The recognition of Series Parallel digraphs

*Jacobo Valdes**
Department of Electrical Engineering and Computer Science
Princeton University
Princeton, N.J. 08540

*Robert E. Tarjan**
Computer Science Department
Stanford University
Stanford, Ca. 94305

Eugene L. Lawler†
Computer Science Division
University of California at Berkeley
Berkeley, Ca. 94720

Abstract: we present an algorithm that recognizes the class of General Series Parallel digraphs and runs in time proportional to the size of its input. To perform this recognition task it is necessary to compute the transitive reduction and transitive closure of any General Series Parallel digraph. Our analysis is based on the relationship between General Series Parallel digraphs and a class of well known models of electrical networks.

1. Introduction

The interest of the directed acyclic graphs that we study in this paper is due to their application to the problem of scheduling under constraints. A number of problems of this type known to be NP-complete when the constraints between the tasks to be scheduled are arbitrary, can be solved efficiently when the constraints form a General Series Parallel (GSP) digraph ([LAW], [MON], [SID]). These efficient algorithms use the simple recursive structure of the GSP constraints in a "divide and conquer" approach.

Our main result is a linear time algorithm that determines whether any given digraph is GSP, and if it is, describes its structure in a concise form suitable to be used by the scheduling algorithms mentioned above. This recognition procedure works by exploiting the relationship between GSP digraphs and the well studied class of Two Terminal Series Parallel (TTSP) multidigraphs ([ADA], [DUF], [RIO], [WAL], [WEI]).

Additionally, our analysis allows us to prove a simple forbidden subgraph characterization of GSP digraphs and design linear time algorithms for the transitive closure and transitive reduction of GSP digraphs as well as for the isomorphism of GSP digraphs that are minimal.

Our work also raises the possibility of the existence of a polynomial time algorithm to solve the subgraph isomorphism problem for transitive and minimal GSP digraphs, and relates this problem to a particular case of the subtree homomorphism problem.

The remainder of this paper is divided into four sections. The first one provides the definitions and elementary facts needed to understand the recognition procedure. In the second, the procedure itself is first outlined and shown correct, and an implementation of it that runs in linear time discussed in detail. The third section presents the forbidden subgraph characterization of GSP digraphs and the last section presents some of the consequences of our work.

* Work supported by NSF Grant MCS-75-22870 and ONR Grant N00014-76-C-0688.

† Work supported by NSF Grant MCS-76-17605.

©1979 ACM 0-89791-003-6/79/0400-001- $00.75

See page ii

2. Basic definitions and relations

2.1. Graph theoretical definitions

Most of the graph theoretical terms used are standard (see [HAR] for instance). We therefore limit ourselves to defining the most commonly used terms and those that may produce confusion.

A *graph* $G=<V,E>$, consists of a finite set of *vertices* V and a finite set of *edges* E. Edges are pairs of distict vertices; if the edges of a graph are unordered pairs the graph is *undirected* and if they are ordered pairs the graph is *directed*. We will abreviate directed graph as *digraph*.

A digraph $G=<V,E>$ is *complete bipartite* if V can be partitioned into H and T so that $E=HxT$. The set H is called the *head* and T is called the *tail* of G.

If the set of edges of a graph may be a multiset, that is, if we allow multiple edges between the same two vertices, the graph is called a *multigraph*. We will abreviate directed multigraph as *multidigraph*. The terms that we define for graphs in the rest of this section can be applied to multigraphs as well.

A vertex v of a digraph G is a *source* if no edge of G enters v. Similarly a vertex v is a *sink* if no edge of G leaves v.

A *path* in a graph (directed or undirected) is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that for all $1<i<n+1$ the pair (v_{i-1}, v_i) is an edge of the graph. If $v_1=v_n$, the path is called a *cycle*. A graph (directed or undirected) that does not contain cycles is called *acyclic*. We will abreviate directed acyclic graph as *dag*.

A dag is *transitive* if it contains an edge (u,v) between any two vertices such that there is a path from u to v. The *transitive closure* of a dag $G=<V,E>$, is the dag $G_T=<V,E_T>$ for which E_T is the minimal subset of VxV that includes E and makes G_T transitive.

An edge (u,v) of a dag is *redundant under transitive closure* or simply *redundant* if there is a path from u to v in the dag that does not include the edge. A dag that does not contain any redundant edge is called *minimal*. The *transitive reduction* of a dag G is the unique minimal dag having the same transitive closure as G.

The *line digraph* of a digraph G is a digraph L(G) that has:

- a vertex f(e) for each edge e of G; and
- an edge $(f(e_1),f(e_2))$ for each pair of edges of G of the form $e_1=(u,v)$, $e_2=(v,w)$.

A graph $G_1=<V_1,E_1>$ is a *subgraph* of another $G=<V,E>$ if V_1 is a subset of V and E_1 is a subset of E. For any subset S of the vertices of a graph G, the *induced* subgraph of S is the maximal subgraph of G with vertex set S. We sat that G contains a subgraph *homeomorphic* to H if H can be obtained from G by a sequence of the following operations:

- removal of an edge;
- replacement of two edges of the form (u,v), (v,w) by an edge (u,w) when v has degree 2.

The assumptions used to analyze our algorithms are standard and can be found in [AHU].

2.2. Minimal Series Parallel digraphs

We define the class of GSP dags in relation to the subclass of its members that are minimal. The dags in this subclass are called *Minimal Series Parallel* (MSP), and are defined recursively as follows:

Definition 1: [Minimal Series Parallel dags]

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

(ii) If $G_1=<V_1,E_1>$ and $G_2=<V_2,E_2>$ are two MSP dags, so is either of the dags constructed by the following operations:

(a) *Parallel composition:* $G_p=<V_1\cup V_2,E_1\cup E_2>$.

(b) *Series composition:* $G_s=<V_1\cup V_2,E_1\cup E_2\cup(N_1 x R_2)>$, where N_1 is the set of sinks of G_1 and R_2 the set of sources of G_2. □

We now define the class of GSP dags as follows:

Definition 2: [General Series Parallel dags]

A dag is GSP if and only if its transitive reduction is a MSP dag. □

Figure 1 shows the construction of a MSP dag by a sequence of series and parallel compositions. Figure 2 shows a GSP dag whose transitive reduction is the MSP dag of fig.1.

A MSP dag constructed by the operations of def.1 can be represented in a natural way by a binary tree as shown in fig.3. This tree has been constructed by (i) associating the trivial tree having one node with the MSP dag having one vertex and no edges, and (ii) using the rules of fig.4 to build larger trees from smaller ones as the process of building the MSP dag by series and parallel compositions progresses.

The result is what we call a *binary decomposition tree:* a binary tree having a leaf for each vertex of the MSP dag it represents, and whose internal nodes are labelled **S** or **P** to indicate respectively the series or parallel composition of the MSP dags represented by the subtrees rooted at the children of the node. Binary decomposition trees provide a concise description of the structure of a MSP dag.

It should be noticed that several non isomorphic binary decomposition trees may represent the same MSP dag. This is due to the symmetry of the parallel composition operation and to the associativity of consecutive series or parallel compositions. The symmetry of parallel compositions makes the left-right ordering of the children of a **P** node irrelevant and the associativity of each of the two operations introduces the ambiguity typical of unparenthesized infix expressions. These characteristics are illustrated in fig.5.

A property of MSP and GSP dags that plays an important role in our recognition procedure, involves the partial order induced by the edges of a MSP dag on the set of its vertices.

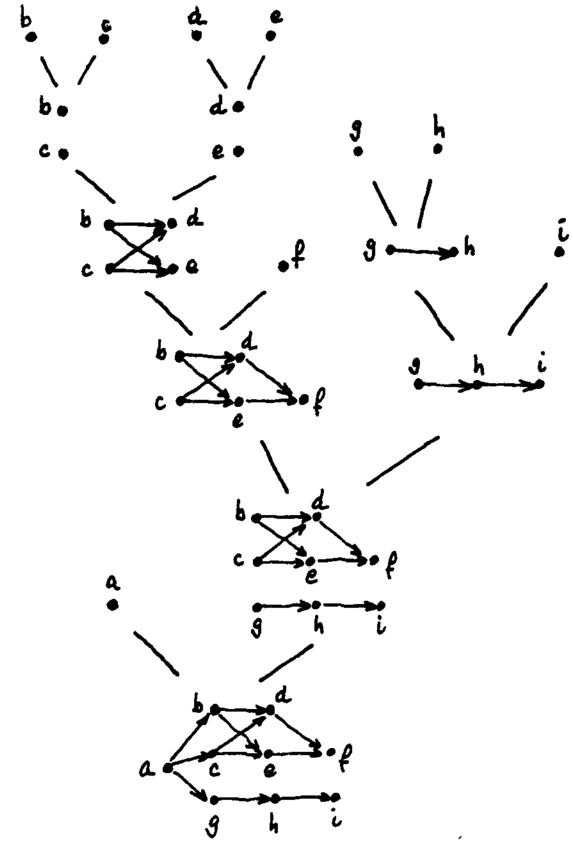

Fig.1
Construction of a MSP dag by series and parallel compositions.

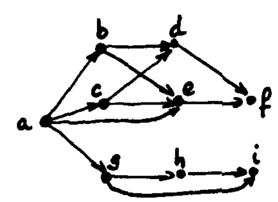

Fig.2
A GSP dag.

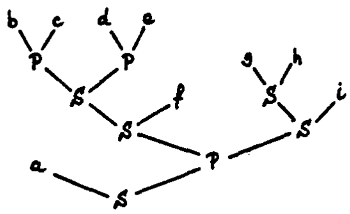

Fig.3
Binary decomposition tree representing the MSP dag of fig.1.

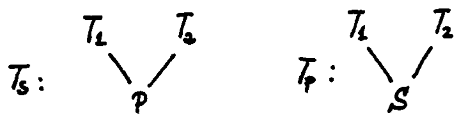

Fig.4
Rules used to construct T_s and T_p (the binary decomposition trees of G_s and G_p of def.1) from T_1 and T_2 (the binary decomposition trees of G_1 and G_2 in the same definition).

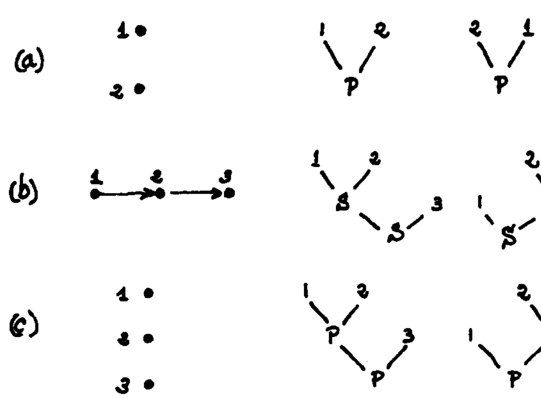

Fig.5
Sources of multiplicity of binary decomposition trees.
(a) Symmetry of parallel compositions.
(b), (c) Associativity of series and parallel compositions.

In general, the binary relation among vertices of a dag G defined by : "*u* → *v if and only if there is a path from u to v in G*" is a partial order. Any partial order on a set can be defined as the intersection of several total orders on the same set, and the minimum number of total orders needed to define the partial order in this fashion is called its *dimension*. As an example, fig.6 shows a MSP dag and two total orders on the set of its vertices. The intersection of the total orders defines the same partial order as the relation "→" described earlier: there is a path from vertex u to vertex v in the dag if and only if u appears before v in the two total orders. Thus the partial order induced by the dag of fig.6 is at most two-dimensional.

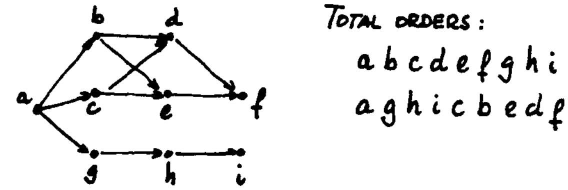

Fig.6
A MSP dag, and two total orders on its vertices whose intersection gives the partial order induced by its edges.

It should be noted that the partial order induced by any dag is the same as the one induced by its transitive closure or its transitive reduction, since the relation "→" is defined in terms of paths between vertices.

The partial order induced by the edges of any MSP dag is at most two-dimensional, that is, it can be obtained as the intersection of at most two total orders. This fact will be proved by describing an algorithm that takes a binary decomposition tree as input and provides two partial orders whose intersection defines the MSP dag represented by the tree. We postpone this description however until a global outline of the GSP recognition procedure in which it is used has been presented.

2.3. Two Terminal Series Parallel multidigraphs

In our recognition algorithm for GSP dags a central role is played by the relationship between MSP dags and the class of Two Terminal Series Parallel (TTSP) multidigraphs. Consequently this section is devoted to the definition of this class and to a review of the relevant properties of its members.

The class of TTSP multidigraphs (named in this fashion because all its members have a single source and a single sink) is defined recursively as follows:

Definition 3: [Two Terminal Series Parallel Multidigraphs]

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

(ii) If G_1 and G_2 are TTSP multidigraphs, so is the multidigraph obtained by either 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_2. □

The construction of a TTSP multidigraph using the operations of def.3 is shown in fig.7. TTSP multidigraphs are obviously acyclic, since the trivial TTSP multidigraph has only one edge, and the operations of def.3 do not create cycles when applied to acyclic multidigraphs.

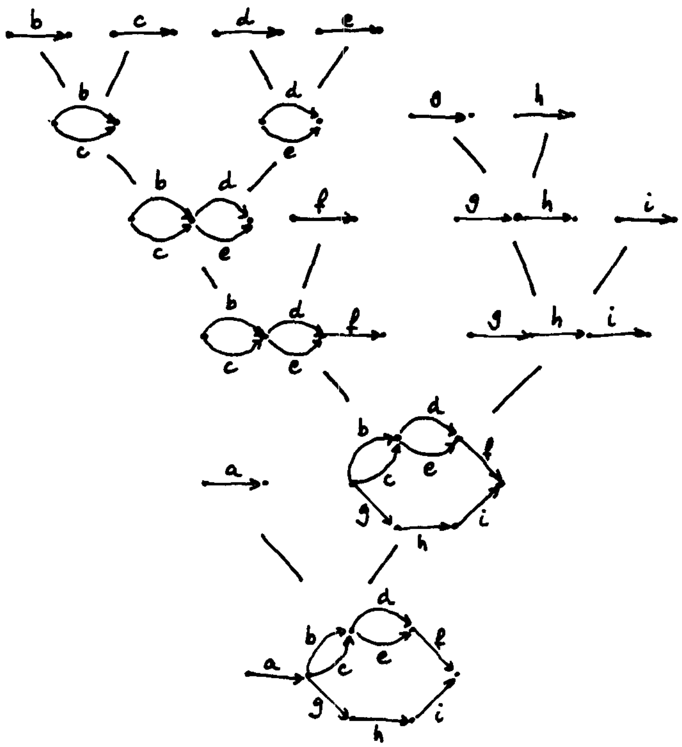

Fig.7
Construction of a TTSP multidigraph by two terminal series and two terminal parallel compositions.

The class of multigraphs containing precisely the undirected versions of all TTSP multidigraphs has been extensively studied ([ADA], [DUF], [RIO], [WAL], [WEI]) because of its relationship with the networks constructed by connection in series or in parallel of electrical components (resistors, capacitors, etc.). The properties of TTSP multidigraphs described in this section are, for the most part, simple extensions of known properties of their undirected versions, and therefore only summary proofs are provided for them. A precise description of the relationship between TTSP multidigraphs and their undirected versions, as well as complete proofs of the properties we describe, can be found in [VAL].

Given the formal similarities between def.3 and def.1, it should come as no surprise that everything said about decomposition trees for MSP dags applies to TTSP multidigraphs almost verbatim. As an example, fig.8 shows the binary decomposition tree corresponding to the construction process of fig.7; note that the decomposition tree has now a leaf for each of the edges of the TTSP multidigraph it represents.

The formal similarity of their definitions suggests also a vertex-edge duality between MSP dags and TTSP multidigraphs. The following lemma shows that this is indeed the case, and relates the two classes through the *line digraph* transformation.

Lemma 1: An acyclic multidigraph with a single source and a single sink is TTSP if and only if its line digraph is a MSP dag.

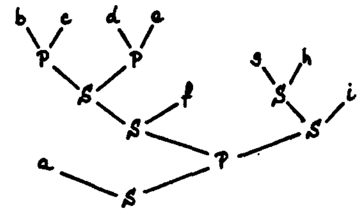

Fig.8
A binary decomposition tree for the TTSP multidigraph of fig.7.

Proof: follows by induction on the number of edges of the multidigraph with the aid of two facts:

(i) the line digraph of the trivial TTSP multidigraph (two vertices joined by a directed edge) is the trivial MSP dag (one vertex and no edges),

(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. □

A further consequence of the relation given by (i) and (ii) in the above proof is that if T is a binary decomposition tree of a TTSP multidigraph G, and we regard it as the binary decomposition tree of a MSP dag, then T represents the line digraph of G. As an example, it is trivial to test that the line digraph of the TTSP multidigraph of fig.7 is the MSP dag of fig.1 and that both can be represented by the same binary decomposition tree (shown in fig.3 and fig.8).

Another important characterization of TTSP multidigraphs based on the reductions shown in fig.9 is given by the following lemma:

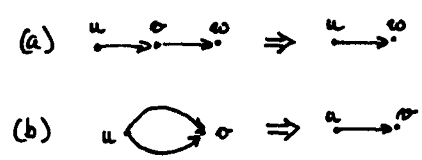

Fig.9
(a) Series reduction. (b) Parallel reduction.

Lemma 2: A multidigraph is TTSP if and only if it can be reduced to the trivial TTSP multidigraph (two vertices joined by a single edge) by a sequence of series and parallel reductions.

Proof: This lemma is a trivial generalization of the results of Duffin [DUF] for undirected TTSP multigraphs, and can be established by an easy induction (on the number of reductions applied for the "if" part, and on the number of edges for the "only if"). The details can be found in [VAL] or [DUF]. □

This characterization is the basis of an efficient algorithm to recognize the class of TTSP multidigraphs that we will use later on as part of our recognition procedure for GSP dags: to test whether a multidigraph is TTSP we repeatedly apply series and parallel reductions to it until no more reductions are possible, and then test whether the remaining digraph consists of a single edge.

Lemma 2 is not sufficient however to guarantee that the recognition procedure just outlined will provide the correct answer. The lemma does indeed say that we will succeed in reducing the multidigraph to a single edge only if it is TTSP. Nevertheless the lemma does not guarantee that we will succeed in reducing a TTSP multidigraph by applying to it arbitrarily selected series and parallel reductions and only states that there exists at least one sequence of such reductions that will reduce the multidigraph.

Fortunately, the reduction system that we are using has a property — known as the *Church-Rosser property* — that guarantees that the characteristic of being reducible to a single edge is preserved by the application of any series or parallel reduction. We can therefore carry out any applicable reduction at any point without fear of hurting our chances of ultimately reducing the multidigraph to a single edge.

Symbol manipulation systems possessing the Church-Rosser (CR) property are useful in many areas of Mathematics and Computer Science, and several sufficient conditions for a system to posses this property are known ([ROS], [SET]). Using these sufficient conditions it is simple to prove that the reduction system consisting of series and parallel reductions has the CR property. The proof requires however a good deal of background irrelevant for our purposes and is omitted (see [HKS] or [WAL] for a proof of the CR property of the undirected version of our reduction system that can be easily generalized to the directed case.)

Just as important for our purposes as the simplicity of the recognition algorithm for TTSP multidigraphs described, is the fact that a binary decomposition tree of the multidigraph being reduced can be obtained as a byproduct of the reduction process.

In order to obtain the decomposition tree, we associate a label with each edge of the multidigraph being reduced. Initially the label of each edge is a trivial binary tree consisting of a single node. As the reduction process introduces new edges we use the rules of fig.10 to compute the binary trees used to label them.

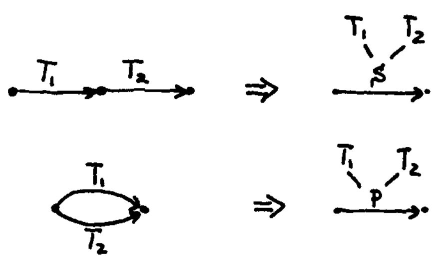

Fig.10
Computing the label of a new edge introduced by a series or parallel reduction.

The binary decomposition tree of the initial multidigraph is obtained as the label of the only remaining edge after the reduction, a fact that can be proved by an easy induction that we omit (see [VAL]). An example of this process is shown in fig.11.

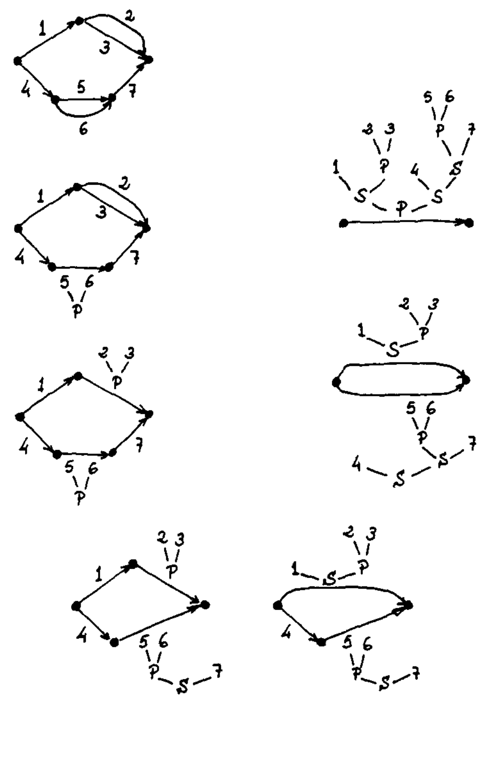

Fig.11
Example of how a binary decomposition tree of a TTSP multidigraph can be obtained from the reduction process.

3. The GSP recognition algorithm

We have finally collected enough facts to be able to outline our procedure to recognize the class of GSP dags and provide a proof of its correctness.

Algorithm 1: [Recognition procedure for the class of GSP dags]

Input: a dag G.
Output: YES if G is GSP, NO otherwise.

Step 1: *Pseudo transitive reduction of G.* Given $G=<V,E>$, partition E into E_T and E_M so that if G is GSP, then $G_M=<V,E_M>$ is its transitive reduction (and therefore MSP). If G is not GSP, G_M may be MSP (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 transitive reduction algorithm exists for arbitrary dags [AGU]).

Step 2: *Compute the line digraph inverse of* G_M. Test whether G_M satisfies a sufficient condition (satisfied by all MSP dags) for having a line digraph inverse $L^{-1}(G_M)$. If G_M does not satisfy this condition we answer NO and stop, otherwise we compute $L^{-1}(G_M)$ so that G_M is MSP if and only if $L^{-1}(G_M)$ is TTSP (lemma 1).

Step 3: *Test whether* $L^{-1}(G_M)$ *is TTSP* using the characterization of lemma 2. If $L^{-1}(G_M)$ is TTSP compute a binary decomposition tree T for it, otherwise answer NO and stop. According to what we said earlier, T is a decomposition tree of $L^{-1}(G_M)$ as a TTSP multidigraph and of G_M (its line digraph) as a MSP dag.

Step 4: *Test whether* G_M *is the transitive reduction of* G. That is, test that the edges in E_T belong to the transitive closure of G_M. If they do, answer YES and output T, otherwise answer NO and stop. This step will be performed by using T to compute two total orders on V whose intersection defines the partial order $\rightarrow$ on G_M, then using them to test, for each edge (u,v) of E_T, whether there is a path from u to v in G_M by testing whether u appears before v on both total orders. □

We can prove this procedure correct by the following argument.

If G is GSP, then G_M will be MSP and will satisfy the test of Step 2. If G_M is MSP, according to lemma 1 $L^{-1}(G_M)$ will be TTSP and thus will satisfy the test of Step 3. Step 4 will simply certify that Step 1 performed the transitive reduction of G and the algorithm will answer YES.

If, on the other hand, G is not GSP we have two possibilities: either G_M is not MSP or it is not the transitive reduction of G. In the first case the algorithm will answer NO in either Step 2 or Step 3, since according to lemma 1 $L^{-1}(G_M)$ cannot be TTSP if G_M is not MSP, and in the second case the algorithm will answer NO in Step 4.

In either case the algorithm produces the right answer, and we conclude that it recognizes the class of GSP dags as claimed.

Unfortunately, the above description of the algorithm is far from being precise enough to establish the linear upper bound on its running time that we want. We will therefore devote the rest of this section to providing enough details about its implementation so this linear bound can be established.

3.1. The transitive reduction of GSP dags

We will now describe how to implement Step 1 of the GSP recognition algorithm so it runs in a number of steps that grows linearly with the size of the input dag. Remember that we want a procedure that computes the transitive reduction of GSP dags and may do anything on a dag that is not GSP.

Consider the following functions defined on a dag G with n vertices:

The *layer* function: L_G: $V \rightarrow \{0,1,2,...,n-1\}$.
$L_G(v)=0$ if v is a source, otherwise the length of the longest path from a source of G to v. □

The *jump* function: J_G: $E \rightarrow \{1,2,...,n-1\}$.
$J_G((u,v))=L_G(v)-L_G(u)$. □

The *minimum jump* function M_G: $V \rightarrow \{0,1,2,...,n-1\}$.
$M_G(v)=0$ if v is a sink of G, otherwise the minimum value of J_G over all edges that leave v. □

Figure 12 shows the values of these three functions for the MSP dag of fig.1.

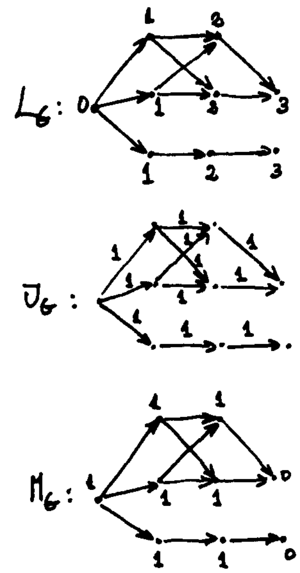

Fig.12
Values of L_G, J_G and M_G for the MSP dag of fig.1.

Our interest in these fuctions is due to the following facts:

Lemma 3: Let G be a dag. For any edge (u,v) of G that is redundant under transitive closure $M_G(u) < J_G((u,v))$.

Proof: Because G has no multiple edges, the path from u to v not including (u,v) has to have at least two edges. Let (u,x) be the first edge on that path; by definition, the values of L_G must increase along any path in G, and there is a path from x to v therefore $L_G(v) > L_G(x)$. By definition $J_G((u,v)) > J_G((u,x))$ and the prosition must be true since $M_G(u)$ cannot be greater than $J_G((u,x))$. □

Lemma 4: If G is MSP then $M_G(u)=J_G((u,v))$ for any edge (u,v) of G.

Proof: We prove the proposition by induction on the number of vertices of G.

If G has one vertex, the proposition is trivially true; otherwise let the proposition hold for all MSP dags with fewer than *k* vertices, and let G be the series or parallel composition of G_1 and G_2, each having at most *k-1* vertices.

We discuss in detail only the case when G is the series composition of G_1 and G_2 since the analysis of the other case is quite similar.

When G is the series composition of G_1 and G_2 there are three possibilities: (i) (u,v) is an edge of G_1, (ii) (u,v) is an edge of G_2, and (iii) (u,v) joins a sink of G_1 to a source of G_2.

When (u,v) is an edge of G_1 the proposition follows immediately from the induction hypothesis and the fact that $J_G((u,v))=J_{G_1}((u,v))$ for all edges of G_1 (this is a trivial consequence of the fact that $L_G(v)=L_{G_1}(v)$ for all vertices of G_1 which in turn follows directly from the definitions of the layer function and series composition).

Let now (u,v) be an edge of G_2 and q be the length of the longest path in G_1. This path has to end in a sink of G_1 and therefore, by definition of the layer function, $L_G(x)=L_{G_2}(x)+q+1$ for any vertex x of G_2. Because J_G is defined by the difference of two layer values, this implies $J_G(e)=J_{G_2}(e)$ for any edge e of G_2; from this fact and the induction hypothesis the proposition follows trivially.

Finally, if (u,v) joins a sink of G_1 to a source of G_2 we know that $L_G(y)=q+1$ for any source y of G_2. Since any edge e leaving a sink u of G_1 must enter a source of G_2 it must be that $J_G(e)=q+1-L_G(u)$ and therefore $M_G(u)=J_G(e)$ for all edges leaving u. From this fact the proposition follows trivially once again. □

The jump and minimum jump functions were defined in terms of the layer function, which in turn was defined in terms of longest paths in a dag. Because a path of this type cannot contain edges that are redundant, the values of these three functions on a dag are insensitive to the addition or removal of redundant edges. As an example, it is trivial to test that the values given in fig.12 for the MSP dag of fig.1 are identical to the values that one would obtain for the GSP dag of fig.2.

This fact together with lemmas 3 and 4 directly implies the following:

Corollary 1: Let G be a GSP dag and (u,v) one of its edges. The edge (u,v) is redundant under transitive closure in G if and only if $M_G(u) < J_G((u,v))$. □

As a consequence, we know that it is enough to compute the values of the jump and minimum jump functions to perform the transitive reduction of a GSP dag. Because these two functions can be trivially computed from the values of the layer function, and the layer values can be computed by a trivial modification of the topological sort algorithm ([KNU]), we can implement Step 1 of the GSP recognition procedure to run in O(n+m) steps for a dag with n vertices and m edges.

3.2. The inverse line digraph of a dag

We now consider the problem of implementing Step 2 of the recognition procedure.

The problem of characterizing the dags that have line digraph inverses has been studied from a non-algorithmic point of view by several authors ([HN], [KLE]), and the problem of computing the inverse line graph for an arbitrary graph has been solved by Lehot [LEH].

Unfortunately Lehot's approach does not work for dags mostly because several nonisomorphic multidigraphs may have the same line digraph, as shown in fig.13.

We will solve the problem in two steps: first we use a characterization due to Harary and Norman [HN] to determine whether the dag has a line digraph inverse, and, once we know that it does, we then compute a specific line digraph inverse out of the several possible ones.

Definition 4: [Complete Bipartite Composite dags]

A dag G is Complete Bipartite Composite (CBC) if there exists a set of complete bipartite subgraphs of G: $B_1,B_2,...,B_k$, called the *bipartite components* of G, such that:

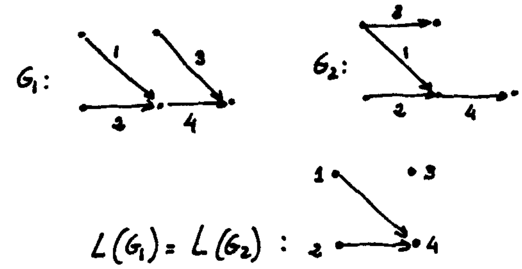

Fig.13
Two nonisomorphic multidigraphs that have the same line digraph.

(a) each edge of G belongs to exactly one bipartite component;

(b) every vertex v of G, except the sinks, belongs to the head of exactly one bipartite component denoted h(v);

(c) every vertex v of G, except the sources, belongs to the tail of exactly one bipartite component denoted t(v). □

It is a trivial exercise to prove that the bipartite components of a CBC dag are unique (see [VAL]).

The first part of the characterization we seek is given by the following lemma:

Lemma 5: A dag has a line digraph inverse if and only if it is CBC.

Proof: See [HN]. □

This lemma solves the question of whether a dag has a line digraph inverse, but says nothing about the multiplicity of inverses mentioned earlier. Fortunately Harary and Norman provide the answer to this problem as well:

Lemma 6: Let G_1 and G_2 be two multidigraphs such that $L(G_1) = L(G_2)$. The multidigraphs obtained from G_1 and G_2 by merging the sources into a single source and the sinks into a single sink are isomorphic.

Proof: Harary and Norman [HN] prove that the inverse line digraph is unique if the sources and sinks are deleted instead of merged. The modification of their argument to prove our lemma is trivial and is omitted. □

From now on any mention of the line digraph inverse $L^{-1}(G)$ of a CBC dag G, will refer to the unique multidigraph having a single source and a single sink whose line digraph is G.

These results would be irrelevant for our purposes but for the following fact:

Lemma 7: Every MSP dag is CBC.

Proof: In the construction of a MSP dag by series and parallel compositions new edges are introduced exclusively by series compositions, and each series composition introduces edges that form a complete bipartite subgraph of the complete MSP dag. It is trivial to check that the subgraphs defined by the series compositions satisfy the conditions of def.4 and are therefore the unique bipartite components of the MSP dag. □

We have therefore solved the first part of our task: we have found a property (being CBC) satisfied by all MSP dags that is a sufficient condition for a dag to posses an inverse line digraph. We will now complete our task by showing (i) how to test a given dag for this property and (ii) how to compute its line digraph inverse in a number of steps proportional to the size of the dag.

We can test whether a dag is CBC as follows. We select an edge (u,v) of the dag that has not been assigned to a bipartite component yet and assign it to a new bipartite component B_i. We now mark all the predecessors of v as belonging to the head, and all the succesors of u as belonging to the tail of B_i. We then test whether there is a complete bipartite subgraph of the dag with the head and tail just identified; if no such subgraph is found, the dag is not CBC. We continue the process by selecting a new edge and repeating the operation until no edge remains unassigned. While performing this process, we decide that the dag is not CBC if we ever attempt to assign an edge to more than one bipartite component, or mark a vertex as belonging to more than one head or one tail.

Because of the uniqueness of the bipartite components, this process will identify a new component every time an unassigned edge is selected and processed as described above. Therefore, by assigning all edges to components, this process proves that the input dag was CBC by identifying its bipartite components.

Because the implementation of this procedure to run in a number of steps proportional to the size of the input dag is a trivial exercise in data structures, we find ourselves closer to our immediate goal of implementing Step 2 of the GSP recognition procedure in linear time. We therefore proceed to consider the remaining problem: computing the inverse line digraph of a CBC dag.

Consider the following transformation of a CBC dag:

Definition 5: [The inverse line digraph of a CBC dag]

Let G be a CBC dag with bipartite components $B_1,B_2,...,B_k$. The vertex set of $L^{-1}(G)$ is $\{B_\alpha,B_1,B_2,...,B_k,B_\omega\}$ and its edge set has an edge for each vertex of G defined as follows:

(a) for each source v of G, $L^{-1}(G)$ has an edge $(B_\alpha,h(v))$;

(b) for each sink v of G, $L^{-1}(G)$ has an edge $(t(v),B_\omega)$;

(c) for each vertex v that is both a source and a sink of G, $L^{-1}(G)$ has an edge (B_α,B_ω); and

(d) in all other cases, the edge of $L^{-1}(G)$ that corresponds to vertex v of G is $(t(v),h(v))$. □

The name given to this transformation is justified by the following property:

Lemma 8: For any CBC dag, $L(L^{-1}(G))=G$.

Proof: For each vertex of G, $L^{-1}(G)$ has an edge, and for each edge of $L^{-1}(G)$ there is a vertex in $L(L^{-1}(G))$ according to the definition of the line digraph. This establishes a one to one relationship between the vertex sets of G and $L(L^{-1}(G))$. The inverse line digraph transformation was defined so that there is an edge between any two vertices of G if and only if there is an edge between the corresponding vertices of $L(L^{-1}(G))$. □

The algorithm sketched earlier to test whether a dag is CBC actually computed the bipartite components of the dag being tested, and given these components it is trivial to compute the inverse line digraph as given by the above definition. Since the line digraph inverse has an edge for each vertex of the CBC dag from which it originates, it should be clear that we have described a procedure to compute the line digraph inverse of a CBC dag G in time proportional to the size of G.

Furthermore, the line digraph inverse of any CBC dag has a single source and a single sink (B_α and B_ω respectively) so it follows from lemmas 1 and 8 that the line digraph inverse of a CBC dag G is a TTSP multidigraph if and only if G is a MSP dag.

We have thus achieved the goal of implementing Step 2 of our recognition procedure so it runs in linear time.

3.3. The recognition of TTSP multidigraphs

The algorithm to be used in Step 3 has already been described in section 1.2: apply series and parallel reductions to the multidigraph given until no more reductions are possible, and then test whether the remaining digraph consists of a single edge. Thus our only task here is to show that this method can be implemented to run in time proportional to the size of the given multidigraph.

The same problem for undirected graphs is suggested as an exercise in [AHU] (exercise 5.8), but unfortunately no solution is presented for it. A detailed discussion of two solutions of this exercise can be found in [VAL] together with their generalization to directed multigraphs. Therefore the description that follows has been reduced to a minimum.

The basic data structure is a list of vertices that we call the *unsatisfied list.* Initially this list includes all vertices of the input multidigraph except the source and the sink, and in general it will contain all the vertices on which some work has to be performed (except the source and sink, which are never added to it).

The algorithm proceeds by removing any vertex v from this list and performing as many parallel reductions on the edges incident to it as it is possible before either leaving the vertex with a single entering edge and a single exiting edge, or discovering that the vertex still has at least two distinct predecessors or two distinct succesors. In the first alternative, the vertex is removed by a series reduction and the two vertices adjacent to it added to the unsatisfied list if they are not there already. This process is repeated until the unsatisfied list becomes empty, at which point the same process is applied to the source and the sink (in order to eliminate any multiple edges between them) before stopping.

We can prove that this method will correctly recognize the class of TTSP multidigraphs using the characterization of lemma 2 as follows. The unsatisfied list becomes empty, either because all vertices (except source and sink of course) have been deleted by series reductions or because every remaining vertex has two distinct predecessors or two distinct successors. In the first case the multidigraph has been reduced (except for possible multiple edges between the source and the sink which will be deleted in the last step) and in the second no vertex can be eliminated by a series reduction until some other vertex is eliminated, which clearly implies that no more vertices can ever be deleted.

The running time of this procedure cannot be analyzed unless we look more closely at the processing of each vertex deleted from the unsatisfied list. Let us assume tnat each vertex has two lists of pointers to edges associated with it. One list contains pointers to all the edges entering the vertex, while the other contains pointers to all the edges leaving the vertex. The processing of a vertex consists of applying to these two lists the following algorithm:

```
while size of the list is greater than one do
   if either of the first two elements points to a deleted edge then
      delete the pointer from the list
   elseif the first two elements point to edges with the same endpoints then
      carry out a parallel reduction and delete the pointers
   else exit
end;
```

Clearly the processing of a vertex terminates when each of its two lists has either a single element or contains pointers to edges with different endpoints. If appropriate data structures are used, this process can be implemented so it takes a constant number of steps every time the process is initaited plus a (different) constant number of steps for every pointer deleted.

We will therefore be able to guarantee a linear time upper bound on the running of the total reduction process if we prove that (a) a linear number of vertices are processed (i.e., deleted from the unsatisfied list) and (b) the total number of pointers to edges deleted is linear.

In a multidgraph with n vertices and m edges, we will have n-2 elements in the unsatisfied list initially. New vertices are added to this list only after a series reduction is performed, an operation that decreases the total number of vertices of the multidigraph by one. Thus at most n-2 series reductions can be performed and no more than 2(n-2) additions to the unsatisfied list will occur, since at most two vertices are added for each reduction.

Initially, we will have a total of 2m pointers to edges in all the lists associated with the vertices since a pointer to (u,v) will appear in the list of edges entering v and the in the list of edges leaving u. New edges, and therefore new pointers, are added by parallel reductions as the algorithm progresses, but since each of these reductions decreases the total number of edges of the multidigraph by one, no more than m-1 of them could possibly be performed and no more than 2(m-1) new pointers introduced. Thus a total of no more than 2m+2(m-1) pointers to edges will be manipulated.

One more problem has to be considered: we want to obtain the decomposition tree of the multidigraph being reduced so we have to compute the labels for the new edges using the rules of fig.10. Clearly any reasonable implementation of this computation will not construct the new label from scratch, but will instead combine the labels of the edges being deleted. In this fashion each new label can be computed in a constant amount of time.

This completes our argument, and we conclude that Step 3 of the GSP recognition procedure can also be implemented to run in time proportional to the number of vertices and edges of its input.

3.4. The two dimensionality of MSP dags

This section completes our description of the implementation of the GSP recognition procedure by showing how Step 4 can be implemented in linear time.

It is useful to remember the task to be performed: given a binary decomposition tree of a MSP dag, we want to compute two total orders on the set of its vertices whose intersection defines the same partial order as the edge set of the dag, that is, two total orders such that for any two vertices of the dag u,v there is a path from u to v if and only if u appears before v in both total orders.

Let us regard a total order on a set of n elements as a one-to-one correspondence between the set and {1,2,...,n}. Thus, given two total orders on a set, we can consider them as assigning two integers to each of the elements of the set, and regard this pair of integers as cartesian coordinates of the element. In this fashion an intuitive correspondence can be established between the two total orders whose intersection defines a MSP dag and an embedding of the MSP dag in the cartesian plane in which the coordinates of any pair of its vertices u,v satisfy the relationship $x_v > x_u$ and $y_v > y_u$ if and only if there is a path from u to v in the dag. As an example fig.14 shows the embedding of the MSP dag of fig.6 resulting from interpreting in this fashion the two total orders given in the same figure. We will use this interpretation in the discussion that follows.

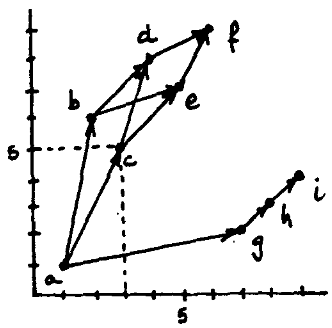

Fig.14
Embedding of the MSP dag of fig.6 in the plane using the two total orders of the same figure as coordinates.

The first observation we make, is that an MSP dag with n vertices can be embedded in an nxn square of the cartesian plane, since the integers assigned to its vertices are in {1,2,...,n}. Knowing this fact, we can use the approach shown in fig.15 to reduce the problem of embedding a MSP dag G to that of embedding two smaller MSP dags, G_1 and G_2, whose series or parallel composition produces G. A look at that figure should convince the reader that for any pair of vertices, $u \in G_1$ and $v \in G_2$, there is a path from u to v if and only if both coordinates of u are smaller than the corresponding coordinates of v, i.e., only in the case of the series composition.

Clearly this approach can be applied recursively to reduce the problem of embedding an MSP dag with n vertices to the n trivial problems of embedding the MSP dag with one vertex and no edges at a specific location of the plane.

To complete the details of how this process may be performed, let us assume that the position of the embedding of a MSP dag with n vertices in the cartesian plane is given by the coordinates of the lower left corner of the nxn square that contains all its vertices. With this convention, if we let n_1 and n_2 denote the number of vertices of G_1 and G_2 in fig.15, the following formulae will provide the positions of G_1 and G_2 given n_1, n_2 and the position (x,y) of G:

Series composition:

$$x_1 = x$$
$$y_1 = y$$
$$x_2 = x + n_1$$
$$y_2 = y + n_2$$

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

The correctness of these formulae can be established by inspection of fig.15.

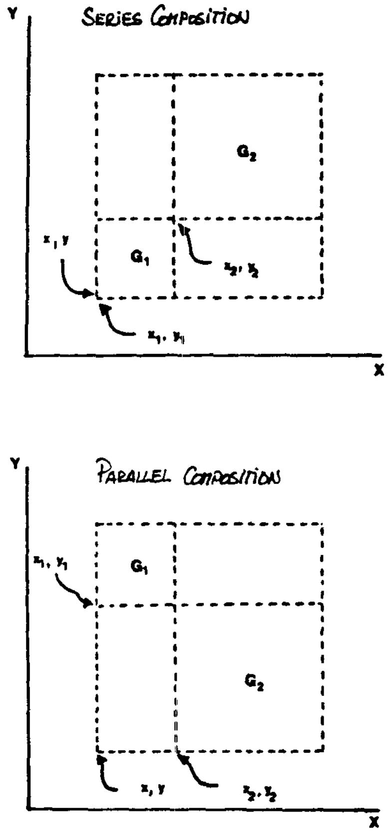

Fig.15
Method used to embed a MSP dag in the plane so the coordinates of its vertices give two total orders whose intersection defines the partial order induced by the edges of the dag.

Given a binary decomposition tree T of a MSP dag G, the embedding process just outlined, can be performed by two traversals of T. First we traverse the tree in postorder and assign a *size* to each of its nodes: we assign the value one to the leaves and the sum of the values of its children to any internal node. Clearly, the size assigned to any node equals the number of vertices of the MSP dag represented by the subtree of T rooted at that node. We then traverse T in preorder assigning a pair of coordinates to each vertex as follows: the root of the tree gets the coordinates (1,1), and the children of each node visited are assigned coordinates using their previously computed sizes in the formulae give above, with the label of their parent determining which set of formulae is used.

As an example, fig.16 shows the tree of fig.3 with the integers associated to its nodes by the two traversals just described. The resulting embedding in the plane of the MSP dag of fig.1 is the one shown in fig.14 since the two total orders produced by this process are identical to the ones given in fig.6.

Clearly these two traversals can be performed in time proportional to the number of nodes of the tree which is in turn proportional to the nuumber of vertices of the MSP dag it represents.

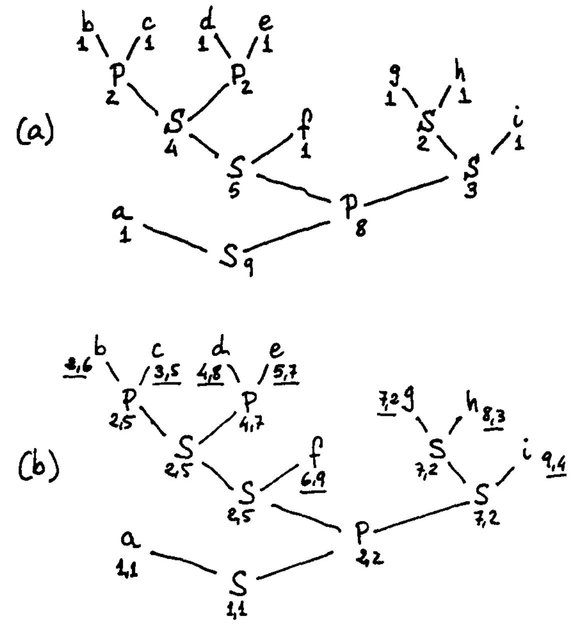

Fig.16
Values produced by the embedding procedure on the decomposition tree of fig.3. (a) Sizes. (b) Coordinates.

Furthermore, once the two total orders are computed, it requires just two comparisons to determine whether a given pair (u,v) of vertices represents an edge of the transitive closure of G: there is a path from u to v in G (i.e., (u,v) is an edge of the transitive closure of G) if and only if u appears before v in the two partial orders.

Therefore Step 4 of the GSP recognition procedure can be implemented to run in time proportional to the size of the decomposition tree plus the number of edges to be tested, and we have completed our description of a linear time implementation of Algorithm 1.

4. Forbidden subgraph characterization

The characterization of a class of graphs by exhibiting a subgraph that no member of the class may contain has been a common goal of the classical theory of graphs. Perhaps the most famous of such *forbidden subgraph characterizations* is due to Kuratowskii's for the class of planar graphs ([HAR]). In this section we present a characterization of this type for the class of GSP dags based on the dag of fig.17 which, for obvious reasons, will be called N.

It can be shown that a dag G is GSP if and only if its transitive closure does not contain N as an induced subgraph. The proof of this fact is rather long and will be omitted. The details can be found in [VAL] where our recognition algorithm is modified so as to exhibit the forbidden N subgraph of its input whenever it gives a negative answer still maintaining its linear running time.

Here we will limit ourselves to a description of the relationship of this forbidden subgraph characterization with a previously

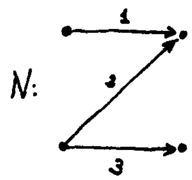

Fig.17
The forbidden subgraph of the class of GSP dags.

known characterization of TTSP multigraphs.

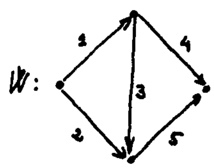

Fig.18
The forbidden subgraph of the class of TTSP multidigraphs.

Duffin [DUF] showed that a multigraph is TTSP if and only if it does not contain a subgraph homeomorphic to K_4 (the complete graph on four vertices). This characterization can be generalized trivially, to show that a multidigraph with a single source and a single sink is TTSP if and only if it does not contain a subgraph homeomorphic to the dag of fig.18, that we will call W.

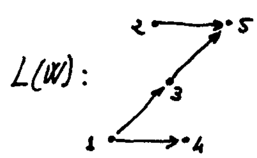

Fig.19
The line digraph of the dag of fig.18.

From this characterization of TTSP multidigraphs and the relationship given by lemma 1, it is not too difficult to show that the transitive closure of a CBC dag contains an embedded N if and only if its line digraph inverse does not have a subgraph homeomorphic to W. That is to say, that a CBC dag is MSP if and only if its transitive closure does not contain N as an induced subgraph. This relationship can be made plausible very quickly if one realizes that the line digraph of W is the dag shown in fig.19, whose transitive closure obviously contains an induced N subgraph.

The relationship that we have just exhibited is the basis of the proof given in [VAL].

5. Concluding remarks

This section presents some of the consequences of the existence of the linear time recognition procedure just described.

First, it should be noticed that Step 1 of the recognition procedure is a linear time transitive reduction algorithm for GSP dags, and that Step 4 computes the transitive closure of an MSP dag (in implicit form) again in linear time. This results compare favorably with the best known algorithms to perform the same tasks in arbitrary dags (see [AGU] for instance).

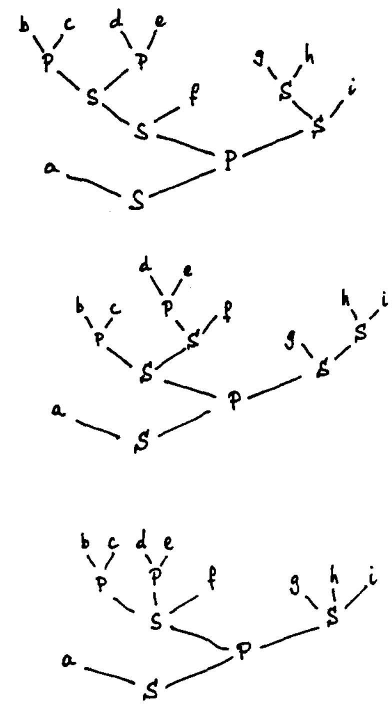

Fig.20
Two binary decomposition trees for the MSP dag of fig.1 and the unique tree obtained by shrinking their edges that join nodes with the same labels.

Second, even though we said that several nonisomorphic binary decomposition trees may represent the same MSP dag, there is a way of modifying these trees to make them represent MSP dags in a quasi-unique way. If one *shrinks* the edges of a binary decomposition tree that join internal nodes with the same label, the result is a tree that is unique up to reordering of the children of **P** nodes, as illustrated in fig.20. A formal proof of this fact can be found in [VAL], where the uniqueness of this tree is related to the uniqueness of the triconnected components of a biconnected multigraph ([HT]).

This unique representation of a MSP dag by a tree, allows the use of the linear time tree isomorphism algorith ([AHU]) to determine whether two MSP dags are isomorphic in linear time. This fact is interesting because there is no known polynomial time graph isomorphism algorithm.

It should be noted however that the isomorphism of GSP dags is as hard as the isomorphism of arbitrary graphs since the decomposition tree does not give any information about redundant edges being or not being present, and one can encode an arbitrary graph into the redundant edges of a GSP dag ([VAL]).

Finally, the existence of a polynomial algorithm for the subtree isomorphism problem ([MAT]), suggests the possibility of a polynomial algorithm for the subgraph isomorphism problem for MSP dags by testing subtree isomorphism of their unique decomposition trees. (The subgraph isomorphism problem for arbitrary graphs is known to be NP-complete [AHU]).

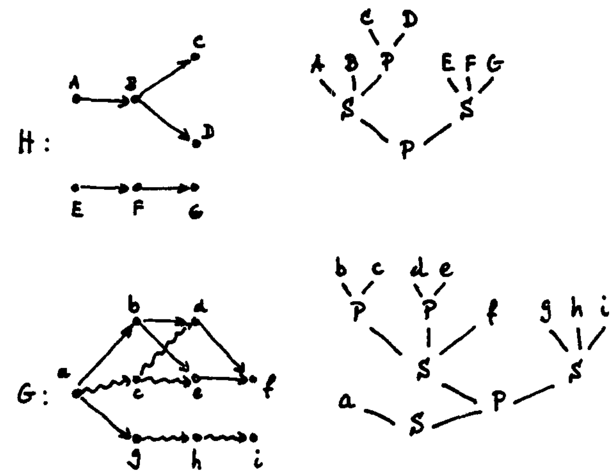

Fig.21
Two MSP dags G and H and their quasi unique decomposition trees. Even though H is isomorphic to a subgraph of G, the same relationship does not hold for the decomposition trees.

Unfortunately matters are not quite that simple, as fig.21 shows: an MSP dag H may be isomorphic to a subgraph of another MSP dag G, and yet the unique decomposition tree of H is not isomorphic to any subtree of the unique decomposition tree of G. Furthermore, the problem seems to lead rather quickly into nontrivial questions related to tree homomorphism that we are currently investigating.

6. References

[ADA] A. Adam, *On graphs in which two vertices are distinguished*, **Acta Math. Acad. Sci. Hungary 12,** 1961, 377-397.

[AGU] A. V. Aho, M. R. Garey, and J. D. Ullman, *The transitive reduction of a directed graph*, **SIAM J. Comput. 1, 2,** 1972, 131-137.

[AHO] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1976.

[DUF] R. J. Duffin, *Topology of Series-Parallel Networks*, **Journal of Math Analysis and Applications 10,** 1965, 303-318.

[HAR] F. Harary, *Graph Theory*, Addison-Wesley, Reading, Mass., 1971.

[HKS] F. Harary, J. Krarup, and A Schwenk, *Graphs suppressible to an edge*, **Canadian Math. Bull. 15, 2** 1972, 201-204.

[HN] F. Harary and R. Norman, *Some properties of line digraphs*, **Rendiconti del Circolo Matematico Palermo 9,** 1960, 149-163.

[HT] J. E. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, **SIAM J. Comput. 2, 3,** 1973, 135-158.

[KLE] J. B. Klerlein, *Characterizing line dipseudographs*, **Proceedings of the Sixth Conference on Combinatorics, Graph Theory, and Computing,** 1975, 429-442.

[KNU] D. E. Knuth, *The Art of Computer Programming*, vol. 1, *Fundamental Algorithms*, Addison-Wesley, Reading, Mass., 1969.

[LAW] E. L. Lawler, *Sequencing Jobs to minimize total weighted completion time subject to precedence constraints*, **Annals of Discrete Math. 2,** 1978, 75-90.

[LEH] P. G. H. Lehot, *An optimal algorithm to detect a line graph and output its root graph*, **Journal ACM 21, 4,** 1974, 569-575.

[MAT] D. W. Matula, *Subtree Isomorphism in* $O(n^{5/2})$, **Annals of Discrete Math. 2,** 1978.

[MON] C. L. Monma and J. B. Sidney, *A general algorithm for optimal job sequencing with Series-Parallel costraints*, Tech. Report No. 347, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, N.Y., July 1977.

[RIO] J. Riordan and C. E. Shannon, *The number of two terminal series parallel networks*, **Journal of Math. Physics 21,** 1942, 83-93.

[ROS] B. K. Rosen, *Tree manipulating systems and Church-Rosser theorems*, **Journal ACM 20, 1,** 1973, 160-187.

[SET] R. Sethi, *Testing for the Church-Rosser property*, **Journal ACM 21, 4,** 1974, 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, November 1976.

[VAL] J. Valdes, *Parsing Flowcharts and Series-Parallel Graphs*, Technical Report STAN-CS-78-682, Computer Science Department, Stanford University, Stanford, Ca., 1978.

[WAL] T. R. S. Walsh, *Counting labelled three-connected and homeomorphically irreducible two-connected graphs*, (Manuscript), 1978.

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