SIAM J. COMPUT.
Vol. 21, No. 5, pp. 927–958, October 1992

008

LAYING OUT GRAPHS USING QUEUES*

LENWOOD S. HEATH† AND ARNOLD L. ROSENBERG‡

Abstract. The problem of laying out the edges of a graph using queues is studied. In a k-queue layout, vertices of the graph are placed in some linear order and each edge is assigned to exactly one of the k queues so that the edges assigned to each queue obey a first-in/first-out discipline. This layout problem abstracts a design problem of fault-tolerant processor arrays, a problem of sorting with parallel queues, and a problem of scheduling parallel processors. A number of basic results about queue layouts of graphs are established, and these results are contrasted with their analogues for stack layouts of graphs (the book-embedding problem). The 1-queue graphs (they are almost leveled-planar graphs) are characterized. It is proved that the problem of recognizing 1-queue graphs is NP-complete. Queue layouts for some specific classes of graphs are given. Relationships between the queuenumber of a graph and its bandwidth and separator size are presented. An apparent tradeoff between the queuewidth and the number of queues allowed in layouts of complete binary trees is indicated.

Key words. queue layout, stack layout, book embedding, graph embedding, bandwidth, separators, NP-completeness, fault-tolerant computing, scheduling parallel processors

AMS(MOS) subject classifications. 05C99, 68Q15, 68Q25, 68R10, 94C15

1. Introduction.

1.1. The problem. We study the use of queues to compute linear layouts of graphs, in the following sense. A *k-queue layout* of an undirected graph $G = (V, E)$ has two aspects. The first aspect is a linear order of V (which we think of as being on a horizontal line). The second aspect is an assignment of each edge in E to one of k queues in such a way that the set of edges assigned to each queue obeys a first-in/first-out discipline. Think of scanning the vertices in order from left to right. When the left endpoint of an edge is encountered, the edge enters its assigned queue (at the back of the queue). When the right endpoint of an edge is encountered, the edge exits its assigned queue (and must, therefore, be at the front of the queue). If a queue is examined at any instant, the edges in the queue are in the order of their right endpoints, with the leftmost of those right endpoints belonging to edges at the head of the queue. The freedom to choose the order of V and the assignment of E so as to optimize some measure of the resulting layout constitutes the essence of the queue layout problem.

More formally, a *k-queue layout* QL of an n-vertex undirected graph $G = (V, E)$ consists of a linear order of V, denoted $\sigma = 1, \cdots, n$, and an assignment of each edge in E to exactly one of k queues, $q_1, \cdots, q_k$. Each queue q_j operates as follows. The vertices of V are scanned in left-to-right (ascending) order. When vertex i is encountered, any edges assigned to q_j that have vertex i 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 i as left endpoint are placed on the back of that queue (enqueued), in ascending order of their right endpoints. k is the *queuenumber* of the layout. The *queuenumber* of G, $\mathrm{QN}(G)$, is the smallest k such that G has a k-queue layout; G is said to be a *k-queue* graph. Let $w(i, q_j)$ be the number of edges in q_j just before vertex i is encountered. Then the *queuewidth* of q_j is $\mathrm{QW}(q_j) = \max_{i \in V} w(i, q_j)$. The *maximum queuewidth*

*Received by the editors October 15, 1990; accepted for publication (in revised form) September 11, 1991. This research was supported by National Science Foundation grants DCI-87-96236, CCR-88-12567, and CCR-90-09953.

†Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061.

‡Department of Computer Science, University of Massachusetts, Amherst, Massachusetts 01003.

of the layout is $\mathrm{QW}(QL) = \max_j \mathrm{QW}(q_j)$. The *cumulative queuewidth* of the layout is $\mathrm{CQW}(QL) = \sum_j \mathrm{QW}(q_j)$.

As an example of a 1-queue layout, consider the graph G in Fig. 1.1. A 1-queue layout of G is shown in Fig. 1.2. The linear order of V is a, f, b, e, c, d. The order in which edges pass through the single queue is

$$(a, f), (a, b), (f, b), (f, e), (b, e), (b, c), (b, d), (e, d), (c, d).$$

Note that edges having the same left endpoint enter the queue in an order determined by their right endpoints. For example, edge (a, f) must enter the queue before edge (a, b) since f is to the left of b.

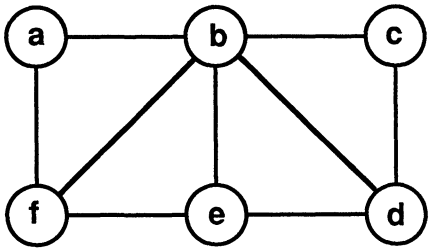

FIG. 1.1. *Example graph G.*

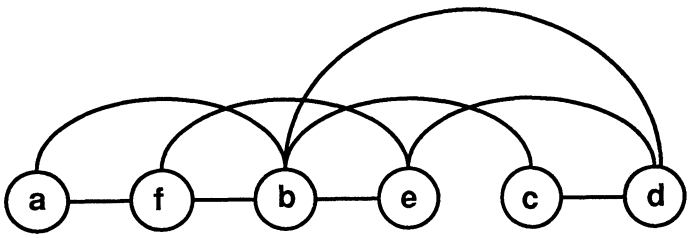

FIG. 1.2. *1-queue layout.*

Dually, a k-stack layout of graph G also has two aspects. The first aspect is again a linear order of V. The second aspect is an assignment of each edge in E to one of k stacks in such a way that the set of edges assigned to each stack obeys a last-in/first-out discipline. Unlike a queue layout, edges do not exit a stack in the same order in which they enter it.

More formally, a *k-stack layout* SL of an undirected graph consists of a linear order of V and an assignment of each edge in E to exactly one of k stacks, $s_1, \cdots, s_k$. Each stack s_j operates as follows. The vertices of V are scanned in left-to-right (ascending) order. When vertex i is encountered, any edges assigned to s_j that have vertex i as their right endpoint must be on the top of that stack; they are removed (popped). Any edges assigned to s_j that have i as left endpoint are placed on the top of the stack (pushed), in descending order of their right endpoints. k is the *stacknumber* of the layout. The *stacknumber* of G, $\mathrm{SN}(G)$, is the smallest k such that G has a k-stack layout; G is said to be a *k-stack* graph. Let $w(i, s_j)$ be the number of edges in s_j just before vertex i is encountered. Then the *stackwidth* of s_j is $\mathrm{SW}(s_j) = \max_{i \in V} w(i, s_j)$. The *maximum stackwidth* of the layout is $\mathrm{SW}(SL) = \max_j \mathrm{SW}(s_j)$. The *cumulative stackwidth* of the layout is $\mathrm{CSW}(SL) = \sum_j \mathrm{SW}(s_j)$.

As an example, Fig. 1.3 shows a 1-stack layout of the graph G in Fig. 1.1. The linear order of V is a, b, c, d, e, f. The order in which edges enter the stack is

$$(a, f), (a, b), (b, f), (b, e), (b, d), (b, c), (c, d), (d, e), (e, f).$$

The order in which edges exit the stack is

$$(a, b), (b, c), (c, d), (b, d), (d, e), (b, e), (e, f), (b, f), (a, f).$$

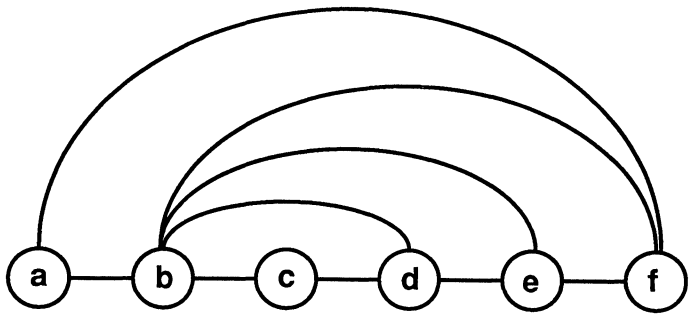

FIG. 1.3. *1-stack layout.*

The queue (respectively, stack) layout problem generalizes the problem of permuting a sequence using parallel queues (respectively, stacks) that was studied by Even and Itai [7] and Tarjan [26]. Let π be a permutation defined on $\{1, \cdots, n\}$. Define the bipartite graph G by

$$V = \{a_1, \cdots, a_n, b_1, \cdots, b_n\},$$
$$E = \{(a_i, b_i) \mid 1 \leq i \leq n\};$$

G is a perfect matching on $2n$ vertices. Then realizing π by k parallel queues (respectively, stacks) is equivalent to laying G out using k queues (respectively, stacks) when V is ordered $a_1, \cdots, a_n, b_{\pi(1)}, \cdots, b_{\pi(n)}$.

1.2. Motivation. Our study and the particular questions we focus on have a tripartite motivation.

Comparing queues and stacks. Queues and stacks are, intuitively, dual in "power" as computing mechanisms, in that queues epitomize a first-in-first-out discipline while stacks epitomize a last-in-first-out discipline. This intuition is strengthened formally when queues and stacks are used to compute fixed permutations (Tarjan [26]), largely as a consequence of the 1936 theorem of Erdös and Szekeres [6] about monotonic sequences in permutations. However, the intuition is called into doubt when queues and stacks are used as worktapes for Turing machines, because a single queue endows a Turing machine with universal computing power, whereas two stacks are needed to achieve comparable power. Here, we compare the powers of queues and stacks as devices for linearizing graphs: one "loads an edge" into the linearization device when its left end is laid out, and one "unloads" it when its right end is laid out. We find this comparison of the powers of queues and stacks to be much more complicated than the other two. To wit,

1. Stacks appear to be simpler than queues, in that the task of recognizing 1-stack graphs is computationally easy (in fact, linear time), while the analogous task for 1-queue graphs is NP-complete;
2. Queues appear to be simpler than stacks, in that, when the linearization of the vertices is preordained, the task of determining the queuenumber of the graph is computationally easy (almost linear time), while the task of determining the stacknumber is NP-complete;
3. Queues appear to be dual in power to stacks, in that a tradeoff inequality of the Erdös–Szekeres type holds for the queuenumber and stacknumber requirements of a graph when the linearization of its vertices is fixed [16];
4. Queues appear to be more powerful than stacks, in that there exist graphs whose minimum queuenumbers are exponentially smaller than their minimum stacknumbers.

These comparisons are a central theme in this paper. Note that stack layouts have been studied extensively, under the aegis of the problem of *embedding graphs in books* [2], [5], while ours is the first major study of queue layouts.

The DIOGENES design methodology. In DIOGENES [24], an array of communicating processors is implemented in a conceptual line, and some number of hardware queues and/or stacks pass over the entire line. The queues and/or stacks implement the communication links among processors in such a way that faulty processors are ignored, and all good processors are utilized. If the processors and their connections are represented by an undirected graph, then the DIOGENES layout problem is equivalent to a graph layout problem, where edges are assigned to conceptual queues and/or stacks. The variant of DIOGENES in which only stacks are used is one motivation for the studies of the book embedding problem: Bernhart and Kainen [2]; Buss and Shor [4]; Chung, Leighton, and Rosenberg [5]; Games [8]; Heath [11], [12], [13]; Heath and Istrail [14], [15]; Obrenić [21]; and Yannakakis [27]. Note that only Rosenberg [24] has considered queues before. The present research intends to investigate the same issues for queues that [5] does for stacks. In particular, we find significant instances of divergence between queue and stack layouts.

Scheduling parallel processors. Consider the following simple model of scheduling parallel computations in an architecture-independent fashion; cf. [22]. We represent the computation to be scheduled as a directed acyclic graph (dag) whose nodes represent the processes to be executed and whose arcs indicate computational dependencies: a process-node cannot be executed until all of its predecessors in the dag have been executed. Processes are queued up in a FIFO *Processor Queue* (PQ) as they become eligible for execution; each idle processor "grabs" the process at the head of the PQ. Our study focuses on the management of data in this scenario: where will the inputs to process P be when P is "grabbed" by a processor? Our solution is to have the PQ be coordinated with a *Data Manager* (DM), which itself is a collection of FIFO queues: When a process terminates, it places its "outputs" on the queues of the DM in such a way that when process P is "grabbed" by a processor, all inputs to P are at the heads of the DM queues. Our queue-based graph linearization problem idealizes this approach to the scheduling problem: The computation dag is the graph to be linearized; the linearization process implicitly specifies the loading of the PQ; the queues that control the linearization comprise the DM. In this abstract we idealize the problem even a step further by replacing the computation dag by an ordinary (undirected) graph. In subsequent work, we plan to study a more faithful version of the scheduling problem.

1.3. Results. Investigating queues at this level of generality has proved a fruitful enterprise. The harvest comprises a number of fundamental results for queue layouts as well as some surprising contrasts with stack layouts.

We summarize the highlights that are included in this paper and in the companion paper [16]. A new class of planar graphs, arched leveled-planar graphs, is shown to be a characterization of the 1-queue graphs. While 1-stack (outerplanar) graphs are easy to recognize (in fact, can be recognized in linear time), the recognition problem for 1-queue graphs is NP-complete. On the other hand, the number of queues in a *fixed-order* layout of an arbitrary graph is easily minimized in polynomial time, while the same problem for stacks is NP-complete [10]. Any 1-queue graph can be laid out with 2 stacks, and any 1-stack graph can be laid out with 2 queues [16]. An obvious generalization of these results fails: there is a class of graphs, the ternary hypercubes, that require exponentially more stacks than queues [16]. We investigate the queuenumber of some specific families of graphs and compare these to known stacknumber results. We show relationships between the queuenumber of a graph G and both the bandwidth and separator size of G. Finally, we expose an apparent tradeoff between queuenumber and queuewidth for layouts of complete binary trees.

The paper is organized as follows. Section 2 contains results on fixed-order layouts, including our polynomial-time algorithm for determining the queuenumber of such a layout. Section 3 characterizes 1-queue graphs and proves that recognizing 1-queue graphs is NP-complete; none of the later results depends on the NP-completeness proof. In §4, we investigate queue layouts for a number of familiar classes of graphs. In §5, we show relationships between the queuenumber of a graph and its bandwidth and separator size. Section 6 indicates an apparent tradeoff between queuenumber and queuewidth for complete binary trees. In the final section, we conclude with some open problems and a table comparing queuenumber and stacknumber for some specific classes of graphs.

2. Fixed-order layouts. In this section, we fix an order $\sigma = 1, 2, \cdots, n$ of V and examine the difficulty of minimizing the number of queues or stacks required to complete σ to a layout. Results include an optimal and efficient algorithm for fixed-order queue layouts. We contrast the existence of this efficient algorithm with the NP-completeness of the analogous problem for stack layouts.

We concentrate on sets of edges that are obstacles to minimizing the number of stacks or queues. A *k-rainbow* is a set of k edges

$$\{e_i = (a_i, b_i)\,,\ 1 \leq i \leq k\}$$

such that

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

in other words, a rainbow is a *nested* matching. A *k-twist* is a set of k edges

$$\{e_i = (a_i, b_i)\,,\ 1 \leq i \leq k\}$$

such that

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

in other words, a twist is a fully intersecting matching.

A rainbow is an obstacle for a queue layout because no two nested edges can be assigned to the same queue.

PROPOSITION 2.1. *Assume that σ has a k-rainbow. Then every queue layout of σ uses at least k queues. There exists a stack layout of σ in which all edges of the k-rainbow are assigned to the same stack.*

A twist is an obstacle for a stack layout because no two intersecting edges can be assigned to the same stack.

PROPOSITION 2.2. *Assume that σ has a k-twist. Then every stack layout of σ uses at least k stacks. There exists a queue layout of σ in which all edges of the k-twist are assigned to the same queue.*

The largest rainbow in σ determines the smallest number of queues needed in a queue layout of σ.

THEOREM 2.3. *If σ has no rainbow of more than k edges, then there is a k-queue layout for σ. Such a layout can be found in time $O(|E| \log\log n)$.*

Proof. We describe an algorithm for assigning the edges of G to k queues, denoted $q_1, q_2, \cdots, q_k$. The algorithm uses an $(n+1)$-position array $R[0..n]$; initially, $R[0] = n+1$, and all other $R[i] = 0$. At each step of the algorithm, each array position $R[i]$, for $1 \leq i \leq n$, contains the larger of 0 and the name of the rightmost vertex of any edge that has been assigned to queue q_i to that point; the assignment $R[0] = n + 1$ simplifies the algorithm, by creating the fiction that there is an edge in fictitious queue q_0 connecting fictitious vertices 0 and $n+1$. The algorithm maintains the invariant that nonzero entries in R are in strictly decreasing order: if $R[i-1] > 0$, then $R[i-1] > R[i]$. Clearly, the initial assignment to R satisfies this condition.

We actually maintain the array R in the balanced search tree data structure of Johnson [18]. The specific purpose of his data structure is to maintain a subset of a bounded set of integers $\{1, 2, \cdots, m\}$. It does so with $O(\log\log m)$ worst-case time for insertions, deletions, and accesses. As the entries in R are between 0 and $n+1$, the $O(\log n)$ time to perform a traditional binary search in R is reduced to $O(\log\log n)$.

Process the vertices in order, left to right. At each vertex s, scan the edges having s as left endpoint twice. When edge $(s,t), s < t$, is reached in the first scan, perform a binary search in R to find the queue q_i such that

$$R[i-1] > t \geq R[i].$$

Assign edge (s,t) to queue q_i. In the second scan of edges leaving s, update R to reflect the assignment of edges to queues. Clearly the algorithm maintains the conditions on R, and the edge assignment yields a queue layout.

It remains to show that, if some edge is assigned to queue q_k, then σ has a k-rainbow. Suppose (s,t) is assigned to queue q_k. Since $R[k-1] > t$ when vertex s is processed, edge (s,t) must nest inside some edge $(v, R[k-1])$ in queue q_{k-1}. Since t is assigned to queue k in the first scan, while $R[k-1]$ is updated in the second scan, the two scans at vertex s guarantee that $v < s$. By an easy induction, this observation extends to show that there are k nested edges, i.e., a k-rainbow.

The time complexity follows from the $O(\log\log n)$ search time for each edge. □

We might expect the dual 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 far from being true. For the fixed order σ, assigning edges to stacks is equivalent to coloring circle graphs [7]. While it is possible to determine the largest twist size for σ in polynomial time (Hsu [17]), minimizing the number of stacks cannot be done in polynomial time unless P = NP because coloring circle graphs is NP-complete (Garey, Johnson, Miller, and Papadimitriou [10]). To summarize, we have the following.

PROPOSITION 2.4 (see [7], [10]). *The problem of minimizing the number of stacks required by a fixed order σ is* NP*-complete.*

In this minimization sense, fixed-order queue layouts are easier than fixed-order stack layouts.

3. One-queue graphs. This section studies the class of 1-queue graphs, with the following results. Just as the 1-stack graphs admit a complete characterization as a class of planar graphs, so do the 1-queue graphs. However, whereas the 1-stack characterization is in terms of a known strengthening of the property of planarity (namely, outerplanarity), the 1-queue characterization employs a new strengthening (namely, arched-leveled planarity). Neither of these subclasses of the planar graphs includes the other. Whereas the 1-stack graphs can be recognized in linear time, we show that the recognition problem for 1-queue graphs is NP-complete. Thus, the recognition problem contrasts with the fixed-order layout problem, in that it points out a sense in which queues are more complicated than stacks.

3.1. Characterizing 1-queue graphs. Bernhart and Kainen [2] give the following characterization of 1-stack graphs.

PROPOSITION 3.1 (see [2]). *G is a 1-stack graph if and only if G is outerplanar.*

(An *outerplanar* graph is a planar graph having a planar embedding in which all vertices appear on a common face.) We show that the 1-queue graphs that have a particular kind of planar embedding are also planar graphs.

Consider the normal cartesian (x, y) coordinate system for the plane. For i an integer, let ℓ_i be the vertical line defined by $\ell_i = \{(i, y) \mid y \in \mathbf{Reals}\}$. A graph $G = (V, E)$ is *leveled-planar* if V can be partitioned into levels $V_1, V_2, \cdots, V_m$ in such a way that

- G has a planar embedding in which all vertices of V_i are on the 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.

Such a planar embedding is called a *leveled-planar embedding.* Figure 3.1 shows a leveled-planar graph having 3 levels. Henceforth, we assume that a valid (but arbitrary) leveled-planar embedding is given along with a leveled-planar graph.

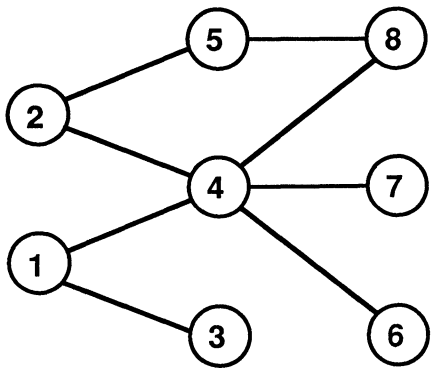

FIG. 3.1. *A leveled-planar graph.*

A leveled-planar embedding induces an order (the *induced order*) on V as follows. As i takes the values $1, 2, \cdots, m$, scan line ℓ_i from bottom to top. Label the vertices $1, 2, \cdots, n$ as they are encountered. For $1 \leq i \leq m$, let b_i be the (bottom) first vertex in level i, and let t_i be the (top) last. Let s_i be the first vertex in level i that is adjacent to some vertex in level $i + 1$, or, if there are no edges between levels i and $i + 1$, let $s_i = t_i$.

Consider augmenting G with new edges. A *level-i arch* for G is an edge connecting vertex t_i with vertex j, where $b_i \leq j \leq \min(t_i - 1, s_i)$. A leveled-planar graph G, augmented by any number of arches, can be embedded in the plane by drawing the arches around level 1; because of the leveling, the arches do not cross. See Fig. 3.2 where $(3, 5)$ and $(6, 8)$ are arches. A leveled-planar graph augmented by (zero or more) arches is called an *arched leveled-planar graph.* The edges that are not arches are called *leveled edges.* An arched leveled-planar graph that cannot be augmented with further arches or leveled edges is *maximal.* See Fig. 3.3 for an example. The above definitions for b_i, s_i, and t_i will be used throughout the paper to refer to vertices in arched leveled-planar graphs.

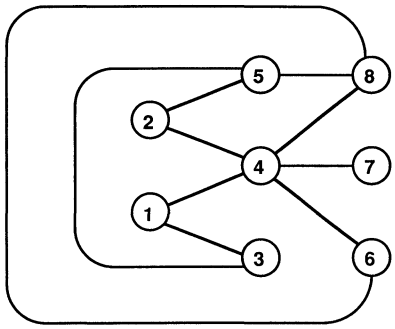

FIG. 3.2. *Drawing arches.*

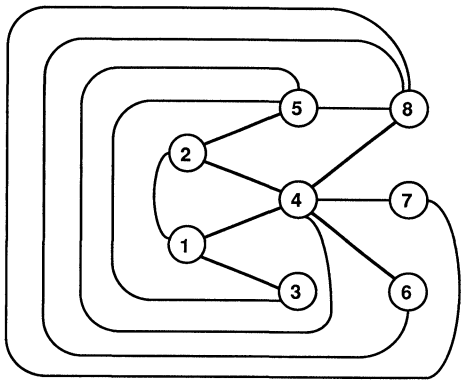

FIG. 3.3. *A maximal arched leveled-planar graph.*

We can now state the characterization of 1-queue graphs.

THEOREM 3.2. *A graph G is a 1-queue graph if and only if G is an arched leveled-planar graph.*

We develop the proof of the theorem through three lemmas.

LEMMA 3.3. *Every leveled-planar graph is a 1-queue graph. The induced order of vertices yields a 1-queue layout.*

Proof. Given a leveled-planar graph $G = (V, E)$ with m levels $V_1, V_2, \cdots, V_m$, order V in the induced order $1, \cdots, n$. We claim that this order yields a 1-queue embedding of G. It suffices to show that no two edges nest. If two edges have a vertex in common, then the edges cannot nest. So consider two edges (u_1, v_1) and (u_2, v_2) such that $u_1 < v_1$, $u_2 < v_2$, $u_1 < u_2$, and $v_1 \neq v_2$. If u_1 and u_2 are in the same level V_i, then v_1 and v_2 are in the same level V_{i+1}, and $v_1 < v_2$ because the edges do not intersect in the leveled-

planar embedding. If u_1 and u_2 are in different levels, ℓ and $m > \ell$, respectively, then v_1 and v_2 are in different levels, $\ell + 1$ and $m + 1$, respectively, and again $v_1 < v_2$. In either case, the two edges do not nest. Hence, the given layout is a 1-queue layout of G. □

LEMMA 3.4. *Every arched leveled-planar graph is a* 1*-queue graph. The induced order of vertices yields a* 1*-queue layout.*

Proof. Let $G = (V, E)$ be an arched leveled-planar graph. By the previous lemma, it suffices to show that no arch nests with another edge.

Let (u_1, t_i) and (u_2, t_j) be two arches. If $t_i = t_j$, then the arches do not nest since they have a vertex in common. If $t_i \neq t_j$, say $t_i < t_j$, then $u_1 < t_i < u_2 < t_j$, and the arches do not nest.

Now say that (u_3, v_3), $u_3 < v_3$, is a leveled edge between levels k and $k + 1$. Since every arch is between two vertices on the same level, no leveled edge can nest inside an arch. Conversely, for the arch (u_1, t_i) to nest inside (u_3, v_3), we must have $u_3 < u_1 < t_i < v_3$. There are two cases:

1. If u_3 is on the same level as the arch, then $u_1 \leq u_3$, a contradiction to $u_3 < u_1$;
2. If v_3 is on the same level as the arch, then $v_3 \leq t_i$, a contradiction to $t_i < v_3$.

Thus, (u_1, t_i) and (u_3, v_3) do not nest. We conclude that we have a 1-queue layout for G. □

LEMMA 3.5. *Every* 1*-queue graph is an arched leveled-planar graph.*

Proof. Let $G = (V, E)$ be an arbitrary 1-queue graph, and let $\sigma = 1, 2, \cdots, n$ be the order of a 1-queue layout of G. It suffices to describe an arched leveled-planar embedding of G. Without loss of generality, we may assume that G is connected.

Partition V into levels, as follows. Level V_1 is the singleton $\{1\}$, so that $b_1 = s_1 = t_1 = 1$. For $i > 1$, until each vertex is placed in some level, set

- $b_i = t_{i-1} + 1$;
- t_i equal to the rightmost vertex incident to some vertex in V_{i-1};
- $V_i = \{b_i, \cdots, t_i\}$;
- s_i equal to the leftmost vertex in V_i that is adjacent to some vertex to the right of t_i; $s_i = t_i$ if $t_i = n$.

Let the resulting partition be $V_1, V_2, \cdots, V_m$. This partition breaks the sequence σ into m contiguous subsequences that end at $1 = t_1, t_2, \cdots, t_m = n$, respectively. By the construction of V_i, it is clear that no edge connects a vertex in V_i with a vertex in V_j if $|i - j| \geq 2$. Let E_ℓ be the subset of E consisting of edges that connect vertices at consecutive levels; that is,

$$E_\ell = \{(u, v) \mid \text{for some } i, u \in V_i, v \in V_{i+1}\}.$$

Construct a leveled-planar embedding of $G_\ell = (V, E_\ell)$. Place the vertices of V_i on line ℓ_i in the order $b_i, b_i + 1, \cdots, t_i$ from bottom to top. Draw the edges in E_ℓ as line segments. We must show that this is indeed a leveled-planar embedding of G_ℓ. It suffices to show that any two distinct edges, (u_1, v_1), $u_1 < v_1$, and (u_2, v_2), $u_2 < v_2$, do not cross in the embedding. Without loss of generality, say that $u_1 \leq u_2$. If $u_1 = u_2$, then the edges do not cross because they share an endpoint. So say that $u_1 < u_2$. If u_1 and u_2 are embedded on different lines, then the edges cannot intersect because all edges go between adjacent lines. If u_1 and u_2 are on the same line, say line ℓ_i, then the queuing discipline guarantees that $u_1 < u_2 < v_1 \leq v_2$ and that v_1 and v_2 are both embedded on line ℓ_{i+1}. Therefore, edges (u_1, v_1) and (u_2, v_2) do not cross.

It remains to show that $E - E_\ell$ contains only arches for G_ℓ. Let $(u_3, v_3) \in E - E_\ell$, where $u_3, v_3 \in V_i$, $u_3 < v_3$. Clearly, $u_3 \leq \min(t_i - 1, s_i)$ since otherwise there is an edge from s_i to some vertex in V_{i+1} that nests over (u_3, v_3). Since t_i is adjacent to some vertex

$x \in V_{i-1}$, we must have $v_3 = t_i$, for otherwise (x, t_i) and (u_3, v_3) nest. We conclude that (u_3, v_3) is an arch for G_ℓ. Since that edge was arbitrary, it follows that $E - E_\ell$ contains only arches, so we have constructed an arched leveled-planar embedding of G. □

Theorem 3.2 follows from Lemmas 3.4 and 3.5.

The structural result of Theorem 3.2 allows us to determine the maximum number of edges in a 1-queue graph. It is well known that a maximal outerplanar graph on n vertices contains $2n-3$ edges. A similar result is now shown for maximal arched leveled-planar graphs. These bound are useful for establishing lower bounds on queuenumber or stacknumber; cf. Proposition 4.11.

THEOREM 3.6. *Let $G = (V, E)$ be a 1-queue graph having a maximal arched leveled-planar embedding of m levels. Assume that A of the levels $V_1, \cdots, V_{m-1}$ are singletons. Then G has exactly*

$$2|V| - 1 - |V_1| - A \;\leq\; 2|V| - 3$$

edges.

Proof. Partition E into levels $E_1, \cdots, E_m$, where an edge is in level E_i if its left endpoint is in V_i. Then all level-i arches are in E_i, and E_m contains only arches. For convenience, let $t_0 = 0$.

First we count the arches in the given embedding. By maximality of the embedding,

- E_i contains $s_i - t_{i-1}$ arches if $s_i \neq t_i$ and $s_i - t_{i-1} - 1$ arches if $s_i = t_i$;
- If $b_i \neq t_i$ (i.e., if $|V_i| > 1$), then there is a leveled edge connecting $t_i - 1$ to level $i+1$, so that $s_i \neq t_i$.

Thus E_i contains $s_i - t_{i-1} - 1 = 0$ arches only when V_i is a singleton.

Now we count the leveled edges in the embedding. Each leveled edge in E_i, $1 \leq i \leq m-1$, has one endpoint among $t_i - s_i + 1$ vertices in level i and one endpoint among $t_{i+1} - t_i$ vertices in level $i+1$. By planarity, there is a bottom-to-top order on the set of leveled edges in E_i. Scanning these edges in order, the first edge connects two vertices, and each subsequent edge connects a new vertex to a previously encountered vertex (because of maximality). Thus, the number of leveled edges in E_i is

$$(t_{i+1} - t_i) + (t_i - s_i + 1) - 1 = t_{i+1} - s_i.$$

Combining the counts of the preceding two paragraphs, for $1 \leq i \leq m-1$,

$$|E_i| = \begin{cases} t_{i+1} - t_{i-1} & \text{if } |V_i| > 1, \\ t_{i+1} - t_{i-1} - 1 & \text{if } |V_i| = 1. \end{cases}$$

By analogous reasoning,

$$|E_m| = t_m - t_{m-1} - 1.$$

The cardinality of E is, therefore,

$$\begin{aligned} |E| &= \sum_{i=1}^{m} |E_i| \\ &= t_m - t_{m-1} - 1 - A + \sum_{i=1}^{m-1} (t_{i+1} - t_{i-1}) \\ &= t_m - t_{m-1} - 1 - A + t_m + t_{m-1} - t_1 \\ &= 2t_m - 1 - t_1 - A \end{aligned}$$

$$= 2|V| - 1 - |V_1| - A$$
$$\leq 2|V| - 3. \qquad \square$$

Thus the greatest number of edges that can be assigned to a single queue is $2|V| - 3$. This value immediately yields a lower bound on the queuenumber of a graph.

COROLLARY 3.7. $\mathrm{QN}(G) \geq \lceil |E|/(2|V| - 3) \rceil$.

3.2. Recognizing 1-queue graphs. The 1-stack graphs are exactly the outerplanar graphs and, therefore, can be recognized in linear time (Syslo and Iri [25]). In contrast, we show that the problem of recognizing 1-queue graphs is NP-complete (see Garey and Johnson [9]). Formally, the recognition problem for 1-queue graphs is the following decision problem.

ARCHED LEVELED-PLANAR.

Instance. A graph $G = (V, E)$, represented by adjacency lists.

Question. Does G have an arched leveled-planar embedding?

Rather than prove the NP-completeness of ARCHED LEVELED-PLANAR directly, we introduce a seemingly simpler decision problem.

LEVELED-PLANAR.

Instance. A graph $G = (V, E)$, represented by adjacency lists.

Question. Does G have a leveled-planar embedding?

Notice that it is not immediate that either of these problems reduces to the other. Using a rather elaborate reduction, we show that LEVELED-PLANAR is NP-complete. At the end of the section, we indicate how the reduction should be modified to show that ARCHED LEVELED-PLANAR is NP-complete.

We now present the known NP-complete problem which, via reduction, establishes the NP-completeness of LEVELED-PLANAR and, thereby, of ARCHED LEVELED-PLANAR. An instance of 3-SAT [9] is a boolean formula ϕ in conjunctive normal form such that each clause contains at most 3 literals. Let $\{v_1, v_2, \cdots, v_n\}$ be the variables of ϕ, and let $\{c_1, c_2, \cdots, c_m\}$ be the clauses. Each c_j is a set containing at most 3 literals, where each literal is either a variable v_i or the complement $\overline{v_i}$ of a variable; call a clause containing exactly k literals a *k-clause.* The *graph* of ϕ, $G(\phi) = (V(\phi), E(\phi))$ has vertex set

$$V(\phi) = \{c_j \mid 1 \leq j \leq m\} \cup \{v_i \mid 1 \leq i \leq n\}$$

and edge set $E(\phi) = E_1 \cup E_2$, where

$$E_1 = \{(c_j, v_i) \mid v_i \in c_j \text{ or } \overline{v_i} \in c_j\},$$
$$E_2 = \{(v_i, v_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(v_n, v_1)\}.$$

The edges of E_2 form a cycle called the *variable cycle.* The graph in Fig. 3.4 represents the graph of the formula having clauses $c_1 = \{v_1, \overline{v_2}, v_5\}$, $c_2 = \{\overline{v_3}, \overline{v_4}, v_5\}$, $c_3 = \{\overline{v_1}, v_2\}$, $c_4 = \{v_2, v_3, v_4\}$, and $c_5 = \{v_2, v_4, \overline{v_5}\}$.

Lichtenstein [19] shows that the following restricted version of 3-SAT is NP-complete.

PLANAR 3-SAT (P3SAT).

Instance. An instance of 3-SAT ϕ such that $G(\phi)$ is planar.

Question. Is ϕ satisfiable?

It always suffices to consider only instances such that each clause contains either two or three literals. From Lemma 1 of [19], we may assume that $G(\phi)$ has a planar embedding such that, for each v_i, all clauses containing the literal v_i are on one side of

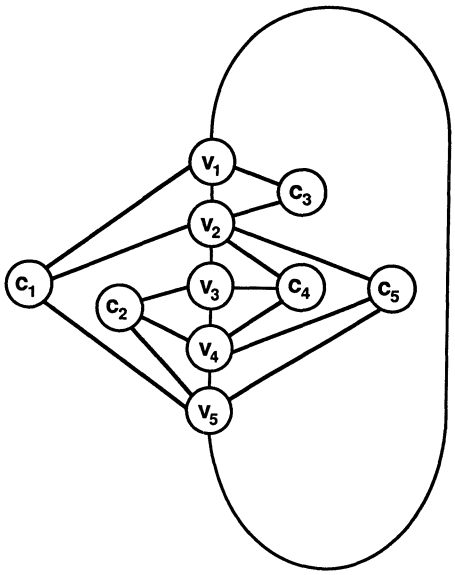

FIG. 3.4. *Example of* PLANAR 3-SAT.

the variable cycle, and all clauses containing the literal $\overline{v_i}$ are on the other side. Call this property of the planar embedding of $G(\phi)$ *consistency.* The planar embedding of Fig. 3.4 is consistent.

While LEVELED-PLANAR is as simple a recognition problem as we could formulate for queue layouts, we show that it is NP-complete in the next theorem. The proof is a long reduction from P3SAT to LEVELED-PLANAR. At the end of the subsection, the NP-completeness of ARCHED LEVELED-PLANAR is resolved by a slight modification of this reduction.

THEOREM 3.8. LEVELED-PLANAR *is* NP*-complete.*

Proof. We reduce P3SAT to LEVELED-PLANAR. As LEVELED-PLANAR is easily in NP, this suffices to prove the theorem.

Let $V = \{v_1, \cdots, v_n\}$ and $C = \{c_1, \cdots, c_m\}$ be an instance of P3SAT. Fix a planar embedding of $G(\phi)$ that is consistent. We will construct an instance H of LEVELED-PLANAR, which is a biconnected planar graph.

Here is an overview of our reduction strategy. We start with a consistent planar embedding of $G(\phi)$ in which the vertices in the variable cycle are in order on a vertical line (e.g., as in Fig. 3.4). Our strategy is to replace all vertices *and* edges in $G(\phi)$ by gadgets that have restricted leveled-planar embeddings. The graph H resulting from these replacements is "rigid" in the sense that in any leveled-planar embedding of H the *relative* levels of any two clause gadgets are fixed. A variable gadget has the flexibility of being in one of two different levels to represent *true* and *false* values of the variable. The gadget for an edge (called a *rod*) is "rigid" in the sense that the number of levels between its two ends is constant. Rods are used both to make H rigid and to transmit the truth value of a variable from the variable gadget to the gadgets of the clauses that use the variable.

As an example, consider Fig. 3.5, the rigid version of the graph from Fig. 3.4. New vertices $u_0, \cdots, u_5$ have been interspersed among $v_1, \cdots, v_5$. New vertices c_0 and c_6 have also been added as dummy clauses. All vertices and edges in Fig. 3.5 represent gadgets used in the construction of H. The construction is such that the subgraph represented

by the cycle c_0, u_0, c_6, u_5, c_0 is a fixed framework that has essentially only one leveled-planar embedding (assuming the c_0 gadget occupies earlier levels than the c_6 gadget). The subgraphs representing $u_0, \cdots, u_5$ must all appear in the same 3 adjacent levels. The subgraphs representing variables $v_1, \cdots, v_5$ have the freedom to be in 2 different positions in a leveled-planar embedding depending on their truth values. The edges that go generally left to right are replaced by rods of appropriate lengths. This is done in a way that fixes the levels occupied by the gadget of any clause. The clause gadgets have the ability to evaluate a boolean OR, in the sense that any clause gadget can successfully be placed in a leveled-planar layout if and only if at least one of its corresponding literals is assigned *true*. H has the property that it has a leveled-planar embedding if and only if ϕ has a satisfying assignment. The truth values in a truth assignment for ϕ specify a leveled-planar embedding for the variable cycle that can be extended to a leveled-planar embedding for H in the case that the assignment is a satisfying assignment.

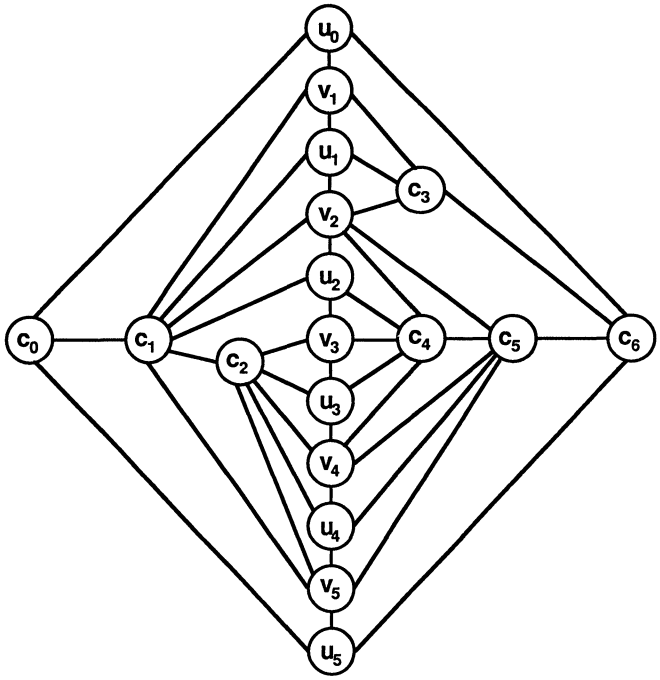

FIG. 3.5. *Rigid* PLANAR 3-SAT.

We begin the proof with some useful building blocks. Consider the copy of $K_{2,3}$ in Fig. 3.6. Suppose this copy is in a leveled-planar embedding. Then a_1 and a_2 are exactly two levels apart, and b_1, b_2, and b_3 are all on the level in between. Further, only two of b_1, b_2, and b_3 can have any additional edges incident to them. In a leveled-planar embedding, the leveling of any copy of $K_{2,3}$ is forced. If b_1, b_2, and b_3 are thought of as a single vertex, then $K_{2,3}$ is thought of as a path of length 2. In a leveled embedding, $K_{2,3}$ differs from a path of length 2 in the sense that it cannot "bend" in the middle in order to bring the ends together; its two endpoints must appear two levels apart. We think of $K_{2,3}$ as a *rigid path of length* 2. By joining $k-1$ copies of $K_{2,3}$ in the manner illustrated in Fig. 3.7 for $k = 3$, rigid paths of any length k can be obtained. In general, we call a rigid path of length k a *k-rod*. (A 1-rod is an edge.) We draw a k-rod as a thick hollow line with intermediate vertices as needed (Fig. 3.8). Note that what appears to

be a single intermediate vertex is actually two different vertices, one on the top and one on the bottom of the rod (the third vertex is inaccessible, hence ignored).

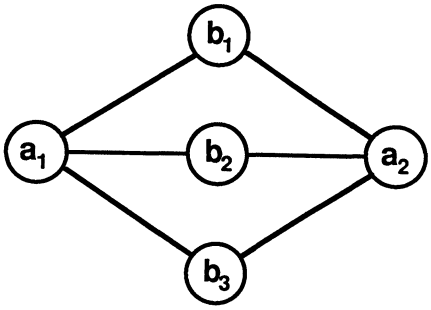

FIG. 3.6. *A 2-rod.*

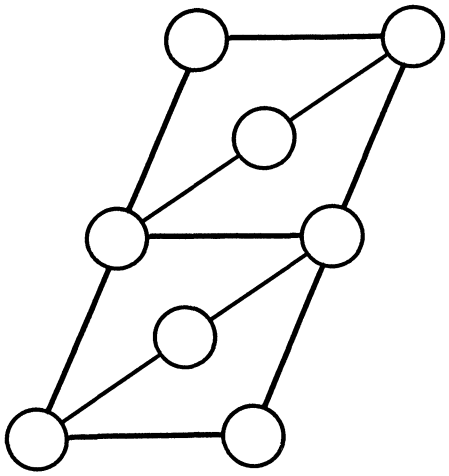

FIG. 3.7. *A 3-rod.*

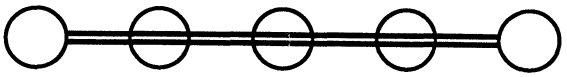

FIG. 3.8. *Representation of a k-rod,* $k = 4$.

A second building block is called a *semi-rod.* It consists of a 3-rod and a 2-rod connected by 2 edges. See Figs. 3.9 and 3.10. A semi-rod has one degree of flexibility that a 5-rod does not have: if x is in level t and y in level $t-1$, then z is either in level $t-5$ (Fig. 3.9, where the semi-rod is extended to its greatest length) or $t-3$ (Fig. 3.10, where the semi-rod is compressed); note that z is always at a lower level than y. We draw a semirod as a 5-rod with a textured interior (Fig. 3.11).

In the construction of H, some vertices will be called *fixed.* If x is a fixed vertex, we intend that, in any leveled-planar embedding of H, the level containing x is always the same (given that a particular vertex, to be specified later, is on the first level). The level in which x should appear is its *preferred level* $\mathcal{L}(x)$. If we fix the two ends of a rod, then the intermediate vertices of the rod are also fixed, with preferred levels derived in the obvious way. During the construction, we designate certain vertices x as fixed and give

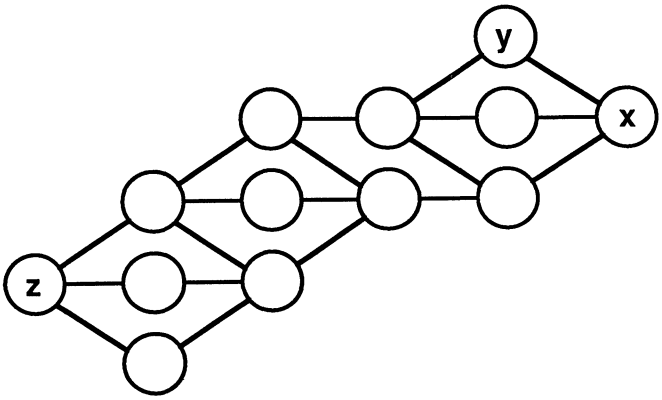

FIG. 3.9. *A semi-rod extended.*

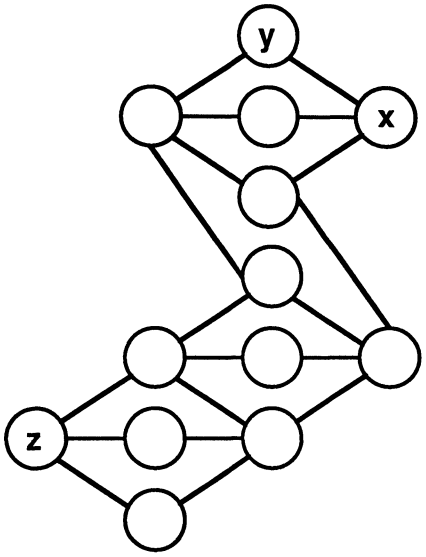

FIG. 3.10. *A semi-rod compressed.*

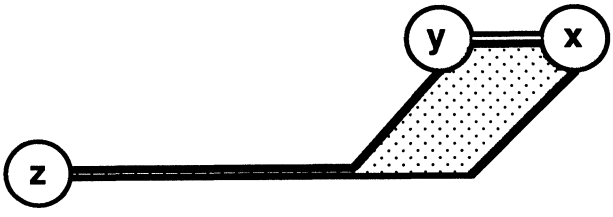

FIG. 3.11. *Representation of a semi-rod.*

a value to $\mathcal{L}(x)$. We show later that there is a leveled-planar embedding of H if and only if there is such an embedding where each fixed x indeed appears on level $\mathcal{L}(x)$.

To begin the construction for formula ϕ, we represent each variable v_i by a 2-rod $VROD[i]$ having left and right endpoints $S[i]$ and $T[i]$. (In other words, think of $S[i]$ appearing in an earlier level than $T[i]$.) Represent the edge (v_i, v_{i+1}) of $G(\phi)$, $1 \leq i \leq n-1$, by a 2-rod $EROD[i]$ having left endpoint $V[i]$ and right endpoint $W[i]$, as shown in Fig. 3.12. Partially represent the edge (v_n, v_1) by a 2-rod $EROD[0]$ connected to $VROD[1]$ and by a 2-rod $EROD[n]$ connected to $VROD[n]$ (the representation of the edge will be completed later). Call the graph constructed so far P. P may be thought of as a path of thickness 3 from $EROD[0]$ to $EROD[n]$. The vertices of each $EROD$ are fixed, and the vertices of each $VROD$ are not fixed. Note that, in any leveled-planar embedding containing P, the levels of $T[i]$ and of $W[i]$ differ by exactly one level. Further, if $W[0]$ is to the right of $V[0]$, then each $W[i]$ is to the right of $V[i]$, and vice versa. By symmetry, we may assume that any leveled-planar embedding of P has each $W[i]$ to the right of $V[i]$, and, therefore, each $\mathcal{L}(V[i]) = \mathcal{L}(W[i]) - 2$, $0 \leq i \leq n$. The intention of the construction (not yet realized) is that all $W[i]$'s appear on the same level, namely, $\Lambda = \mathcal{L}(W[0])$. For the time being, we use the level Λ as a relative reference for other $\mathcal{L}$ values. Call the property of all $W[i]$'s appearing on level Λ *line up.*

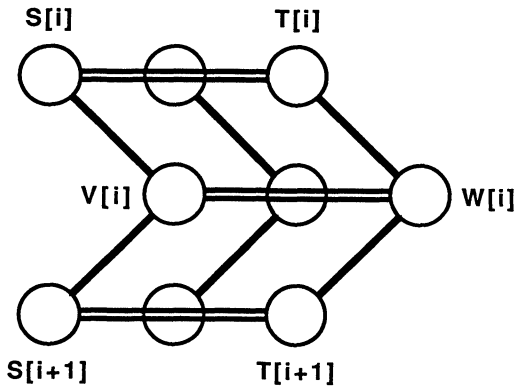

FIG. 3.12. *Representing the variable path.*

Because the embedding of $G(\phi)$ is planar, the variable cycle partitions the clause set C into two subsets, C_1 and C_2, in such a way that the clauses in C_1 nest and the clauses in C_2 nest. We will place the clauses in C_1 to the left of P and the clauses in C_2 to the right. (In Fig. 3.4, $C_1 = \{c_1, c_2\}$ and $C_2 = \{c_3, c_4, c_5\}$. Clause c_2 is nested under clause c_1, and clause c_4 is nested under clause c_5.) A clause $c_j \in C_2$ is associated with the 2 or 3 $T[i]$'s that correspond to its variables. (Similarly, a clause in C_1 is associated with 2 or 3 $S[i]$'s.) Because the embedding of $G(\phi)$ is consistent, each $T[i]$ can be associated consistently with either v_i or $\overline{v_i}$ (the corresponding $S[i]$ is associated with the complementary literal). If the $W[i]$'s line up (on level Λ), then each $T[i]$ appears either on level $\Lambda + 1$ or $\Lambda - 1$. If $T[i]$ is on level $\Lambda - 1$, then we say that $T[i]$ is *intruded* and $S[i]$ is *extruded;* otherwise (i.e., $T[i]$ is on level $\Lambda + 1$), $T[i]$ is *extruded* and $S[i]$ is *intruded.* We interpret the literal associated with each $T[i]$ (or $S[i]$) as *true* or *false*, respectively, according as whether $T[i]$ (or $S[i]$) is intruded or not.

We need gadgets for each clause in C. By mirror-image symmetry in P, we consider only clauses in C_2 and place their gadgets to the right of P. Construct the gadgets for the clauses in C_2 in any order that obeys the nesting of clauses, taking the more deeply

nested clauses earlier. (In Fig. 3.4, construct the gadget for c_4 before the gadget for c_5; the gadget for c_3 is unconstrained.) Figure 3.13 shows the gadget for a clause. The gadget contains two semirods that share an edge. $U[j]$, $X[j]$, and $Y[j]$ (among others) are fixed vertices. $X[j]$ and $Y[j]$ must appear in the same level, which is 4 levels before the level of $U[j]$. $Q[j,1]$, $Q[j,2]$, and $Q[j,3]$ are connected to the literals in the clause by rods. They may be either 3 or 5 levels before $U[j]$, depending on the truth value of the corresponding literal.

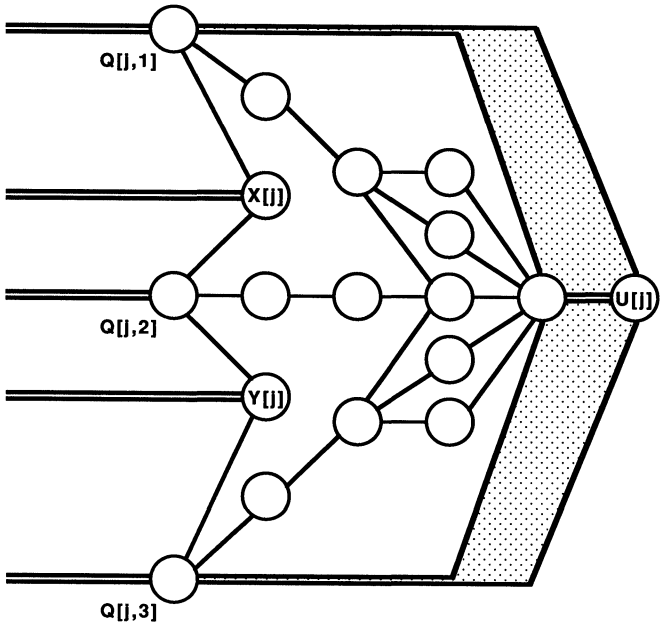

FIG. 3.13. *The gadget for a* 3*-clause.*

We first assume that $c_j \in C_2$ is a 3-clause. Let c_j be associated with $T[i_1]$, $T[i_2]$, and $T[i_3]$ in order from top to bottom. By the construction regimen, the gadgets for any clauses nested inside c_j have already been constructed. The gadget for each $c_s \in C_2$ contains a fixed vertex $U[s]$ that is visible on the right side of the gadget. If no clauses nest under c_j, then $\mathcal{L}(U[j]) = \Lambda+6$. If there are one or more clauses nested under c_j, let c_s be one that maximizes $\mathcal{L}(U[s])$. Put $\mathcal{L}(U[j]) = \mathcal{L}(U[s])+6$. Let $k = \mathcal{L}(U[j])-\Lambda-4$. Place a k-rod on each of $T[i_1], T[i_2], T[i_3]$, and connect them to $Q[j,1], Q[j,2], Q[j,3]$, as shown by the k-rods on the left in Fig. 3.13. $Q[j,a]$ is *intruded* (*extruded*) exactly when $T[i_a]$ is intruded (extruded). $X[j]$ and $Y[j]$ are fixed with $\mathcal{L}(X[j]) = \mathcal{L}(Y[j]) = \mathcal{L}(U[j]) - 4$. There will be $U[s]$'s or $W[i]$'s visible under $X[j]$ (or $Y[j]$). Connect a rod of the appropriate length from $X[j]$ (or $Y[j]$) to each visible $U[s]$ and $W[i]$. For example, between $X[j]$ and $U[s]$, connect a $(\mathcal{L}(X[j]) - \mathcal{L}(U[s]))$-rod.

In the case that c_j is a 2-clause, the gadget is the same. There are only two vertices, $T[i_1]$ and $T[i_2]$, associated with c_j. Connect $T[i_1]$ to $Q[j,1]$ and $T[i_2]$ to $Q[j,3]$ with rods as before. There will be at least one fixed vertex visible under $X[j]$, $Y[j]$, and $Q[j,2]$ (that is, some $U[s]$ or $W[i]$). Connect rods of appropriate lengths between one such

fixed vertex and $X[j]$, $Y[j]$, and $Q[j,2]$, so that $Q[j,2]$ is always extruded. Then c_j is represented by the gadget in the same manner as a 3-clause in which the second literal is always false. This allows us to treat every clause as though it were a 3-clause.

Once gadgets have been constructed for each clause to the right of P, *cap* the right end of H with a path around the right end, in the following sense. Let $c_s \in C_2$ have maximum level $\mathcal{L}(U[s])$. (If $C_2 = \emptyset$, by abuse of notation, take $\mathcal{L}(U[s]) = \Lambda$.) Let $k = \mathcal{L}(U[s]) - \Lambda + 3$. Place a k-rod at each of $W[0]$ and $W[n]$; identify their free ends (Fig. 3.14). Notice that two edges, one from each rod, are also identified as the edge $(X[m+1], Z[right])$. $X[m+1]$ is a fixed vertex with $\mathcal{L}(X[m+1]) = \Lambda + k - 1$. Some $U[s]$'s or $W[i]$'s will be visible from $X[m+1]$. Connect $X[m+1]$ to each of them with a rod of appropriate length. The cap around the right end may be thought of as a dummy clause containing no literals whose purpose is to provide a rod for any $U[s]$'s and $W[i]$'s that have yet to be connected to a rod.

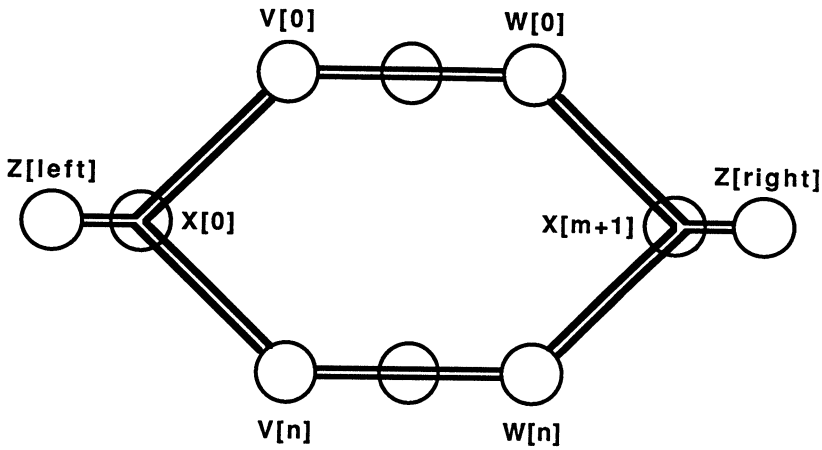

FIG. 3.14. *Capping the left and right ends.*

After the gadgets for the clauses to the left of P are constructed, cap the left end by two rods from $V[0]$ to $X[0]$ and from $V[n]$ to $X[0]$ in a manner similar to the preceding paragraph. This completes the construction of H.

So far, all the $\mathcal{L}$ values have been relative to $\Lambda = \mathcal{L}(W[0])$. Now fix $\mathcal{L}(Z[left]) = 1$, so that

$$\Lambda = \mathcal{L}(W[0]) - \mathcal{L}(Z[left]) + 1.$$

Remaining $\mathcal{L}$ values are adjusted according to the value of Λ.

Clearly, the construction can be accomplished in polynomial time. Also, H is a planar graph with a planar embedding that is essentially unique except for some inconsequential freedom in embedding the intermediate vertices in k-rods.

Every $V[i]$, $W[i]$, and $U[j]$ in H has a rod connecting it to either an $X[j]$ or a $Y[j]$. If H has a leveled-planar embedding, then it has a leveled-planar embedding in which (1) $Z[left]$ is in level 1; (2) every vertex in the capping cycle

$$Z[left], \cdots, V[0], \cdots, W[0], \cdots, Z[right], \cdots, W[n], \cdots, V[n], \cdots, Z[left]$$

is in its preferred level; (3) $W[0]$ is above $W[n]$ in level Λ. Because the capping cycle has only one leveled-planar embedding satisfying these constraints, all other vertices are

forced to be inside the capping cycle. In such an embedding, we want each fixed vertex to be in its preferred level. We show in the following two claims that this must be the case. In Claim 1, we assume that, for a particular clause $c_j \in C_2$, $U[j]$ is in level t, $X[j]$ and $Y[j]$ are in level $t-4$, the rod connected to $U[j]$ goes right, and the rods connected to $Q[j,1], X[j], Q[j,2], Y[j], Q[j,3]$ go left.

CLAIM 1. The gadget for c_j has such a leveled-planar embedding if and only if at least one of $Q[i,1], Q[i,2], Q[i,3]$ is intruded.

Proof. If $Q[j,1]$, $Q[j,2]$, and $Q[j,3]$ are all intruded, Fig. 3.13 shows such an embedding. If only $Q[j,2]$ is intruded, Fig. 3.15 shows such an embedding. If only $Q[j,1]$ (or, by symmetry, only $Q[j,3]$) is intruded, Fig. 3.16 shows such an embedding. If two of $Q[j,1]$, $Q[j,2]$, and $Q[j,3]$ are intruded, moving an appropriate $Q[j,a]$ in one of these figures easily gives an embedding. From these three figures, it is clear that there is no leveled-planar embedding of the gadget if all three vertices $Q[j,1]$, $Q[j,2]$, and $Q[j,3]$ are extruded. □

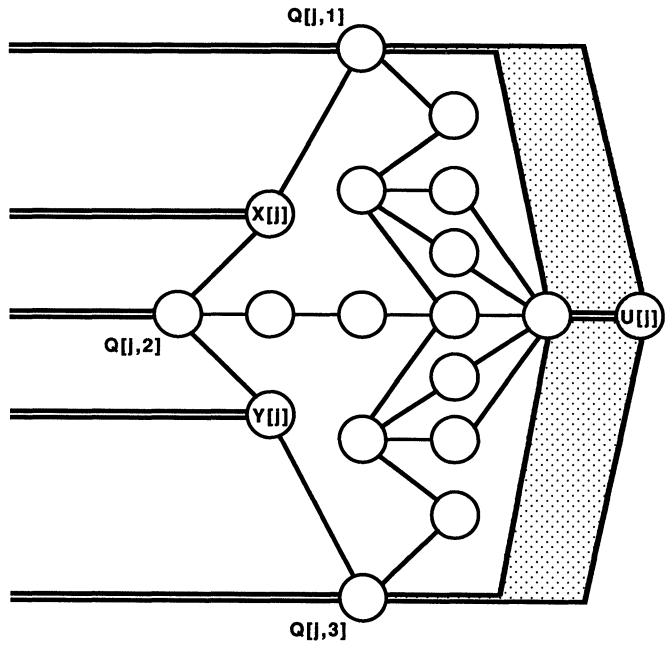

FIG. 3.15. $Q[j,2]$ *intruded.*

CLAIM 2. If H has a leveled-planar embedding such that each vertex in the capped cycle is in its preferred level, then each fixed vertex of H is in its preferred level.

Proof. Say that there is a fixed vertex not in its preferred level. If there is such a vertex in an $EROD$, let i be a smallest index for which $EROD[i]$ contains such a vertex. By left-right symmetry, it suffices to consider the case that $W[i]$ is in a level $t > \Lambda$. Because i is minimum, $t = \Lambda + 2$. $W[i]$ is connected by a rod to either an $X[j]$ or a $Y[j]$. Without loss of generality, say that the rod is to $X[j]$. The rod forces $X[j]$ to be in level $\mathcal{L}(X[j]) + 2 = \mathcal{L}(U[j]) - 2$. We claim that $U[j]$ is in a level higher than $\mathcal{L}(U[j])$.

Suppose $U[j]$ is in level $\mathcal{L}(U[j])$. The semi-rod to $Q[j,1]$ forces $Q[j,1]$ to be in level $\mathcal{L}(U[j]) - 3$. (Otherwise, $W[i]$ could not be in level $\Lambda + 2$.) The rods of $X[j]$ and $Q[j,1]$

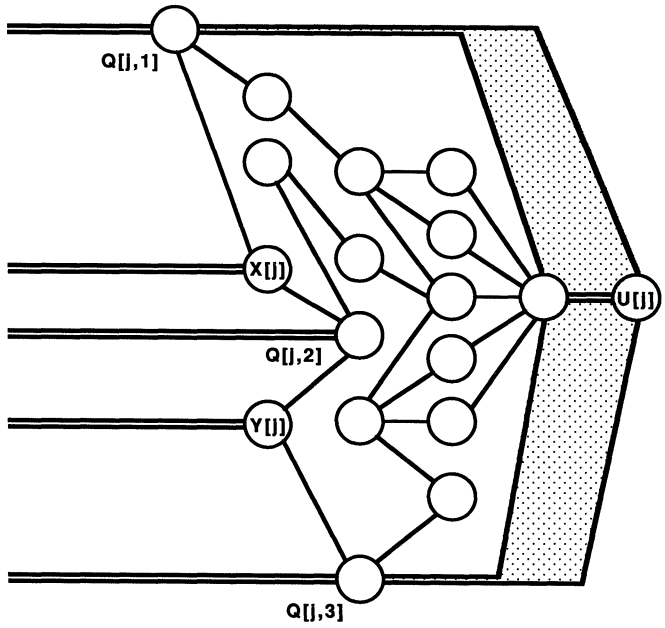

FIG. 3.16. $Q[j, 1]$ *intruded.*

force $Q[j, 1]$ to be above $X[j]$. Therefore, it is not possible to embed the paths from $Q[j, 1]$ to $U[j]$ properly in levels.

We conclude that $U[j]$ is in level $\geq \mathcal{L}(U[j]) + 2$. The above argument repeats with the rod from $U[j]$ connecting to some $X[j']$ or $Y[j']$, shifting $U[j']$ to a level higher than $\mathcal{L}(U[j'])$. Repetition of the argument ends at $X[m+1]$, which must be in level $\mathcal{L}(X[m+1])$, not higher. This contradiction proves that each $W[i]$ is in level Λ.

A similar argument shows that any fixed vertex that is not a $W[i]$ must also be in its preferred level. □

We need to show that ϕ is satisfiable if and only if H has a leveled-planar embedding.

Assume that ϕ is satisfiable. Choose a satisfying assignment for ϕ. Embed P first. Place all fixed vertices of P on their assigned levels. If v_i is true, let whichever of $S[i]$ and $T[i]$ corresponds to the literal v_i be intruded. If v_i is false, let whichever of $S[i]$ and $T[i]$ corresponds to the literal $\overline{v_i}$ be intruded. Then each $u[j]$ has at least one intruded $Q[i, j]$ and can be level embedded by Claim 1. Thus H has a leveled-planar embedding.

Now assume that H has a leveled-planar embedding. By Claim 2, we may assume that each fixed vertex F is on level $\mathcal{L}(F)$. Let $Z[i]$ be whichever of $S[i]$ and $T[i]$ corresponds to the literal v_i. If $Z[i]$ is intruded, assign v_i the value true; otherwise, assign v_i the value false. By Claim 1, every $U[j]$ has an intruded $Q[i, j]$. Therefore, each clause c_j contains a literal that is true under this assignment. This truth assignment satisfies ϕ; that is, ϕ is satisfiable.

Thus P3SAT reduces to LEVELED-PLANAR. As P3SAT is NP-complete, we conclude that LEVELED-PLANAR is NP-complete. □

It appears that the graph H is arched leveled-planar if and only if it is leveled-planar. To be certain of this, we modify the construction slightly by adding an *arched cap* on the left and right ends of H. The cap on the right end is shown in Fig. 3.17. The rightmost

edge of the right cap and the leftmost edge of the left cap must be arches, and no other edges may be arches. With this change to H, H is an arched leveled-planar graph if and only if ϕ is satisfiable. This proves the following corollary.

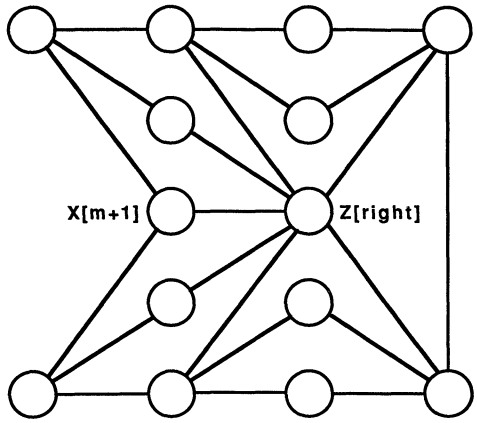

FIG. 3.17. *An arched cap.*

COROLLARY 3.9. ARCHED LEVELED-PLANAR *is* NP*-complete. Thus, the problem of recognizing* 1*-queue graphs is* NP*-complete.*

4. Layouts for specific graphs. In this section, we present queue layouts having small queuenumber for a variety of specific families of graphs. We make contrasts with the corresponding stack layouts. The intuition that an easily leveled graph has a good queue layout is supported by most of these families. Some details are left to the reader.

4.1. Trees and meshes. We begin with trees and meshes, two natural leveled-planar families of graphs.

A *tree* T is a connected graph that has no cycles. Choose an arbitrary vertex r to be the *root* of T. Each vertex in T has a well-defined *depth*, i.e., distance from r. Let $\mathrm{DEPTH}(i)$, $i \geq 0$, consist of all vertices at depth i.

PROPOSITION 4.1. *Every tree* T *is a leveled-planar, hence* 1*-queue, graph.* T *has a* 1*-queue layout such that the first level is* $\{r\}$ *and the queuewidth of the layout is the cardinality of the largest* $\mathrm{DEPTH}(i)$, $i > 0$.

Proof. Lay T out breadth-first starting from the root r. The result is a 1-queue layout of the tree with the stated properties. □

An $m \times n$ *mesh* is a graph with vertices

$$\{v_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$$

and edges

$$\{(v_{ij}, v_{i,j+1}) \mid 1 \leq j \leq n-1\} \cup \{(v_{ij}, v_{i+1,j}) \mid 1 \leq i \leq m-1\}.$$

PROPOSITION 4.2. *An* $m \times n$ *mesh is a leveled-planar, hence* 1*-queue, graph. There is a* 1*-queue layout* QL *of the mesh having queuewidth* $\mathrm{QW}(QL) \leq \min\{m, n\}$.

Proof. An $m \times n$ mesh has a natural embedding in the plane with vertices in m rows and n columns. If this embedding is rotated 45°, vertices line up on $m + n - 1$ vertical lines. The result is a leveled-planar embedding with the stated queuewidth. □

For $m > 2$, $n > 2$, an $m \times n$ mesh is not an outerplanar graph and, in fact, has stacknumber 2 [5]. Thus, the mesh provides an example of a 1-queue graph that fails to be a 1-stack graph.

4.2. Unicyclic graphs. A *unicyclic* graph is an undirected graph in which each connected component contains at most one cycle. The family of unicyclic graphs includes trees, forests, and cycles of all lengths.

PROPOSITION 4.3. *A unicyclic graph is an arched leveled-planar, hence* 1*-queue, graph. Each connected component contributes at most one arch.*

Proof. Let $G = (V, E)$ be a unicyclic graph. We may assume that G is connected. By Proposition 4.1, we need only treat the case that G contains a cycle. Let

$$C = u_1, u_2, \cdots, u_k, u_1$$

be that cycle. If k is even, level C into $\frac{k}{2} + 1$ levels

$$U_1 = \{u_1\}, U_2 = \{u_2, u_k\}, \cdots, U_i = \{u_i, u_{k-i+2}\}, \cdots, U_{(k/2)+1} = \{u_{(k/2)+1}\};$$

a leveled-planar embedding of C results. If k is odd, level C into $\frac{k+1}{2}$ levels

$$U_1 = \{u_1, u_k\}, U_2 = \{u_2, u_{k-1}\}, \cdots, U_i = \{u_i, u_{k-i+1}\}, \cdots, U_{(k+1)/2} = \{u_{(k+1)/2}\};$$

an arched leveled-planar embedding of C results, with the single arch (u_1, u_k).

Let G' be G without the edges of C. G' contains one connected component for each u_i; this connected component is a tree T_i, which we root at u_i. We convert the (arched) leveled-planar embedding of C into one for G by expanding T_i from u_i in a breadth-first manner, as prescribed in Proposition 4.1. □

4.3. X-trees. The *depth-d complete binary tree* $CBT(d)$ has vertex set

$$\{1, 2, \cdots, 2^{d+1} - 1\}$$

and edge set

$$\{(\alpha, 2\alpha), (\alpha, 2\alpha + 1) \mid 1 \leq \alpha \leq 2^d - 1\}.$$

The root of $CBT(d)$ is 1, and $CBT(d)$ has $d+1$ levels in the leveling starting at the root. The *depth-d X-tree* $X(d)$ is the supergraph of $CBT(d)$ that has edges added across each of the levels from left to right. See Fig. 4.1.

Every $X(d)$ is a 2-stack graph; when $d \leq 2$, $X(d)$ is a 1-stack graph [5]. In contrast, even small X-trees require two queues.

PROPOSITION 4.4. *For $d \geq 1$, $X(d)$ admits a 2-queue layout with queuewidths 2^d and 1. For $d \geq 2$, $X(d)$ is not a 1-queue graph.*

Proof. For the upper bound, choose the order $\sigma = 1, 2, \cdots, 2^{d+1} - 1$. The edges of $CBT(d)$ are assigned to one queue and the edges across each level are assigned to a second queue.

For the lower bound, since $X(2)$ is a subgraph of $X(d)$, $d \geq 2$, it suffices to show that $X(2)$ is not a 1-queue graph.

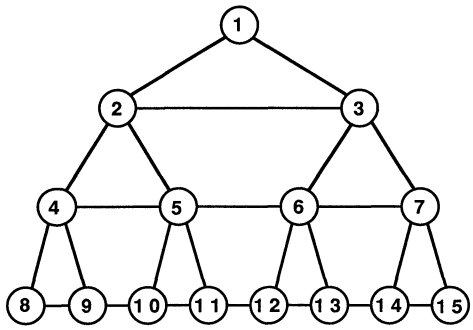

FIG. 4.1. *X-tree* $X(3)$.

We exploit an alternate means of constructing $X(2)$. Given any graph G and any edge (x, y) in G, define the operation of *hatting* (x, y) as adding a new vertex (the *peak*) z and new edges (x, z) and (y, z). Start with a cycle of length 4:

$$C = u_1, u_2, u_3, u_4, u_1.$$

Choose any three of the four edges of C. Hat each of the chosen edges. The resulting graph is isomorphic to $X(2)$.

To obtain a contradiction, suppose that $X(2)$ has a 1-queue layout. Let σ be the order of the vertices. Without loss of generality, assume that u_1 is the leftmost vertex of C in σ. Neither u_2 nor u_4 can be the rightmost vertex of C in σ, for then two edges of C would nest. By symmetry we may assume that the order of the vertices of C in σ is u_1, u_2, u_4, u_3. Three of the four edges of C must be hatted. In particular, either u_1 or u_3 has both of its incident edges hatted. By symmetry, we may assume that (u_1, u_4) and (u_1, u_2) are hatted. Let w be the peak of (u_1, u_4).

There are five possible placements of w within the order u_1, u_2, u_4, u_3. Only placement of w between u_1 and u_2 fails to yield two nested edges. But, with w between u_1 and u_2, there is no placement of the peak of (u_1, u_2) that does not yield two nested edges. This is a contradiction to σ giving a 1-queue layout of $X(2)$. □

Since $X(2)$ is outerplanar, we have the following corollary.

COROLLARY 4.5. $X(2)$ *is a* 1*-stack graph that is not a* 1*-queue graph.*

4.4. DeBruijn graphs. The *order-d deBruijn graph* $DB(d)$ has vertex set

$$\{0, 1, \cdots, 2^d - 1\}$$

and edges connecting each vertex x with vertices $2x \bmod 2^d$ and $2x+1 \bmod 2^d$. See Fig. 4.2. Note that multiple edges and loops are discarded.

PROPOSITION 4.6. $DB(d)$ *admits a* 2*-queue layout with queuewidths* 2^{d-1}. $DB(d)$, $d \geq 4$, *does not admit a* 1*-queue layout.* $DB(3)$ *does admit a* 1*-queue layout.*

Proof. The edges of $DB(d)$ of the forms $(x, 2x)$ and $(x, 2x+1)$, $x \in \{1, 2, \cdots, 2^{d-1} - 1\}$, are the edges of a depth-$(d-1)$ complete binary tree rooted at vertex 1 and containing all vertices except 0. Similarly, the edges of the forms $(2^{d-1}+x, 2x)$ and $(2^{d-1}+x, 2x+1)$, $x \in \{0, 1, \cdots, 2^{d-1} - 2\}$, are the edges of a depth-$(d-1)$ complete binary tree rooted at vertex $2^{d-1} - 2$ and containing all vertices except $2^{d-1} - 1$. Choose the order $\sigma = 0, 1, \cdots, 2^d - 1$. Assign edges of the forms $(x, 2x)$ and $(x, 2x+1)$ to one queue and edges of the forms $(2^{d-1} + x, 2x)$ and $(2^{d-1} + x, 2x + 1)$ to a second queue. (When edges are assigned to both queues, break ties arbitrarily.)

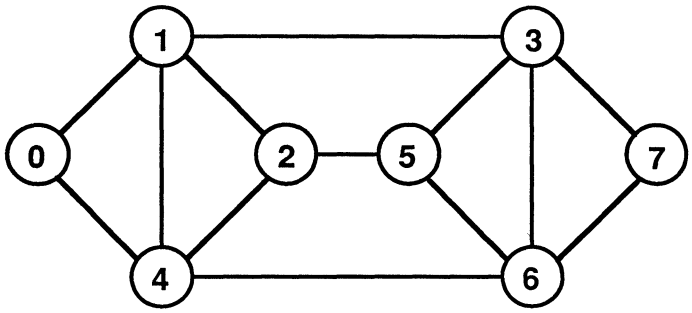

FIG. 4.2. *The deBruijn graph* $DB(3)$.

$DB(d)$, $d \geq 4$, is not planar, hence not a 1-queue graph. The order

$$\sigma = 1, 0, 2, 3, 4, 5, 7, 6$$

yields a 1-queue layout of $DB(3)$. □

4.5. Complete graphs. The *complete graph* K_n has a vertex set of size n and an edge connecting every pair of vertices.

PROPOSITION 4.7. $\mathrm{QN}(K_n) = \lfloor n/2 \rfloor$.

Proof. Every vertex order for K_n is symmetric, so fix any order $\sigma = 1, 2, \cdots, n$. The maximum size of a set of nesting edges is exactly $\lfloor n/2 \rfloor$. By Proposition 2.1 and Theorem 2.3, the result follows. □

An explicit assignment of edges of K_n to queues is easily described. In the fixed order σ, every edge (i, j) has length $|i - j|$. There are edges of every length from 1 to $n - 1$. For $i \in \{1, 2, \cdots, \lfloor n/2 \rfloor\}$, assign all edges of lengths $2i - 1$ and $2i$ to queue q_i. No two edges having the same length or having lengths differing by 1 can nest.

4.6. Complete bipartite graphs. The *complete bipartite graph* $K_{m,n}$ has $m + n$ vertices, partitioned into two sets:

$$\{a_1, a_2, \cdots, a_m\} \cup \{b_1, b_2, \cdots, b_n\};$$

its edges connect every a vertex with every b vertex.

PROPOSITION 4.8. $\mathrm{QN}(K_{m,n}) = \min(\lceil m/2 \rceil, \lceil n/2 \rceil)$.

Proof. Without loss of generality, assume that $m \leq n$. We need to show that

$$\mathrm{QN}(K_{m,n}) = \lceil m/2 \rceil .$$

Upper bound. Choose the layout order

$$\sigma = a_1, a_2, \cdots, a_{\lceil m/2 \rceil}, b_1, \cdots, b_n, a_{\lceil m/2 \rceil + 1}, \cdots, a_m.$$

Partition the edges of $K_{m,n}$ into $\lceil m/2 \rceil$ sets, each of which will be assigned to a distinct queue. The ith set, $1 \leq i \leq \lceil m/2 \rceil$, comprises all edges of the form (a_i, b_j) and (a_{m+1-i}, b_j), for $1 \leq j \leq n$. Since none of the edges in the ith set nest, they can all be assigned to a single queue, whence $\lceil m/2 \rceil$ queues suffice.

Lower bound. Let σ be an order of the vertices in a $\mathrm{QN}(K_{m,n})$-queue layout of $K_{m,n}$. By symmetry, we may assume that the a_i's appear in the order $a_1, a_2, \cdots, a_m$ in σ and that the b_j's appear in the order $b_n, b_{n-1}, \cdots, b_1$ in σ. Because we may reverse σ and

still have a $\mathrm{QN}(K_{m,n})$-queue layout, we may assume that $b_{\lceil m/2 \rceil}$ appears after $a_{\lceil m/2 \rceil}$ in σ. Then the set of edges

$$\{(a_i, b_i) \mid 1 \leq i \leq \lceil m/2 \rceil\}$$

nest. By Proposition 2.1, $\mathrm{QN}(K_{m,n}) \geq \lceil m/2 \rceil$. □

This straightforward determination of $\mathrm{QN}(K_{m,n})$ contrasts with the current status of $\mathrm{SN}(K_{m,n})$ as reported in [20]. Even after much effort, the exact stacknumber of $K_{m,n}$, or even of $K_{n,n}$, has not been determined, though Muder, Weaver, and West [20] have obtained nontrivial bounds.

4.7. FFT and Beneš networks. We now consider two related families of graphs that have importance as computational networks. The *FFT network* represents the data dependencies of the Fast Fourier Transform algorithm. The *Beneš rearrangeable permutation network* is a switching network capable of realizing at its n outputs any permutation of its n inputs (Beneš [1]).

The n-input Beneš network $B(n)$, $n = 2^m$, is defined inductively as follows.

1. $B(2)$ is the complete bipartite graph $K_{2,2}$ on the two input vertices $I[1,1]$ and $I[1,2]$ and the two output vertices $O[1,1]$ and $O[1,2]$.

2. $B(n)$ is obtained from two copies of $B(n/2)$, together with n new input vertices $I[m,1], I[m,2], \cdots, I[m,n]$ and n new output vertices $O[m,1], O[m,2], \cdots, O[m,n]$. In the second copy of $B(n/2)$, each vertex $I[k,i]$ is relabeled $I[k, i+n/2]$, and each vertex $O[k,i]$ is relabeled $O[k, i+n/2]$; all vertices then have distinct labels. For $1 \leq i \leq n$, add edges to create a copy of $K_{2,2}$ on vertices $I[m,i]$ and $I[m, i+n/2]$ and vertices $I[m-1,i]$ and $I[m-1, i+n/2]$; also, add edges to create a copy of $K_{2,2}$ on vertices $O[m,i]$ and $O[m, i+n/2]$ and vertices $O[m-1,i]$ and $O[m-1, i+n/2]$.

As shown in Fig. 4.3, the Beneš network has a natural level structure with $2m$ levels. The n-input FFT network is the graph consisting of the first $m+1$ levels of $B(n)$.

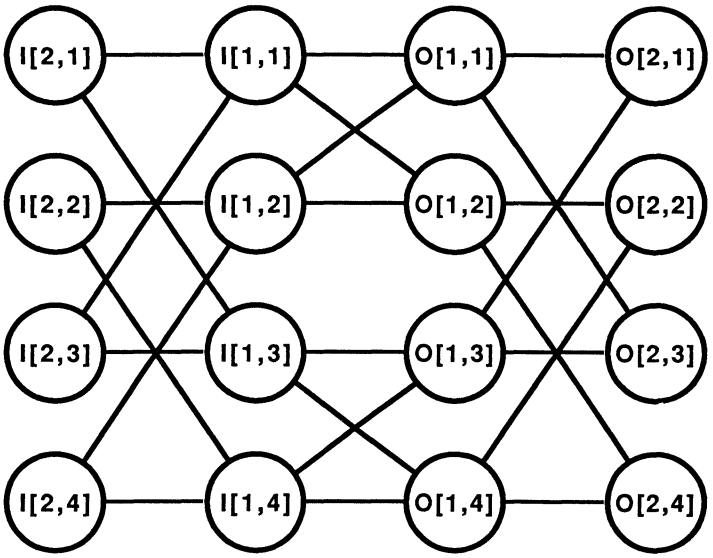

FIG. 4.3. *The Beneš network* $B(4)$.

As $B(n)$, $n > 2$, is not planar, its queuenumber is at least 2. The level structure (of either network) provides a straightforward 3-queue layout: order the vertices level by

level, going up each level; one queue for the "cross" edges, one queue for the "upward" edges, and one queue for the "downward" edges suffices. A more complicated 2-queue layout of $B(n)$ is due to Reibman [23].

PROPOSITION 4.9 (see [23]). *The Beneš network $B(n)$ admits a 2-queue layout with each queue of width n. The layout is optimal in queuenumber and within a factor of 2 of optimal in queuewidth.*

Proof. The layout of $B(n)$ follows its inductive definition. The inductive hypothesis is that $B(n)$ has a 2-queue layout which respects the leveling of $B(n)$; that is, all level i vertices appear before any level $i+1$ vertices, though no restriction is placed on the relative order of vertices within each level.

1. The vertex order for $B(2)$ is $I[1,1], I[1,2], O[1,1], O[1,2]$. The two edges incident to $I[1,1]$ are assigned to one queue and the two edges incident to $I[1,2]$ are assigned to the second queue. The layout satisfies the inductive hypothesis.

2. We assume that $B(n/2)$ has a 2-queue layout satisfying the inductive hypothesis. Let B_1 and B_2 be two copies of $B(n/2)$. Lay each out in the 2-queue order that is guaranteed by the induction. Merge the two layouts level by level so that the level-i vertices of B_2 always appear immediately to the right of the level-i vertices of B_1. In particular, $I[k, i+n/2]$ (respectively, $O[k, i+n/2]$) is always $n/2$ vertices to the right of $I[k,i]$ (respectively, of $O[k,i]$). Because the leveling of $B(n)$ is honored in the layout, each level-i edge of B_1 crosses every level-i edge of B_2, and vice versa, so no nesting results from the merging; hence, a 2-queue layout of the "sum" of B_1 and B_2 results. Add n new input vertices to the left and n new output vertices to the right of the entire layout. View the n new inputs as consisting of $n/2$ consecutive pairs of vertices. Add edges from the first pair to the first vertices of B_1 and B_2 to form a copy of $K_{2,2}$. In general, add edges from the ith pair to the ith vertices of B_1 and B_2. Assign the added edges incident to B_1 (which form a twist) to the first queue and the added edges incident to B_2 (which also form a twist) to the second queue. Similarly, connect the n new outputs to the last vertices of B_1 and B_2. The result is a 2-queue layout of $B(n)$. □

Because the FFT network is a subgraph of the Beneš network, it also has a 2-queue layout. This compares favorably with the stacknumber optimal 3-stack layouts of the Beneš and FFT networks in Games [8]. The natural leveling of these networks is a definite advantage in constructing queue layouts that are good, at least in the sense of queuenumber.

4.8. Hypercube. The *d-dimensional hypercube* $Q(d)$ has vertex set $\{0,1\}^d$, the set of all bit strings of length d; its edges connect every pair of vertices that differ in exactly one bit position. View the vertex set of $Q(d)$ as the set of integers $\{0, 1, \cdots, 2^d - 1\}$ by identifying a d-bit string with the corresponding integer in binary notation. The hypercube admits a very regular layout strategy.

PROPOSITION 4.10. *For $d \geq 2$, $Q(d)$ admits a $(d-1)$-queue layout with queuewidths*

$$2^{d-1}, 2^{d-2}, \cdots, 2^2, 2^1.$$

Proof. We lay out $Q(d)$ inductively. The order $\sigma = 0, 1, 2, 3$ gives a 1-queue layout of $Q(2)$ with queuewidth 2. To obtain a layout for $Q(d)$, $d > 2$, inductively lay out two adjacent copies of $Q(d-1)$, similarly ordered. By induction, each of the copies of $Q(d-1)$ uses $d-2$ queues with queuewidths

$$2^{d-2}, 2^{d-3}, \cdots, 2^2, 2^1;$$

hence, their disjoint sum does also. The 2^{d-1} edges connecting one copy of $Q(d-1)$ to the other form a 2^{d-1}-twist; hence they require only one additional queue of width 2^{d-1}. □

The queuenumber of the preceding layout is optimal to within a constant factor.

PROPOSITION 4.11. $\mathrm{QN}(Q(d)) = \Omega(d)$.

Proof. $Q(d)$ has $d2^{d-1}$ edges. By Corollary 3.7, therefore,

$$\mathrm{QN}(Q(d)) \geq \left\lceil \frac{d2^{d-1}}{2^{d+1}-3} \right\rceil = \Omega(d). \qquad \square$$

5. Queuenumber and graph structure. We now explore two structural properties of graphs that provide bounds on queuenumber. These properties are bandwidth and separator size.

5.1. Bandwidth. Let $\sigma = 1, 2, \cdots, n$ be any order of the vertices of G. The *bandwidth* of σ is the length of the longest edge; that is,

$$\mathrm{BW}(\sigma) = \max_{(i,j)\in E} |i-j|.$$

The *bandwidth* of G is the minimum bandwidth of any σ; that is,

$$\mathrm{BW}(G) = \min_{\sigma} \mathrm{BW}(\sigma).$$

Assume that $n \geq B+1$. The *maximal bandwidth-B graph on n vertices* $M(B,n)$ has vertex-set $\{1, 2, \cdots, n\}$; its edges form a copy of the complete graph K_{B+1} on each subset of vertices

$$\{i, i+1, \cdots, i+B\}, \quad 1 \leq i \leq n-B.$$

See Fig. 5.1.

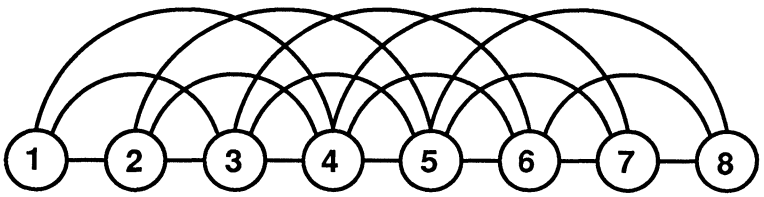

FIG. 5.1. *Maximal bandwidth-B graph* $M(3,8)$.

The following theorem establishes a relationship between bandwidth and queuenumber.

THEOREM 5.1. $\mathrm{QN}(M(B,n)) = \lceil B/2 \rceil$. *Hence, if* $\mathrm{BW}(G) = B$, *then* $\mathrm{QN}(G) \leq \lceil B/2 \rceil$.

Proof.

Upper bound. Choose the order $\sigma = 1, 2, \cdots, n$ for the vertices of $M(B,n)$. There are edges of every length from 1 to B. Assign all edges of lengths $2i-1$, $2i$, $1 \leq i \leq \lceil B/2 \rceil$, to queue q_i. No two edges having the same length or having lengths differing by 1 can nest. A $\lceil B/2 \rceil$-queue layout of $M(B,n)$ results.

Lower bound. $M(B,n)$ contains complete graphs on $B+1$ vertices. By Proposition 4.7,

$$\mathrm{QN}(M(B,n)) \geq \left\lfloor \frac{B+1}{2} \right\rfloor = \left\lceil \frac{B}{2} \right\rceil.$$

Since every bandwidth-B graph G is a subgraph of some maximal bandwidth-B graph $M(B,n)$, $\mathrm{QN}(G) \leq \lceil B/2 \rceil$. □

5.2. Separator size. Let $S(x)$ be a nondecreasing integer function. A graph $G = (V,E)$ has a $(\frac{1}{3},\frac{2}{3})$-*vertex-separator of size* $S(x)$ (or just *separator of size* $S(x)$) if either $|V| < 3$ or if there is a subset of cardinality at most $S(|V|)$ whose removal leaves connected components of cardinality less than or equal to $\frac{2}{3}|V|$, each having a separator of size $S(x)$.

Suppose G has maximum degree d and has a separator of size $S(x)$. Let R be the following function of S:

$$R(x) = (d+1)\sum_i S(x/2^i).$$

A *bucket tree for* G is a complete binary tree whose level-j *buckets* (vertices) have *bucket capacity*

$$C(j) = cdR(|V|/2^j),$$

where c is a constant. Bhatt et al. [3] demonstrate the following.

LEMMA 5.2 (see [3]). *If $G = (V,E)$ is a graph of maximum degree d that has a separator of size $S(x)$, then V can be mapped onto the bucket tree for G in such a way that*

1. *At most $C(j)$ vertices are mapped to each level-j vertex of the bucket tree;*
2. *If two vertices that are adjacent in G are mapped to two distinct buckets, then these two buckets are at most distance d apart in the bucket tree, and one of the buckets is an ancestor of the other.*

The following relates the separator size of a graph to its queuenumber and stacknumber.

THEOREM 5.3. *If $G = (V,E)$ is a graph of maximum degree d that has a separator of size $S(x)$, then G has queuenumber and stacknumber $O(d^2R(|V|))$.*

Proof.

Queuenumber. Construct a queue layout of G in two steps. First, use Lemma 5.2 to map V onto the bucket tree for G, and lay out the bucket tree in a breadth-first order. Second, use this 1-queue layout of the bucket tree to obtain a queue layout of G; replace each bucket B by the contents of B placed contiguously in any order. We analyze the number of queues needed in this layout of G. The two endpoints of any edge of G are mapped to a pair of buckets B_1 and B_2 such that B_1 is an ancestor of B_2, and B_1 and B_2 are at most distance d apart in the bucket tree; call an edge an i-*edge*, $0 \leq i \leq d$, if i is the distance between B_1 and B_2 in the bucket tree. If the endpoints of two i-edges are mapped to different pairs of buckets, then the two edges cannot nest. Fix i, $0 \leq i \leq d$. Since each bucket contains $O(dR(|V|))$ vertices, $O(dR(|V|))$ queues suffice for all i-edges. Since i has $d+1$ possible values, the queue layout for G has queuenumber $O(d^2R(|V|))$.

Stacknumber. A similar construction that begins by laying out the bucket tree in preorder suffices to show the bound on stacknumber. □

We remark that Theorem 4.5 of Chung, Leighton, and Rosenberg [5] gives an upper bound on the stacknumber of a graph as a function of its bifurcator, rather than separator, size.

6. A queuenumber/queuewidth tradeoff. In this section, we provide evidence of an apparent tradeoff between queuenumber and queuewidth for queue layouts of complete

binary trees. Then we relate the queuewidth of a graph G to the *diameter* of G, i.e., the greatest distance between any pair of vertices of G.

By Proposition 4.1, a depth-d complete binary tree has a 1-queue layout with queuewidth 2^d. This queuewidth is exponentially greater than the $O(d)$ stackwidth of a 1-stack layout of a complete binary tree [5]. We consider the question of whether a larger number of queues can be traded off for a smaller (cumulative) queuewidth. The following theorem suggests an apparent tradeoff.

THEOREM 6.1. *A depth-d complete binary tree T with $n = 2^d$ leaves has a k-queue layout QL with* $\mathrm{CQW}(QL) = O(kn^{1/k})$.

Proof. We give the proof for $k = 2$. To simplify the construction, we assume that $d = 2d'$ is even. Let $n' = 2^{d'} = \sqrt{n}$. Let $T^\star$ be the upper $d' + 1$ levels of T, and let $1, 2, \cdots, n'$ be the leaves of $T^\star$ in canonical order. Each leaf i is the root of a subtree T_i of depth d'. Order the vertices of $T^\star$ in breadth-first order from the root, so that vertices $1, \cdots, n'$ appear rightmost in the order. For each $i, 1 \leq i \leq n'$, place the vertices of T_i in breadth-first order immediately to the right of its root i. Assign the edges of $T^\star$ to one queue and the edges of $T_1, T_2, \cdots, T_{n'}$ to a second queue. Each queue has queuewidth $n' = \sqrt{n}$. The cumulative queuewidth of the 2-queue layout is $2n' = O(2n^{1/2})$.

For general k, $2 \leq k \leq d$, cut T every $\lceil d/k \rceil$ levels, and use one queue for the edges in each of the produced "meta-levels." Details are left to the reader. □

To show that the apparent tradeoff is a real one, we need lower bound techniques for queuewidth. The next theorem provides a lower bound on queuewidth as a function of diameter for arbitrary 1-queue graphs.

THEOREM 6.2. *Suppose $G = (V, E)$ is a connected 1-queue graph having diameter D. Let QL be a 1-queue layout of G. Then,*

$$\mathrm{QW}(QL) \geq \frac{|E|}{2D + 1}.$$

Proof. By Theorem 3.2, layout QL yields an arched leveled-planar embedding of G; denote the induced levels by $V_1, V_2, \cdots, V_m$. Since each edge of G connects vertices that are either in the same or adjacent levels, it is immediate that $D \geq m - 1$. Consider now the following $2m - 1 \leq 2D + 1$ cuts of layout QL:[1]

$$\mathrm{CUT}(t_1 - 1), \mathrm{CUT}(t_1), \cdots, \mathrm{CUT}(t_i - 1), \mathrm{CUT}(t_i), \cdots, \mathrm{CUT}(t_m - 1).$$

Since every edge of G either has some t_i as its right endpoint or passes over some t_i, the enumerated set of cuts collectively exhausts E. The proof is completed by appealing to two facts:

- The queuewidth of the layout QL is (clearly) as big as the biggest cut.
- The biggest cut contains no fewer edges than the average cut. □

For a depth-d complete binary tree T, there are $n = 2^d$ leaves, $|V| = 2n - 1$, $|E| = 2n - 2$ and $D = 2d = 2 \log n$. We have this corollary.

COROLLARY 6.3. *Any 1-queue layout of a depth-d complete binary tree has queuewidth at least* $(2n - 2)/(4d + 1) = \Omega(n/\log n)$.

The breadth-first layout of T starting at the root has queuewidth n. The lower bound on queuewidth in the corollary is close to this upper bound. We do not know how to achieve this lower bound and doing so appears difficult. The higher width of queue

[1] $\mathrm{CUT}(i)$, the *cut* at vertex i, is the set of all edges whose left endpoint is less than or equal to i and whose right endpoint is greater than i.

layouts over stack layouts suggests that stack layouts of trees are preferable to queue layouts.

We do not yet have a lower bound on cumulative queuewidth for arbitrary k-queue layouts of T, along the lines of Theorem 6.2 for the case $k = 1$. Thus, we do not know whether there is a real tradeoff between queuenumber and queuewidth here, but we conjecture that there is. We further conjecture that the cumulative queuewidth announced in Theorem 6.1 is within a factor of $O(d)$ of optimal.

7. Future directions. In Table 7.1, we summarize our queuenumber results for specific graphs alongside the corresponding stacknumber results. Further comparison of the relative merits of queues and stacks is warranted. In particular, queues appear to be more appropriate than stacks for graphs with a leveled structure; can this insight be formalized? Chung, Leighton, and Rosenberg [5], and Heath [13] show that there are tradeoffs between stacknumber and stackwidth in the sense that, for certain graphs, devoting more stacks to a layout decreases the cumulative stackwidth. We expect analogous tradeoffs between queuenumber and queuewidth. We conjecture the following.

CONJECTURE 1. There are graphs that exhibit a tradeoff between queuenumber and queuewidth, and Theorem 6.1 exposes such a tradeoff.

TABLE 7.1
Queuenumbers of specific graphs.

Graph Class	Queuenumber	Stacknumber
Trees	1	1 [5]
X-trees	2	2 [5]
DeBruijn Graph	2	≤ 5 [21]
Complete Graph K_n	$\lfloor n/2 \rfloor$	$\lfloor n/2 \rfloor$ [5]
Complete Bipartite Graph $K_{m,n}$	$\min(\lceil m/2 \rceil, \lceil n/2 \rceil)$ (Exact)	$\leq \lceil (m+2n)/4 \rceil$ [20]
FFT Network	2	3 [8]
Beneš Network	2	3 [8]
Boolean n-cube	$\leq n-1$	$\leq n-1$ [5]
Ternary n-cube	$\leq 2n-2$ [16]	$\Omega(3^{\alpha n})$, $\alpha < 1/9$ [16]
Planar Graphs	Unknown (Conjecture bounded)	4 [27]

Often, good queue layouts seem easier to obtain than good stack layouts. Planar graphs may be an exception to this. However, in harmony with the fact that planar graphs can be laid out in a bounded number of stacks (Yannakakis [27]), we conjecture the following.

CONJECTURE 2. Planar graphs can be laid out in a bounded number of queues.

The notions of stack and queue layouts may be generalized in several directions. One approach is to define layouts that simultaneously utilize queues and stacks. We conjecture the following.

CONJECTURE 3. Each planar graph admits a 1-stack, 1-queue layout.

Another approach is to utilize deques or more general permutation mechanisms. In the realm of such generality, it becomes necessary to consider relative cost measures for the various mechanisms.

Note added in proof. The interested reader should be aware of the Ph.D. thesis of Sriram V. Pemmaraju (*Exploring the Powers of Stacks and Queues via Graph Layouts*, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1992). It contains further results in the theory of queue and stack layouts. The thesis develops a number of new tools for studying such layouts. It also initiates the study (promised in §1.2) of queue and stack layouts of *dags*. Copies are available from the first author.

Acknowledgments. Part of this research was conducted while the first author was at the University of North Carolina at Chapel Hill and at the Massachusetts Institute of Technology. A portion of the work of the second author was done during a visit to the Department of Applied Mathematics and Informatics of the University of the Saarlands.

We wish to thank Reuven Bar-Yehuda, Sandeep Bhatt, Fan Chung, Sergio Fogel, Tom Leighton, Bojana Obrenić, Sriram Pemmaraju, and Andrew Reibman for helpful conversations. We especially thank Bojana for a careful reading of the completed paper, for many useful suggestions, and for pointing out the construction in the proof of Theorem 5.3.

REFERENCES

[1] V. E. Beneš, *Optimal rearrangeable-multistage connecting networks*, Bell System Technical Journal, 43 (1964), pp. 1641–1656.

[2] F. Bernhart and B. Kainen, *The book thickness of a graph*, J. Combin. Theory B, 27 (1979), pp. 320–331.

[3] S. N. Bhatt, F. R. K. Chung, J.-W. Hong, F. T. Leighton, B. Obrenić, A. L. Rosenberg, and E. J. Schwabe, *Optimal emulations by butterfly-like networks*, Tech. Rpt. 90-108, Department of Computer Science, University of Massachusetts, Amherst, MA, 1990; J. Assoc. Comput. Mach., to appear.

[4] J. Buss and P. Shor, *On the pagenumber of planar graphs*, in Proceedings of the 16th Annual ACM Symposium on Theory of Computing, Washington, DC, 1984, pp. 98–100.

[5] F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg, *Embedding graphs in books: A layout problem with applications to VLSI design*, SIAM J. Algebraic Discrete Meth., 8 (1987), pp. 33–58.

[6] P. Erdös and E. Szekeres, *A combinatorial problem in geometry*, Compositio Math., 2 (1935), pp. 463–470.

[7] S. Even and A. Itai, *Queues, stacks and graphs*, in Theory of Machines and Computations, Z. Kohavi and A. Paz, eds., Academic Press, New York, 1971, pp. 71–86.

[8] R. Games, *Optimal book embeddings of the FFT, Benes, and barrel shifter networks*, Algorithmica, 1 (1986), pp. 233–250.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, New York, 1979.

[10] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, *The complexity of coloring circular arcs and chords*, SIAM J. Algebraic Discrete Meth., 1 (1980), pp. 216–227.

[11] L. S. Heath, *Embedding planar graphs in seven pages*, in Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, Singer Island, FL, 1984, pp. 74–83.

[12] ———, *Algorithms for Embedding Graphs in Books*, Ph.D. thesis, Department of Computer Science, University of North Carolina, Chapel Hill, NC, 1985.

[13] ———, *Embedding outerplanar graphs in small books*, SIAM J. Algebraic Discrete Meth., 8 (1987), pp. 198–218.

[14] L. S. Heath and S. Istrail, *The pagenumber of genus g graphs is $O(g)$*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 388–397.

[15] ———, *The pagenumber of genus g graphs is $O(g)$*, J. Assoc. Comput. Mach., 1992, to appear.

[16] L. S. Heath, F. T. Leighton, and A. L. Rosenberg, *Comparing queues and stacks as mechanisms for laying out graphs*, SIAM J. Discrete Math., 5 (1992), to appear.

[17] W.-L. HSU, *Maximum weight clique algorithms for circular-arc graphs and circle graphs*, SIAM J. Comput., 14 (1985), pp. 224–231.

[18] D. B. JOHNSON, *A priority queue in which initialization and queue operations take $O(\log \log D)$ time*, Math. Systems Theory, 15 (1982), pp. 295–309.

[19] D. LICHTENSTEIN, *Planar formulae and their uses*, SIAM J. Comput., 11 (1982), pp. 329–343.

[20] D. J. MUDER, M. L. WEAVER, AND D. B. WEST, *Pagenumber of complete bipartite graphs*, J. Graph Theory, 12 (1988), pp. 469–489.

[21] B. OBRENIĆ, *Embedding de Bruijn and shuffle-exchange graphs in five pages*, in Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, Hilton Head, SC, 1991, pp. 137–146.

[22] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Towards an architecture-independent analysis of parallel algorithms*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 510–513.

[23] A. REIBMAN, *DIOGENES layouts using queues*, typescript, Department of Computer Science, Duke University, Durham, NC, 1984.

[24] A. L. ROSENBERG, *The DIOGENES approach to testable fault-tolerant arrays of processors*, IEEE Trans. Comput., C-32 (1983), pp. 902–910.

[25] M. M. SYSLO AND M. IRI, *Efficient outerplanarity testing*, Fund. Inform., 2 (1979), pp. 261–275.

[26] R. E. TARJAN, *Sorting using networks of queues and stacks*, J. Assoc. Comput. Mach., 19 (1972), pp. 341–346.

[27] M. YANNAKAKIS, *Embedding planar graphs in four pages*, J. Comput. System Sci., 38 (1989), pp. 36–67.