{i,3}, a_{i,4}, a_{i,5}, a_{i,6}$ in that order. To construct B_i, make another copy of A_i and change the labels of nodes from $a_{i,j}$ to $b_{i,j}$ for $1 \leq j \leq 6$, from $x_{i,j}$ to $y_{i,j}$ for

$1 \leq j \leq n$ and from $\overline{x_{i,j}}$ to $\overline{y_{i,j}}$ for $1 \leq i \leq n$. The literal dag X_i contains arcs from nodes in A_i to corresponding nodes in B_i.

The connections between the literal dag X_i and the clause dag C_i are now described. The arc $(a_{i,6}, c_{i,1})$ ensures that in any layout of TS_i, the clause dag C_i appears after A_i. The remaining arcs between the literal dag and the clause dag depend on the literals in the clause c_i. As in the proof of Theorem 4.28, these connections are best described with an example. Let $c_i = \{x_2, \overline{x_4}, x_4\}$. Then TS_i contains the arcs

$$(x_{i,2}, c_{i,2}) \quad (\overline{x_{i,2}}, c_{i,1})$$

corresponding to the positive literal x_2,

$$(x_{i,4}, c_{i,3}) \quad (\overline{x_{i,4}}, c_{i,4})$$

corresponding to the negative literal $\overline{x_4}$, and the arcs

$$(x_{i,7}, c_{i,6}) \quad (\overline{x_{i,7}}, c_{i,5})$$

corresponding to the positive literal x_7. Note that the connections corresponding to each literal in the clause intersect with the connections corresponding to other literals in any layout of TS_i. Further note that depending upon the order of the nodes in the literal dag, the connections describe above may cause a nesting of size at least 3 and at least 6.

The enforcer dag E_i simply contains two disjoint directed paths each of length 7. More precisely, the enforcer dag E_i contains the nodes $\{e_{i,j} \mid 1 \leq j \leq 16\}$ and the arcs

$$\{(e_{i,j}, e_{i,j+1}) \mid 1 \leq i \leq 7\} \cup \{(e_{i,j}, e_{i,j+1}) \mid 8 \leq i \leq 15\}.$$

The enforcer dag E_i is connected to the rest of the truth-setting dag TS_i by the arcs

$$(c_{i,5}, e_{i,1}) \quad (e_{i,8}, b_{i,1}) \quad (b_{i,6}, e_{i,9}).$$

This completes the description of a truth-setting dag TS_i.

Each truth-setting dag TS_i has arcs to the truth-setting dag TS_{i+1}. The arcs that participate in this connection are

$$\{(y_{i,j}, x_{i+1,j}) \mid 1 \leq j \leq n\} \cup \{(\overline{y_{i,j}}, \overline{x_{i+1,j}}) \mid 1 \leq j \leq n\}$$

and the arcs

$$(b_{i,5}, a_{i,6}) \quad (b_{i+1,6}, a_{i+1,4}) \quad (b_{i,6}, a_{i+1,1})$$

To complete the description of the dag $\vec{G}$ we need to describe a final component. This a called the *gate keeper* dag and is simply a directed path of length 7. The contains the nodes $\{g_i \mid 1 \leq j \leq 8\}$ and arcs $\{(g_i, g_{i+1}) \mid 1 \leq i \leq 7\}$. It is connected to the enforcer dag in each truth-setting dag TS_i by the following arcs

$$\{(g_j, e_{i,j}), (g_j, e_{i,j+8}) \mid 1 \leq j \leq 8, 1 \leq i \leq m\}.$$

In addition it is connected to the first truth-setting dag TS_1 by the arc $(g_8, a_{1,1})$. This ensures that the gate keeper occurs before the rest of the dag in any layout of $\vec{G}$ This completes the description of $\vec{G}$.

Based on an ordering σ of nodes of $\vec{G}$ a variable x_j is defined to occur in *true order* in a clause c_i as in the proof of Theorem 4.28. Consider the following two properties that an ordering of the nodes of $\vec{G}$ may have.

1. Every clause c_i contains at least one variable that occurs in true order.

2. For each j, $1 \leq j \leq n$, either $x_{i,j}$ occurs before $\overline{x_{i,j}}$ for all i, $1 \leq i \leq m$ or $x_{i,j}$ occurs after $\overline{x_{i,j}}$ for all i, $1 \leq i \leq m$ in σ. In other words, the pair of nodes $x_{i,j}$ and $\overline{x_{i,j}}$ occur in the same relative order in all literal dags X_i, $1 \leq i \leq m$ in σ.

We show that if σ is an ordering of the nodes of $\vec{G}$ that yields a 9-stack layout, then σ satisfies properties (1) and (2) If σ does not satisfy property (1), then there exists a a clause c_k, such that none of the variables that it contains occur in true order σ. Then the arcs between the literal dag X_k and the clause dag C_k yield a twist of size 6. The arcs

$$(a_{k,1}, a_{k,3}) \quad (a_{k,2}, a_{k,4}) \quad (b_{k-1,6}, a_{k,5}) \quad (a_{k,6}, b_{k,6})$$

add to the size of this twist thereby producing a twist of size 10. If σ does not satisfy property (2), then for some k, $1 \leq k \leq m$ and some j, $1 \leq j \leq n$ either the pair $(x_{k,j}, \overline{x_{k,j}})$ and the pair $(y_{k,j}, \overline{y_{k,j}}$ do not occur in the same relative order or the pair $(y_{k,j}, \overline{y_{k,j}})$ and the pair $(x_{k+1,j}, \overline{x_{k+1,j}})$ do not occur in the same relative order in σ. In either case they participate in a twist of size 10.

Now we show that if σ satisfies properties (1) and (2) then σ yields a 9-stack layout of $\vec{G}$. 2 stacks suffice for the arcs within A_i and the same two can be reused for the arcs within B_i. 5 stacks suffice for th arcs between the literal dag X_i and the clause dag C_i, while 1 stack is sufficient for the arcs between A_i and B_i. These 8 stacks can be reused for one portion of the enforcer dag. A ninth stack can be used for the connections between the truth-setting dags and the same stack can be used for the other portion of the enforcer dag.

Clearly, if there exists a truth assignment for the variables that satisfies all the clauses, then there exists an ordering σ that satisfies properties (1) and (2). Hence, there exists a truth assignment for the variables that satisfies all clauses if and only if $\vec{G}$ has a layout in 9 stacks. Thus DAGSN is NP-complete. □

4.7 Conclusion

In this chapter, we have initiated the study of stack and queue layouts of dags that was promised in Heath and Rosenberg [36]. In keeping with the spirit of Chung, Leighton, and Rosenberg [16] and Heath and Rosenberg [36], we obtain structural as well as algorithmic results in this area. As is apparent throughout the chapter, the presence of arcs (directed edges) instead of edges lends a distinct flavor to the problems in stack and queue layouts of dags as compared to stack and queue layouts of undirected graphs. For example, obtaining small upper bounds on the stacknumber or on the queuenumber of classes of dags is more difficult than obtaining small upper bounds on the stacknumber and the queuenumber of the corresponding classes of covering graphs. On the other hand, it is easier to algorithmi-

cally recognize dags with small stacknumber and queuenumber as compared to undirected graphs with small stacknumber and queuenumber. An evidence of this is our algorithm that recognizes leveled-planar dags in linear time in the size of the node set of the dags as compared to the NP-completeness of the problem of recognizing leveled-planar graphs.

There are several open questions that arise out of this work and here we emphasize two that we consider important.

1. What is the time complexity of recognizing 1-queue dags?

2. What is the stacknumber of the class of outerplanar dags?

We conjecture that our algorithm that recognizes leveled-planar dags can be extended to recognize arched leveled-planar dags.

Conjecture 1 *The problem of determining whether a dag is a 1-queue dag can be solved in polynomial time.*

Based on our examination of a subclass of the class of outerplanar dags, we conjecture that the stacknumber of the class of outerplanar dags is bounded above by a small constant.

Conjecture 2 *Let $\mathcal{O}$ be the class of outerplanar dags. Then there exists a constant c such that $SN_{\mathcal{O}}(n) \leq c$, for all n.*

Chapter 5

Stack and Queue Layouts of Posets

In this chapter, we study stack and queue layouts of posets. Posets are ubiquitous mathematical objects and various measures of their structure have been defined. In fact, in chapter 2, we make extensive use of posets to prove results related to matrix covers. Some of these measures are bumpnumber, jumpnumber, length, width, dimension, and thickness [7,26]. Nowakowski and Parker [53] define the stacknumber of a poset as the stacknumber of its Hasse diagram viewed as a dag. They derive a lower bound on the stacknumber of a planar poset and an upper bound on the stacknumber of a lattice. Nowakowski and Parker conclude by asking whether the stacknumber of the class of planar posets is unbounded. Hung [38] shows that there exists a planar poset with stacknumber 4; moreover, no planar poset with stacknumber 5 is known. Sysło [60] provides a lower bound on the stacknumber of a poset in terms of its bumpnumber. He also shows that while posets with jumpnumber 1 have stacknumber at most 2, posets with jumpnumber 2 can have an arbitrarily large stacknumber.

The organization of this chapter is as follows. Section 5.1 contains notation and definitions related to stack and queue layouts of posets. In Section 5.2, we derive upper bounds on the queuenumber of a poset in terms of its jumpnumber, its length, its width, and the queuenumber of its covering graph. In Section 5.3, we show that the queuenumber of the class of planar posets is unbounded. A complementary upper bound result shows that the queuenumber of a planar poset is within a small constant factor of the width of the poset. In Section 5.4, we show that the stacknumber of the class of posets with planar covering graphs is $\Theta(n)$. In Section 5.5, we present several open questions and conjectures concerning stack and queue layouts of posets.

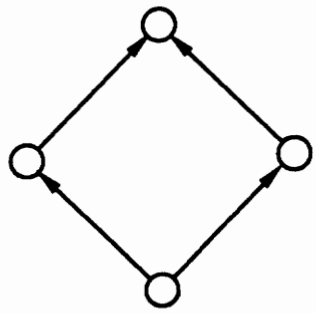


Figure 5.1: A 2-stack poset.

5.1 Definitions

In Chapter 1 we defined posets and some terminology related to posets. This section contains the definitions of stack and queue layouts of posets. Other relevant measures of the structure of posets are also defined. Let $P = (V, \leq)$ be a poset. The *size* of a poset, denoted by $|P|$ is simply $|V|$. In this chapter, we restrict our attention to posets of finite size. The Hasse diagram of P is a dag $\vec{H}(P) = (V, \vec{E})$ such that if $(u, v) \in \vec{E}$, then $u \leq v$ and there is a no other directed path from u to v. Thus, $\vec{H}(P)$ contains no arcs that ae implied by transitivity and can be thought of as a minimal representation of P. The *stacknumber* $SN(P)$ of P is $SN\left(\vec{H}(P)\right)$; P is a *k-stack poset* if $SN(P) = k$. Similarly, the *queuenumber* $QN(P)$ of P is $QN\left(\vec{H}(P)\right)$; P is a *k-queue poset* if $QN(P) = k$. The covering graph, $H(P)$, of $\vec{H}(P)$ is called the *covering graph* of P. Clearly, for any poset P,

$$SN(H(P)) \leq SN(P)$$

and

$$QN(H(P)) \leq QN(P).$$

An example of a 2-stack poset is given in Figure 5.1. An example of a 2-queue poset is given in Figure 5.2. Each of these two posets has a covering graph that is a 1-stack and a 1-queue graph.

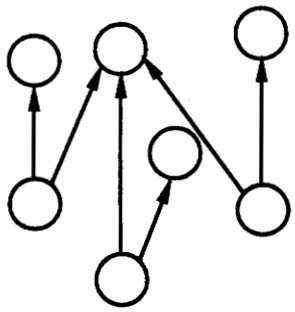


Figure 5.2: A 2-queue poset.

A *diagram invariant* is a property of posets that is shared by all posets with the same covering graph. It is easy to see that neither stacknumber nor queuenumber is a *diagram invariant*. In fact, in the subsequent sections we show that stacknumber and queuenumber are not even *approximate diagram invariants* in the sense that we are able to exhibit pairs of posets that share a covering graph, but one poset has a constant stacknumber (queuenumber), while the other has a stacknumber (queuenumber) that is arbitrarily large (see Theorem 5.8 and Theorem 5.12).

A poset P is *planar* if its Hasse diagram $\vec{H}(P)$ has a planar embedding in which all arcs are drawn as straight line segments with the tail of each arc strictly below its head with respect to a Cartesian coordinate system. Note that $H(P)$ may be planar even though the poset P is not.

Example 5.1. The posets shown in Figure 5.1 and in Figure 5.2 are both planar. But, Figure 5.3 shows the Hasse diagram of a poset that is not planar. Nevertheless, it is easy to see that the covering graph of this poset is planar graph. □

Let γ be a fixed topological order on $\vec{H}(P)$. Two elements u and v are *adjacent* in γ if there is no w such that $u <_\gamma w <_\gamma v$ or $v <_\gamma w <_\gamma u$. A *spine arc* in $\vec{H}(P)$ with respect to γ is an arc (u, v) in $\vec{H}(P)$ such that u and v are adjacent in γ. A *break* in $\vec{H}(P)$ with respect to γ is a pair (u_1, u_2) of adjacent elements such that $u_1 <_\gamma u_2$ and (u_1, u_2) is not an arc in $\vec{H}(P)$. A *connection* C in $\vec{H}(P)$ with respect to γ is a maximal sequence of elements

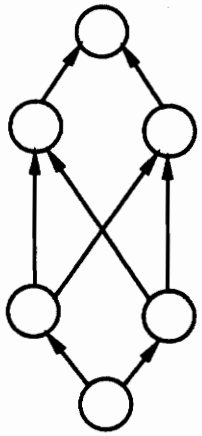


Figure 5.3: A non-planar poset whose covering graph is planar.

$u_1 <_\gamma u_2 <_\gamma \ldots <_\gamma u_k$ such that (u_i, u_{i+1}) is a spine arc for all $i, 1 \leq i < k$; in other words, a connection is a maximal path of spine arcs. The *breaknumber* $BN(\gamma, P)$ of a topological order γ on $\vec{H}(P)$ is the number of breaks in $\vec{H}(P)$ with respect to γ. The *jumpnumber* $JN(P)$ of P is the minimum of $BN(\gamma, P)$ over all topological orders γ on $\vec{H}(P)$.

Recall the definitions of a chain and an antichain (given in Section 1.1). The *length* of a poset P, denoted by $L(P)$, is the size of the largest chain in P. The *width* of a poset P, denoted by $W(P)$, is the size of the largest antichain in P. Just as stacknumber and queuenumber are defined for classes of graphs (see Section 1.1) and for classes of dags (see Section 4.1), stacknumber and queuenumber can be defined for classes of posets also. Similarly, length, width, and jumpnumber can be defined for classes of posets.

5.2 Upper Bounds on Queuenumber

In this section, we derive upper bounds on the queuenumber of a poset in terms of its jumpnumber, its length, its width, and the queuenumber of its covering graph.

5.2.1 Jumpnumber and Queuenumber

Some results of Sysło [60] relate the stacknumber of a poset to its jumpnumber.

Proposition 5.2 *(Sysło [60])* *Suppose P is a poset with jumpnumber 1; then the stack-number of P is at most 2. There exists a class $\mathcal{P}$ of posets such that for all $n \geq 1$,*

$$JN_{\mathcal{P}}(n) = 2,$$

but

$$SN_{\mathcal{P}}(n) = \Omega(n).$$

In contrast to the second result of Sysło, we show that, for any poset P, $JN(P) + 1$ is an upper bound on $QN(P)$. This bound is tight within a constant factor.

Theorem 5.3 *Suppose P is a poset; then, $QN(P) \leq JN(P) + 1$. For every n, there exists a poset P such that $|P| = 2n$ and $JN(P)/2 \leq QN(P)$.*

Proof: To show the upper bound, suppose P is a poset with $JN(P) = k$. Let γ be a topological order on $\vec{H}(P)$ that has exactly k breaks and hence $k+1$ connections. Lay out $\vec{H}(P)$ according to γ and label the connections $C_0, C_1, \ldots, C_k$ from left to right. Suppose (u_1, v_1) and (u_2, v_2) are two nonspine arcs such that u_1 and u_2 are in C_i, v_1 and v_2 are in C_j, and $1 \leq i < j \leq k$. If (u_1, v_1) and (u_2, v_2) nest, then one of (u_1, v_1) and (u_2, v_2) is a transitive arc. Since $\vec{H}(P)$ contains no transitive arc, no pair of arcs from C_i to C_j nest, and all arcs from C_i to C_j can be assigned to a single queue. Assign all arcs between a pair of connections C_i and C_j, where $i \neq j$ to queue $q_{|i-j|}$. Assign all the spine arcs to a queue q_0. Hence, we use at most k queues for non-spine arcs and one queue for spine arcs, for a total of at most $k+1$ queues, as claimed.

To show a lower bound, construct a poset P whose Hasse diagram is the complete bipartite graph $K_{n,n} = (V_1, V_2, E)$ with each edge directed from V_1 to V_2. Hence, $JN(P) = 2(n-1)$, $QN(P) = n$, and

$$QN(P) = \frac{n}{2(n-1)} JN(P).$$

□

Proposition 5.2 and Theorem 5.3 have the following corollary.

Corollary 5.4 *There exists a class of posets $\mathcal{P}$ for which the ratio $SN(\mathcal{P})/QN(\mathcal{P})$ is unbounded.*

Theorem 5.8, in contrast, shows a class of posets $\mathcal{Q}$ for which $QN(\mathcal{Q})/SN(\mathcal{Q})$ is unbounded.

5.2.2 Length and Queuenumber

In this subsection, we show an upper bound on the queuenumber of a poset in terms of its length and the queuenumber of its covering graph. The result appeals to Lemma 3.19.

Theorem 5.5 *For any poset P,*

$$QN(P) \leq 2 \cdot (L(P) - 1) \cdot QN(H(P)).$$

There exists a class of posets $\mathcal{P} = \{P \mid |P| = n,$ for each $n \geq 1\}$ such that $L_{\mathcal{P}}(n) = 2$ and for all $P \in \mathcal{P}$,

$$\left\lceil \frac{QN(P)}{2} \right\rceil = (L(P) - 1) \cdot QN(H(P)).$$

Proof: Suppose P is any poset and $QN(H(P)) = k$. Let $\vec{H}(P) = (V, \vec{E})$, and let σ be a total order on V that yields a k-queue layout of $H(P)$. The nodes of $\vec{H}(P)$ can be labeled by a function $l : V \rightarrow \{1, \ldots, L(P)\}$ such that $l(u) < l(v)$ if $u < v$ in P, as follows. Let $\vec{H}_0 = \vec{H}(P)$. Label all the nodes with indegree 0 in $\vec{H}_0$ with the label 1. Delete all the labeled nodes in $\vec{H}_0$ to obtain $\vec{H}_1$. In general, label the nodes with indegree 0 in $\vec{H}_i$ with the label $i + 1$. Delete the labeled nodes in $\vec{H}_i$ to obtain $\vec{H}_{i+1}$. By an inductive proof, it can be checked that the labeling so obtained satisfies the required conditions. Let $V_i = \{u \in V \mid l(u) = i\}$. For any arc $(u, v) \in \vec{E}$, if $u \in V_i$ and $v \in V_j$, then $i < j$. Therefore $\vec{H}(P) = \left(V_1, V_2, \ldots, V_{L(P)}, \vec{E}\right)$ is an $L(P)$-partite dag. Construct a total order γ on the nodes of $\vec{H}(P)$ such that

1. The elements in each set V_i, $1 \leq i \leq L(P)$ occur contiguously and in the order prescribed by σ.

2. The elements in V_i occur before the elements in V_{i+1} for all i, $1 \leq i < L(P)$.

Since every arc in $\vec{H}(P)$ is from a node in V_i to a node in V_j, $1 \leq i < j \leq L(P) - 1$, γ is a topological order on $\vec{H}(P)$. By Lemma 3.19, γ yields a layout that requires no more than $2 \cdot (L(P) - 1) \cdot k$ queues.

We now prove the second part of the theorem. Construct for each n, $n \geq 1$, the Hasse diagram of a poset P of size n as follows: Let $p = \lfloor n/2 \rfloor$ and let $q = \lceil n/2 \rceil$. Let $K_{p,q} = (V_1, V_2, E)$ such that $|V_1| = p$ and $|V_2| = q$. We get the Hasse diagram of a poset of size n by directing the edges in $K_{p,q}$ from V_1 to V_2. Clearly, $L(P) = 2$ and $QN(P) = p$. In Chapter 2 and Chapter 3, we present two rather different proofs of the fact: $QN(K_{p,q}) = \min(\lceil p/2 \rceil, \lceil q/2 \rceil)$. In this case, since $p \leq q$, $QN(K_{p,q}) = \lceil p/2 \rceil$. Therefore,

$$\left\lceil \frac{QN(P)}{2} \right\rceil = (L(P) - 1) \cdot QN(H(P)).$$

Let $\mathcal{P}$ be the class of all posets constructed in the manner described above. The second part of the theorem follows. □

Note that Theorem 5.5 holds for dags as well as for posets as its proof does not rely on the absence of transitive arcs.

Corollary 5.6 *For any poset P,*

$$QN(H(P)) \leq QN(P) \leq 2 \cdot (L(P) - 1) \cdot QN(H(P)).$$

Suppose $\mathcal{P}$ is a class of posets. If there exists a constant K such that $L(P) \leq K$, for all $P \in \mathcal{P}$, then $QN(P) = \Theta(QN(H(P))$ for all $P \in \mathcal{P}$.

5.2.3 Width and Queuenumber

In this section, we establish an upper bound on the queuenumber of a poset in terms of its width.

Theorem 5.7 *The largest rainbow in any layout of a poset P has size at most $W(P)^2$. Hence, $QN(P) \leq W(P)^2$.*

Proof: Dilworth's theorem (Proposition 1.4) allows us to partition a poset P into $W(P)$ chains. For a poset $P = (S, \leq)$, let $S_1, S_2, \ldots, S_{W(P)}$ be a partition of S into $W(P)$ chains. Define an *i-chain arc* as an arc in $\vec{H}(P)$, both of whose end points belong to chain S_i, $1 \leq i \leq W(P)$. An *(i,j)-cross arc*, $i \neq j$ is an arc whose tail belongs to chain S_i and whose head belongs to chain S_j. Fix an arbitrary topological order of $\vec{H}(P)$. For any i, no two i-chain arcs nest. Therefore, the largest rainbow of chain arcs has size no greater than $W(P)$. If $i \neq j$, then no two (i,j)-cross arcs can nest without one of them being a transitive arc. Therefore, the largest rainbow of cross arcs has size no greater than $W(P)(W(P)-1)$. The size of the largest rainbow is at most $W(P) + W(P)(W(P)-1) = W(P)^2$. By Proposition 1.15, the theorem follows. □

The bound established in the above theorem is not known to be tight. In fact, we conjecture that the queuenumber of a poset is bounded above by its width (see Conjecture 3 in Section 5.5).

5.3 The Queuenumber of Planar Posets

In this section, we first show that the queuenumber of the class of planar posets is unbounded. We then establish an upper bound on the queuenumber of a planar poset in terms of its width.

5.3.1 A Lower Bound on the Queuenumber of Planar Posets

We construct a sequence of planar posets P_n with $|P_n| = 3n+3$ and $QN(P_n) = \Theta(\sqrt{n})$. In fact, we determine the queuenumber of P_n almost exactly. To prove the theorem, we need the Theorem of Erdös and Szekeres (see Proposition 1.7).

Theorem 5.8 *There exists a planar poset P_n with $3n+3$ elements such that*

$$\lfloor \sqrt{n} \rfloor + 1 \leq QN(P_n) \leq \lfloor \sqrt{n} \rfloor + 2.$$

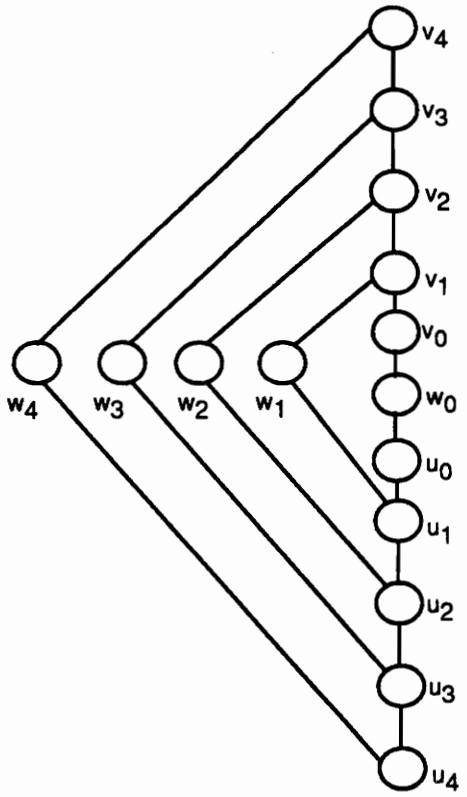


Figure 5.4: The planar poset P_4.

Proof: Define the disjoint sets U, V, and W as follows.

$$\begin{aligned} U &= \{u_i \mid 0 \leq i \leq n\} \\ V &= \{v_i \mid 0 \leq i \leq n\} \\ W &= \{w_i \mid 0 \leq i \leq n\} \end{aligned}$$

Let $S = U \cup V \cup W$. The planar poset $P_n = (S, \leq)$ is given by

$$\begin{aligned} u_i &< u_{i-1}, && 1 \leq i \leq n \\ v_i &< v_{i+1}, && 0 \leq i \leq n-1 \\ u_i &< w_i < v_i, && 1 \leq i \leq n \end{aligned}$$

Figure 5.4 shows the Hasse diagram of P_4. Let σ be an arbitrary topological order on S. The elements of $U \cup V \cup \{w_0\}$ appear in the order $u_n, u_{n-1}, \ldots, u_0, w_0, v_0, v_1, \ldots, v_n$ in σ,

and all elements of W appear between u_n and v_n. Define a total order δ on the elements of W by $w_i <_\delta w_j$ if $i < j$. Suppose

$$w_{i_1}, w_{i_2}, \ldots, w_{i_k}$$

is an increasing sequence of nodes in W with respect to δ. Since w_{i_1} appears after u_{i_1} in any topological order of $\vec{H}(P_n)$, the following sequence of nodes is a subsequence of σ:

$$u_{i_k}, u_{i_{k-1}}, \ldots, u_{i_1}, w_{i_1}, w_{i_2}, \ldots, w_{i_k}.$$

Therefore, the set $\{(u_{i_j}, w_{i_j}) \mid 1 \leq j \leq k\}$ is a k-rainbow in σ. Similarly, if

$$w_{i_1}, w_{i_2}, \ldots, w_{i_k}$$

is a decreasing sequence of nodes in W with respect to δ, then the set $\{(w_{i_j}, v_{i_j}) \mid 1 \leq j \leq k\}$ is a k-rainbow in σ. By Proposition 1.7, there is an increasing subsequence of size $\lceil\sqrt{n}+1\rceil$ or a decreasing subsequence of size $\lceil\sqrt{n}+1\rceil$ in W with respect to δ. Thus there is a rainbow of size $\lceil\sqrt{n}+1\rceil \geq \lfloor\sqrt{n}\rfloor + 1$ in any topological order on $\vec{H}(P_n)$ and therefore $QN(P_n) \geq \lfloor\sqrt{n}\rfloor + 1$.

We now lay out P_n in $\lfloor\sqrt{n}\rfloor + 2$ queues. Let $s = \lceil\sqrt{n}\rceil$, and let $t = \lceil n/s\rceil \leq \lceil\sqrt{n}\rceil$. Partition $W - \{w_0\}$ into s nearly equal-sized subsets

$$W_1, W_2, \ldots, W_s$$

such that

$$W_i = \begin{cases} \{w_j \mid 1 \leq j \leq t\} & 1 \leq i \leq s-1 \\ \{w_j \mid (s-1)t \leq j \leq n\} & i = s \end{cases}$$

Construct an order σ on the elements of S by first placing the elements in $U \cup V \cup \{w_0\}$ in the order

$$u_n, u_{n-1}, \ldots, u_0, w_0, v_0, v_1, \ldots, v_n.$$

Now place the elements of $W - \{w_0\}$ between u_0 and v_0 such that the elements belonging to each set W_i appear contiguously and the sets themselves appear in the order

$$W_s, W_{s-1}, \ldots, W_1.$$

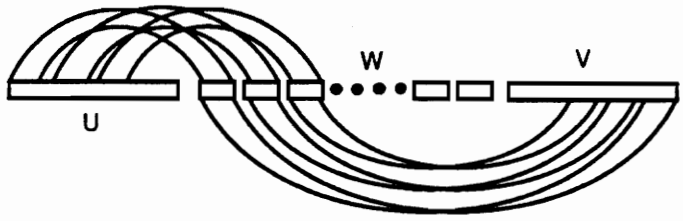


Figure 5.5: Layout of planar poset P_n.

Within each set W_i, $1 \leq i \leq s$, place the elements in increasing order. The arcs from U to W form s mutually intersecting rainbows each of size at most t. Therefore t queues suffice for these arcs. The arcs from W to V form s nested twists each of size at most t. Therefore s queues suffice for these arcs. Since an arc from U to W and an arc from W to V do not nest, the arcs from U to W can reuse the s queues that are used by the arcs from W to V. An additional queue is required for the remaining arcs. This is a layout of P_n in $\lceil \sqrt{n} \rceil + 1$ queues (Figure 5.5). Thus $QN(P_n) \leq \lfloor \sqrt{n} \rfloor + 2$. □

We conjecture that the upper bound in the above proof can be tightened to exactly match the lower bound. In fact, we have been able to show that for $m^2 \leq n \leq m(m+1)$, $QN(P_n) = m + 1 = \lfloor \sqrt{n} \rfloor + 1$.

The situation for stacknumber of planar posets is somewhat different in that there is no known example of a sequence of planar posets with unbounded stacknumber. Two observations about the sequence P_n in Theorem 5.8 are in order. The first observation is that $SN(P_n) = 2$. A 2-stack layout of $\vec{H}(P_4)$ is shown in Figure 5.6. The second observation is that the stacknumber *and* the queuenumber of $H(P_n)$ is 2. A 2-queue layout of $H(P_4)$ is shown in Figure 5.7. Theorem 5.8 and the above observations imply the following corollaries.

Corollary 5.9 *There exists a sequence of planar posets P_n, $n \geq 1$, $|P_n| = 3n+3$ such that*

$$\frac{QN(P_n)}{SN(P_n)} = \Omega\left(\sqrt{n}\right).$$

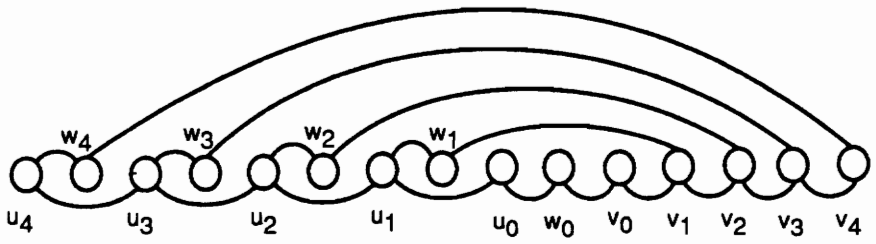


Figure 5.6: A 2-stack layout of the planar poset P_4.

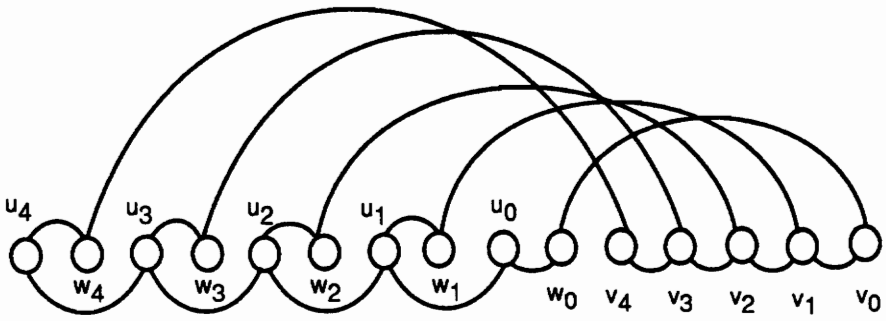


Figure 5.7: A 2-queue layout of the covering graph of P_4.

Corollary 5.10 *There exists a sequence of planar posets P_n, $n \geq 1$, $|P_n| = 3n + 3$, such that*

$$\frac{QN(P_n)}{QN(H(P_n))} = \Omega\left(\sqrt{n}\right).$$

While Theorem 5.8 establishes a lower bound of $\Omega(\sqrt{n})$ on the queuenumber of the class of planar posets with n elements, a matching upper bound is not known (see Conjecture 4 in Section 5.5).

5.3.2 An Upper Bound on the Queuenumber of Planar Posets

In this subsection, we show that the queuenumber of a planar poset is within a small constant factor of its width.

Theorem 5.11 *For any planar poset P, any topological order of $\vec{H}(P)$ has queuenumber no greater than $3 \cdot W(P) - 2$.*

Proof: Without loss of generality, assume that $\vec{H}(P) = (V, \vec{E})$ is embedded in the plane such that every arc is drawn as a straight arrow pointing upwards and no two nodes are on the same horizontal line. (If two nodes are on the same horizontal line, a slight vertical perturbation of either of them yields another planar embedding with the nodes on different horizontal lines). Furthermore, we may assume that $\vec{H}(P)$ has a unique source (the lowest node) and a unique sink (the highest node) by the following argument. If v is a source and not the lowest node, then some node u that is below v is visible from v along a straight line. If the arc (u, v) is added, then we get a Hasse diagram of a new poset with a planar embedding in which every arc is drawn as a straight arrow pointing upwards. The width of the new poset is no greater than $W(P)$, and it has one fewer source than $\vec{H}(P)$. By induction we may assume that $\vec{H}(P)$ has a unique source, call it s. By an analogous argument we may assume that $\vec{H}(P)$ has a unique sink, call it t.

By Proposition 1.4, V can be partitioned into $W(P)$ chains

$$V_1, V_2, \ldots, V_{W(P)}.$$

For each chain V_i, there is a directed path D_i from s to t that contains every element of V_i. We conclude that the nodes of $\vec{H}(P)$ can be covered by $W(P)$ directed paths, not necessarily disjoint, from s to t.

For an arbitrary directed path D from s to t, there is an obvious notion of a node or an arc being to the left of D, to the right of D, or on D. We say that a directed path D' from s to t is *to the left (right) of* D if every arc in D' is either to the left (right) of D or on D. Clearly, D' is to the left of D if and only if D is to the right of D'. Write $D' <_L D$ if D' is to the left of D. It is easy to see that $<_L$ is a partial order on paths from s to t.

Without loss of generality, we may assume that the paths $D_1, D_2, \ldots, D_{W(P)}$ covering the nodes of $H(P)$ are in the order

$$D_1 <_L D_2 <_L \cdots <_L D_{W(P)}$$

by the following argument. Suppose that D_i and D_j are unrelated in the partial order $<_L$ and that $i < j$. Construct two paths D_i' and D_j' from s to t. Place all arcs that are on both D_i and D_j in both D_i' and D_j'. Place all arcs in D_i to the left of D_j and all arcs in D_j to the left of D_i in D_i'. Place all arcs in D_i to the right of D_j and all arcs in D_j to the right of D_i in D_j'. It is easy to see that D_i' and D_j' are directed paths from s to t and that $D_i' <_L D_j'$. An inductive proof completes the argument.

We are now prepared to complete the proof of the theorem by an induction on $W(P)$. In the base case, $W(P) = 1$ and the entire Hasse diagram is covered by a single path D_1 from s to t. Assigning all arcs in D_1 to a single queue yields the claimed bound.

To complete the induction, assume that $W(P) > 1$ and that the bound holds for all posets of smaller width. Cover the nodes of $\vec{H}(P)$ with paths $D_1, D_2, \ldots, D_{W(P)}$ from s to t so that

$$D_1 <_L D_2 <_L \cdots <_L D_{W(P)}.$$

Clearly, $D_{W(P)}$ is the directed path from s to t along the rightmost arcs in the planar embedding of $\vec{H}(P)$. There must be at least one node in $D_{W(P)}$ that is in no other D_i (otherwise, the width of P is less than $W(P)$, a contradiction). Let P' be the poset derived

by removing all such nodes from P. We have that $W(P') = W(P) - 1$ and that the nodes of $\vec{H}(P')$ are covered by $D_1, D_2, \ldots, D_{W(P)-1}$. By the inductive assumption, any topological order of $H(P')$ has queuenumber at most $3 \cdot W(P') - 2 = 3 \cdot W(P) - 5$.

Consider any topological order of $\vec{H}(P)$. Assign the arcs of $\vec{H}(P')$ to $3 \cdot W(P) - 5$ queues. Any arcs not in $\vec{H}(P')$ fall into one of three classes: (i) on $D_{W(P)}$, (ii) *incoming* into $D_{W(P)}$, or (iii) *outgoing* from $D_{W(P)}$. Note that an incoming arc must have its tail on $D_{W(P)-1}$ and an outgoing arc must have its head on $D_{W(P)-1}$. Use one queue for each of these classes of arcs. By the fact that $\vec{H}(P)$ is in topological order, no two arcs *on* $D_{W(P)}$ can nest. Suppose (u, v) and (x, y) are two incoming arcs. Then u and x are on $D_{W(P)-1}$ and v and y are on $D_{W(P)}$. Suppose u precedes x on $D_{W(P)-1}$ and hence in the topological order. By the planarity of the embedding of $\vec{H}(P)$ with arcs drawn as straight arrows pointing upwards, v precedes y in topological order. Similarly, if x precedes u on $D_{W(P)-1}$, then y precedes v on $D_{W(P)}$. In either case (u, v) and (x, y) do not nest. Similarly, it can be shown that no two outgoing arcs nest. Hence, the assignment of arcs to queues described above results in a queue layout of $\vec{H}(P)$ in $3 \cdot W(P) - 2$ queues.

By induction, the theorem follows. □

In the above theorem, we show that any topological order can be used to obtain a $(3W(P) - 2)$-queue layout of $\vec{H}(P)$. We conjecture that a carefully constructed topological order will yield a $W(P)$-queue layout of $\vec{H}(P)$ (see Conjecture 3).

5.4 Stacknumber of Posets with Planar Covering Graphs

In this section, we construct $3n$-element posets P_n, $n \geq 1$, such that $H(P_n)$ is planar and hence $SN(H(P_n)) \leq 4$ (Yannakakis [64]), and yet $SN(P_n) = \Theta(n)$.

Theorem 5.12 *There exist posets P_n, $n \geq 1$, such that $|P_n| = 3n$, $H(P_n)$ is planar, and*

$$\lfloor n/2 \rfloor \leq SN(P_n) \leq n.$$

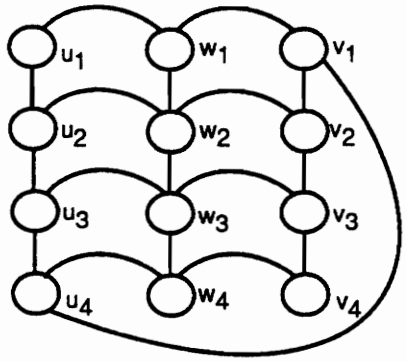


Figure 5.8: The covering graph of P_4.

Proof: Let U, V, and W be disjoint sets

$$\begin{aligned} U &= \{u_i \mid 1 \leq i \leq n\} \\ V &= \{v_i \mid 1 \leq i \leq n\} \\ W &= \{w_i \mid 1 \leq i \leq n\}. \end{aligned}$$

The poset $P_n = (U \cup V \cup W, \leq)$ is given by

$$\begin{aligned} u_i < u_{i+1}, \quad v_i < v_{i+1}, \qquad & w_i < w_{i+1}, \quad 1 \leq i \leq n-1 \\ u_i < w_i < v_i, \quad & 1 \leq i \leq n \\ & u_n < v_1. \end{aligned}$$

Figure 5.8 shows $H(P_4)$.

To prove the lower bound on $SN(P_n)$, suppose σ is any topological order on P_n. The order σ contains the elements of $U \cup V$ in the order $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n$, and the elements of W in the order $w_1, w_2, \ldots, w_n$. The elements of W are mingled among the elements $U \cup V$. If $w_{\lfloor n/2 \rfloor} <_\sigma u_n$, then

$$(w_1, v_1), (w_2, v_2), \ldots, (w_{\lfloor n/2 \rfloor}, v_{\lfloor n/2 \rfloor})$$

form an $\lfloor n/2 \rfloor$-twist. If $w_{\lfloor n/2 \rfloor} >_\sigma u_n$, then

$$(u_{\lfloor n/2 \rfloor+1}, w_{\lfloor n/2 \rfloor+1}), (u_{\lfloor n/2 \rfloor+2}, w_{\lfloor n/2 \rfloor+2}), \ldots, (u_n, w_n)$$

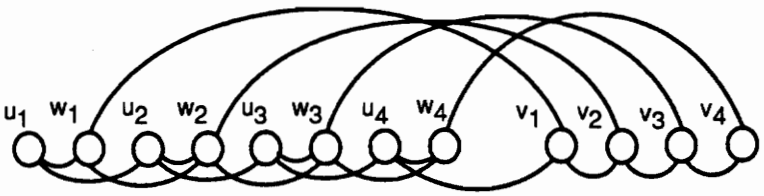


Figure 5.9: A 2-queue layout of P_4.

form an $\lceil n/2 \rceil$-twist. In either case, the layout contains an $\lfloor n/2 \rfloor$ twist. Therefore, $SN(P_n) \geq \lfloor n/2 \rfloor$.

An n-stack layout of P_n is obtained by laying out the elements of $U \cup V$ in the only possible order, and then placing each element w_i immediately after u_i for all i, $1 \leq i \leq n$. Figure 5.9 shows a 2-queue layout of P_4. By using the same total order as used for this layout, we obtain an n-stack layout of P_n. The assignment of arcs to stacks is as follows. Assign each arc in the set $\{(u_i, w_i), (w_i, v_i), (w_i, w_{i+1})\}$ to stack s_i for all i, $1 \leq i \leq n-1$ and assign each arc in the set $\{(u_n, w_n), (w_n, v_n)\}$ to stack s_n. Note that no two arcs assigned to the same stack intersect. The only arcs remaining to be assigned are the arcs in the set

$$\{(u_i, u_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(v_i, v_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(u_n, v_1)\}.$$

The arcs (v_i, v_{i+1}) for i, $1 \leq i \leq n-1$ do not intersect any other arc and can be assigned to any stack. Each arc (u_i, u_{i+1}) can be assigned to stack s_{i+1} for all i, $1 \leq i \leq n-1$ and arc (u_n, v_1) can be assigned to stack s_1. □

Two observations about the sequence of posets P_n constructed in the above proof are in order. The first observation is that $QN(P_n) = 2$. A 2-queue layout of P_4 is shown in Figure 5.9. In general, the n-stack layout of P_n described in the above proof yields a 2-queue layout of P_n. The second observation is that the stacknumber and the queuenumber of $H(P_n)$ is 2. A 2-stack layout of $H(P_4)$ is shown in Figure 5.10. In general, a 2-stack layout of $H(P_n)$ can be obtained because $H(P_n)$ is a hamiltonian planar graph [6].

Theorem 5.12 and the above observations lead to the following corollaries.

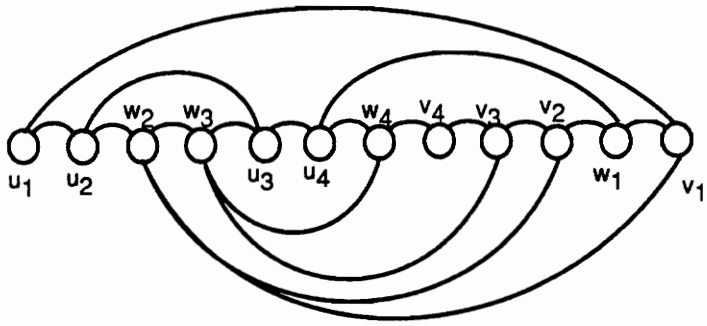


Figure 5.10: A 2-stack layout of the covering graph of P_4.

Corollary 5.13 *There exists a sequence of posets P_n, $n \geq 1$, $|P_n| = 3n$, such that $H(P_n)$ is planar and*

$$\frac{SN(P_n)}{QN(P_n)} = \Omega(n).$$

Corollary 5.14 *There exists a sequence of posets P_n, $n \geq 1$, $|P_n| = 3n$, such that $H(P_n)$ is planar and*

$$\frac{SN(P_n)}{SN(H(P_n))} = \Omega(n).$$

5.5 Conclusions and Open Questions

In this chapter we have initiated the study of queue layouts of posets and have also proved a lower bound result for stack layouts of posets with planar covering graph. The upper bounds on the queuenumber of a poset in terms of its jumpnumber, its length, its width, and the queuenumber of its covering graph, proved in Section 5.2, may be useful in proving specific upper bounds on the queuenumber of various classes of posets. We conjecture that the upper bound of $W(P)^2$ on the queuenumber of an arbitrary poset P, proved in Section 5.2, and the upper bound of $3 \cdot W(P) - 2$ on the queuenumber of any planar poset P, proved in Section 5.3, are not tight.

Conjecture 3 *For any poset P, $QN(P) \leq W(P)$.*

We have established a lower bound of $\Omega(\sqrt{n})$ on the queuenumber of the class of planar posets. We conjecture that this bound is tight:

Conjecture 4 *For any n-element planar poset P, $QN(P) = O(\sqrt{n})$.*

Another upper bound that we conjecture exists on the queuenumber of a planar poset P is given by the length $L(P)$. We conjecture that it is possible to embed a planar poset in an "almost" leveled-planar fashion with $L(P)$ levels. From such an embedding, a queue layout of P in $L(P)$ queues can be obtained.

Conjecture 5 *For any planar poset P, $QN(P) \leq L(P)$.*

In Section 5.4 we show that the stacknumber of posets whose covering graph is planar is $\Theta(n)$. However, the stacknumber of planar posets is still unresolved.

Chapter 6

Queue Layouts of Planar Graphs

Heath, Leighton, and Rosenberg [35] show that every 1-stack (respectively, 1-queue) graph is a 2-queue (respectively, 2-stack) graph. This result suggests the conjecture that for any family of graphs $\mathcal{G}$, the ratios $\frac{SN_n(\mathcal{G})}{QN_n(\mathcal{G})}$ and $\frac{QN_n(\mathcal{G})}{SN_n(\mathcal{G})}$ are bounded above by a constant, perhaps even the constant 2. Heath, Leighton, and Rosenberg [35] disprove the first half of this conjecture by showing that $\frac{SN_n(\mathcal{G})}{QN_n(\mathcal{G})}$ is unbounded; in fact, they present a class of graphs

$$\mathcal{T} = \{TC(n) | n \geq 0\},$$

whose stack requirement grows exponentially as compared to its queue requirement. Each graph $TC(n)$ in $\mathcal{T}$ is called a *ternary n-cube*. The vertices of $TC(n)$ comprise all strings of length n over the alphabet $\Sigma = \{0, 1, 2\}$. The edges of $TC(n)$ connect all triples of vertices of the forms $x0y$, $x1y$, and $x2y$, $x, y \in \Sigma^*$ into a triangle (i.e., a copy of K_3). Hence, $TC(n)$ has $N = 3^n$ vertices and $n3^n$ edges connected into $n3^{n-1}$ triangles.

$\mathcal{T}$ requires exponentially more stacks than queues. Heath, Leighton, and Rosenberg [35] show, by a counting argument, that every layout of $TC(n)$ contains a separated layout of a large sum of triangles graph with respect to its natural partition. Chung, Leighton, and Rosenberg [16] have shown a lower bound on the separated stack layout of a sun of triangles graph with respect to its natural partition (see Theorem 3.21). This result, along with the fact that any layout $TC(n)$ contains a large separated layout of a sum of triangles graph with respect to its natural partition, provides a lower bound on the stacknumber of $TC(n)$. This bound is exponential in n. An upper bound on the queuenumber of $TC(n)$ is determined the way in which the upper bound on the queuenumber of binary hypercubes is determined. This bound is linear in n. The precise statement of the theorem is the

following.

Theorem 6.1 (Heath, Leighton, and Rosenberg [35]) *$TC(n)$ admits a queue layout using $2n$ queues but requires $\Omega\left(3^{n(\frac{1}{9}-\epsilon)}\right)$ stacks in any stack layout, for any $\epsilon > 0$.*

6.1 Stellated K_3

An upper bound on the ratio $\frac{QN_n(\mathcal{G})}{SN_n(\mathcal{G})}$ is unknown. Heath, Leighton, and Rosenberg [35] have conjectured that there does not exist a class of graphs that requires many more queues than stacks. This conjecture was motivated by a lack of a class of graphs that would play the same role for queue layouts that the ternary hypercube does for stack layouts. But, recent investigations of a class of planar graphs that can be laid out in 3 stacks seem to indicate that this class requires an unbounded number of queues. In addition to resolving the $\frac{QN_n(\mathcal{G})}{SN_n(\mathcal{G})}$ conjecture, an unbounded queuenumber for this class would show that the queuenumber of the class of planar graphs is unbounded. This class, called the *Stellations of K_3*, is defined in the following paragraph.

Given an arbitrary planar graph G, with a fixed planar embedding, the *stellation* $ST(G)$ of G results from placing a new vertex in each interior face of G and connecting it by an edge to each vertex on that face. Define $ST^0(G) = G$, and $ST^n(G) = ST\left(ST^{n-1}(G)\right)$, $n \geq 1$. Let

$$\mathcal{S}(G) = \{ST^n(G) \mid n \geq 0\}$$

be the class of graphs by stellating G. Bernhart and Kainen [6] conjecture that, for any planar graph G, $\mathcal{S}(G)$ has an unbounded stacknumber. Buss and Shor [11] disprove this conjecture by showing that $ST^n(K_3)$ can be laid out in 5 stacks for all n, while Heath [32] reduces this bound to the optimal value of 3. The corresponding queue layout problem for $ST^n(K_3)$ is open, although we suspect that $QN_n(\mathcal{S}(K_3)) = \Theta(\log_n)$. In this chapter, we show that for various natural layouts of $ST^n(K_3)$, $QN(ST^n(K_3)) = \Omega(n)$. We conjecture that the techniques used in proving the lower bound results mentioned above can be extended to show that the queuenumber of $ST^n(K_3)$ is $\Theta(n)$.

An idea that has been very useful in our investigations of queue layouts of $ST^n(K_3)$ has been the labeling of vertices of $ST^n(K_3)$ devised by Buss and Shor [11]. The labeling allows us to view a total order on the vertices of $ST^n(K_3)$ as a sequence of labels and reveals structural properties of the graphs as relationship between labels. This labeling scheme is described inductively as follows. The vertices of K_3 are labeled a^{-1}, b^{-1} and c^{-1}. Let $Grade(i)$ be the set of vertices of $ST^n(K_3)$ added at the ith stellation. Hence $Grade(0) = \{a^{-1}, b^{-1}, c^{-1}\}$. The remaining vertices of $ST^n(K_3)$ are labeled with strings over the alphabet $\Sigma = \{a, b, c\}$. The vertex in the center obtained by stellating K_3 once is the sole vertex in $Grade(1)$ and is labeled ϵ, the empty string. Three directions a, b, and c are defined as shown in Figure 6.1. From the labeling of the vertices in $Grade(i-1)$, the labeling of the vertices of $Grade(i)$ is obtained as follows. Let v be a vertex in $Grade(i)$ and let u be the vertex in $Grade(i-1)$ that it is connected to (if $i \geq 2$, a vertex in $Grade(i)$ is connected to exactly one vertex in $Grade(i-1)$). If u is labeled $\boldsymbol{x}$ ($\boldsymbol{x} \in \Sigma^*$) and (u, v) is in direction d ($d \in \Sigma$), then label v with $\boldsymbol{x}d$ ($\boldsymbol{x}d \in \Sigma^+$). Figure 6.1 shows the labeling of $ST^3(K_3)$. This labeling has the following properties.

1. Each vertex in $Grade(i)$, $i > 0$, is labeled with a unique string of length $i-1$. This implies a one-one correspondence between vertices of $ST^n(K_3)$ and the strings in $\cup_{i=0}^{n-1}\Sigma^i \cup \{a^{-1}, b^{-1}, c^{-1}\}$. This correspondence allows us to view strings over Σ as vertices of $ST^n(K_3)$.

2. Let vertex r have the label $\boldsymbol{x}$ and vertex s have the label $\boldsymbol{x}\alpha y$, where $\boldsymbol{x}, y \in \Sigma^*$ and $\alpha \in \Sigma$. If α does not occur in y, then $(\boldsymbol{x}, \boldsymbol{x}\alpha y)$ is an edge in $ST^n(K_3)$. The edges in $ST^n(K_3)$ are exactly these edges together with (a^{-1}, b^{-1}), (b^{-1}, c^{-1}), (c^{-1}, a^{-1}), (a^{-1}, ϵ), (b^{-1}, ϵ), and (c^{-1}, ϵ).

6.2 Bounds on the Queuenumber of Stellated K_3

Let δ be a total order on the vertices of $ST^n(K_3)$ defined as follows. $a^{-1} <_\delta b^{-1} <_\delta c^{-1}$ and $\forall \boldsymbol{x} \in \Sigma^*, c^{-1} <_\delta \boldsymbol{x}$. δ orders the remaining vertices, that are strings over Σ,

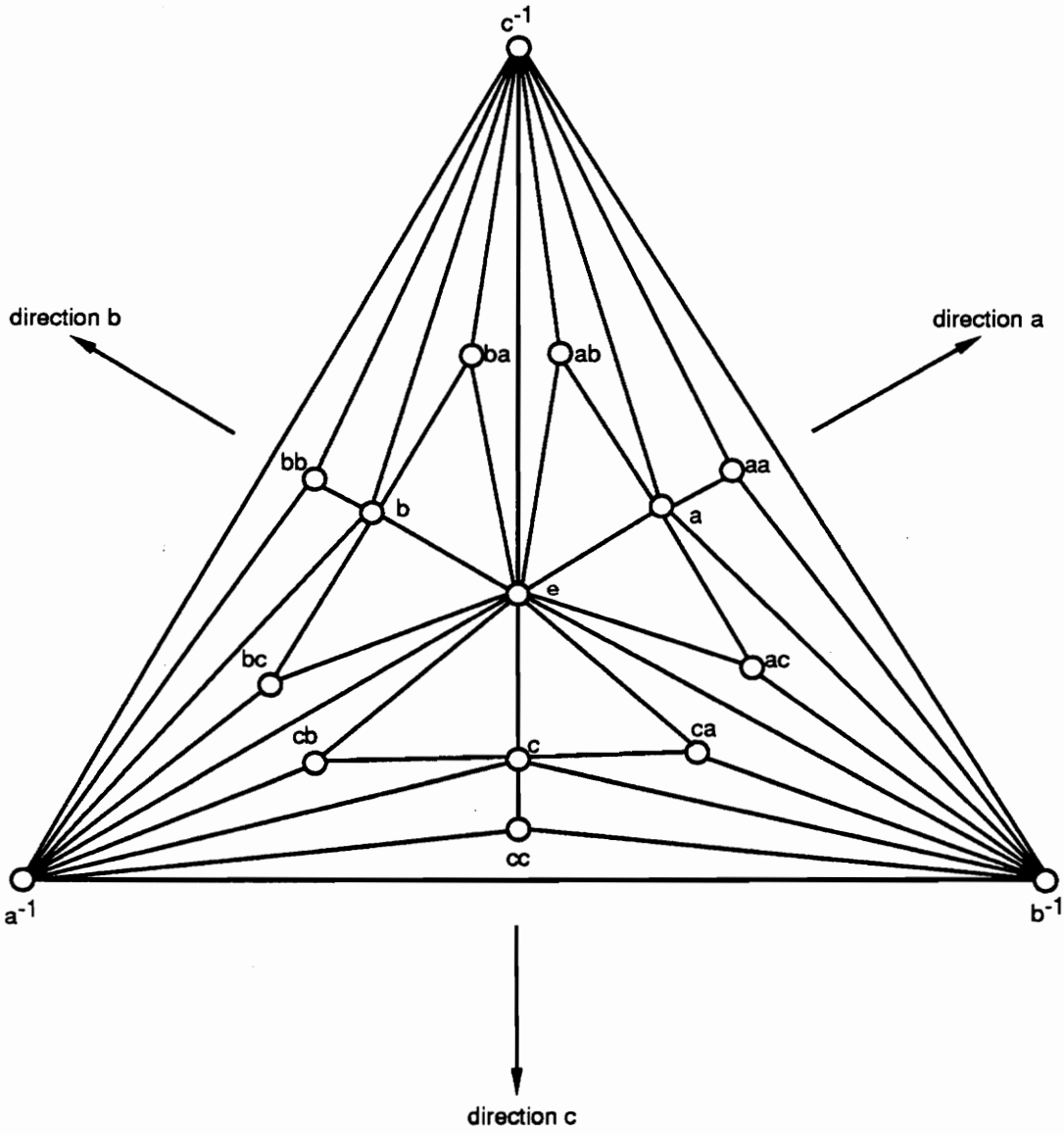


Figure 6.1: Labeled $ST^3(K_3)$.

according to the dictionary order. Call δ the *lexicographic order* of the vertices of $ST^n(K_3)$ and call any layout according to δ, a *lexicographic layout.* Buss and Shor [11] show that $SN(\delta, ST^n(K_3)) \leq 5$, $n \geq 0$. The lexicographic order turns out to be particularly bad for queues as we show in the following theorem.

Theorem 6.2 *The queuenumber of the lexicographic layout of $ST^n(K_3)$ is at least $n-1$.*

Proof: If δ is the lexicographic order on the vertices of $ST^n(K_3)$, then

$$\epsilon <_\delta a <_\delta ab <_\delta ab^2 <_\delta \cdots <_\delta ab^{n-3} <_\delta ab^{n-3}c <_\delta \cdots <_\delta ab^2c <_\delta abc <_\delta ac <_\delta c$$

occur in the order shown above. Since, (ϵ, c) and edges of the form (ab^i, ab^ic) occur in $ST^n(K_3)$, for $0 \leq i \leq n-3$, the lexicographic layout of $ST^n(K_3)$ contains a rainbow of size $(n-1)$. This implies that the queuenumber of the lexicographic layout of $ST^n(K_3)$ is at least $n-1$. □

We now prove a stronger result in which we show that vertices belonging to large grades cannot appear in lexicographic order in any layout with a bounded queuenumber. The result by Erdös and Szekeres [21] (see Proposition 1.7) is used in the proof. We are now ready to state and prove the theorem.

Theorem 6.3 *If $i > 9k^2 + 2$, then the vertices in Grade(i) cannot appear in lexicographic order in a k-queue layout of $ST^n(K_3)$.*

Proof: The proof is by contradiction. Assume that there exists a k-queue layout of $ST^n(K_3)$ and further assume that there exists an integer j such that $j > 9k^2 + 2$ and the vertices in *Grade*(j) occur in the lexicographic order. By Theorem 3.13 it follows that there exists a $3k$-queue separated layout of $ST^n(K_3)$ in which all the vertices in *Grade*(j) appear in lexicographic order and to the right of the rest of the vertices. Let σ be the total order of the vertices in this separated layout. We obtain a contradiction by showing that any layout of $ST^n(K_3)$ in which the vertices are laid out according to σ contains a rainbow of size greater than $3k$ and hence will have a queuenumber greater than $3k$.

Consider the following set of $j-2$ vertices of $ST^n(K_3)$ that are not in $Grade(j)$,

$$\{b, b^2, \ldots, b^{j-2}\}.$$

This set of vertices occurs in some total order in the layout according to σ. But, from Proposition 1.7 it follows that there is a subset $\{b^{i_1}, b^{i_2}, \ldots, b^{i_{m-1}}, b^{i_m}\}$ of size $m = \lfloor\sqrt{j-2}\rfloor$ such that

$$b^{i_1} <_\sigma b^{i_2} <_\sigma \cdots <_\sigma b^{i_{m-1}} <_\sigma b^{i_m}$$

where either $i_1 < i_2 < \ldots < i_m$ or $i_1 > i_2 > \ldots > i_m$. In the former case ($i_1 < i_2 < \ldots < i_m$) this subsequence forms a rainbow of size m with the $Grade(j)$ vertices

$$b^{i_1}ca^{j-i_1-2}, b^{i_2}ca^{j-i_2-2}, \ldots, b^{i_m}ca^{j-i_m-2},$$

while in the latter case ($i_1 > i_2 > \ldots > i_m$) this subsequence forms a rainbow of size m with the $Grade(j)$ vertices

$$b^{i_1}ac^{j-i_1-2}, b^{i_2}ac^{j-i_2-2}, \ldots, b^{i_m}ac^{j-i_m-2}.$$

This implies that in any layout of $ST^n(K_3)$ which contains vertices ordered according to σ there is a rainbow of size $\lfloor\sqrt{j-2}\rfloor$. From $j > 9k^2+2$ it follows that $\lfloor\sqrt{j-2}\rfloor > 3k$. This is a contradiction because the size of the largest rainbow in a $3k$ queue layout cannot be greater than $3k$. □

The collection of sets $\{Grade(i) \mid i \geq 0\}$ partitions the vertices of $ST^n(K_3)$ into $n+1$ blocks. Let $\gamma = \{Grade(i) \mid i \geq 0\}$. The class of separated layouts of $ST^n(K_3)$ with respect to γ is a natural one to investigate. We call a layout belonging to this class a *graded layout.* We have almost tight bounds on the queuenumber of graded layouts of $ST^n(K_3)$.

Theorem 6.4 *There is a graded layout of $ST^n(K_3)$ that requires no more than $n+2$ queues. Any graded layout of $ST^n(K_3)$ requires at least $\left\lfloor\frac{2}{3}(n-1)\right\rfloor$ queues.*

Proof: We first show a graded layout of $ST^n(K_3)$ that requires no more than $(n+2)$ queues. Let σ be the lexicographic order of vertices in $ST^n(K_3)$ and let $\gamma = \{Grade(i) \mid i \geq 0\}$. Let $\delta \in I(\gamma, \sigma)$ such that for each i, j if $u \in Grade(i)$ and $v \in Grade(j)$ then $u < v$ in δ if and only if $i < j$. In other words, δ is a graded layout such that the vertices in each grade appear in the lexicographic order and the grades themselves occur in the order $Grade(0), Grade(1), \ldots, Grade(n-1)$. We show that, for any $i, j > 0$, there are no 2-rainbows in the sublayout induced by the vertices in $Grade(i) \cup Grade(j)$. There are no edges between two vertices of $Grade(i)$ or two vertices of $Grade(j)$. Hence, we only have to look for rainbows between vertices in $Grade(i)$ and vertices in $Grade(j)$. Assume, without loss of generality, that $i < j$. Let $x, z \in Grade(i)$ and let x be lexicographically smaller than z implying that $x < z$ in δ. Let $x\alpha y_1, z\beta y_2 \in Grade(j)$ be vertices such that $(x, x\alpha y_1)$ and $(z, z\beta y_2)$ are edges in $ST^n(K_3)$. Since $i < j$, $x < x\alpha y_1, x < z\beta y_2, z < x\alpha y_1$, and $z < z\beta y_2$. Also, $x\alpha y_1$ is lexicographically smaller than $z\beta y_2$ because x is the same length as z and is lexicographically smaller than z. This implies that $x\alpha y_1 < z\beta y_2$ in δ and hence the edges $(x, x\alpha y_1)$ and $(z, z\beta y_2)$ cross rather than nest. Since, $(x, x\alpha y_1)$ and $(z, z\beta y_2)$ are arbitrary edges between $Grade(i)$ and $Grade(j)$ we conclude that there is no 2-rainbow in the sublayout induced by the the vertices in $Grade(i) \cup Grade(j)$. The largest possible rainbow in the layout then is suggested by Figure 6.2. This rainbow has size $n-1$. When the three vertices $a^{-1}, b^{-1}, c^{-1} \in Grade(0)$, are added to the layout the largest rainbow can be of size at most $(n+2)$. Proposition 1.15 shows that the size of the largest rainbow in a layout is equal to the queuenumber of the layout.

We now prove the lower bound of $\left\lfloor \frac{2}{3}(n-1) \right\rfloor$ on the number of queues required by a graded layout of $ST^n(K_3)$. Consider an arbitrary graded layout of $ST^n(K_3)$ according to total order σ. Let $i, j, k > 1$ be such that the vertices in $Grade(i)$ occur before the vertices in $Grade(j)$ which occur before the vertices in $Grade(k)$. We first show that the layout of the subgraph induced by the vertices in $Grade(i) \cup Grade(j) \cup Grade(k)$ contains a rainbow of size 2. There are 3 cases depending on the values of i, j, k.

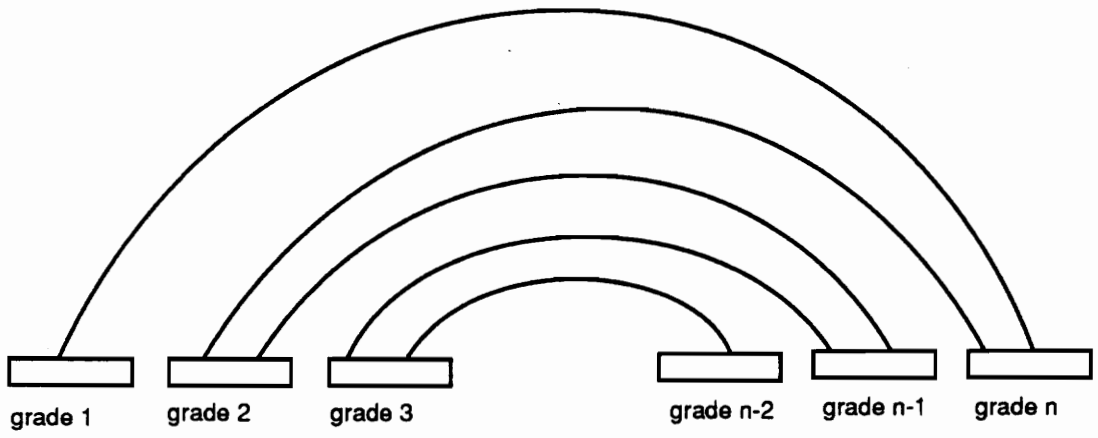


Figure 6.2: The largest possible rainbow in a graded layout.

Case 1: $i < j$ and $i < k$. Let $x, y \in Grade(i)$, such that x is lexicographically smaller than y. Then

$$\begin{aligned} xab^{k-i-1}, yab^{k-i-1} &\in Grade(k) \\ xab^{j-i-1}, yab^{j-i-1} &\in Grade(j), \end{aligned}$$

and

$$\begin{aligned} (x, xab^{k-i-1}) \quad & (y, yab^{j-i-1}) \\ (x, xab^{j-i-1}) \quad & (y, yab^{k-i-1}) \end{aligned}$$

are edges in $ST^n(K_3)$. If $j < k$ then (x, xab^{k-i-1}) and (y, yab^{j-i-1}) nest and if $j > k$ then (x, xab^{j-i-1}) and (y, yab^{k-i-1}) nest.

Case 2: $i > k$ and $j > k$. Let $p = \min(j, k)$ and $q = \max(j, k)$. Let $x \in Grade(p)$. Then

$$u_1 = xab^{q-p-1}, u_2 = xac^{q-p-1} \in Grade(q)$$

and

$$v_1 = xab^{q-p}c^{i-q-1}, v_2 = xac^{q-p}b^{i-q-1} \in Grade(i).$$

(x, v_1), (x, v_2), (u_1, v_1), and (u_2, v_2) are edges in $ST^n(K_3)$. Without loss of generality assume that v_1 occurs before v_2 in σ. If $j < k$ then (u_1, v_1) nests over (x, v_2) and if $j > k$ then (x, v_1) nests over (u_2, v_2).

Case 3: $j < i < k$ or $k < i < j$. Let $p = \min(j, k)$ and $q = \max(j, k)$. Let $x \in Grade(p)$. Then

$$u_1 = xab^{i-p-1}, u_2 = xac^{i-p-1} \in Grade(i)$$

and

$$v_1 = xab^{i-p}c^{q-i-1}, v_2 = xac^{i-p}b^{q-i-1} \in Grade(q).$$

(x, u_1), (x, u_2), (u_1, v_1), and (u_2, v_2) are edges in $ST^n(K_3)$. Without loss of generality assume that u_1 occurs before u_2 in σ. If $j < k$ then (u_1, v_1) nests over (x, u_2) and if $j > k$ then (x, u_1) nests over (u_2, v_2).

Hence, we have shown that the sublayout induced by the vertices in $Grade(i) \cup Grade(j) \cup Grade(k)$ contains a rainbow of size 2, for any $i, j, k > 1$.

Let $i_1, i_2, \ldots, i_{n-1}$ be a permutation of $2, 3, \ldots, n$ such that the graded layout contains the vertices of $Grade(i_p)$ before the vertices of $Grade(i_q)$ if $p < q$. The vertices of $\cup_{i=2}^{n} Grade(i)$ can be partitioned into $\left\lfloor \frac{1}{3}(n-1) \right\rfloor$ groups in the following manner. For $r = 1, 2, \ldots, \left\lfloor \frac{1}{3}(n-1) \right\rfloor$ let

$$Group(r) = Grade(i_{2r-1}) \cup Grade(i_{2r}) \cup Grade(i_{n-r+1}).$$

For each r, $r > 0$, the sublayout induced by the vertices in $Group(r)$, contains a 2-rainbow and for each $r, r > 1$, the 2-rainbow in the sublayout induced by $Group(r)$ nests under the 2-rainbow in the sublayout induced by $Group(r-1)$. This implies the existence of a rainbow of size $\left\lfloor \frac{2}{3}(n-1) \right\rfloor$ □

6.3 Conclusions and Open Questions.

The results proved in this chapter strengthen our belief that $QN\left(ST^n(K_3)\right)$ increases as n increases. The technique that Heath, Leighton, and Rosenberg [35] use to show that the class of ternary n-cubes requires exponentially more stacks than queues is powerful but does not seem to apply to $ST^n(K_3)$. This indicates the need for a new lower bound technique to show that $QN\left(ST^n(K_3)\right)$ is unbounded. Separated layouts provide us with 2 new approaches.

A possible lower bound technique arises from our result about the queuenumber of the separated layout of a graph with respect to a partition of size m. We have shown that a k-queue, m-partite graph has a $(2(m-1)k$-queue separated layout with respect to its natural partition of size m. We also know that a lower bound on the queuenumber of graded layouts of $ST^n(K_3)$ is $\left\lfloor \frac{2}{3}(n-1) \right\rfloor$. But, a graded layout of $ST^n(K_3)$ is a separated layout with respect to the natural partition of size $n+1$. If we were able to tighten our result about the queuenumber of a k-queue, m-partite graph, to show that any k-queue, m-partite graph has a separated layout with respect to its natural partition in $f(m) \cdot k$ queues, where $f(m)$ is a sublinear function of m, then we would have a lower bound of $\left\lceil \frac{2(n-1)}{3f(n+1)} \right\rceil$ on $QN(ST^n(K_3))$. Since, f is a sublinear function, $\left\lceil \frac{2(n-1)}{3f(n+1)} \right\rceil$ will be unbounded.

Another possible approach arises from the fact that $ST^n(K_3)$ is a 4-partite graph. We hope to show that the separated layout of $ST^n(K_3)$ with respect to its natural partition into four sets of vertices, requires an unbounded number of queues. This would immediately imply that $ST^n(K_3)$ does not have a layout in a bounded number of queues.

A demonstration that $QN\left(ST^n(K_3)\right)$ is unbounded has an additional implication. It would imply that the queuenumber of the class of planar graphs is unbounded. This would be in contrast to the result that every planar graph can be laid out using 4 stacks [64]. This would expose another asymmetry in the relative abilities of stacks and queues as mechanisms to lay out graphs. While we have demonstrated some progress towards showing that $QN\left(ST^n(K_3)\right)$ is unbounded, 2 important questions that still remain open are

Open question 1 *Is there a class of graphs whose queue requirements are unbounded as compared to its stack requirements?*

Open question 2 *Is the queue number of planar graphs bounded?*

The following conjecture, based on the results presented in this chapter, if proved would answer both the above questions.

Conjecture 6 $QN(ST^n(K_3)) = \Theta(n)$.

Chapter 7
Conclusion

In this research, we have studied stack and queue layouts of graphs from various points of view and have obtained algorithmic and graph-theoretic results. In the process, we have developed tools, whose utility, we believe, extends beyond this dissertation. We have also strengthened known applications of stack and queue layouts of graphs and have discovered and developed a new application. We believe that this new application, staircase covers of matrices and a scheme for matrix-vector multiplication, provides the strongest practical motivation as yet for studying queue layouts. We conclude this dissertation with the hope that the progress made in this dissertation will provide a new impetus to research in the area of stack and queue layouts of graphs.

7.1 Main Contributions

The main contributions of this research are as follows:

1. In Chapter 2, we have established a strong correspondence between queue layouts and *matrix covers*. This correspondence has led to a harvest of results in the realm of queue layouts as well as in the realm of matrix covers. Furthermore, this correspondence has led to a new and efficient scheme for performing matrix computations in parallel on a data-driven network. This scheme is provably more efficient than any existing schemes in terms of time and hardware.

2. Heath and Rosenberg [36] promise a study of stack and queue layouts of dags (see Section 1.2 in [36]). This is because stack and queue layouts of dags form a more faithful model than stack and queue layouts of undirected graphs of the problem

of process scheduling in a parallel processing system. In Chapter 4, we fulfill their promise by initiating the study of *stack and queue layouts of dags.* In keeping with the spirit of Chung, Leighton, and Rosenberg [16] and Heath and Rosenberg [36], we answer a host of algorithmic and graph-theoretic questions in our study.

3. Nowakowski and Parker [53], Hung [38], and Sysło [60] have studied stack layouts of posets and have established connections between the stacknumber of a poset and other structural measures. Motivated by this work, we have initiated the study of *queue layouts of posets* in Chapter 5. The connections between the queuenumber and other structural measures of posets, as revealed by our work, are evidence to the remarkably rich structure that posets possess.

4. The utility of *separated stack and queue layouts of graphs* as a tool to prove lower bound results should be clear from their use in the various results proved in this dissertation. In Chapter 3, we have shown that separated layouts, along with mingled layouts, can lead to significant new lower bound results and simplify existing lower bound arguments.

5. Due to the efforts of Buss and Shor [11], Heath [31], and Yannakakis [64], we know that the stacknumber of the class of planar graphs is 4. The corresponding question for queue layouts is open. In Chapter 6, we investigated a class of planar graphs called stellated triangles and have shown that the queuenumber of various natural layouts of a graph belonging to this class is not bounded by any constant. Our belief is that these results will pave the way for new research that will eventually determine the *queuenumber of planar graphs.*

6. An underlying theme of this research, as the title of this dissertation suggests, is a comparison of the powers of stacks and queues. Based on the various results obtained in this dissertation, we conclude that the duality that stacks and queues exhibit in simple situations, vanishes in more complicated situation. A few examples of results

that suggest this are:

(1) There is a simple connection between queue layouts and matrix covers. No similar connection seems to exist between stack layouts and matrix covers.

(2) Any k-queue graph $G = (V, E)$ has a $(2m - 1)k$-queue separated layout with respect to any partition of V into m blocks. But, for $m \geq 3$, there exists a class of k-stack graphs $\mathcal{G}$ with the property that, for each $G = (V, E)$ in $\mathcal{G}$, there exists a partition γ of V into m blocks such that the minimum stacknumber of any separated layout of G with respect to γ is $\Omega\left(n^{\frac{1}{3}}\right)$ (see Theorem 3.13 and Theorem 3.21).

(3) There exists a class of outerplanar dags whose queuenumber grows arbitrarily as compared to its stacknumber, and there exists a class of planar dags whose stacknumber grows arbitrarily as compared to its queuenumber (see Theorem 4.9 and Theorem 4.12).

(4) There exists a class of planar posets whose queuenumber grows arbitrarily large as compared to its stacknumber, and there exists a class of posets with planar covering graphs whose stacknumber grows arbitrarily as compared to its queuenumber (see Theorem 5.8 and Theorem 5.12).

7.2 Future Directions

First we present a list of precisely stated open questions that arise from our work. To motivate potential researchers in this area, we also provide a conjecture corresponding to each question. These conjectures are based on partial results and examples.

Open Question 1 What is the minimum stacknumber of a separated layout of a k-stack bipartite graph with respect to its natural partition?

Conjecture 1 The minimum stacknumber of a separated layout of a bipartite graph G with respect to its natural partition is not bounded above by a constant multiple of

the stacknumber of G.

Open Question 2 What is the stacknumber of the class of outerplanar dags?

Conjecture 2 Let $\mathcal{O}$ be the class of outerplanar dags. There exists a constant c such that $SN_{\mathcal{O}}(n) \leq c$ for all n.

Open Question 3 What is the time complexity of recognizing 1-queue dags?

Conjecture 3 The time complexity of recognizing 1-queue dags is linear.

Open Question 4 What is the stacknumber of the class of planar posets?

Conjecture 4 The stacknumber of the class of planar posets is bounded above by a constant.

Open Question 5 What is the queuenumber of the class of planar graphs?

Conjecture 5 Let $\mathcal{S}$ be the class of stellations of a triangle. Then $QN_{\mathcal{S}}(n) = \theta(\log n)$. Furthermore, the queuenumber of the class of planar graphs is a logarithmic function of graph size.

In addition to the open questions stated above, we suggest the following three broad research areas.

Parallel Matrix Computations: In Chapter 2, we developed an efficient scheme for performing matrix-vector multiplication using a data-driven network. Based on Melhem's [51] work, we know that this scheme can be extended to other simple matrix computations such as solving a triangular system of linear equations. Extending our scheme to other fundamental matrix computations such as matrix-matrix multiplication is a potentially fruitful research area.

Queue Layouts of Posets: We have initiated the study of queue layouts of posets and have established connections between the queuenumber of posets and other structural

measures of posets. Nevertheless, this area is ripe for exploration in at least two directions. One direction would be to tighten the bounds that we have proven and the other direction would be to establish connections between the queuenumber of posets and structural measures of posets that we have not considered, such as the dimension of posets.

Other Measures of Stack and Queue Layouts: In this research, we have focused exclusively on stacknumber and queuenumber as measures of the quality of stack layouts and queue layouts. Many new questions are obtained by introducing other measures such as cutwidth (see Heath and Rosenberg [36]). Of particular interest are questions of interactions or tradeoffs among multiple measures (Heath [33]).

REFERENCES

[1] Alok Aggarwal and Subhash Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the Third ACM Symposium on Computational Geometry*, pages 278–290, 1987.

[2] Mikhail J. Atallah and Greg N. Frederickson. A note on finding a maximum empty rectangle. *Discrete Applied Mathematics*, 13(1):87–91, 1986.

[3] Mikhail J. Atallah and S. Rao Kosaraju. An efficient algorithm for maxdominance, with applications. *Algorithmica*, 4:221–236, 1989.

[4] Mikhail J. Atallah, Glenn K. Manacher, and Jorge Urrutia. Finding a minimum independent dominating set in a permutation graph. *Discrete Applied Mathematics*, 21:177–183, 1988.

[5] Reuven Bar-Yehuda and Sergio Fogel. Monotone sequences with applications. Typescript, 1987.

[6] Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27:320–331, 1979.

[7] Garrett Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, 1940.

[8] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[9] M. Brady and Donna Brown. VLSI routing: four layers suffice. *Advances in Computing Research*, 2:245–254, 1985.

[10] Jonathan F. Buss, Arnold L. Rosenberg, and Judson D. Knott. Vertex types in book embeddings. *SIAM Journal on Computing*, 54:156–175, 1987.

[11] Jonathan F. Buss and Peter W. Shor. On the page number of planar graphs. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pages 98–100, 1984.

[12] Bernard Chazelle, R. L. Drysdale, and D. T. Lee. Computing the largest empty rectangle. *SIAM Journal on Computing*, 15(1):300–315, 1986.

[13] K. Y. Chen. Minimizing the bandwidth of sparse symmetric matrices. *Computing*, 11:103–110, 1973.

[14] K. Y. Chen. Note on minimizing the bandwidth of sparse symmetric matrices. *Computing*, 11:27–30, 1973.

[15] Norishige Chiba, Takao Nishizeki, Shigenobu Abe, and Takao Ozawa. A linear algorithm for embedding planar graphs using *PQ*-trees. *Journal of Computer and System Sciences*, 30:54–76, 1985.

[16] Fan R. K. Chung, Frank T. Leighton, and Arnold L. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic and Discrete Methods*, 8:33–58, 1987.

[17] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th National ACM Conference*, pages 157–172, 1969.

[18] E. Cuthill and J. McKee. Several strategies for reducing the bandwidth of matrices. In Donald Rose and R. Willoughby, editors, *Sparse Matrices and their Application*. Plenum Press, 1972.

[19] Edsger. W. Dijkstra. Some beautiful arguments using mathematical induction. *Acta Informatica*, 13:1–8, 1980.

[20] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[21] Paul Erdös and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[22] Shimon Even. *Graph Algorithms*. Computer Science Press, Rockville, MD, 1979.

[23] Shimon Even and Alon Itai. Queues, stacks and graphs. In Zvi Kohavi and A. Paz, editors, *Theory of Machines and Computations*, pages 71–86. Academic Press, New York, 1971.

[24] Shimon Even, Amir Pnueli, and Abraham Lempel. Permutation graphs and transitive graphs. *Journal of the ACM*, 19:400–410, 1972.

[25] M. Farber and J. Keil. Domination in permutation graphs. *Journal of Algorithms*, 6:309–321, 1985.

[26] Peter C. Fishburn. Thickness of ordered sets. *SIAM Journal of Discrete Mathematics*, 3:489–501, 1990.

[27] Michael Garey and David S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, San Fransisco, 1979.

[28] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics A Foundation for Computer Science.* Addison-Wesley Publishing Company, Reading, MA, 1988.

[29] Eitan M. Gurari and Ivan Hal Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *Journal of Algorithms*, 5:531–546, 1984.

[30] Frank Harary and Geert Prins. The block-cutpoint-tree of a graph. *Publicationes Mathematica Debrecen*, 13:103–107, 1966.

[31] Lenwood S. Heath. Embedding planar graphs in seven pages. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pages 74–83, 1984.

[32] Lenwood S. Heath. *Algorithms for Embedding Graphs in Books.* PhD thesis, University of North Carolina at Chapel Hill, 1985.

[33] Lenwood S. Heath. Embedding outerplanar graphs in small books. *SIAM Journal on Algebraic and Discrete Methods*, 8:198–218, 1987.

[34] Lenwood S. Heath and Sorin Istrail. The pagenumber of genus g graphs is $O(g)$. *Journal of the ACM*, 39(3):479–501, 1992.

[35] Lenwood S. Heath, Frank T. Leighton, and Arnold L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM Journal of Discrete Mathematics*, 5(3):398–412, 1992.

[36] Lenwood S. Heath and Arnold L. Rosenberg. Laying out graphs using queues. *SIAM Journal of Computing*, 21(5):927–958, 1992.

[37] John E. Hopcroft and Robert E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[38] Le Tu Quoc Hung. A planar poset which requires 4 pages. Typescript, 1989.

[39] Donald B. Johnson. A priority queue in which initialization and queue operations take $O(\log \log D)$ time. *Mathematical Systems Theory*, 15:295–309, 1982.

[40] Paul C. Kainen. The bookthickness of a graph. In *Proceedings of the 20th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 127–132, 1989.

[41] Rolf G. Karlsson and Mark H. Overmars. Scanline algorithms on a grid. *BIT*, 28:227–241, 1988.

[42] Donald E. Knuth. *Fundamental Algorithms: The Art of Computer Programming Volume 1.* Addison-Wesley Publishing Company, Reading, MA, 1971.

[43] H. T. Kung. Why systolic architecture? *Computer*, 15:37–46, 1982.

[44] H. T. Kung and Charles E. Leiserson. Systolic arrays (for VLSI). In Iain S. Duff and Gilbert W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 256–282. SIAM, 1979.

[45] Charles E. Leiserson. *Area-Efficient VLSI Computation*. MIT Press, Cambridge, MA, 1982.

[46] Abraham Lempel, Shimon Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs, International Symposium, Rome*, pages 215–232. Gordon and Breach, New York, 1967.

[47] Seth M. Malitz. Genus g graphs have pagenumber $O(\sqrt{g})$. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 458–468, 1988.

[48] Udi Manber. *Introduction to Algorithms*. Addison-Wesley Publishing Company, Reading, MA, 1989.

[49] Rami Melhem. Determination of stripe structures for finite element matrices. *SIAM Journal on Numerical Analysis*, 24(6):1419–1433, 1987.

[50] Rami Melhem. A study of data interlock in computational networks for sparse matrix multiplication. *IEEE Transactions on Computing*, 36(9):1101–1107, 1987.

[51] Rami Melhem. Parallel solution of linear systems with striped sparse matrices. *Parallel Computing*, 6:165–184, 1988.

[52] A. Naamad, D. T. Lee, and W. L. Hsu. On the maximum empty rectangle problem. *Discrete Applied Mathematics*, 8:267–277, 1984.

[53] Robert Nowakowski and A. Parker. Ordered sets, pagenumbers and planarity. *Order*, 6:209–218, 1989.

[54] M. Orlowski and M. Pachter. An algorithm for the determination of a longest increasing subsequence in a sequence. *Computers Math. Applic.*, 17(7):1073–1075, 1989.

[55] Christos H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16:263–270, 1976.

[56] Christos H. Papadimitriou and Mihalis Yannakakis. Towards an architecture-independent analysis of parallel algorithms. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 510–513, 1988.

[57] Sriram V. Pemmaraju. Exploring the powers of stacks and queues. Ph. D. proposal, Department of Computer Science, Virginia Tech, 1991.

[58] Arnold L. Rosenberg. The DIOGENES approach to testable fault-tolerant arrays of processors. *IEEE Transactions on Computers*, 10:902–910, 1983.

[59] K. Stoffers. Scheduling of traffic lights—a new approach. *Transportation Research 2*, pages 199–234, 1968.

[60] Maciej M. Sysło. Bounds to the page number of partially ordered sets. Technical Report 227/1989, Technische Universität Berlin, 1989.

[61] Maciej M. Syslo and M. Iri. Efficient outerplanarity testing. *Fundamenta Informaticae*, 2:261–275, 1979.

[62] Robert E. Tarjan. Sorting using networks of stacks and queues. *Journal of the ACM*, 19:341–346, 1972.

[63] William Trotter. Order preserving embeddings of graphs. In Y. Alavi and D. Lick, editors, *Theory and Applications of Graphs, Lecture Notes in Computer Science*, pages 572–579. Springer-Verlag, Berlin, 1978.

[64] Mihalis Yannakakis. Four pages are necessary and sufficient for planar graphs. *SIAM Journal of Discrete and Algebraic Methods*, 3:351–358, 1982.

VITA

Sriram V. Pemmaraju was born in Calcutta, West Bengal, India on March 31, 1966. He came to the United States in 1987 and had the fortune of meeting Santhi Hejeebu in 1987 and Lenwood Heath in 1988. He is currently employed in the Department of Computer Science at the Universty of Iowa, Iowa City.

Sriram Pemmaraju
Dec 7, 1992