CHAPTER 11

Not So Perfect Graphs

1. Sorting a Permutation Using Stacks in Parallel

Let π be a permutation of the numbers $\{1, 2, \ldots, n\}$, which we will regard as the sequence $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$. We would like to sort π into natural order using a system of stacks arranged in parallel, illustrated in Figure 11.1. Initially, the permutation sits on the input queue. Two types of moves are allowed: (i) moving the number at the head of the input queue onto the top of one of the stacks or (ii) moving a number from the top of a stack to the tail of the output queue. A successful sorting is accomplished by transferring all numbers to the output queue in the order $[1, 2, \ldots, n]$ by repeatedly applying (i) and/or (ii).

Given a sufficient number of stacks, any permutation can be sorted in this manner. But when can a permutation π be sorted using a system of only m stacks in parallel? For example, the sequence $\pi = [3, 5, 4, 1, 6, 2]$ requires three stacks, since the numbers 3, 5, and 6 must be stored on different stacks until 2 has reached the output queue. The observation that 3, 5, and 6 occur in their natural order but are followed by the smaller number 2 is the key to converting this sorting problem into a graph coloring problem.

Let $H[\pi]$ be the undirected graph having vertices $\{1, 2, \ldots, n\}$ with j and k adjacent if there exists an i such that

$$i < j < k \qquad \text{and} \qquad \pi_j^{-1} < \pi_k^{-1} < \pi_i^{-1}.$$

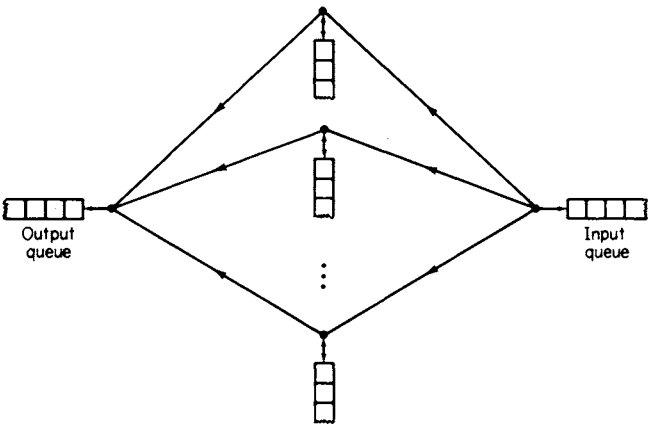


Figure 11.1. A system of stacks in parallel.

We can read $\pi_i^{-1} = (\pi^{-1})_i$ as "the position of i in π." An example of this construction is given in Figure 11.2. We call $H[\pi]$ the *stack sorting graph* of π. It is a straightforward exercise to show the following:

π can be sorted in a system of m stacks in parallel if and only if the chromatic number of $H[\pi]$ is at most m.

One possible application of this sorting technique is in rearranging the railroad cars of a train in a switching yard (see Knuth [1969, Section 2.2.1; 1973, pp. 169–170], Even and Itai [1971], and Tarjan [1972]).

Let $\mathscr{H}$ be the collection of all graphs G such that G is isomorphic to $H[\pi]$ for some permutation π. Very little is known about the class $\mathscr{H}$. There have been no efficient recognition or coloring algorithms produced for the graphs in $\mathscr{H}$ which are in general *not perfect graphs*. Neither is there a good graph theoretic characterization. Our reason for introducing the class $\mathscr{H}$ is to show an equivalence between $\mathscr{H}$ and another class of graphs which has frustrated mathematicians for some years, namely, the circle graphs.

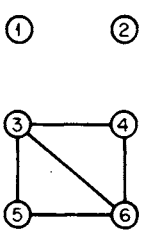


Figure 11.2. The graph $H[3, 5, 4, 1, 6, 2]$.

2. Intersecting Chords of a Circle

An undirected graph G is called a *circle graph* if it is isomorphic to the intersection graph of a finite collection of chords of a circle (see Figure 11.3). Without loss of generality, we may assume that no two chords share a common endpoint.

Theorem 11.1 (Even and Itai [1971]). An undirected graph G is a graph of intersecting chords of a circle if and only

$$G - \{\text{all isolated vertices}\} \cong H[\pi] - \{\text{all isolated vertices}\}$$

for some permutation π.

This theorem will be proved constructively by demonstrating two techniques, Algorithms 11.1 and 11.2, whose correctness will be shown in Propositions 11.2 and 11.3, respectively. The algorithms transform one representation into the other.

Remark 1. The subtraction of isolated vertices in the theorem is required. Two intersecting chords would give the complete graph on two vertices, whereas any graph $H[\pi]$ which has an edge must have at least three vertices.

Remark 2. From the point of view of coloring, covering by cliques, and finding a maximum stable set or maximum clique, isolated vertices neither add to nor subtract from the essential complexity of the problem.

Remark 3. A circle with intersecting chords enables us to generalize the notion of a matching diagram which we encountered in Section 7.4. Furthermore, sorting a permutation using stacks in parallel is very much like the problem of sorting a permutation using parallel queues discussed in Section

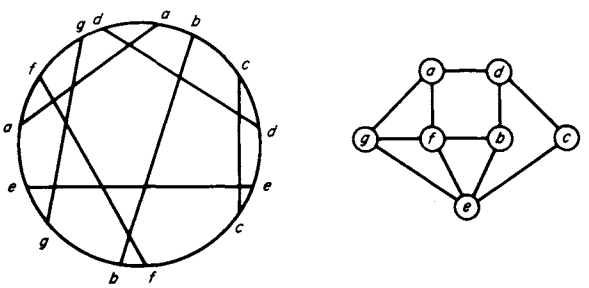


Figure 11.3. A set of chords and its intersection graph.

7.5. The remarkable feature of Theorem 11.1 is that the equivalence established in Chapter 7 between permutation graphs and sorting in parallel queues extends to an equivalence between circle graphs and sorting in parallel stacks.

Algorithm 11.1. Constructing a circle with chords from a permutation.

Input: A permutation π of the numbers $1, 2, \ldots, n$.
Output: A circle $\mathscr{C}$ with n chords.
Method: The algorithm is as follows:

1. DRAW A CIRCLE. Label nodes $\pi_1, \pi_2, \ldots, \pi_n$ in a clockwise manner.
2. We go once around the circle clockwise starting just prior to π_1.
3. **for** $i \leftarrow 1$ **to** n **do**
4. **if** you have not passed by node i
5. **then** SKIP clockwise to i;
6. Draw another node i immediately clockwise (but before the next node);
7. **next** i;
8. DRAW chords matching the pairs of numbers.

Example 11.1. We apply Algorithm 11.1 to the permutation $\pi = [2, 9, 4, 6, 7, 1, 3, 8, 5]$. The instructions executed by the algorithm are given in Figure 11.4 along with the stack sorting graph $H[\pi]$ and the initial and final configurations for the circle $\mathscr{C}$ of chords.

Proposition 11.2. Given a permutation π, Algorithm 11.1 constructs a set of chords of a circle whose intersection graph is isomorphic to $H[\pi]$.

Proof. Suppose j and k are adjacent in $H[\pi]$ and assume $j < k$. Then there is an i such that $i < j < k$ and $\pi_j^{-1} < \pi_k^{-1} < \pi_i^{-1}$, which implies that after the ith iteration of Algorithm 11.1 j and k have already been passed

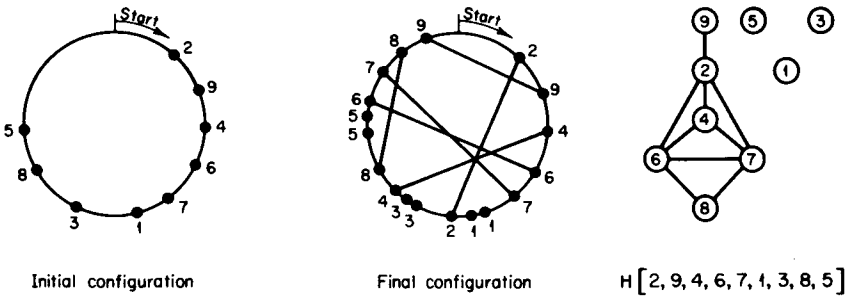


Figure 11.4. Algorithm 11.1 applied to the permutation $\pi = [2, 9, 4, 6, 7, 1, 3, 8, 5]$. The instructions executed are as follows.
Skip to 1; draw a 1. Draw a 2. Skip to 3; draw a 3. Draw a 4. Skip to 5; draw a 5. Draw 6–9.

over. Hence we shall write the new j before the new k, and their chords will therefore intersect.

Conversely, suppose two chords intersect and, reading clockwise around the circle from the starting point, their endpoints are labeled j, k, j, k.* Thus $j < k$ and $\pi_j^{-1} < \pi_k^{-1}$. Since in the jth iteration we had already passed the first k, there must be an i, $i < j$, such that during the ith iteration we skipped over k to the first occurrence of i. Thus, $\pi_k^{-1} < \pi_i^{-1}$, so j and k are adjacent in $H[\pi]$. ∎

Algorithm 11.2. Constructing a permutation from a circle with chords.

Input: A circle $\mathscr{C}$ with chords.
Output: A permutation π of the numbers $1, 2, \ldots, n$.
Method: The algorithm is as follows:

```
1.  Pick a number n (a lucky choice will eliminate renumbering later);
2.  Initialize: i ← n; pick a starting point (not an endpoint of a chord);
3.  for once around the circle going counterclockwise do
      begin
4.      if next endpoint p is unlabeled
          then
5.          label it i; label its opposite endpoint i';
6.          decrement: i ← i − 1;
          else
7.          create a dummy endpoint on the circle just preceding p;
8.          label the dummy i';
9.          decrement: i ← i − 1;
10.         skip to just prior to the next unlabeled endpoint;
      end
11. renumber everything so that the smallest label is 1 (by subtracting the final value of i from
    each);
12. print the sequence of primed numbers running clockwise from the starting point and call
    them π1, π2, ..., respectively;
```

Example 11.2. Applying Algorithm 11.2 to the circle $\mathscr{C}$ in Figure 11.5, we obtain the permutation $\pi = [7, 4, 2, 10, 6, 1, 8, 3, 9, 5]$. The instructions executed by the algorithm and the final (labeled) configuration for $\mathscr{C}$ are also given.

Proposition 11.3. Given a set of chords of a circle $\mathscr{C}$, Algorithm 11.2 finds a permutation π such that the intersection graph of $\mathscr{C}$ is isomorphic to $H[\pi]$ − {some isolated vertices}.

* The second occurrences of j and k were created from the index of the loop, which is increasing.

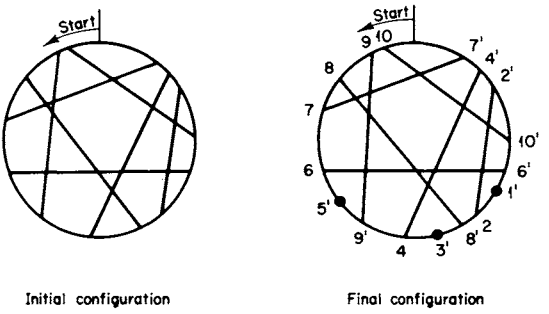


Figure 11.5. Algorithm 11.2 applied to the circle $\mathscr{C}$. The instructions executed are the following.

We cheat and pick $n = 10$. Label chords 10, 9, 8, 7, and 6. Create dummy 5′ and skip over 9′. Label chord 4. Create dummy 3′ and skip over 8′. Label chord 2. Create dummy 1′ and skip over 6′, 10′, 2′, 4′, and 7′. The permutation is $\pi = [7, 4, 2, 10, 6, 1, 8, 3, 9, 5]$.

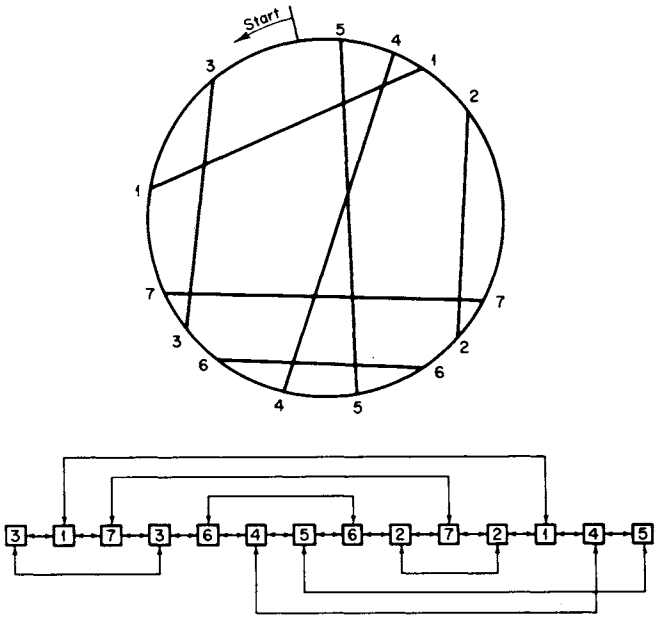
Proof. Suppose j and k are adjacent in $H[\pi]$ and assume $j < k$. Then there is an i such that $i < j < k$ and their primed versions appear in the clockwise order j', k', i'. This implies that j' and k' are not dummy endpoints.* Since unprimed numbers occur in decreasing order going counterclockwise, it follows that the jth and kth chords intersect.

Conversely, if the jth and kth chords intersect with $j < k$, then k' was skipped over after labeling some dummy endpoint i', where $i < j$. So $\pi_j^{-1} < \pi_k^{-1} < \pi_i^{-1}$ and j and k are adjacent in $H[\pi]$. ∎

For small examples these algorithms are easy to do by hand. We would like to suggest a data structure suitable for performing the algorithms on a computer. A circle $\mathscr{C}$ with chords may be represented by either a list or an array consisting of the endpoints of the chords given in the counterclockwise order, beginning with a fixed but arbitrary starting point. There will be pointers providing direct access from one endpoint of a chord to the opposite endpoint. An example of this data structure is given in Figure 11.6. Notice that ARRAY(i) = ARRAY(OPPOSITE(i)) for all i in the example.

To implement Algorithm 11.2 we scan the data structure corresponding to $\mathscr{C}$ once from left to right, labeling endpoints appropriately The property of an endpoint being primed can be coded into the label. On the other hand, Algorithm 11.1 would receive its input π as the reversed list $[\pi_n, \ldots, \pi_2, \pi_1]$ into which the new nodes are inserted. As the list is scanned from right to left, we keep track of which numbers have been passed by using an auxiliary bit vector. Both of these implementations can be carried out in time and space proportional to the size of the input.

* Because any dummy endpoint following i' counterclockwise would have smaller value.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Array :	3	1	7	3	6	4	5	6	2	7	2	1	4	5
Opposite :	4	12	10	1	8	13	14	5	11	3	9	2	6	7

Figure 11.6. Data structures for a circle with chords.

Mark Buckingham has suggested the following algorithm to construct the adjacency sets of the intersection graph obtained from a circle with chords.

Algorithm 11.3.

Input: The data structure DS, as described above, representing a collection of n chords of a circle.

Output: The adjacency sets of the intersection graph.

Method: The algorithm is given in Figure 11.7. We traverse DS (i.e., the circle counterclockwise) exactly once. Chords j and k intersect if and only if their endpoints occur in the order $[k, j, k, j]$ or $[j, k, j, k]$. Each chord k is added to the end of a list called LIST when its first endpoint is encountered (line 4). It remains there until the second endpoint is reached at which time all chords j on the LIST following k are discovered to intersect chord k (lines 5–7). Then k is removed from the LIST (line 8). An array POINTER(k),

```
    begin
1.    initialize: LIST ← ∅; for i ← 1 to n do Adj(i) ← ∅;;
2.    for each entry k of DS do in order
3.      if k is not on LIST
              then
4.              append k to the tail LIST;
              else
5.              for each j to the right of k on LIST do
6.                add j to Adj(k);
7.                add k to Adj (j);
                for next j;
8.              delete k from LIST;
      next k;
    end
```

Figure 11.7. Algorithm 11.3.

initially undefined, may be used to execute efficiently the test in line 3 and the access to the starting point in line 5. A proof of correctness is left as Exercise 8.

Remark. Touchard [1952], Riordan [1975], and Read [1979] investigate a generating function for the number of ways of drawing n chords of a circle so as to obtain k intersections.

3. Overlap Graphs

The circle graphs are equivalent to yet another popular class of graphs, namely, the *overlap graphs*. Given a collection of intervals on a line, each pair of intervals will satisfy exactly one of the following properties.

Overlap. The two intervals intersect but neither properly contains the other.

Containment. One of the two intervals properly contains the other.

Disjointness. The two intervals have empty intersection.

A graph G is called an *overlap graph* if its vertices may be put into one-to-one correspondence with a collection of intervals on a line such that two vertices are adjacent in G if and only if their corresponding intervals overlap (not just intersect). Without loss of generality we may assume that the intervals are either open or closed and that no two intervals have a common endpoint.

Let $\mathcal{I} = \{I_x\}_{x \in V}$ be a collection of intervals on a line, and assume that no two intervals have a common endpoint. The pairs of distinct indices are

partitioned into three mutually disjoint sets A, B, D as follows: For distinct $x, y \in V$,

$$xy \in A \quad \text{if} \quad \varnothing \neq I_x \cap I_y \neq I_x, I_y$$

(i.e., the intervals overlap);

$$xy \in B \quad \text{if} \quad \text{either } I_x \subset I_y \quad \text{or} \quad I_y \subset I_x$$

(i.e., one interval properly contains the other);

$$xy \in D \quad \text{if} \quad I_x \cap I_y = \varnothing$$

(i.e., the intervals are disjoint).

Clearly, A, B, and D are symmetric relations partitioning all pairs. Thus we have that (V, A) is the overlap graph represented by $\mathcal{I}$, $(V, A + B)$ is the interval graph represented by $\mathcal{I}$, and (V, D) is a comparability graph since its complement is an interval graph. Furthermore, defining

$$xy \in C \quad \text{if} \quad I_x \subset I_y$$

and

$$xy \in F \quad \text{if} \quad I_x \text{ lies entirely to the left of } I_y,$$

it follows that (V, C) and (V, F) are transitive orientations of (V, B) and (V, D), respectively. An example of these graphs is illustrated in Figure 11.8.

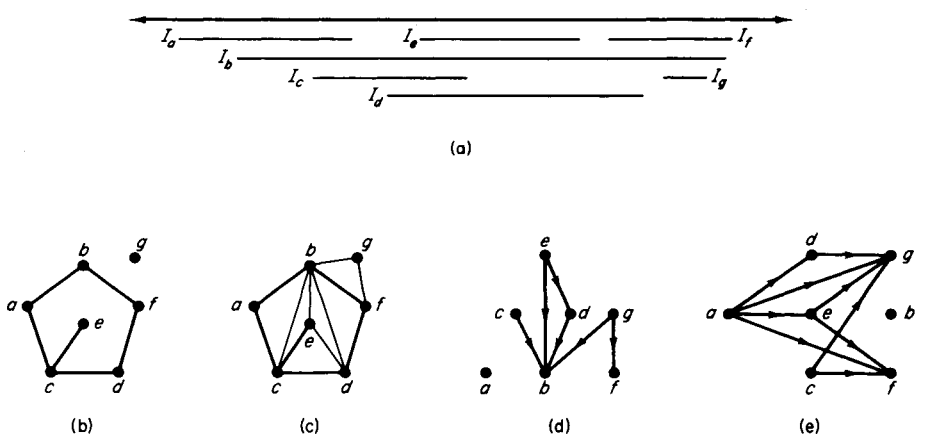


Figure 11.8. (a) A collection $\mathcal{I}$ of intervals. (b) The overlap graph (V, A) of $\mathcal{I}$. (c) The interval graph $(V, A + B)$ of $\mathcal{I}$. (d) The transitive orientation (V, C) representing proper containment. (e) The transitive orientation (V, F) representing disjointness.

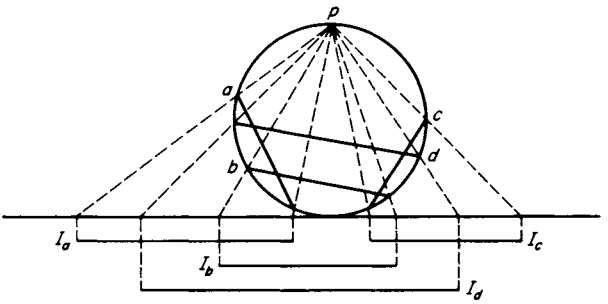


Figure 11.9. Intersecting chords of the circle correspond to overlapping intervals on the line. This projection method is suggested by Gavril [1973]. Relative to this choice of p, we have I_a and I_b overlapping, I_b and I_c overlapping, I_a and I_c disjoint, and I_d containing I_b.

We have the following easy result.

Proposition 11.4. An undirected graph G is a circle graph if and only if G is an overlap graph.

Proof. Given a circle with chords, choose a point p on the circle which is not an endpoint of a chord. To each chord with endpoints c and c' we associate the arc along the circle from c to c' which does not contain p. If the circle is cut at point p, then we will obtain a line with a collection of intervals having the desired property.

This process may be reversed by wrapping a collection of intervals of a line around a circle and then drawing a chord between the two endpoints of each interval. ∎

An alternate way to visualize this equivalence is by placing the circle tangent to a line, with p as the north pole, and the point of tangency as the south pole. Projecting the chords down to the line, as in Figure 11.9, we obtain the correspondence. Our data structure for representing a circle with chords is simply a discrete version of this linearization.

4. Fast Algorithms for Maximum Stable Set and Maximum Clique of These Not So Perfect Graphs

In the preceding sections we demonstrated the equivalence of the stack sorting graphs, the circle graphs, and the overlap graphs. We are therefore free to choose whichever model suits us best in order to prove properties about the class.

As we mentioned earlier, there are a number of open problems concerning this class of graphs.

(i) Find an algorithm which recognizes circle graphs and constructs a representation for the graph as intersecting chords of a circle.

(ii)* Are the coloring and clique-cover problems NP-complete for circle graphs?

(iii) Is the strong perfect graph conjecture true for circle graphs?

The stable set problem and the clique problem are tractable when restricted to our not so perfect graphs. In the context of open problem (i), it is essential that we be given, *a priori*, a representation of the graph as overlapping intervals, intersecting chords, or a permutation to be sorted. We choose to use the overlap graph model. We first present a solution to the stable set problem, due to Gavril [1973], which can be implemented to run in polynomial time.

Let $\mathcal{I} = \{I_x\}_{x \in V}$ be a collection of intervals, let $G = (V, A)$ be the overlap graph of $\mathcal{I}$, and let C and F be the oriented containment and disjointness relations as defined in the preceding section. For all $x \in V$, let

$$U(x) = \{v \in V \mid vx \in C\}$$

be the set of indices whose corresponding intervals are (properly) contained in I_x. The algorithm is as follows.

Algorithm 11.4. Maximum stable set of an overlap graph.

Input: The (transitively) oriented containment relation (V, C) and the (transitively) oriented disjointness relation (V, F) of a collection $\mathcal{I}$ of intervals whose overlap graph is $G = (V, A)$.
Output: A maximum stable set S of G.
Method: We assign, to each vertex $x \in V$, a weight $w(x)$ and a maximum stable set $S(x)$ of $G_{\{x\} \cup U(x)}$, where $w(x) = |S(x)|$. This is carried out recursively in such a way that vertices are assigned weights in a topologically sorted order with respect to C. At the heart of the algorithm is the subroutine (from Chapter 5) MAXWEIGHT CLIQUE, which finds a set of pairwise disjoint intervals $\{I_v\}_{v \in T}$ that generates, in line 6 of MAXSTABLE, the best possible stable set. The entire algorithm consists of the single call,

```
begin
  S ← MAXSTABLE(V);
end
```

and uses the recursive procedure in Figure 11.10. The assumption that F is transitive is crucial since it allows line 5 to be executed efficiently.

* Garey, Johnson, Miller and Papadimitrious [1979] report that the coloring problem for circle graphs *is* NP-complete.

```
    procedure MAXSTABLE(X):
      begin
1.      if X = ∅ then return ∅;;
2.      while there exists x ∈ X with w(x) undefined do
3.        S(x) ← {x} ∪ MAXSTABLE(U(x));
4.        w(x) ← |S(x)|;;
5.      T ← MAXWEIGHT CLIQUE(X, F_X);
6.      return ⋃_{v∈T} S(v);
      end
```

Figure 11.10

Theorem 11.5 (Gavril [1973]). Algorithm 11.4 correctly finds a maximum stable set of an overlap graph.

An example of Algorithm 11.4 applied to the intervals in Figure 11.8 follows the proof of the theorem.

Proof. We shall show that the procedure returns a maximum stable set of the subgraph $G_X = (X, A_X)$ for any subset $X \subseteq V$ satisfying $U(x) \subset X$ for all $x \in X$. The claim is certainly true if $|X| = 0$. Assume that it is true for all subsets smaller than X. In particular, by induction, $S(x) - \{x\}$ is a maximum stable set of $G_{U(x)}$, so $S(x)$ is a maximum stable set of $G_{\{x\} \cup U(x)}$ for each $x \in X$.

Let T be as defined in line 5. Since the intervals $\{I_v\}_{v \in T}$ are pairwise disjoint, and since $I_x \subset I_v$ for all $x \in S(v) - \{v\}$ and $v \in T$, it follows that $J = \bigcup_{v \in T} S(v)$ is a stable set of G_X. Thus

$$\alpha(G_X) \geq \sum_{v \in T} w(v) = |J|. \tag{1}$$

We must show that J is maximum.

Let J' be a maximum stable set of G_X, and let T' be the set of sinks of $(J', C_{J'})$; i.e.,

$$T' = \{y \in J' \mid I_y \subset I_z \text{ implies } z \notin J'\}.$$

Note that the intervals represented by T' are also pairwise disjoint, so, by the correctness of MAXWEIGHT CLIQUE,

$$\sum_{y \in T'} w(y) \leq \sum_{v \in T} w(v). \tag{2}$$

Now, clearly, if $S'(y) = \{x \in J' \mid I_x \subset I_y\}$, then $|S'(y)| = w(y)$ for all $y \in T'$, for otherwise we could replace $S'(y)$ with $S(y)$ and obtain a larger stable set. Hence,

$$\alpha(G_X) = |J'| = \sum_{y \in T'} w(y). \tag{3}$$

Combining (1)–(3), we obtain

$$\alpha(G_X) = |J|,$$

which concludes the required proof. ∎

Example 11.3. We apply Algorithm 11.4 to the intervals of Figure 11.8 in order to find a maximum stable set of their overlap graph $G = (V, A)$. The vertices can be assigned weights in any topologically sorted order with respect to C. We arbitrarily choose the order a,c,e,g,d,b,f. Clearly, $w(a) = w(c) = w(e) = w(g) = 1$ and $S(a) = \{a\}$, $S(c) = \{c\}$, $S(e) = \{e\}$, and $S(g) = \{g\}$ since they are all sources of C. Next, $S(d) = \{d, e\}$ and $w(d) = 2$ since MAXWEIGHT CLIQUE $(U(d), F_{U(d)}) = U(d) = \{e\}$. Now $U(b) = \{c, d, e, g\}$ and the heaviest clique in $F_{U(b)}$ is $\{d, g\}$; thus $S(b) = \{b, d, g\}$ and $w(b) = 3$. Similarly, $S(f) = \{f, g\}$ and $w(f) = 2$. Finally, MAXWEIGHT CLIQUE (V, F) could be either $\{a, d, g\}$ or $\{a, e, f\}$, both having weight 4. They give, respectively, the stable sets $\{a, d, e, g\}$ and $\{a, e, f, g\}$.

Next we provide an algorithm due to Gavril [1973] which solves the clique problem for circle graphs in polynomial time. The notions of matching diagram and permutation graph from Chapter 7 will be used.

Let $G = (V, E)$ be a circle graph with representing family $\{C_v\}_{v \in V}$ of chords of a circle $\mathscr{C}$, and let $N(v) = \{v\} + \text{Adj}(v)$.

Lemma 1. For every vertex $v \in V$, the induced subgraph $G_{N(v)}$ is a permutation graph.

Proof. We may assume that no two chords have a common endpoint. Thus, the chord C_v cuts $\mathscr{C}$ into two pieces such that for $x \in \text{Adj}(v)$ the chord C_x has one endpoint in each piece. Therefore, the subset $D = \{C_x\}_{x \in \text{Adj}(v)}$ is a matching diagram whose permutation graph is $G_{\text{Adj}(v)}$. Since connecting a new vertex to every vertex of a permutation graph results in another permutation graph, it follows that $G_{N(v)}$ is a permutation graph. ∎

Lemma 2. If K is a clique of G, then K is a clique of $G_{N(v)}$ for each $v \in K$.

Proof. Trivial. ∎

Algorithm 11.5. Maximum clique of a circle graph.
The algorithm is as follows:

```
begin
1. for v ∈ V do K_v ← MAXCLIQUE(G_N(v));
2. return the largest K_v;
end
```

By Lemma 1, statement 1 can be executed efficiently. By Lemma 2, the algorithm is correct. Details are left to the reader as Exercise 11.

5. A Graph Theoretic Characterization of Overlap Graphs

Although we have shown the equivalence of circle graphs, stack sorting graphs, and overlap graphs, we have not really characterized them from a traditional graph theoretic point of view. In this section we shall present such a characterization. The solution, however, will fall short of providing us with an efficient recognition algorithm.

Theorem 11.6 (Fournier [1978]). An undirected graph $G = (V, E)$ is an overlap graph if and only if there exists an acyclic orientation P of G and two linear extensions L_1 and L_2 of P such that the relation $F = (L_1 \cap L_2) - P$ satisfies

$$xy \in F,\ yz \in L_1 \Rightarrow xz \in F \tag{4}$$

and

$$xy \in L_2,\ yz \in F \Rightarrow xz \in F. \tag{5}$$

Remark. Such a relation F is transitive.

Proof. ($\Rightarrow$) Let $\mathcal{I} = \{I_x\}_{x \in V}$ be a collection of closed intervals on the real line, no two intervals sharing an endpoint, such that xx' is an edge of G if and only if I_x and $I_{x'}$ overlap (i.e., they intersect but neither contains the other). We denote $I_x = [a, b]$ and $I_{x'} = [a', b']$. Consider the binary relations defined on V as follows:

$$\begin{aligned} xx' \in P &\Leftrightarrow a < a' < b < b', \\ xx' \in L_1 &\Leftrightarrow a < a', \\ xx' \in L_2 &\Leftrightarrow b < b'. \end{aligned}$$

Clearly P is an acyclic orientation of G, and L_1 and L_2 are linear extensions of P. The relation $F = (L_1 \cap L_2) - P$ satisfies

$$xx' \in F \Leftrightarrow a < b < a' < b'$$

and represents I_x being entirely to the left of $I_{x'}$. It is easy to see that (4) and (5) are satisfied.

($\Leftarrow$) In the remainder of the proof, for any binary relation R we denote

$$R(x) = \{y \mid xy \in R\}.$$

Let $G = (V, E)$ be an undirected graph on n vertices and let P, L_1, L_2, and F satisfy the conditions of the theorem. To a vertex x of G we associate the interval $I_x = [a, b]$ as follows:

$$a = 1 + |L_1^{-1}(x)| + |P^{-1}(x)|,$$
$$b = 2n - |L_2(x)| - |P(x)|.$$

We shall show that $\mathcal{I} = \{I_x\}_{x \in V}$ is an overlap representation of G.

Claim 1. $1 \leq a < b \leq 2n$.

We shall prove the inequality $a < b$, the others being trivial. By the definitions of a and b, it is enough to prove the inequality

$$|L_1^{-1}(x)| + |F^{-1}(x)| + |L_2(x)| + |F(x)| \leq 2(n-1). \tag{6}$$

If $x' \in F^{-1}(x)$, then $x' \notin L_2(x)$, because $F \subset L_2$ and L_2 is antisymmetric; thus $F^{-1}(x) \cap L_2(x) = \varnothing$. Similarly, $F(x) \cap L_1^{-1}(x) = \varnothing$. Thus, each member x' of V is counted at most twice on the left side of (6) except for x itself, which is not counted at all. This proves Claim 1.

For vertices x and x' $(x \neq x')$, where $I_x = [a, b]$ and $I_{x'} = [a', b']$ are defined as above, we shall show the following three implications:

Claim 2.

(i) $xx' \in L_1 - (P \cup F) \Rightarrow a < a' < b' < b$,

(ii) $xx' \in F \Rightarrow b < a'$,

(iii) $xx' \in P \Rightarrow a < a' < b < b'$.

Notice that $xx' \in L_1 - (P \cup F)$ if and only if $xx' \in L_1 - L_2$, and since L_2 is a total order, implication (i) would follow directly from

(i_1) $xx' \in L_1 \Rightarrow a < a'$ and (i_2) $xx' \in L_2 \Rightarrow b < b'$.

Let $xx' \in L_1$. Since L_1 is a total order we have $L_1^{-1}(x) \subset L_1^{-1}(x')$, and thus $|L_1^{-1}(x)| < |L_1^{-1}(x')|$. (The strict inequality is due to $x \in L_1^{-1}(x')$ and $x' \notin L_1^{-1}(x)$.) Also, from property (i_1), we have $F^{-1}(x) \subset F^{-1}(x')$, so $|F^{-1}(x)| < |F^{-1}(x')|$. Combining these inequalities we obtain

$$|L_1^{-1}(x)| + |F^{-1}(x)| < |L_1^{-1}(x')| + |F^{-1}(x')|,$$

which yields $a < a'$ and proves (i_1). Implication (i_2) is proved in the same fashion. This proves (i).

For implication (ii) we shall show that if $xx' \in F$, then

$$|L_1^{-1}(x')| + |F^{-1}(x')| + |L_2(x)| + |F(x)| \geq 2n. \tag{7}$$

Let $x'' \in V$. If $x'' \notin L_1^{-1}(x')$ (i.e., $x'x'' \notin L_1^{-1}$), then $x'x'' \in L_1$ since L_1 is a total order; moreover, (4) implies that $xx'' \in F$ (i.e., $x'' \in F(x)$), and hence $x'' \in L_2(x)$. In an analogous manner, if $x'' \notin L_2(x)$, then $x'' \in F^{-1}(x')$ and

$x'' \in L_1^{-1}(x')$. Thus, each vertex of V, including x and x', is counted at least twice on the left side of (7). This proves (ii).

For implication (iii), since (i_1) and (i_2) hold, it is sufficient to show that if $xx' \in P$ then $a' < b$, or equivalently, if $xx' \in P$, then

$$|L_1^{-1}(x')| + |F^{-1}(x')| + |L_2(x)| + |F(x)| \leq 2(n-1). \tag{8}$$

It is easy to verify that if $x'' \in F^{-1}(x')$ then $x'' \notin L_2(x)$ and $x'' \notin F(x)$ (since F is a transitive relation). Similarly, if $x'' \in F(x)$, then $x'' \notin L_1^{-1}(x')$ and $x'' \notin F^{-1}(x')$. Thus, each element of V is counted at most twice on the left side of (8), except for x and x', which are counted exactly once each ($x \in L_1^{-1}(x')$ and $x' \in L_2(x)$). This proves (iii) and concludes Claim 2.

Finally, the three conditions of (i)–(iii) are mutually exclusive and cover all possibilities. Therefore, the opposite implications also hold in (i)–(iii). In particular,

$$xx' \in P \Leftrightarrow a < a' < b < b'.$$

Since P is an orientation of G, we conclude that

$$xx' \in G \Leftrightarrow I_x \text{ and } I_{x'} \text{ overlap},$$

and $\mathscr{I} = \{I_x\}_{x \in V}$ is the desired overlap model of G. ∎

Remark. If the relation F in Theorem 11.6 is empty, then G is a permutation graph; conversely, for every permutation graph there exist relations P, L_1, and L_2 as in the theorem with $P = L_1 \cap L_2$ (as in the proof of Theorem 7.1.) However, even when G is a permutation graph there may very well exist other relations P, L_1, and L_2 satisfying the conditions of the theorem for which $P \neq L_1 \cap L_2$. For example, letting $G = \bar{K}_3$ with $P = \varnothing$, $L_1 = [x < y < z]$, and $L_2 = [x < z < y]$, we obtain $L_1 \cap L_2 \neq \varnothing$.

Historical Note

We began this chapter by discussing a problem using stacks. We conclude with an historical note on one of the oldest written references to the notion of "last-in, first-out."* The reference occurs in a commentary by Rashi (Rabbi Solomon ben Isaac) on the Biblical verse

> Then his brother emerged, his hand seizing Esau's heel; so they named him Jacob.† Isaac was sixty years old when they were born [Genesis XXV, 26].

* We are indebted to Gideon Ehrlich for pointing out this reference in a communication with Edward M. Reingold, who then passed it on to the author. The translation of the Rashi quotation is due to E. M. Reingold.

† In Hebrew, Ya'akov, play on the word 'aqev meaning "heel."

Rashi lived from 1040 to 1105 A.D., residing primarily in Troyes, France, his birthplace, where he founded one of the leading schools of the time. He is the most famous biblical and talmudic commentator in all of Jewish history. His commentary on this verse is as follows:

> I heard a Midrashic legend expounding on the meaning [of the phrase "Then his brother emerged ..."]. It was his [Jacob's] right, the grasping of his [Esau's] heel: Jacob was conceived from the first drop and Esau from the second. Consider a tube with narrow mouth. Put two stones into it, one after the other—the first to enter exits last and the last to enter exists first. It is [thus] found [that] Esau, conceived last, exited first, and Jacob, conceived first, exited last. [Thus] Jacob went to delay him [Esau] so that he [Jacob] would be first born [just] as he was first produced and to be a first fruit of the womb and to take the birthright [as he deserved] according to the law.

Rashi is clearly describing a stack mechanism.

EXERCISES

1. Find a permutation π whose graph $H[\pi]$ is a chordless pentagon plus some isolated vertices.

2. Using the data structure suggested in the text, write computer programs to implement Algorithms 11.1 and 11.2 and test them on the examples given in this chapter.

3. Does $H[\pi]$ always have some isolated vertices? Prove that if $H[\pi]$ has exactly one isolated vertex then $H[\pi] = \overline{G[\pi]}$ but not conversely (see Chapter 7).

4. Show that if the output of Algorithm 11.2 is used as the input of Algorithm 11.1, then the resulting composition may change the set of chords. Modify Algorithm 11.2 so that this does not happen.

5. By an arbitrary convention we have discussed sorting a permutation π in a parallel system of stacks from right to left. The problem of sorting π from left to right* is equivalent to forming π from $[1, 2, \ldots, n]$ from right to left. In this case one should study the coloring problem on the undirected graph $H^{\rho}[\pi]$ having vertices $\{1, 2, \ldots, n\}$ with i and j adjacent if there is a k such that $i < j < k$ and $\pi_k^{-1} < \pi_i^{-1} < \pi_j^{-1}$. In general the chromatic numbers of $H[\pi]$ and $H^{\rho}[\pi]$ are different.

* n would emerge first and 1 last.

Let the function $\rho: i \mapsto n + 1 - i$ act on the labeled vertices of a graph* and be composed with other permutations in the obvious way. Also let $\mathscr{H}^\rho = \{G \mid G \cong H^\rho[\pi] \text{ for some } \pi\}$. Prove the following:

(i) $\rho \circ H^\rho[\pi] = H[\rho \circ \pi \circ \rho]$,
(ii) $\rho \circ H[\pi] = H^\rho[\rho \circ \pi \circ \rho]$,
(iii) $\mathscr{H} = \mathscr{H}^\rho$.

6. Let the graph $H[\pi]$ be properly colored using t colors. Show that the following algorithm correctly sorts π using a network of t stacks in parallel.
Input: The permutation π with the numbers properly colored.
Output: The permutation $[1, 2, \ldots, n]$ sorted.
Method: The t stacks are in one-to-one correspondence with the colors. The algorithm is as follows:

```
begin
  k ← 1;
  while k ≤ n do
    if k can be moved onto the output queue then
      move k onto the output queue;
    else
      move the next number of the input queue onto the stack of its color;;
end
```

7. Show that a permutation π can be sorted in a network of k stack in parallel under the restriction that all numbers *must* be loaded into the stacks before any unloading can begin if and only if its reversal π^ρ can be sorted in a network of k queues in parallel. Give some additional equivalent conditions.

8. Prove that Algorithm 11.3 correctly calculates the adjacency sets of the graph $G = (V, E)$ of intersecting chords of a circle. Show that the algorithm can be implemented to run in $O(|V| + |E|)$.

9. Let $f(n)$ be the length of the shortest string of numbers from $\{1, 2, \ldots, n\}$ which contains all $n!$ permutations as subsequences. Prove that $f(n) \leq n^2 - 2n + 4$ $(n \geq 3)$ (Koutas and Hu [1975]).

10. Determine the computational complexity of Algorithm 11.4.

11. Determine the computational complexity of Algorithm 11.5 taking into consideration that each subgraph $G_{N(v)}$ must be calculated.

12. A circle $\mathscr{C}$ with chords *admits an equator* if an additional chord may be added to $\mathscr{C}$ which will intersect every other chord.

(i) Prove that G is a permutation graph if and only if G is the intersection graph of a circle of chords which admits an equator.

(ii) Give an example of a circle $\mathscr{C}$ which does *not* admit an equator but whose intersection graph is a permutation graph.

13. The sequence of operations used in sorting the permutation [3,5,4,1,6,2]

*denoted $\rho \cdot G$.

as in Section 11.1 can be abbreviated by the code

$$S_1S_2S_2S_1X_1S_3S_1X_1X_1X_2X_2X_3,$$

where S_i stands for "move the next number from the input queue onto stack i" and X_i stands for "move the number at the top of stack i to the output queue." Some sequences of S_i's and X_i's specify meaningless operations; for example, the sequence $S_1S_2S_1X_2X_1S_3X_2X_1S_2X_3$ cannot be carried out.

We call a sequence of S_i's and X_i's *admissible* if it contains the same number of S_i's and X_i's for each integer i, and if it specifies no operation that cannot be performed. Formulate a rule which distinguishes between admissible and inadmissible sequences (Knuth [1969, Exercise 2.2.1, No. 3]).

Bibliography

Even, Shimon, and Itai, Alon

[1971] Queues, stacks and graphs, *in* "Theory of Machines and Computations," pp. 71–86. Academic Press, New York.

Fournier, Jean-Claude

[1978] Une caractérization des graphes de cordes, *C.R. Acad. Sci. Paris* **286A**, 811–813.

Garey, M. R., Johnson, D. S., Miller, G. L., and Papadimitrious, C. H.

[1979] The complexity of coloring circular arcs and chords, submitted for publication.

Gavril, Fanica

[1973] Algorithms for a maximum clique and a minimum independent set of a circle graph, *Networks* **3**, 261–273. MR49 #4862.

Knuth, Donald E.

[1969] "The Art of Computer Programming," Vol. 1. Addison-Wesley, Reading, Massachusetts.

[1973] "The Art of Computer Programming," Vol. 3. Addison-Wesley, Reading, Massachusetts.

Koutas, P. J., and Hu, T. C.

[1975] Shortest string containing all permutations, *Discrete Math.* **11**, 125–132. MR50 #12740.

Read, Ronald C.

[1979] The chord intersection problem. *Ann. N.Y. Acad. Sci.* **319**, 444–454.

Riordan, John

[1975] The distribution of crossings of chords joining pairs of $2n$ points on a circle, *Math. Comp.* **29**, 215–222. MR51 #2933.

Tarjan, Robert Endre

[1972] Sorting using networks of queues and stacks, *J. Assoc. Comput. Mach.* **19**, 341–346. MR45 #7852.

Touchard, Jacques

[1952] Sur un problème de configurations et sur les fractions continues, *Canad. J. Math.* **4**, 2–25. MR13, p. 716.

Zelinka, Bohdan

[1965] The graph of the system of chords of a given circle, *Mat.-Fyz. Casopis Sloven. Akad. Vied* **15**, 273–279. MR33 #2575.