# Permutation Graphs and Transitive Graphs

S. EVEN AND A. PNUELI

*The Weizmann Institute of Science, Rehovot, Israel*

AND

A. LEMPEL

*Sperry Rand Research Center, Sudbury, Massachusetts*

ABSTRACT. A graph $G$ with vertex set $N = \{1, 2, \cdots, n\}$ is called a permutation graph there exists a permutation $P$ on $N$ such that for $i, j \in N$, $(i - j)[P^{-1}(i) - P^{-1}(j)] < 0$ if ar only if $i$ and $j$ are joined by an edge in $G$.

A structural relationship is established between permutation graphs and transitive graph An algorithm for determining whether a given graph is a permutation graph is given. Efficie algorithms for finding a maximum size clique and a minimum coloration of transitive grap are presented. These algorithms are then shown to be applicable in solving problems in memo allocation and circuit layout.

KEY WORDS AND PHRASES: graphs, permutations, permutation graphs, transitive graph cliques, maximal cliques, chromatic decomposition of graphs, minimal chromatic decompos tion, memory allocation problems

CR CATEGORIES: 3.73, 4.43, 5.32

## 1. *Introduction*

Let $P = [P(1), P(2), \cdots, P(n)]$ be a permutation of the positive integers 2, $\cdots$, $n$. Let $N = \{1, 2, \cdots, n\}$ and $\Pi$ be a subset of $N \times N$ defined as follow

$$\Pi = \{(i, j) \mid i < j \quad \text{and} \quad P^{-1}(i) > P^{-1}(j) \quad \text{or} \quad i > j \quad \text{and} \quad P^{-1}(i) < P^{-1}(j)$$

where $P^{-1}(i)$ is the element of $N$ which $P$ maps into $i$. In a more pictorial wa draw the matching diagram for the permutation. In this matching diagram th line connecting the two $i$'s intersects the line connecting the two $j$'s if and only $(i, j) \in \Pi$.

*Example* 1. Let $P = [2, 5, 4, 1, 3]$. The matching diagram is shown in Figure We now define the *permutation graph* of $P$ to be $G(N, \Pi)$; that is, $G$ is an undirecte graph whose vertices are 1, 2, $\cdots$, $n$ and its edges are specified by the relation I Clearly, $G$ has no loops and no parallel edges. The corresponding graph for $P$ Example 1 is shown in Figure 2.

Authors' present addresses: S. Even and A. Pnueli, The Weizmann Institute of Scienc Rehovot, Israel; A. Lempel, Department of Electrical Engineering, Technion—Israel Inst tute of Technology, Haifa, Israel.

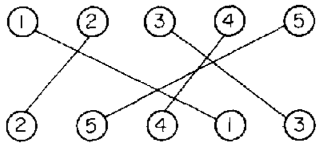

FIG. 1. Matching diagram for $P = [2, 5, 4, 1, 3]$

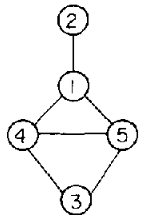

FIG. 2. The permutation graph for Example 1

A directed graph $\vec{G}$ is called *transitive* if the existence of edges $i \rightarrow j$ and $j \rightarrow k$ in $\vec{G}$ implies the existence of an edge $i \rightarrow k$ in $\vec{G}$.

In Section 2 we establish the structural relationship between permutation graphs and transitive graphs and describe an algorithm for determining whether a given graph is a permutation graph. Because of their special structural properties, these two families of graphs are very helpful in modeling and solving various problems as described in the remainder of the paper. In Section 3 we derive an efficient algorithm for finding a maximum size clique of a transitive graph. In Section 4 we present a problem of memory allocation and show that it reduces to that of finding a maximum clique of a permutation graph and therefore is easily solved by the technique of Section 3. In Section 5 we describe an efficient procedure for minimal chromatic decomposition of transitive graphs, and show that it provides a solution to the minimum plane connection problem described by Liu [3].

In view of the efficiency of these algorithms for transitive and permutation graphs, it is of interest to have procedures for deciding whether a given finite graph is transitively orientable and whether a given graph is isomorphic to a permutation graph. We have solved these two problems in a later paper [4].

## 2. *Characterization of Permutation Graphs*

Our first aim is to characterize the graphs $G(N, R)$ which are permutation graphs of some $P$, and to devise an algorithm for finding a $P$ if such exists. Notice that we assume a fixed labeling $1, 2, \cdots, n$ for the graph vertices. We do not consider here the problem of deciding whether there exists a relabeling for which the graph becomes a permutation graph for some $P$.

Let us introduce a few additional terms. Assume $G(N, R)$ is a given undirected graph with no loops and no parallel edges; namely, $R$ is an irreflexive symmetric relation. Define $R'$ to be $(N \times N) - \{(i, i) \mid i \in N\} - R$. The graph $G'(N, R')$ is, therefore, the complementary undirected graph of $G(N, R)$. Now define

$$\vec{R} = \{(i, j) \mid i < j \quad \text{and} \quad (i, j) \in R\}.$$

Thus $\vec{G}(N, \vec{R})$ is actually $G(N, R)$, where all its edges are now directed from low to high. Similarly, define

$$\overleftarrow{R}' = \{(i, j) \mid i > j \quad \text{and} \quad (i, j) \in R'\}.$$

Therefore, $\overleftarrow{G}'(N, \overleftarrow{R}')$ is actually $G'(N, R')$ where all its edges are now directed from high to low. Finally, define

$$\overleftrightarrow{R} = \vec{R} \cup \overleftarrow{R}'$$

and $\overleftrightarrow{G}(N, \overleftrightarrow{R})$ is now the "union" of $\overrightarrow{G}$ and $\overleftarrow{G}'$. It consists of all the edges of $G$, directed from low to high, and of all edges missing from $G$ (except the loops) now added and directed from high to low.

THEOREM 1. *$G(N, R)$ is a permutation graph if and only if $\overrightarrow{G}(N, \overrightarrow{R})$ and $\overleftarrow{G}'(N, \overleftarrow{R}')$ are transitive.*

PROOF. Assume $G(N, R)$ is a permutation graph. The definitions of a permutation graph and of $\overrightarrow{G}$ and $\overleftarrow{G}'$ imply that

$$\overrightarrow{R} = \{(i, j) \mid i < j \qquad \text{and} \qquad P^{-1}(i) > P^{-1}(j)\}$$

and

$$\overleftarrow{R}' = \{(i, j) \mid i > j \qquad \text{and} \qquad P^{-1}(i) > P^{-1}(j)\}.$$

It is easy to see that both $\overrightarrow{R}$ and $\overleftarrow{R}'$ are transitive.

We postpone the proof of the "if" part of the theorem until we develop some preliminary results.

A directed graph (with no loops) is called *complete* if for every pair of vertices $a$, $b$, either $(a, b)$ is an edge or $(b, a)$ is an edge, but not both; it is called *circuit-free* if there exists no directed circuit in the graph.

LEMMA 1. *A complete directed graph is circuit-free if and only if it contains no directed triangles.*

PROOF. The "only if" part is obvious. Assume now that the graph is not circuit-free. Let $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v_l \rightarrow v_1$ be a directed circuit of minimum length. If $l = 3$, Lemma 1 follows. If not, consider the edge between vertex $v_1$ and $v_3$. If $v_1 \rightarrow v_3$, $l$ is not the minimum length of a directed circuit. If $v_3 \rightarrow v_1$, the graph contains a directed triangle. Q.E.D.

LEMMA 2. *If $\overrightarrow{G}(N, \overrightarrow{R})$ and $\overleftarrow{G}'(N, \overleftarrow{R}')$ are both transitive, then $\overleftrightarrow{G}(N, \overleftrightarrow{R})$ is a complete circuit-free directed graph.*

PROOF. It follows directly from definition that $\overleftrightarrow{G}(N, \overleftrightarrow{R})$ is a complete directed graph. If it contains directed circuits, then by Lemma 1 it contains a directed triangle $a \rightarrow b \rightarrow c \rightarrow a$. Without loss of generality, we may assume that either $a < b < c$ or $a > b > c$. In the former case the edges $(a, b)$ and $(b, c)$ were contributed by $\overrightarrow{R}$, and by the transitivity of $\overrightarrow{G}(N, \overrightarrow{R})$ we also have $(a, c)$ both in $\overrightarrow{R}$ and $\overleftrightarrow{R}$. This contradicts the assumption that $(c, a) \in \overleftrightarrow{R}$. In the latter case a similar contradiction arises from the transitivity of $\overleftarrow{G}'(N, \overleftarrow{R}')$. Q.E.D.

We now return to the proof of the "if" part of Theorem 1. We assume that both $\overrightarrow{G}(N, \overrightarrow{R})$ and $\overleftarrow{G}'(N, \overleftarrow{R}')$ are transitive. By Lemma 2, $\overleftrightarrow{G}(N, \overleftrightarrow{R})$ is a complete, circuit-free directed graph. Thus it must contain a sink, namely, a vertex which has no edge emanating from it. (Clearly it cannot have more than one sink; consider the edge connecting two vertices, both of which are supposed to be sinks.)

Let the unique sink of $\overleftrightarrow{G}(N, \overleftrightarrow{R})$ be the vertex $s_1 \in N$ and let $N_1 = N - \{s_1\}$. The subgraph $\overleftrightarrow{G}_1$ of $\overleftrightarrow{G}$, spanned by the vertices belonging to $N_1$, is also a complete, circuit-free graph and therefore $\overleftrightarrow{G}_1$ also contains a unique vertex $s_2 \in N_1$ with no edges emanating from it. The same is true for the next subgraph $\overleftrightarrow{G}_2$, spanned by $N_2 = N_1 - \{s_2\}$, and so on.

Consider the sequence of successive sinks $s_1, s_2, \cdots, s_n$ obtained as described above. It is clear that $\overleftrightarrow{R}$ is the set of all ordered pairs $(s_i, s_j)$ for which $i > j$, and that $\overrightarrow{R}$ is the subset of $\overleftrightarrow{R}$ for which the additional condition $s_i < s_j$ holds. Also, since $s_i = s_j$ if and only if $i = j$, the mapping $P: i \rightarrow s_i$ $(i \in N)$ is a permutation on $N$, with $P(i) = s_i$ and $P^{-1}(s_i) = i$. Thus $R$ is the set of all unordered pairs $(s_i, s_j)$ for which either $s_i < s_j$ and $P^{-1}(s_i) > P^{-1}(s_j)$ or $s_i > s_j$ and $P^{-1}(s_i) < P^{-1}(s_j)$. Hence $G(N, R)$ is a permutation graph with $[P(1), P(2), \cdots, P(n)] = [s_1, s_2, \cdots, s_n]$. Q.E.D.

This proof implies an algorithm for deciding whether a given graph $G(N, R)$ is a permutation graph. The defining permutation is obtained by the sequence of sinks of $\overleftrightarrow{G}(N, \overleftrightarrow{R})$ when sinks are successively eliminated from the graphs. In case one of the resulting graphs does not have a sink, the original graph is not a permutation graph.

*Example* 2. Let us apply the suggested algorithm to the graphs given in Figure 3. First we construct $\overleftrightarrow{G}(N, \overleftrightarrow{R})$ for the graph given in Figure 3(a). This is shown in Figure 4, where the solid lines show the arcs of $\overrightarrow{G}$ and the dashed lines are those of $\overleftarrow{G'}$. As is easily observed, this graph does not have any sink, and therefore the graph of Figure 3(a) is not a permutation graph. Next, the graph $\overleftrightarrow{G}$ for the graph of Figure 3(b) is shown in Figure 5(a). The sink of this graph is vertex 3, and therefore we assign $P(1) = 3$. Now, vertex 3 is eliminated from the graph to yield the graph shown in Figure 5(b). Vertex 5 is now a sink, thus $P(2) = 5$. The successive steps are shown in Figure 5(c) and 5(d). The resulting permutation is [3, 5, 4, 1, 2].

There are several properties of permutation graphs which we shall mention here. It follows immediately from our discussion that if $G(N, R)$ is a permutation graph defined by $P = [P(1), P(2), \cdots, P(n)]$, then $G'(N, R')$ is also a permutation graph and its permutation is $P(n + 1 - i)$, namely, $[P(n), P(n - 1), \cdots, P(1)]$

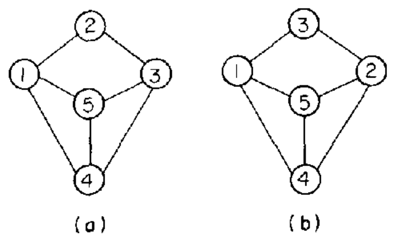

FIG. 3. The graphs of Example 2

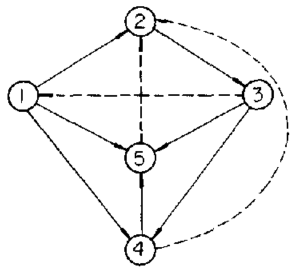

FIG. 4. $\overleftrightarrow{G}$ for the graph of Example 2(a)

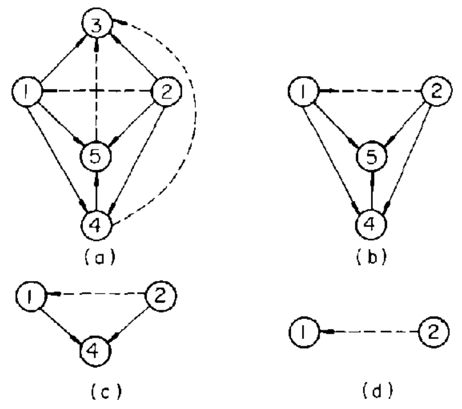

FIG. 5. The successive steps in the testing of the graph of Example 2(b)

Also, it is not difficult to see that the graph defined by $P^{-1}$ is isomorphic to that of $P$, where the image of $i$ in the graph defined by $P$ is the vertex $P^{-1}(i)$ of the graph defined by $P^{-1}$.

## 3. *Cliques and Independent Sets of Circuit-Free Transitive Directed Graphs*

Let $G(V, E)$ be any finite directed graph with the following properties: (1) $G$ has no loops and no parallel edges; (2) $G$ is circuit-free; (3) $G$ is transitive. We can find an order of $V$ which will have the property that edges always go from low to high. This is easily done in the following way: find all the sinks of $G$ and order them in any arbitrary way; eliminate them from the graph and find all the sinks of the new graph again ordered arbitrarily—they are next in the order; etc. Once this is done we may assume that the vertices are $1, 2, \cdots, n$ where $n$ is the number of vertices of $G$.

*Independent sets* of vertices are usually defined for undirected graphs in the following way: a set of vertices is called independent if no two vertices of the set are connected in the graph by an edge. A *maximal independent set* is an independent set to which no vertex can be added without violating this condition. These definitions are easily extended to directed graphs by simply referring to their underlying graph, namely, by ignoring the directions.

A set of vertices of an undirected graph is called *complete* if every two vertices of the set are connected by an edge in the graph. A *clique* is a maximal complete set of vertices. There is an obvious relation between independent sets and complete sets; the independent sets are complete sets of vertices of the complementary graph, and the same relation holds between maximal independent sets and cliques.

It is often necessary to find the set of all cliques or maximal independent sets of a given graph. Also, it is sometimes necessary to find a maximum size clique or maximum size independent set. Several algorithms were published for generating the lists of all cliques of a given graph, and the best one known to the authors is that of Paull and Unger [1, Process D]. This is still a cumbersome procedure, although we do not believe that a more efficient one can be found for graphs with no given structural properties. The situation with the problem of finding a maximum clique is similar. The authors know of no algorithm which solves this problem in $n^k$ steps.

In this section we show that, in the case of directed transitive graphs, the problem of detecting the cliques is easier and that of finding a maximum clique is almost immediate.

Let $G(N, E)$ be a given directed transitive graph, where $N = \{1, 2, \cdots, n\}$ and edges are always directed from low to high. Assume $1 \leq m < n$ and that $S$ is a clique of the section graph defined on $\{1, 2, \cdots, m\}$, namely, the subgraph with vertices $1, 2, \cdots, m$ and the same edges between them as in $G$. Let $k$ be the highest vertex in the clique. It is now observed that $S \cup \{m + 1\}$ is a clique of the section graph defined by $\{1, 2, \cdots, m, m + 1\}$ if and only if $(k, m + 1)$ is an edge of $G$. For, if $S \cup \{m + 1\}$ is a clique, then $(k, m + 1)$ must be an edge in $G$; and if $(k, m + 1)$ is an edge in $G$, then every other vertex in $S$ must also be connected to $m + 1$. This provides an immediate simplification of the Paull and Unger procedure.

A more significant result is provided in view of the above-mentioned observation

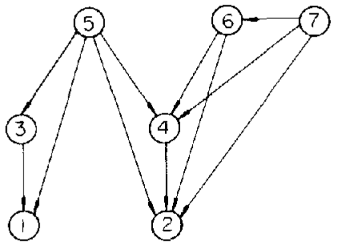

FIG. 6. $G(N, E)$ for Example 3

for finding a maximum clique. Let us construct a sequence of $n$ integers $c(1)$, $c(2), \cdots, c(n)$, where $c(i)$ is the number of vertices in a maximum clique containing vertex $i$ of the section graph defined on $\{1, 2, \cdots, i\}$. Clearly, $c(1) = 1$. In general, when $c(1), c(2), \cdots, c(i-1)$ are known, we look for those vertices among $1, 2, \cdots, i-1$ which are connected by an edge to $i$. If the set $J_i = \{j \mid (j, i) \in E\}$ of these vertices is empty, then $c(i) = 1$; otherwise, $c(i) = 1 + \max_{j \in J_i} c(j)$. The size of the maximum clique of $G(N, E)$ is $\max_{j \in N} c(j)$.

After all the $c(i)$ have been obtained, a traceback operation is performed to group all the nodes belonging to a maximum clique. We locate first a $j$ for which $c(j)$ is maximal. This $j$ must participate in a maximum clique and is, in fact, the highest vertex in it. After gathering vertices $i_1 > i_2 \cdots > i_m$ and $c(i_m) > 1$, we locate the next vertex by searching for $i$ such that $(i, i_m) \in E$ and $c(i) + 1 = c(i_m)$ $(i < i_m)$, etc.

The outlined algorithm is extendable to a weighted graph where weight $\omega_i$ is assigned to vertex $i$, and we look for a clique with the maximum weight sum of its vertices. In this case the maximum clique weights are generated by:

$$c(i) = \omega_i \qquad \text{if } |J_i| = 0 \text{ (as is always the case for } i = 1\text{)},$$

$$c(i) = \omega_i + \max_{j \in J_i} c(j) \qquad \text{if } |J_i| > 0.$$

*Example* 3. Let $G(N, E)$ be the graph shown in Figure 6. First we have $c(1) = 1$, and since $(1, 2) \notin E$, then $c(2) = 1$. Since 3 is connected to 1 (among 1 and 2), $c(3) = c(1) + 1 = 2$. Similarly, $c(4) = 2$. Vertex 5 is connected to 1, 2, 3, 4. The maximum value of the corresponding $c$'s is 2; thus $c(5) = 3$. Also, $c(6) = 3$. Vertex 7 is connected to 2, 4, and 6. Thus $c(7) = 4$. We now know that a maximum clique is of size 4 and its highest member is the vertex 7. We search for a lower vertex connected to it whose $c$ value is 3; this is vertex 6, etc. In this way we trace a maximum clique $\{2, 4, 6, 7\}$.

This is a very efficient dynamic programming solution to this problem; the number of elementary operations it requires is of the order $n^2$. A natural question then arises: are there any interesting problems in which the helpful transitivity is present? A few problems of this type are discussed in the next two sections.

## 4. *Memory Reallocation Problem*

In this section we bring an application of the maximum clique algorithm to the field of system programming.

In a multiprogramming computer system environment, the computer's memory holds at one time $n$ programs, whose starting addresses are respectively $x_1, x_2, \cdots, x_n$, with $x_i < x_j$ for $i < j$.

After a certain time, some of these programs change their memory space requirements, with the new length of the $i$th program being $l_i$. To satisfy these requirements, some of the programs are shifted bodily from one address to another so as to make new space.

With each program $i$ we associate a cost of transplantation $\omega_i$ which incurs if the program is shifted but is independent of the distance shifted. We stipulate that the shifts permitted preserve the order of the programs in memory, and also that the overall memory requirement fits the available space [2].

The problem is to minimize the reallocation costs. To tackle this problem, we define an undirected graph $G(N, R)$ whose vertex $i$ corresponds to the $i$th program. We draw an edge $(i, j) \in R$ if the local memory requirements are such that programs $i$ and $j$ can both start at the same initial addresses $x_i$ and $x_j$, respectively. This is expressed for $i < j$ by

$$(i, j) \in R \Leftrightarrow \sum_{k=i}^{j-1} l_k \leq x_j - x_i$$

and similarly for $i > j$.

A maximal clique in the graph $G$ corresponds to a set of programs that can be left in place simultaneously while the others must be shifted. Thus the cost minimization problem is equivalent to the problem of finding a maximum clique (with maximum sum of the $\omega_i$'s) in $G$.

We proceed to show that $\overrightarrow{G}(N, \overrightarrow{R})$ and $\overleftarrow{G}'(N, \overleftarrow{R}')$ are both transitive. Let $i < j < k$. If $(i, j) \in R$ and $(j, k) \in R$, then by definition

$$\sum_{m=i}^{j-1} l_m \leq x_j - x_i, \qquad \sum_{m=j}^{k-1} l_m \leq x_k - x_j .$$

Adding these two inequalities, we obtain

$$\sum_{m=i}^{k-1} l_m \leq x_k - x_i$$

and thus $(i, k) \in R$.

Similarly, by using the oppositely directed inequalities we derive

$$(i, j) \in R' \quad \text{and} \quad (j, k) \in R' \Rightarrow (i, k) \in R'.$$

Thus any memory reallocation of the type described generates a permutation graph. In particular, it generates a transitive graph and we may use the maximum clique procedure of the previous section for finding an optimal reallocation.

To ensure that the programs are not shifted out of memory, we modify our description by requiring that programs 1 and $n$, which are dummy programs, never require additional space and are not to be moved. This can be taken care of either by directly looking for the maximum clique which contains vertices 1 and $n$, or indirectly, by assigning them very high shift costs $\omega_1$ and $\omega_n$.

*Example* 4. In TABLE I we summarize a reallocation problem and its solution. We assume that programs 1 and 8 are dummy programs and should not be moved. Also, we assume that all programs have the same cost of transplantation. Therefore, the problem reduces to that of finding a maximum (in number of vertices in it) clique among those which contain both vertex 1 and 8.

The graph $G(N, R)$ of our example is shown in Figure 7. There is a unique clique of maximum size which contains both 1 and 8: {1, 2, 5, 8}. Thus programs 2 and 5

TABLE I

| *Program number* | $x_i$ *Start address* | $l_i$ *Length required* | *Size of a maximum clique (containing 1) whose highest vertex is i* |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 2 | 0 | 300 | 2 |
| 3 | 200 | 100 | 0 |
| 4 | 600 | 600 | 3 |
| 5 | 1000 | 200 | 3 |
| 6 | 1100 | 100 | 0 |
| 7 | 1500 | 300 | 4 |
| 8 | 1700 | 0 | 4 |

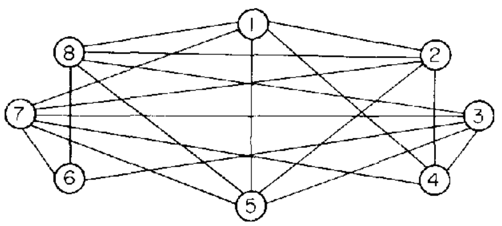

FIG. 7. $G(N, R)$ of Example 4

can be left in place while the others must shift. As the reader can establish for himself, there is no need to construct $G(N, R)$, and the last column of the table can be computed directly.

We conclude this section by showing that not only do memory reallocation problems generate permutation graphs, but also that any permutation graph may be generated by an appropriate memory reallocation problem.

Let $G(N, R)$ be a permutation graph. We want to show that we can always find $x_i$, $i = 1, \cdots, n$ and $l_i$, $i = 1, \cdots, n - 1$ such that for every $i < j$

$$(i, j) \in \overrightarrow{R} \Rightarrow \sum_{k=i}^{j-1} l_k \le x_j - x_i,$$

$$(j, i) \in \overleftarrow{R}' \Rightarrow \sum_{k=i}^{j-1} l_k > x_j - x_i.$$

This is a set of $n(n - 1)/2$ inequalities in $2n - 1$ unknowns. It will be convenient if we can find a basic smaller subset of inequalities which, when satisfied, ensure satisfaction of all the others.

Consider the inequality corresponding to a pair $(i, j)$ denoted by $I(i, j)$. In the complete graph $\overleftrightarrow{G}(N, \overleftrightarrow{R})$, the type of constraint satisfied indicates whether there is an arc of $\overrightarrow{G}$ leading from $i$ to $j$ or an arc of $\overleftarrow{G}'$ leading from $j$ to $i$ (assuming $i < j$).

Enumerate the sequence of sinks in the order they are selected for constructing the permutation corresponding to the graph:

$$s_1 \leftarrow s_2 \leftarrow \cdots \leftarrow s_n .$$

This is a Hamiltonian directed path of $\overleftrightarrow{G}$ leading from $s_n$ to $s_1$. Note that the direction of all other arcs can be deduced by transitivity from this path. This observation suggests that it may be sufficient to consider the basic inequalities subset $\{I(s_1, s_2), I(s_2, s_3), \cdots, I(s_{n-1}, s_n)\}$.

Suppose we have a solution of $l_i$ $(i = 1, \cdots, n-1)$, $x_i$ $(i = 1, \cdots, n)$ satisfying this basic set. Construct the memory reallocation problem involving these $l_i$, $x_i$. It generates a complete graph $\overleftrightarrow{G}$. The satisfaction of the basic subset of inequalities implies the existence of the Hamiltonian path $s_1 \leftarrow s_2 \leftarrow \cdots \leftarrow s_n$ also in $\overleftrightarrow{G}$. Since all other arcs are deducible by transitivity, $\overleftrightarrow{G}$ must be identical to $\overleftrightarrow{G}$, and therefore all the other inequalities implied by $\overleftrightarrow{G}$ are satisfied as well.

We are now left with the task of solving the basic set. The inequalities of the basic set are:

$$I(s_i, s_{i+1}) = \begin{cases} \sum_{k=s_{i+1}}^{s_i-1} l_k \leq x_{s_i} - x_{s_{i+1}} & \text{for } s_{i+1} < s_i, \\ \sum_{k=s_i}^{s_{i+1}-1} l_k > x_{s_{i+1}} - x_{s_i} & \text{for } s_i < s_{i+1}. \end{cases}$$

We look for a solution of a particular form. Set $l_1 = l_2 = \cdots = l_{n-1} = L$ and $x_i = (i-1)\cdot L + \delta_i$. The basic inequalities are then

$$I(s_i, s_{i+1}) = \begin{cases} \delta_{s_i} - \delta_{s_{i+1}} \geq 0 & \text{for } s_{i+1} < s_i, \\ \delta_{s_{i+1}} - \delta_{s_i} < 0 & \text{for } s_i < s_{i+1}. \end{cases}$$

We set therefore $\delta_{s_1} = n$ and

$$\delta_{s_{i+1}} = \begin{cases} \delta_{s_i} & \text{for } s_{i+1} < s_i, \\ \delta_{s_i} - 1 & \text{for } s_i < s_{i+1}. \end{cases}$$

It is clear that $1 \leq \delta_i \leq n$. If we choose now $L \geq n$ then $x_i = (i-1)\cdot L + \delta_i$ satisfies $x_i \geq 0$ and $x_i \leq x_{i+1}$ as necessary.

## 5. *Minimal Chromatic Decomposition*

A *chromatic decomposition* of a graph is a decomposition of its vertex set into disjoint independent sets $N = S_1 \cup S_2 \cup \cdots \cup S_k$, where $k$ is the chromatic number of the decomposition. A minimum decomposition is one of minimum $k$. The following algorithm is proposed for finding a minimum chromatic decomposition for a transitive graph $\vec{G}(N, \vec{R})$.

At the $k$th stage we generate a minimum decomposition for the section graph of $\{1, 2, \cdots, k\}$. We denote this decomposition by $D_k = (S_1^k, S_2^k, \cdots, S_{m_k}^k)$ where each $S_i^k$ is an independent set. Clearly $D_1 = (\{1\})$.

Having derived $D_k$, we add vertex $k+1$ to the decomposition by the following rule. Locate the first $S_i^k$ to which vertex $k+1$ can be joined without destroying its independence (i.e. such a set that none of its members is connected to $k+1$). We then add $k+1$ to that first possible recipient. If none exists, a new monochromatic set is generated, containing vertex $k+1$ alone. After the $n$th step we have $N = S_1^n \cup S_2^n \cup \cdots \cup S_{m_n}^n$. It is claimed that this decomposition is a minimum one. For, take any element of $S_{m_n}^n$. The reason that it was not put into $S_{m_n-1}$ when it was introduced is that a previous and therefore smaller member in $S_{m_n-1}$ is connected to it. Consider now this member of $S_{m_n-1}$; it must be connected to some member of $S_{m_n-2}$. We can thus display a monotone decreasing chain of connected elements which by transitivity is a clique, and whose size is $m_n$. It is therefore clear that any chromatic decomposition must have at least $m_n$ sets.

The number of comparisons required by this algorithm is of the order $n^2$. In case of a permutation graph, the process may be further simplified by observing that at the $k$th stage only the last element of each independent set of the decomposition has to be observed. For, in order to test whether a new element can be joined to a particular set, it is sufficient to test connectivity with the last element only. The following example uses this particular simplification for finding minimum decomposition of a permutation graph.

*Example* 5. Consider the graph of Figure 7, where all edges should be directed from low to high. We get:

$$\begin{aligned}
D_1 &= (\{1\}),\\
D_2 &= (\{1\}, \{2\}),\\
D_3 &= (\{1, 3\}, \{2\}),\\
D_4 &= (\{1, 3\}, \{2\}, \{4\}),\\
D_5 &= (\{1, 3\}, \{2\}, \{4, 5\}),\\
D_6 &= (\{1, 3\}, \{2, 6\}, \{4, 5\}),\\
D_7 &= (\{1, 3\}, \{2, 6\}, \{4, 5\}, \{7\}),\\
D_8 &= (\{1, 3\}, \{2, 6\}, \{4, 5\}, \{7, 8\}).
\end{aligned}$$

For illustration, consider the construction of $D_7$ from $D_6$. The vertex 7 is connected to 3, 6, and 5 (it is also connected to 1, 2, and 4, but these we do not have to check since the last elements of the blocks in $D_6$ are 3, 6, and 5); thus 7 cannot be added to any of the blocks of $D_6$ and a block $\{7\}$ has to be constructed. Our conclusion is that the graph of Figure 7 is 4-chromatic, and $D_8$ describes a minimal chromatic decomposition.

In some cases, all we want to know is the chromatic number and we do not insist on specifying a minimal decomposition. In these cases it is sufficient to find a maximum clique, using the technique of the previous section.

Liu in [3] describes a problem of realizing a connection board with the least number of planes. The original connection requirements are prescribed by a bipartite graph. For illustration consider the following example.

*Example* 6. Consider the connection requirements given in Figure 8. To determine the least number of planes in which these connections can be realized, without two connections intersecting in one plane, Liu constructs a new undirected graph in which the vertices correspond to the edges (connections) of the original problem. Two vertices are connected by an edge if and only if the corresponding edges intersect. It is then observed that the chromatic number of the resulting graph is the same as the required number of planes and that a chromatic decom-

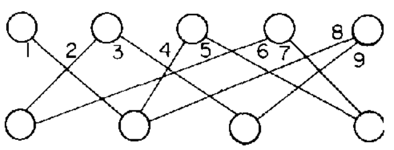

FIG. 8. The connections for Example 6

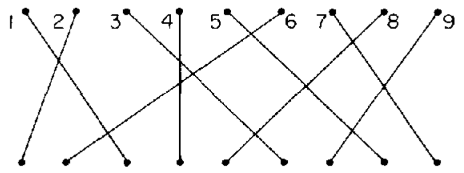

FIG. 9. Figure 8 after "trimming"

position prescribes an assignment of edges to planes. This can be further improved by observing that in essence we have a permutation graph. In our example, simply "trim" the vertices above and below, to obtain the graph shown in Figure 9, in which the endpoints of all edges are disjoint and all edge intersections are preserved. The defining permutation is $P = [2, 6, 1, 4, 8, 3, 9, 5, 7]$. Now, instead of using any of the conventional algorithms for coloration, all of which are comparatively inefficient, we can use the algorithm described earlier in this section:

$$\begin{aligned}
D_1 &= (\{1\}),\\
D_2 &= (\{1\}, \{2\}),\\
D_3 &= (\{1, 3\}, \{2\}),\\
D_4 &= (\{1, 3\}, \{2, 4\}),\\
D_5 &= (\{1, 3, 5\}, \{2, 4\}),\\
D_6 &= (\{1, 3, 5\}, \{2, 4\}, \{6\}),\\
D_7 &= (\{1, 3, 5, 7\}, \{2, 4\}, \{6\}),\\
D_8 &= (\{1, 3, 5, 7\}, \{2, 4, 8\}, \{6\}),\\
D_9 &= (\{1, 3, 5, 7\}, \{2, 4, 8, 9\}, \{6\}).
\end{aligned}$$

Also, the chromatic number of this graph could be determined directly by the maximum clique procedure of Section 3.

Finally, we want to mention a slightly different way of finding a minimal chromatic decomposition of a permutation graph:

The blocks (subsets) are constructed directly from $P$. Let us illustrate on Example 6. The first block contains 2, next is 6 (which is greater than 2 and to its right in $P$ and therefore not connected to it), next is 8, and finally 9. Thus the first block is $\{2, 6, 8, 9\}$. It is clear that we always adjoin the first element which can be adjoined to the present set. The procedure is continued on the remaining sequence: $[1, 4, 3, 5, 7]$. We derive the block $\{1, 4, 5, 7\}$. The last block is $\{3\}$.

## REFERENCES

1. PAULL, M. C., AND UNGER, S. H. Minimizing the number of states in incompletely specified sequential switching functions. *IRE Trans. on Electronic Computers EC-8*, 3 (Sept. 1959), 360.
2. KNUTH, D. E. *The Art of Computer Programming, Volume 1: Fundamental Algorithms.* Addison-Wesley, Reading, Mass., 1968, Sec. 2.2.2.
3. LIU, C. L. *Introduction to Combinatorial Mathematics.* McGraw-Hill, New York, 1968, Ch. 9, Ex. 9-5.
4. PNUELI, A., LEMPEL, A., AND EVEN, S. Transitive orientation of graphs and identification of permutation graphs. *Canad. J. Math. 23* (1971), 160–175.

RECEIVED MARCH 1970; REVISED SEPTEMBER 1971