SIAM J. COMPUT.
Vol. 11, No. 2, May 1982

THE RECOGNITION OF SERIES PARALLEL DIGRAPHS*

JACOBO VALDES,† ROBERT E. TARJAN‡ AND EUGENE L. LAWLER§

Abstract. We present a linear-time algorithm to recognize the class of vertex series-parallel (VSP) digraphs. Our method is based on the relationship between VSP digraphs and the class of edge series-parallel multidigraphs. As a byproduct of our analysis, we obtain efficient methods to compute the transitive closure and transitive reduction of VSP digraphs, and to test isomorphism of minimal VSP digraphs.

Key words. algorithms, complexity, graph decomposition, graph isomorphism, series-parallel graphs, scheduling, transitive closure

1. Introduction. In this paper we study a class of directed acyclic graphs arising in certain scheduling problems. In these problems, the tasks to be scheduled are subject to a partial order. The scheduling problems are NP-complete for an arbitrary partial order but have efficient algorithms if the partial order defines a vertex series-parallel (VSP) digraph ([LAW1], [LAW2], [MON], [SID]). These algorithms apply a "divide-and-conquer" approach to the recursive structure of VSP digraphs.

Our main result is a linear-time algorithm that determines whether an arbitrary digraph is VSP. The algorithm represents the structure of a VSP digraph in a concise form suitable for use by the scheduling algorithms mentioned above. Our method exploits the relationship between VSP digraphs and the class of edge series-parallel (ESP) multidigraphs ([ADA], [DUF], [RIO], [WAL], [WEI]), which arise in the analysis of electrical networks.

Our analysis allows us to prove a simple forbidden subgraph characterization of VSP digraphs. We are also able to give efficient algorithms to compute the transitive closure and transitive reduction of VSP digraphs, and to test two minimal VSP digraphs for isomorphism.

The remainder of this paper is divided into four sections. Section 2 provides the concepts and elementary facts used in the recognition procedure. Section 3 outlines the procedure, proves it correct, and describes in detail a linear-time implementation. Section 4 presents the forbidden subgraph characterization, and § 5 discusses some additional consequences of our work.

2. Basic concepts.

2.1. Graph-theoretic definitions. This section reviews the standard graph-theoretic concepts we shall employ. A *multigraph* $G = \langle V, E \rangle$ consists of a finite set of *vertices* V and a finite multiset of *edges* E. Each edge is a pair (v, w) of distinct vertices. If the edges of G are unordered pairs, then G is an *undirected multigraph*; if the edges are ordered pairs, G is a *directed multigraph* (*multidigraph*). If E is a set,

* Received by the editors May 22, 1980, and in revised form February 4, 1981. A preliminary version of this work was presented at the Eleventh Annual ACM Symposium on Theory of Computing, Atlanta, Georgia, 1979.

† Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey 08544. The work of this author was partially supported by the National Science Foundation under grant MCS-75-22870 and the Office of Naval Research under contract N00014-76-C-0688.

‡ Computer Science Department, Stanford University, Stanford, California 94305. The work of this author was partially supported by the National Science Foundation under grant MCS-75-22870 and the Office of Naval Research under contract N00014-76-C-0688. Present address: Bell Laboratories, Murray Hill, New Jersey 07974.

§ Computer Science Division, University of California at Berkeley, Berkeley, California 94720. The work of this author was partially supported by the National Science Foundation under grant MCS-76-17605.

then G is a *graph*. The terms we define in the remainder of this section for graphs apply equally well to multigraphs.

If (v, w) is an edge of a graph G, then v and w are *adjacent* and (v, w) is *incident* to v and w. If G is a directed graph (*digraph*), then each edge (v, w) is an ordered pair which *leaves* v and *enters* w; v is a *predecessor* of w and w is a *successor* of v. The *degree* of a vertex v in a graph is the number of vertices adjacent to v; a vertex of degree zero is *isolated*. A vertex v in a digraph is a *source* if no edges enter v and a *sink* if no edges leave v.

A *path* of *length* k in a graph is a sequence of vertices $v_0, v_1, \cdots, v_k$ such that (v_i, v_{i+1}) is an edge for $0 \leqq i < k$. If $v_0 = v_k$ and $k \geqq 2$, the path is a *cycle*. A graph which contains no cycles is *acyclic*. The path *contains* vertices $v_0, v_1, \cdots, v_k$ and edges $(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)$, and *avoids* all other vertices and edges.

A directed acyclic graph (*dag*) is *transitive* if for any two vertices v and w such that there is a path from v to w, either $v = w$ or (v, w) is an edge. The *transitive closure* $G_T = \langle V, E_T \rangle$ of a dag $G = \langle V, E \rangle$ is the dag such that $(v, w) \in E_T$ if and only if $v \neq w$ and there is a path from v to w in G.

An edge (v, w) in a dag is *redundant under transitive closure* or simply *redundant* if there is a path from v to w which avoids (v, w). A dag with no redundant edges is *minimal*. The *transitive reduction* of a dag G is the unique minimal dag having the same transitive closure as G (see [AGU]).

The *line digraph* of a digraph G is the digraph $L(G)$ having a vertex $f(e)$ for each edge e of G and an edge $(f(e_1), f(e_2))$ for each pair of edges e_1, e_2 in G of the form $e_1 = (u, v)$, $e_2 = (v, w)$.

A graph $G = \langle V_1, E_1 \rangle$ is a *subgraph* of another graph $G = \langle V_2, E_2 \rangle$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. For any subset S of vertices in a graph G, the subgraph *induced* by S is the maximal subgraph of G with vertex set S. A graph G contains a subgraph *homeomorphic* to a graph H if H can be obtained from G by a sequence of the following operations: (i) remove an edge; (ii) remove an isolated vertex; (iii) if a vertex v has degree two, delete v and replace the two edges (u, v), (v, w) incident to v by an edge (u, w).

2.2. Vertex series-parallel digraphs.

We define the class of VSP dags in terms of the subclass of its minimal members. The dags in this subclass are called *minimal vertex series-parallel* (MVSP) and are defined recursively as follows:

DEFINITION 1 (*minimal vertex series-parallel dags*).

(i) The dag having a single vertex and no edges is MVSP.

(ii) If $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ are two MVSP dags, so are the dags constructed by each of the following operations:

(a) *Parallel composition*: $G_P = \langle V_1 \cup V_2, E_1 \cup E_2 \rangle$.

(b) *Series composition*: $G_S = \langle V_1 \cup V_2, E_1 \cup E_2 \cup (T_1 \times S_2) \rangle$, where T_1 is the set of sinks of G_1 and S_2 is the set of sources of G_2.

We define the class of VSP dags as follows:

DEFINITION 2 (*vertex series-parallel dags*). A dag is VSP if and only if its transitive reduction is MVSP.

Figure 1 shows the construction of an MVSP dag by a sequence of series and parallel compositions. Figure 2 shows a VSP dag whose transitive reduction is the MVSP dag of Fig. 1.

An MVSP dag can be represented in a natural way by a binary tree as shown in Fig. 3. This tree is constructed by (i) associating a tree of one node with the MVSP dag having one vertex and no edges, and (ii) using the rules of Fig. 4 to build larger

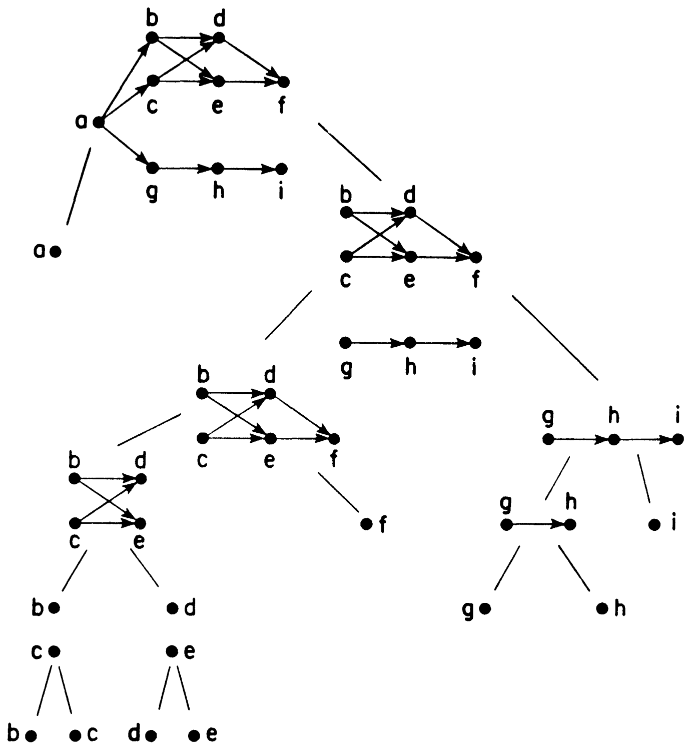

FIG. 1. *Construction of an* MVSP *dag by series and parallel compositions.*

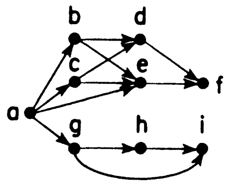

FIG. 2. *A* VSP *dag.*

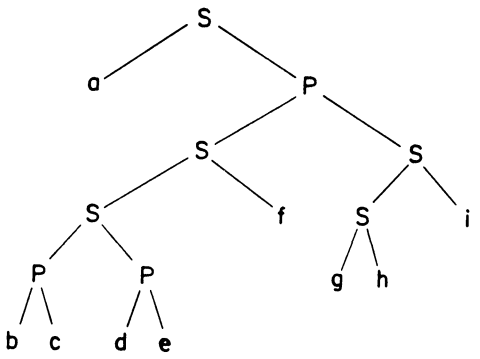

FIG. 3. *Binary decomposition tree representing the* MVSP *dag of Figure* 1.

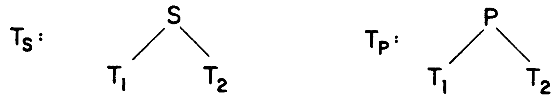

FIG. 4. *Rules to construct T_S and T_P (the binary decomposition trees of G_S and G_P in Definition* 1) *from T_1 and T_2 (the binary decomposition trees of G_1 and G_2).*

trees from smaller ones as the process of building the MVSP dag by series and parallel compositions progresses. We call such a tree a *binary decomposition tree.* Each external node of the tree represents a vertex in the MVSP dag; each internal node is labeled S or P and represents the series or parallel composition of the MVSP dags represented by the subtrees rooted at the children of the node. A binary decomposition tree thus provides a concise description of the structure of an MVSP dag.

Note that several nonisomorphic binary trees may represent the same dag. Since parallel composition is commutative, the children of any P node may be reordered without changing the MVSP dag represented. Furthermore, both series and parallel composition are associative, which allows the ambiguity typical of unparenthesized infix expressions.

Any dag G induces a partial order on its vertices, defined by $v < w$, if and only if there is a path from v to w in G. Since this definition depends only on the paths in G, the partial order induced by the transitive closure of G or by the transitive reduction of G is the same as that induced by G. The partial orders induced by MVSP and VSP dags are of a special type that plays a role in our algorithm.

Any partial order on a set is the intersection of several total orders on the same set; the minimum number of total orders needed to define the partial order in this fashion is the *dimension* of the partial order. For instance, the MVSP dag in Fig. 1 is two-dimensional, since there is a path from v to w in the dag if and only if v appears before w in both of the following total orders; $a\,b\,c\,d\,e\,f\,g\,h\,i$; $a\,g\,h\,i\,c\,b\,e\,d\,f$. Indeed, *any* partial order induced by an MVSP dag is at most two-dimensional. We shall prove this fact in § 3 after describing how the recognition procedure uses it.

2.3. Edge series-parallel multidigraphs. The relationship between MVSP dags and the class of *edge series-parallel* (ESP) *multidigraphs* plays a central role in our recognition algorithm. In this section we review the relevant properties of ESP multidigraphs (sometimes called two-terminal series-parallel multidigraphs). We define the class of ESP multidigraphs recursively as follows:.

DEFINITION 3 (*edge series-parallel multidigraphs*).

(i) A digraph consisting of two vertices joined by a single edge is ESP.

(ii) If G_1 and G_2 are ESP multidigraphs, so are the multidigraphs constructed by each of the following operations:

(a) *Two-terminal parallel composition*: Identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.

(b) *Two-terminal series composition*: Identify the sink of G_1 with the source of G_1.

Figure 5 illustrates the construction of an ESP multidigraph using the operations in Definition 3. Note that every ESP multidigraph is acyclic, since the one-edge ESP multidigraph is acyclic, and the operations of Definition 3 do not create any cycles.

An ESP *multidigraph* is a multigraph formed from an ESP multidigraph by ignoring edge directions. The ESP multigraphs have been extensively studied ([ADA], [DUF], [RIO], [WAL], [WEI]) because of their use in modeling electrical networks. The properties of ESP multidigraphs that we need are simple extensions of known

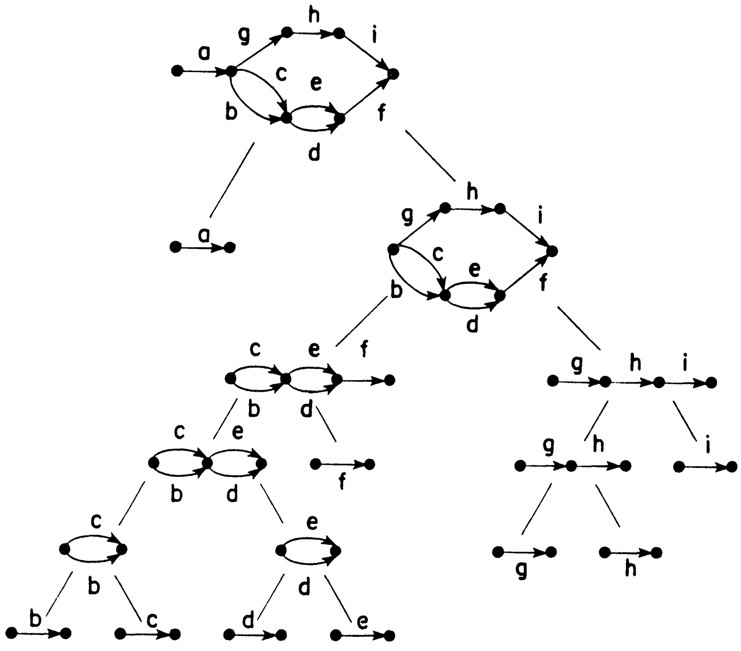

FIG. 5. *Construction of an* ESP *multidigraph by two-terminal series and parallel compositions.*

properties of ESP multigraphs, and we shall provide only summary proofs. A complete description of the relationship between ESP multidigraphs and ESP multigraphs appears in [VAL].

The similarity of Definitions 1 and 3 suggests a vertex-edge duality between MVSP dags and ESP multidigraphs. The following lemma expresses this duality.

LEMMA 1. *An acyclic multidigraph with a single source and a single sink is* ESP *if and only if its line digraph is an* MVSP *dag.*

Proof. By induction on the number of edges in the multidigraph using two facts:

(i) The line digraph of the one-edge ESP multidigraph is the one-vertex MVSP dag.

(ii) The line digraph of the two-terminal series (parallel) composition of G_1 and G_2 is the series (parallel) composition of the line digraph of G_1 and the line digraph of G_2. □

Everything we have said about binary decomposition trees for MVSP dags applies almost verbatim to ESP multidigraphs. In general, if T is a binary decomposition tree of an ESP multidigraph G, then T also represents the corresponding MVSP dag $L(G)$. For instance, the binary decomposition tree in Fig. 3 represents both the ESP multidigraph in Fig. 5 and the MVSP dag in Fig. 1. The external nodes of the tree represent the edges of the ESP multidigraph and the vertices of its line digraph.

The following lemma gives an alternative characterization of ESP multidigraphs based on the reductions in Fig. 6.

LEMMA 2. *A multidigraph is* ESP *if and only if it can be reduced to the one-edge* ESP *multidigraph by a sequence of series and parallel reductions.*

Proof. This lemma corresponds to a result of Duffin [DUF] for ESP multigraphs and follows by an easy induction (on the number of reductions applied for the "if" part, and on the number of edges for the "only if" part). For details see [DUF] or [VAL]. □

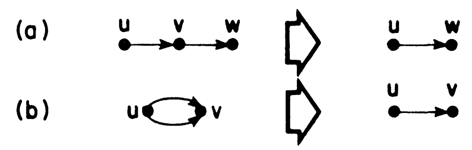

FIG. 6. (a) *Series reduction* (*requires that v have in-degree one and out-degree one*). (b) *Parallel reduction.*

From Lemma 2 we obtain an efficient procedure to recognize ESP multidigraphs. Given a multidigraph G, we repeatedly apply series and parallel reductions until no reduction is possible. If the result is a graph consisting of a single edge, then the original graph G is ESP. If not, G is not ESP. The validity of this procedure depends upon the fact that series and parallel reductions have the *Church–Rosser property* ([ROS], [SET]): if reductions are applied in any order until no reduction is possible, the result is a unique graph independent of the specific reductions applied. Harary, Krarup, and Schwenk [HKS] and Walsh [WAL] prove that the corresponding reduction system for undirected graphs is Church–Rosser; the proof extends easily to the directed case (see [VAL]).

If the reduction process succeeds, we can obtain as a byproduct a decomposition tree of the original ESP multidigraph. We associate a label consisting of a binary tree with each edge of the multidigraph being reduced. Initially the label of each edge is a single-node binary tree. As the reduction process proceeds we use the rules of Fig. 7 to update the edge labels. The label of the last edge remaining after all reductions is the binary decomposition tree of the original multidigraph, as can be proved by an easy induction (see [VAL]).

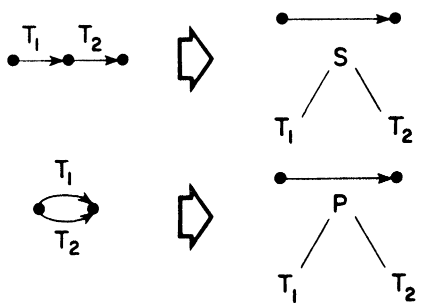

FIG. 7. *Computing the label of an edge introduced by a series or parallel reduction.*

3. The VSP recognition algorithm. We are now able to outline our procedure for recognizing VSP dags and to prove it correct. The input to the algorithm is a dag G. If G is VSP, the algorithm answers YES and produces a binary decomposition tree for G. If G is not VSP, the algorithm answers NO.

Recognition procedure for the class of VSP *dags.*

Step 1. (*Compute the pseudo-transitive reduction of* G.) Given $G = \langle V, E\rangle$, partition E into E_T and E_M such that, if G is VSP, then $G_M = \langle V, E_M\rangle$ is the transitive reduction of G (G_M is thus MVSP). If G is not VSP, G_M may still be MVSP. (We have to pay this price in order to be able to implement this step in linear time, since it is unlikely that a linear-time algorithm exists for transitive reduction of arbitrary dags [AGU].)

Step 2. (*Compute the line digraph inverse of G_M.*) Test whether G_M satisfies a condition (satisfied by all MVSP dags) that guarantees the existence of a line digraph inverse $L^{-1}(G_M)$. If G_M does not satisfy the condition, answer NO and stop. Otherwise, compute a multidigraph $L^{-1}(G_M)$ such that $L(L^{-1}(G_M)) = G_M$. By Lemma 1, G_M is MVSP if and only if $L^{-1}(G_M)$ is ESP.

Step 3. (*Test whether $L^{-1}(G_M)$ is* ESP.) Apply Lemma 2 to determine whether $L^{-1}(G_M)$ is ESP. If not, answer NO and stop. Otherwise compute a binary decomposition tree T for $L^{-1}(G_M)$ as an ESP multidigraph. T is also a decomposition tree for G_M as an MVSP dag.

Step 4. (*Test whether G_M is the transitive reduction of G.*) Use T to compute two total orders whose intersection defines the partial order $<$ on G_M. Use these orders to test each edge in E_T for redundancy. If every edge in E_T is redundant, answer YES, output T, and stop. Otherwise answer NO and stop.

The following argument shows that this procedure is correct. If G is VSP, then G_M will be MVSP and will pass the test of Step 2. If G_M is MVSP, then by Lemma 1 $L^{-1}(G_M)$ will be ESP and will pass the test of Step 3. Step 4 will certify that Step 1 correctly performed the transitive reduction of G and the algorithm will answer YES.

If, on the other hand, G is not VSP, then either G_M as constructed by the algorithm will not be MVSP, or G_M will not be the transitive reduction of G. In the former case the algorithm will answer NO in either Step 2 or Step 3, since by Lemma 1 $L^{-1}(G_M)$ cannot be ESP if G_M is not MVSP. In the latter case the algorithm will answer NO in Step 4.

In order to verify the linearity of the recognition procedure, we must provide more details of the implementation. The remaining parts of this section describe how to implement each step of the algorithm so that it runs in linear time.

3.1. The transitive reduction of VSP dags. Step 1 requires a method to compute the transitive reduction of any VSP dag; the method may do anything to a non-VSP dag. Our method uses the following functions defined on a dag $G = \langle V, E\rangle$.

DEFINITION 4 (*level, jump, and minimum jump functions*). The *level* function L_G is the function from vertices to non-negative integers such that $L_G(v)$ is the length of the longest path from a source of G to v. Note that $L_G(v) = 0$ if v is a source.

The *jump* function J_G is the function from edges to positive integers defined by $J_G((u, v)) = L_G(v) - L_G(u)$.

The *minimum jump* function M_G is the function from vertices to integers defined by $M_G(v) = 0$ if v is a sink of G, $M_G(v) = \min \{J_G((v, w)) \mid (v, w) \in E\}$ if v is not a sink of G.

The following lemmas justify our interest in these functions.

LEMMA 3. *Let G be a dag. If (v, w) is an edge of G that is redundant under transitive closure, then $M_G(v) < J_G((v, w))$.*

Proof. The vertices along any path in G have strictly increasing levels. If (v, w) is redundant, there must be a path of length at least two from v to w. Thus if (v, x) is the first edge on this path then $J_G(c, x) < J_G(v, w)$, so $M_G(v) < J_G(v, w)$. □

LEMMA 4. *If G is* MVSP *then $M_G(v) = J((v, w))$ for every edge (v, w) in G.*

Proof. We prove the lemma by induction on the number of vertices in G. The lemma is immediate if G has one vertex. Suppose that G has $n \geqq 2$ vertices and the lemma is true for MVSP dags with fewer than n vertices.

If $G = \langle V_1 \cup V_2, E_1 \cup E_2 \rangle$ is the parallel composition of $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$, then the level of each vertex in V_1 is exactly the same in G as in G_1, and the level of each vertex in V_2 is the same in G as in G_2. The lemma follows by the induction hypothesis applied to G_1 and G_2.

Suppose on the other hand that $G = \langle V_1 \cup V_2, E_1 \cup E_2 \cup (T_1 \times S_2) \rangle$ is the series composition of $G_1 = \langle V_1, E_1 \rangle$ and $\langle V_2, E_2 \rangle$. Then $L_G(v) = L_{G_1}(v)$ for all vertices $v \in V_1$, and $L_G(v) = L_{G_2}(v) + k + 1$ for all vertices $v \in V_2$, where k is the length of the longest path in G_1 ($k = \max\{L_{G_1}(v) \mid v \in V_1\}$). Thus if $(v, w) \in E_1$, $J_G((v, w)) = J_{G_1}((v, w))$; if $(v, w) \in E_2$, $J_G((v, w)) = J_{G_2}((v, w))$; and if $(v, w) \in T_1 \times S_2$, then $J_G((v, w)) = k + 1 - L_{G_1}(v)$. Since each vertex of G has exiting edges which come entirely from one of the sets E_1, E_2, $T_1 \times S_2$, the lemma for G follows from the induction hypothesis applied to G_1 and G_2. □

The jump and the minimum jump functions are defined in terms of the level function, which in turn is defined in terms of longest paths. Because a longest path cannot contain any redundant edge, the values of these three functions are insensitive to the addition and removal of redundant edges. Combining this fact with Lemmas 3 and 4, we obtain the following corollary.

COROLLARY 1. *Let G be a* VSP *dag and let (v, w) be one of its edges. Then (v, w) is redundant under transitive closure if and only if $M_G(v) < J_g(v, w)$.*

We carry out Step 1 of the recognition procedure by computing L_G, J_G, and M_G, and letting $E_M = \{(v, w) \mid M_G(v) = J_G(v, w)\}$. By Corollary 1 this method correctly performs Step 1. We can compute L_G in linear time by processing the vertices of G in topologically sorted order ([KNU]); the rest of the computation clearly requires only linear time.

3.2. The inverse line digraph of a dag.

Implementing Step 2 of the recognition procedure requires an understanding of line digraph inverses. Several authors have characterized dags having line digraph inverses by using a nonalgorithmic approach ([HN], [KLE]), and Lehot [LEH] has developed a fast algorithm for computing the inverse line graph of an arbitrary undirected graph. However, Lehot's method does not seem to apply to dags, because several nonisomorphic dags may have the same line digraph. See Fig. 8.

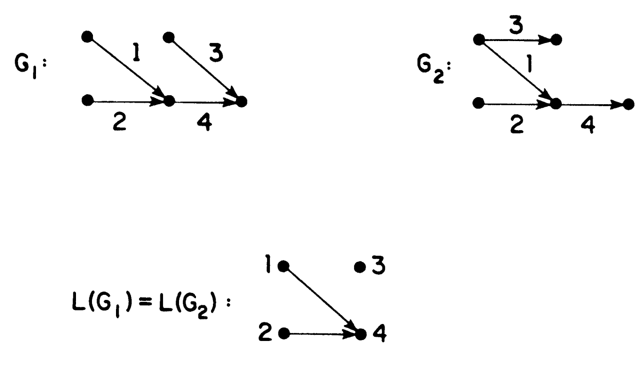

FIG. 8. *Two nonisomorphic multidigraphs that have the same line digraph.*

We compute the line digraph inverse of G_M in two steps. First, we apply a characterization of Harary and Norman [HN] to determine whether G_M has any line digraph inverse. If so, we select a specific inverse as $L^{-1}(G_M)$.

DEFINITION 5 (*complete bipartite composite dags*). A dag G is *complete bipartite composite* (CBC) if there is a set $\{B_1, B_2, \cdots, B_k\}$ of complete bipartite subgraphs of G, called the *bipartite components* of G, such that

(i) every edge of G belongs to exactly one bipartite component;

(ii) for each non-sink vertex v, all edges leaving v belong to the same bipartite component, denoted by $B_H(v)$; and

(iii) for each non-source vertex v, all edges entering v belong to the same bipartite component, denoted by $B_T(v)$.

It is easy to prove that the bipartite components of a CBC dag are unique (see [VAL]). The following lemma gives Harary and Norman's characterization.

LEMMA 5. *A dag has a line digraph inverse if and only if it is* CBC.

Proof. See [HN]. □

In order to select a unique inverse, we use another result of Harary and Norman.

LEMMA 6. *Let G_1 and G_2 be two multidigraphs such that $L(G_1) = L(G_2)$. Let G_i' for $i = 1, 2$ be the multidigraph obtained by merging all sources of G_i into a single source and all sinks of G_i into a single sink. Then G_1' and G_2' are isomorphic.*

DEFINITION 6 (*line digraph inverse*). If G is a CBC dag, then the *line digraph inverse* of G, denoted by $L^{-1}(G)$, is the multidigraph with a single source and a single sink such that $L(L^{-1}(G)) = G$. By Lemma 5, $L^{-1}(G)$ exists. By Lemma 6, $L^{-1}(G)$ is unique.

To use these results in our recognition procedure, we require one more fact.

LEMMA 7. *Every* MVSP *dag is* CBC.

Proof. In the construction of an MVSP dag, new edges are introduced exclusively by series compositions, and each series composition introduces a set of edges forming a complete bipartite subgraph. It is easy to check that these subgraphs satisfy the conditions of Definition 5. □

Lemmas 5–7 give us a way to carry out Step 2. First, we test whether G_M is CBC, as it must be if G_M is MVSP. We perform this test as follows. We select an edge (v, w) that has not yet been assigned to a bipartite component, and we assign it to a new component B_i. We mark all successors of v as belonging to the tail T_i of the component, and we mark all predecessors of w as belonging to the head H_i of the component. We check to see whether G_M contains $\langle T_i \cup H_i, T_i \times H_i \rangle$ as a (complete bipartite) subgraph. If not, G_M is not CBC. It so, we assign all edges in this subgraph to B_i. Then we select a new unassigned edge and repeat the process. We continue until either all edges of G_M are assigned (G_M is CBC), or we attempt to assign an edge to more than one bipartite component (G_M is not CBC), or we mark a vertex as belonging to more than one head or more than one tail (G_M is not CBC). It is straightforward to prove the correctness of this method. It is also not difficult to implement it to run in time linear in the size of the input dag.

Once we have verified that G_M is CBC, we apply the following transformation to compute $L^{-1}(G_M)$. As the vertex set of $L^{-1}(G_M)$, we use $\{B_\alpha, B_1, \cdots, B_k, B_\omega\}$, where $B_1, \cdots, B_k$ represent the complete bipartite components found by the CBC testing method, and B_α and B_ω are two additional vertices. For each vertex v of G_M, we add one edge to $L^{-1}(G_M)$ as follows:

(a) if v is an isolated vertex, we add an edge (B_α, B_ω) to $L^{-1}(G_M)$;

(b) if v is a source but not a sink, we add an edge $(B_\alpha, B_H(v))$ to $L^{-1}(G_M)$;

(c) if v is a sink but not a source, we add an edge $(B_T(v), B_\omega)$ to $L^{-1}(G_M)$;

(d) if v neither a source nor a sink, we add an edge $(B_T(v), B_H(v))$ to $L^{-1}(G_M)$.

This transformation requires linear time given the complete bipartite components of G_M as computed by the CBC testing method. It is routine to verify that $L^{-1}(G_M)$

as constructed by this transformation satisfies $L(L^{-1}(G_M)) = G_M$. We have thus provided a way to carry out Step 2 in time linear in the number of vertices and edges in G_M.

3.3. The recognition of ESP multidigraphs. In section 1.2 we described the algorithm to be used in Step 3 of the recognition procedure: we apply series and parallel reductions until no more are applicable, and we test whether the graph remaining has a single edge. It remains for us to describe how to implement this algorithm so that it runs in linear time. Aho, Hopcroft, and Ullman ([AHU], exercise 5.8), pose a similar problem for undirected graphs but provide no solution to it. Two solutions and their extension to multidigraphs appear in [VAL], and a solution to a special case of the problem appears in [PS]. We shall sketch one of these solutions (suggested by Hopcroft) as applied to the directed case.

We assume that the input multidigraph has a single source and a single sink. We maintain a list of vertices called the *unsatisfied list*. Initially this list contains all vertices except the source and the sink. In general the list contains all vertices other than the source and the sink on which reductions must still be tried. The algorithm repeats the following step until no vertices remain on the unsatisfied list.

General step

(a) Remove some vertex v from the unsatisfied list.

(b) Examine edges entering v. If two edges of the form (u, v) are found, apply a parallel reduction to them. Continue examining edges entering v and applying parallel reductions until either (i) only one edge enters v, or (ii) v is found to have two distinct predecessors.

(c) Examine edges leaving v. If two edges of the form (v, w) are found, apply a parallel reduction to them. Continue examining edges leaving v and applying parallel reductions until either (i) only one edge leaves v, or (ii) v is found to have two distinct successors.

(d) If only one edge (u, v) now enters v and only one edge (v, w) now leaves v, carry out the following steps.

   (i) Apply a series reduction to delete v and replace (u, v) and (v, w) by a new edge (u, w).

   (ii) If u is not the source and not on the unsatisfied list, add it to the unsatisfied list.

   (iii) If w is not the sink and not on the unsatisfied list, add it to the unsatisfied list.

When the unsatisfied list is empty, we test whether any vertices other than the source and sink remain. It so, the multidigraph is not reducible to a single edge. If not, we complete the reduction to a single edge by applying parallel reductions to the edges joining the source and sink.

The following observation guarantees the correctness of this algorithm. Let v be a vertex which is neither the source nor the sink. When v is removed from the unsatisfied list, either v is deleted from the multidigraph or v is verified to have either at least two predecessors or at least two successors. The number of predecessors of v cannot be decreased without adding v to the unsatisfied list. Similarly the number of successors of v cannot decrease without adding v to the list. Now suppose the unsatisfied list is empty and the multidigraph still contains a vertex other than the source and the sink. Then every vertex other than the source and the sink has either at least two predecessors or at least two successors. No number of parallel reductions can change this fact. Thus there is no way to delete additional vertices (by series reduction), and the multidigraph cannot be reduced to a single edge.

In order to implement this algorithm to run in linear time, we maintain for each vertex a (doubly linked) list of the edges entering it and a list of the edges leaving it. Then each edge examination, parallel reduction, and series reduction requires constant time. Suppose the input multidigraph has n vertices and m edges. Each parallel reduction reduces the number of edges by one. Each series reduction deletes a vertex and reduces the number of edges by one. Thus there are at most $m-1$ parallel reductions and at most $n-2$ series reductions. At most $3(n-2)$ additions to the unsatisfied list occur, $n-2$ initially and two for each series reduction. Thus there are at most $3(n-2)$ executions of the general step.

Consider a given execution of the general step. Each edge examination in (b) (except at most two) causes a parallel reduction. Each edge examination in (c) (except at most two) causes a parallel reduction. Thus the total number of edge examinations in (b) and (c) during all executions of the general step is at most $12(n-2)+m-1$. It follows that the entire algorithm requires $O(m+n)$ time. (Each execution of step (a) or (d) takes constant time.)

It is a simple matter to modify the algorithm so that it computes a binary decomposition tree by applying the rules of Fig. 7 as it carries out reductions. Combining two trees which are edge labels into a new tree labeling the edge produced by a reduction takes only constant time; thus the linear time bound is not affected by this modification.

3.4. The two-dimensionality of MVSP dags. In order to complete our implementation of the recognition procedure, we must show that every MVSP dag is two-dimensional, and we must provide a way to compute two total orders whose intersection is the relation $<$ such that $v<w$ if and only if there is a path from v to w in the dag.

We shall regard a total order on a set of n elements as a bijection between the set and $\{1, 2, \cdots, n\}$. Two total orders on a set thus map each element e into a pair of integers (x_e, y_e), which we can interpret as a point in the Cartesian plane. Our problem is thus to map the vertices of an arbitrary MVSP dag into the plane so that there is a path from v to w if and only if $x_v \leqq x_w$ and $y_v \leqq y_w$; i.e., (x_v, y_v) is below and to the left of (x_w, y_w). Figure 9 shows an embedding of the MVSP dag of Fig. 1 which satisfies this criterion.

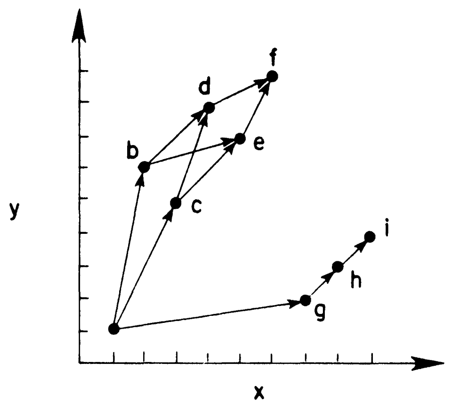

FIG. 9. *Embedding of the* MVSP *dag of Figure* 1 *in the Cartesian plane using the two total orders given in Section* 2.2 *as coordinates.*

We can build up the embedding step-by-step using the constructions of Fig. 10 to deal with series and parallel compositions. If G is formed from G_1 and G_2 by a series composition, there is a path from every vertex of G_1 to every vertex of G_2; in the embedding of Fig. 10, every vertex of G_1 is below and to the left of every vertex in G_2. If G is formed from G_1 and G_2 by a parallel composition, there is no path between G_1 and G_2; in the embedding of Fig. 10, no vertex of G_1 is below and to the right of any vertex in G_2.

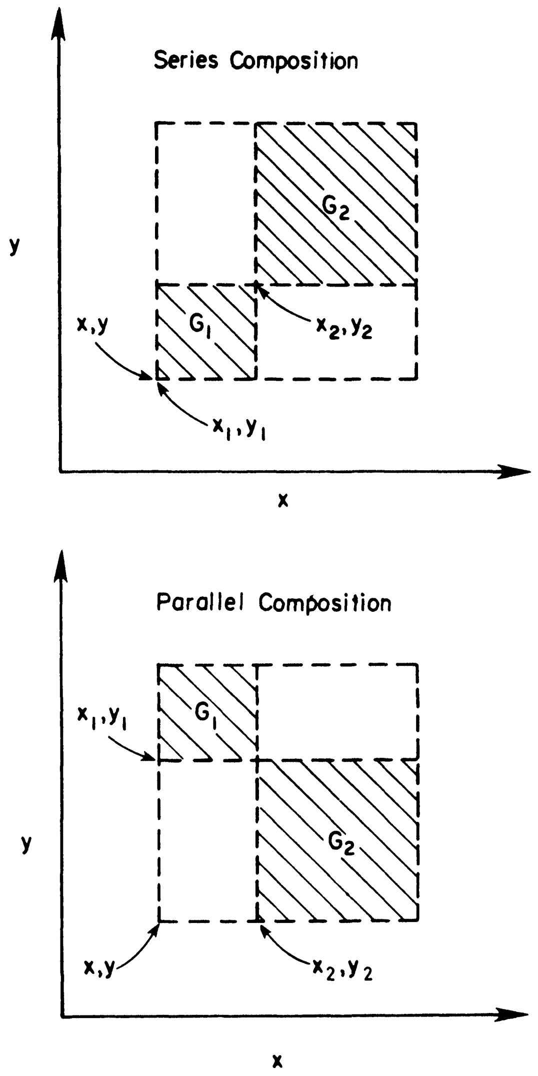

FIG. 10. *Method used to embed an* MVSP *dag in the plane.*

To formally specify this embedding, we shall represent the position of a subgraph by the coordinates of the lower left-hand corner of the smallest square which contains all its vertices. If we let n_1 and n_2 denote the number of vertices in G_1 and G_2, the following formulas provide the positions of G_1 and G_2 given the position of G.

Series composition: $x_1 = x$, $y_1 = y$; $x_2 = x + n_1$, $y_2 = y + n_1$.

Parallel composition: $x_1 = x$, $y_1 = y + n_2$; $x_2 = x + n_1$, $y_2 = y$.

We can compute an embedding in two traversals of the binary decomposition tree T of G. First, we traverse the tree in postorder and assign a *size* to each node; each external node has size one and each internal node has size equal to the sum of the sizes of its children. Next we assign coordinates $(1, 1)$ to the root of the tree. Finally, we traverse the tree in postorder, assigning coordinates to the children of each vertex using the coordinates of the node, the label of the node (S or P), the sizes of the children, and the formulas above. Computing the embedding requires $O(n)$ time, where n is the number of vertices in the MVSP dag. Once we have the embedding, we can test any edge for redundancy in constant time. This provides a linear-time implementation of Step 4, and completes our description of the recognition procedure.

4. Forbidden subgraph characterization.

A common goal in classical graph theory is the characterization of a class of graphs by exhibiting a set of *forbidden subgraphs* such that a graph is in the class if and only if it does not contain any forbidden subgraph. Perhaps the most famous of such results is Kuratowski's characterization of planar graphs ([HAR]). We can provide such a result for VSP dags.

THEOREM 1. *A dag G is* VSP *if and only if its transitive closure does not contain the N graph of Fig.* 11 *as an induced subgraph.*

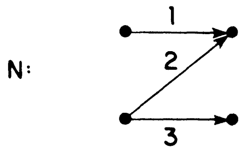

FIG. 11. *The forbidden subgraph for* VSP *dags.*

Proof. (along lines suggested by Peter Avery). The N graph has neither a series nor a parallel decomposition. Furthermore if a transitive dag G contains the N graph as an induced subgraph, and G is composed in either a series or a parallel fashion, then the N graph appears intact in one of the components. For parallel decomposition, this is because no edges join the component, but any partition of the vertices of the N has an edge joining the two blocks of the partition. For series composition this is because every pair of vertices in different components are joined by an edge, but any partition of the vertices of the N produces two vertices in different blocks that are not joined by an edge. Thus, if the transitive closure of a dag G contains an N, G cannot be VSP.

To prove the converse, suppose G is a transitive dag containing at least two vertices and having neither a series nor a parallel decomposition. We want to show that G contains an N. Let s be any source of G and let T be the set of sinks that are successors of s. T is nonempty, or G would have a parallel decomposition into s and the remaining vertices. Let X be the set of vertices with at least one successor in T. We distinguish two cases.

Case 1. There is some vertex $v \notin X \cup T$. Since G has no parallel decomposition, there must be such a vertex v with either a successor or a predecessor in $X \cup T$. Since T contains only sinks, v cannot have a predecessor in T; since $v \notin X$ and G is transitive, v cannot have a successor in $X \cup T$. Thus v has a predecessor, say u, in X. Let t be a successor of u in T. We claim u, v, s, t are the vertices of an N in G.

Certainly, (u, v), (u, t), and (s, t) are edges. Since t is a sink (t, v) is not an edge; since $v \notin X$, (v, t) is not an edge. Since s is a source, neither (u, s) nor (v, s) is an edge.

If v is a sink, (s, v) is not an edge because $v \notin T$. If v is not a sink, there is some sink $w \notin T$ that succeeds v, and (s, v) is not an edge because $w \notin T$ and G is transitive. Finally, since G is transitive and (s, v) is not an edge, (s, u) is also not an edge. Thus u, v, s, t form an N.

Case 2. Every vertex in G is in $X \cup T$. Then T contains all the sinks in G. Let Y be the subset of vertices in X that precede all the sinks. ($Y = \{x \in X \mid$ for all $t \in T$, (x, t) is an edge$\}$.) Y cannot equal X, or G would have a series decomposition into X and T. Let Z be the subset of vertices in $X - Y$ that succeed all the vertices in Y. ($Z = \{x \in X - Y \mid$ for all $y \in Y$, (y, x) is an edge$\}$.) Z cannot equal $X - Y$, or G would have a series decomposition into Y and $(X - Y) \cup T$. Let u be a vertex in $X - Y - Z$. Since $u \in (X - Y - Z)$, there is some vertex $y \in Y$ such that (y, u) is not an edge. Since $u \in X - Y$, there is some vertex $v \in T$ such that (u, v) is not an edge. Finally, since $u \in X$, there is some vertex $t \in T$ such that (u, t) is an edge. We claim y, v, u, t are the vertices of an N in G.

We know (u, t) is an edge. Since $y \in Y$ both (y, t) and (y, v) are edges. Since both t and v are sinks, neither (t, v), (v, t), nor (v, u) is an edge. We know (u, v) is not an edge; by transitivity (u, y) is not an edge. Finally, we know (y, u) is not an edge. Thus y, v, u, t form an N. □

Our VSP recognition procedure can be modified, while preserving its linear time bound, so that it finds an induced N subgraph in any non-GSP dag. The details of this modification appear in [VAL].

Another way to prove Theorem 1 is to apply a forbidden subgraph characterization of Duffin [DUF] for undirected ESP multigraphs. Duffin showed that a multigraph is undirected ESP if and only if it does not contain a subgraph homeomorphic to K_4 (the complete graph on four vertices). A trivial modification of his argument shows that a multigraph with a single source and a single sink is ESP if and only if it does not contain a subgraph homeomorphic to the W dag of Fig. 12. Using this characterization of ESP multidigraphs and Lemma 1, it is not hard to show that the transitive closure of a CBC dag G contains N as an induced subgraph if and only if $L^{-1}(G)$ contains a subgraph homeomorphic to W. (Note that the line digraph of W is the dag in Fig. 13, whose transitive closure contains an induced N.) From this the theorem is immediate.

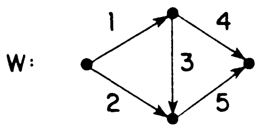

FIG. 12. *The forbidden subgraph for* ESP *multidigraphs.*

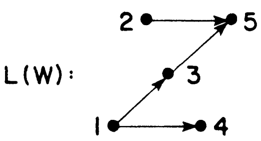

FIG. 13. *The line digraph of the dag of Figure* 12.

5. Remarks.

In this section we mention some additional consequences of our results. Step 1 of the recognition procedure provides a linear-time algorithm to compute the transitive reduction of a VSP dag; Step 4 gives a linear-time method to compute the transitive closure of a VSP dag (in implicit form). These methods are much faster than the best algorithms for arbitrary dags (see [AGU]).

Although several nonisomorphic binary decomposition trees may represent the same MVSP dag, there is a way of modifying these trees to represent MVSP dags in a quasi-unique way: we contract into a single node each connected group of S nodes and each connected group of P nodes. See Fig. 14. The result is a decomposition tree, no longer binary, which is unique up to reordering the children of each P node.

Using these *canonical decomposition trees*, we can test two MVSP dags for isomorphism in linear time by adapting a linear-time tree isomorphism algorithm ([AHU]). The isomorphism problem for VSP dags is as hard as isomorphism of arbitrary graphs; the decomposition tree gives no information about the presence or absence of redundant edges, and we can encode an arbitrary graph into the redundant edges of a VSP dag ([VAL]).

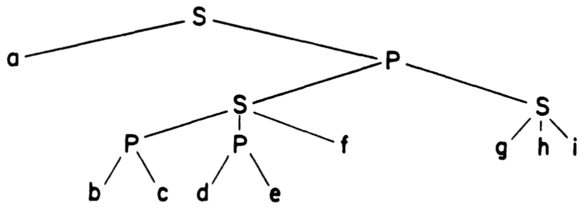

FIG. 14. *Canonical decomposition tree for the* MVSP *dag of Figure* 1.

The subgraph isomorphism problem for MVSP dags is NP-complete, because it contains as a special case the following known NP-complete problem [GJ]: given a (rooted, directed) tree T and a forest (collection of rooted, directed trees) F, determine whether T contains a subgraph isomorphic to F.

Acknowledgment. We thank the referee for finding an error in our original proof of Theorem 1.

REFERENCES

[ADA] A. ADAM, *On graphs in which two vertices are distinguished*, Acta Math. Acad. Sci. Hungary, 12 (1961), pp. 377–397.

[AGU] A. V. AHO, M. R. GAREY AND J. D. ULLMAN, *The transitive reduction of a directed graph*, this Journal, 1 (1972), pp. 131–137.

[AHU] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[DUF] R. J. DUFFIN, *Topology of series-parallel networks*, J. Math. Anal. Appl., 10 (1965), pp. 303–318.

[GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[HAR] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1971.

[HKS] F. HARARY, J. KRARUP AND A. SCHWENK, *Graphs suppressible to an edge*, Canadian Math. Bull., 15 (1971), pp. 201–204.

[HN] F. HARARY AND R. NORMAN, *Some properties of line digraphs*, Rendiconti del Circolo Mathematico Palermo, 9 (1960), pp. 149–163.

[KLE] J. B. KLERLEIN, *Characterizing line dipseudographs*, Proc. Sixth Conference on Combinatorics, Graph theory, and Computing (1975), pp. 429–442.

[KNU] D. E. KNUTH, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

[LAW1] E. L. LAWLER, *Sequencing jobs to minimize total weighted completion time subject to precedence constraints*, Annals of Discrete Math., 2 (1978), pp. 75–90.

[LAW2] E. L. LAWLER, *Sequencing problems with series parallel precedence constraints*, Proc. Conf. on Combinatorial Optimization, Urbino, Italy, 1978, to appear.

[LEH] P. G. H. LEHOT, *An optimal algorithm to detect a line graph and output its root graph*, J. Assoc. Comput. Mach., 21 (1974), pp. 569–575.

[MON] C. L. MONMA AND J. B. SIDNEY, *A general algorithm for optimal job sequencing with series-parallel constraints*, Technical Report No. 347, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, N.Y., 1977.

[PS] B. PRABHALA AND R. SETHI, *Efficient composition of expressions with common subexpresssions*, J. Assoc. Comput. Mach., 27 (1980), pp. 146–163.

[RIO] J. RIORDAN AND C. E. SHANNON, *The number of two terminal series parallel networks*, J. Math. Physics, 21 (1942), pp. 83–93.

[ROS] B. K. ROSEN, *Tree manipulating systems and Church–Rosser theorems*, J. Assoc. Comput. Mach., 20 (1972), pp. 160–187.

[SET] R. SETHI, *Testing for the Church–Rosser property*, J. Assoc. Comput. Mach., 21 (1974), pp. 671–679.

[SID] J. B. SIDNEY, *The two machine flow line problem with series-parallel precedence relations*, Working paper 76-19, Faculty of Management Sciences, University of Ottawa, Ottawa, Ontario, Canada, 1976.

[VAL] J. VALDES, *Parsing flowcharts and series-parallel graphs*, Technical Report STAN-CS-78-682, Computer Science Department, Stanford University, Stanford, California, 1978.

[WAL] T. R. S. WALSH, *Counting labeled three-connected and homeomorphically irreducible two-connected graphs*, unpublished manuscript, 1978.

[WEI] L. WEINBERG, *Linear graphs: theorems, algorithms, and applications*, Aspects of Network and System Theory, R. E. Kalman and N. DeClaris, eds., Holt, Rinehart, and Winston, N.Y., 1971.