Graph Structures

JOHN L. PFALTZ

University of Virginia, Charlottesville, Virginia

ABSTRACT. The concept of a graph structure S_G in which a tree-like assemblage of subgraphs is used to represent a directed graph G is developed. This is directly analogous to the representation of lists by list structures. It is shown that with suitable restrictions, given S_G, G can be uniquely determined, and conversely, given G, S_G can be effectively constructed. A computer implementation which minimizes storage to represent G is presented, together with algorithms that illustrate the utility of graph structures, in particular, one that efficiently determines the existence of paths in G.

KEY WORDS AND PHRASES: data structures, directed graphs, list structures, path finding, graph homomorphism

CR CATEGORIES: 5.32, 3.74

1. *Introduction*

A simple list is a finite set of points (or items) that can be totally ordered. Such simple lists have a straightforward computer representation as a simply linked chain of cells, each of which represents a single point. The list concept becomes more interesting if we admit the ability to identify all the points of some subsequence and treat them as if they were a single item. Thus a more common definition is: a list is a finite sequence of elements in which each element may be either a point or itself a list (called a sublist). Computer representations of this more general concept are called list structures. As before, an individual list may be a simple linked chain of cells, but now those cells which represent sublist elements must contain an additional link to the sublist itself.

The utility of these tree-like list structures is well documented in the literature. One immediately considers generalizing this approach to the case of a set of points (or items of information) where the relation between them is other than a total ordering—in short, to arbitrary directed graphs. Thus we would be led to define a directed graph as a relation on a set of elements, each of which is either a point or itself a directed graph. And, in effect, given an original finite graph G, we would be identifying arbitrary subgraphs H of G and treating them as single subgraph elements.

Unfortunately this appealing idea simply doesn't work in general. Given a sublist in a list structure, we know how to reinsert it into its higher level list, so that by a simple traversal of the list structure tree we can reconstruct the original list of

Author's address: Department of Applied Mathematics and Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22901. This research was supported in part by National Science Foundation grant GJ-31626.

points. But given that H is a subgraph of G, how do we know how H is to be "reinserted" into G? We have lost the very valuable property of reconstructibility that is present with list structures.

We will show, however, that by suitably restricting the kinds of graph G to be considered, and by restricting the kinds of subgraph H to be identified as subgraph elements, a workable and useful concept of a "graph structure" can be developed which is directly analogous to that of a "list structure."

First we will review the graph notation to be used, and will establish precisely what we mean when we "identify" a subgraph H of a graph G, and what we mean by "reconstructibility." We will prove several results about graphs, which we will use in Section 3 to show that graph structures actually have the properties we want and that they can be effectively constructed. Graph structures themselves are defined in Section 3. Finally, we will illustrate the utility of graph structures by presenting algorithms that operate on them, and will suggest various modifications of the concept.

2. *σ and τ Contractions*

A simple *directed graph* G is a relation E (or set of ordered pairs) on a set of points P, denoted $G = (P, E)$. A *subgraph* H of G is E restricted to a subset $P_H \subset P$. With this definition of subgraph, one of several encountered in the literature, a subgraph is completely determined by its point set. Consequently we will often let H denote both the point set and the subgraph, trusting that context will make the precise meaning clear. For example, $G \sim H$ could denote the set of points in G but not H, together with the subgraph on these points.

In general, the relation E (for edge) is neither transitive nor reflexive. We can, however, let E induce another relation ρ (for path) on P which has both these properties. We say there is a path from p to q, denoted $\rho(p, q)$, if there exists a sequence of points $x_0, x_1, \cdots, x_n \in P$ with $n \geq 0$ such that (1) $(x_{i-1}, x_i) \in E$; (2) $x_{i-1} \neq x_i$, for $1 \leq i \leq n$; and (3) $x_0 = p$, $x_n = q$. The path from p to q is said to have length n, denoted $|\rho(p, q)| = n$. Transitivity and reflexivity are evident; in particular, for all $p \in P$ there exists $\rho(p, p)$ of length zero. If, in addition, ρ is a partial order relation on P (that is, weakly antisymmetric), then G is said to be an *acyclic* graph.

If H is any subgraph of G, then by the *left context* of H, denoted $L(H)$, we mean the set $\{p \notin H \mid (p, q) \in E \text{ for some } q \in H\}$. The right context is similarly defined, $R(H) = \{r \notin H \mid (q, r) \in E \text{ for some } q \in H\}$.

By the set of *minimal* points of H, denoted m_H, and *maximal* points, denoted M_H, we mean the sets $\{q \in H \mid \text{for all } p \in H, \rho(p, q) \text{ implies } p = q\}$ and $\{q \in H \mid \text{for all } r \in H, \rho(q, r) \text{ implies } q = r\}$ respectively.

Let R denote any relation on a set P and suppose we have, in addition, an equivalence relation Σ on P. It is natural to call Σ an *R-congruence* on P if $(p_1, q_1) \in R$ implies $(p_2, q_2) \in R$ whenever (1) $(p_1, p_2), (q_1, q_2) \in \Sigma$, and (2) $(p_1, q_1) \notin \Sigma$. (The usual definition of a congruence relative to a binary operator would omit condition (2), but this seems too restrictive in the case of most relations.) Thus an R-congruence preserves the relation R between elements of distinct congruence classes.

Now given any subgraph H, it, together with the singleton sets $\{p_i\}$ for all $p_i \in G \sim H$, defines a partition on P and hence an equivalence. The natural question to ask is, when does such a subgraph H induce an E-congruence? It turns out that this is too stringent a requirement for our intended application so we instead ask, when does such a subgraph H induce a ρ-congruence? These H have the property that for any $p \in G \sim H$, if $\rho(p, q_i)$ for some $q_i \in H$, then $\rho(p, q_i)$ for all $q_i \in H$.

By a *homomorphism* of $G = (P, E)$ onto $G' = (P', E')$, denoted $\varphi : G \to G'$, we mean a simple point function $\varphi : P \to P'$ with the properties: (1) $(p, q) \in E$ implies $(\varphi(p), \varphi(q)) \in E'$; and (2) $(p', q') \in E'$ implies that for some $x \in \varphi^{-1}(p')$, $y \in \varphi^{-1}(q')$ we have $(x, y) \in E$. We will use primes to denote points, edges, and paths in a homomorphic image of G. Note that G and the point function φ alone define a unique homomorphic image G' on $\varphi(P)$, and that G and G' are isomorphic if and only if φ is one to one. A homomorphic mapping $\varphi : G \to G'$ is called a *contraction* if G' is acyclic.

PROPOSITION 1. *If G is acyclic, then any ρ-congruence Σ induces a contraction $\varphi : G \to G'$.*

PROOF. Define φ by $\varphi(p) = \varphi(q)$ iff $(p, q) \in \Sigma$. It is now straightforward but tedious to show that if G' is not acyclic, then G cannot have been acyclic either [4].

The converse to this proposition does not hold; we may have contractions φ whose pre-image partition is not a ρ-congruence.

A *simple contraction* $\varphi_H : G \to G'$ is one where H is a nonempty subgraph of G, and (1) $\varphi(q_i) = q'$ for all $q_i \in H$, and (2) φ is an isomorphism of $G \sim H$ onto $G' \sim \{q'\}$. Because φ_H is a homormophism we have in addition, (3) (p, q_i) or $(q_i, r) \in E$, where $p, r \in G \sim H$ implies $(\varphi(p), q')$ or $(q', \varphi(r)) \in E'$. In effect then, a simple contraction φ_H on G maps the entire subgraph H onto a single point, where q' is its new identifier, while acting as the identity map on the rest of G. Clearly, if the subgraph H induces a ρ-congruence on G, then φ_H is a simple contraction. Also, any contraction φ can be regarded as a composition of simple contractions $\varphi_{H_1}, \cdots, \varphi_{H_n}$ where $H_i = \varphi^{-1}(p_i')$.

We have observed that given G and any homomorphism φ, G' is completely determined. Of prime concern now are those contractions φ_H with the property that H, G', and $q' \in G'$ completely characterize G. Since H is given and $G \sim H \cong G' \sim \{q'\}$, all that is necessary to complete the characterization is to be able to reconstruct those edges between H and its left and right contexts. $L(H)$ and $R(H)$ themselves are known since they correspond to $L(q')$ and $R(q')$, respectively. Contractions φ_H which have this property we call *reconstructable*.

In many computer applications, such as the representation of transitive implications in a semantic model, accessibility in a directory, or task dependence in a PERT network, it is redundant to store that information which can be inferred from transitivity. Regarding the relations as a directed graph G, the most compact representation would have the property that $|\rho(p, q)| \geq 2$ implies $(p, q) \notin E$. Such graphs are said to be *basic*. (Conversely, the most redundant representation of G would be by its transitive closure G^t in which $\rho(p, q)$ implies $(p, q) \in E$.) Now suppose the subgraph H induces a ρ-congruence on G; then clearly φ_H preserves all the path information contained in G. We next show that if G is basic, φ_H also preserves the edge structure of G, in that we can reconstruct the missing edges, even though H does not in general induce an E-congruence.

PROPOSITION 2. *If a subgraph H induces a ρ-congruence on a basic acyclic graph G, then H has the property that for any $q \in H$ and any $p, s \in G \sim H$,*

(1) $(p, q) \in E$ *implies* $q \in m_H$ *and for all* $r \in m_H$, $(p, r) \in E$, *and*
(2) $(q, s) \in E$ *implies* $q \in M_H$ *and for all* $r \in M_H$, $(r, s) \in E$.

PROOF. Suppose $q \notin m_H$; then since G is acyclic, there exists $r \in m_H$ such that $\rho_1(r, q)$ of length ≥ 1. Since H is a ρ-congruence class, $\rho(p, q)$ implies that $\rho_2(p, r)$ of length ≥ 1. Now by transitivity ρ_1 and ρ_2 imply $| \rho_3(p, q) | \geq 2$ which, together with $(p, q) \in E$, contradicts G basic. So $q \in m_H$. Now let $r \in m_H$ be any minimal point. As before, we know $\rho(p, r)$. Suppose $| \rho(p, r) | \geq 2$, implying there exists $x_1 \in \rho$, $x_1 \notin H$ (since r is minimal). Again, since H is a ρ-congruence class, we have $\rho(x_1, q)$ of length ≥ 1, yielding $| \rho(p, q) | \geq 2$, which with $(p, q) \in E$ again contradicts G basic. So $| \rho(p, r) | = 1$. The proof of assertion (2) is similar.

In effect then, every edge between the left context $L(H)$ and H is between $L(H)$ and its set of minimal points m_H. Further, for all $p \in L(H)$, $q \in m_H$, we have $(p, q) \in E$, and similarly for the right context. For this reason, subgraphs H which exhibit the two properties of the preceding proposition will be called *m-M subgraphs*. It is now readily apparent (1) that G is reconstructible from H, G', and $q' \in G'$ provided H is a *m-M* subgraph; (2) that in a graph G (not necessarily acyclic or basic), any *m-M* subgraph H induces a ρ-congruence on G; and (3) if H induces a ρ-congruence on a basic acyclic graph G, then H must be an *m-M* subgraph (Proposition 2 above).

Now the question becomes, how does one discover such *m-M* subgraphs in a practical sense; or equivalently, how does one discover simple contractions φ_H for which H is an *m-M* subgraph?

In particular, we would like a method of deriving such φ_H by combining contractions which are in some sense primitive, and which can themselves be discovered by local search procedures. In [4] it is shown that for G acyclic,

PROPOSITION 3. *If* $\varphi_H : G \rightarrow G'$ *is any contraction, then* φ_H *can be represented as a composition of simple contractions* φ_{H_i} *where each* H_i *is a subgraph on exactly two points.*

In such two-point contractions, the contracted subgraph H_i is either the trivial graph with no edge between the two points, or a linear string with exactly one edge from one point to the other. This leads us to the following definitions. φ_H is called a τ *contraction* if H is an *m-M* subgraph and H is the trivial graph. φ_H is called a σ *contraction* if H is an *m-M* subgraph and the points of H are totally ordered by $\rho |_H$, that is, the path relation restricted to H. In these cases we will call H a τ or σ subgraph, respectively, which simply denotes a special kind of *m-M* subgraph.

A graph G will be called *irreducible* if it contains no τ or σ subgraphs. In Figure 1, $H = [d, e] \subset G$ is a τ subgraph and $\varphi_{[d,e]} : G \rightarrow G'$ is a τ contraction; $H' = [b', q'] \subset G'$ is a σ subgraph and $\varphi_{[b',q']} : G' \rightarrow G''$ is a σ contraction; G'' is irreducible.

We can make several observations from this example. First, given the reduced graph G'' and the contractions $\varphi_{[d,e]}$, $\varphi_{[b',q']}$, we can reconstruct G. We will use this observation as the key to defining graph structures. Also, the *m-M* subgraph $[b, d, e]$ in G can be represented as a composition of τ and σ subgraphs, but the *m-M* subgraph $[q'', f'', c'', g'']$ in G'' cannot. In particular, this shows that we cannot strengthen Proposition 3 and assert that if H is an *m-M* subgraph, then φ_H can be represented by the composition of φ_{H_i} where each H_i is an *m-M* subgraph on two points, even though in many cases this does turn out to be possible. We can, however, show that:

PROPOSITION 4. *If* G *is basic and* φ_H *is any* τ *(or* σ*) contraction, then* φ_H *can be represented as a composition of two-point* τ *(or* σ*) contractions.*

PROOF. If H is a τ subgraph, then evidently any subgraph H_1 on two points

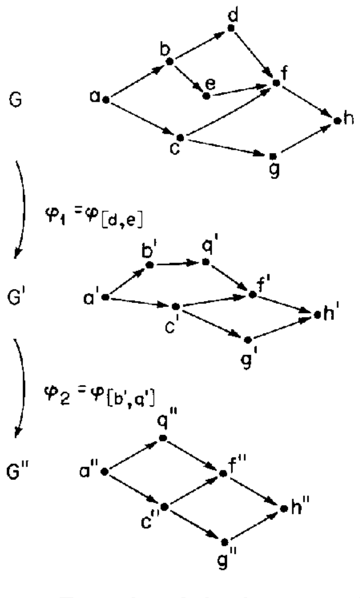
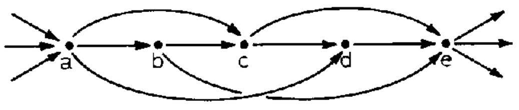


FIG. 1. Examples of simple contractions

contained in H is a τ subgraph, as is $H \sim H_1 \cup \{\varphi(H_i)\}$. If $H = [q_1, \cdots, q_n]$ is a σ subgraph, we let $H_1 = [q_1, q_2]$ or $[q_{n-1}, q_n]$ for the same result. ▌

The requirement that G be basic is shown necessary from the following example:



$H = [a, b, c, d, e]$ is readily a σ subgraph (it is an *m-M* subgraph totally ordered by ρ). However, no two-point subgraph $H_1 \subset H$ is a σ subgraph.

PROPOSITION 5. *If H_1, H_2 are both τ (or σ) subgraphs of G acyclic, and $H_1 \cap H_2 \neq \varnothing$, then $H_1 \cup H_2$ is a τ (or σ) subgraph.*

PROOF. Suppose H_1 and H_2 are τ subgraphs, so that $L(p_i) = L(p_j)$ for all $p_i, p_j \in H_1$, $L(q_i) = L(q_j)$ for all $q_i, q_j \in H_2$. Since $H_1 \cap H_2 \neq \varnothing$, $p_i = q_j$ for some i, j, so that $L(r_i) = L(r_j)$ for all $r_i, r_j \in H_1 \cup H_2$. Similarly $R(r_i) = R(r_j)$. Readily, $E|_{H_1 \cup H_2} = \varnothing$, so every point r_i is both minimal and maximal, implying $H_1 \cup H_2$ is an *m-M* subgraph, hence a τ subgraph.

Now suppose $H_1 = [p_i]$, $H_2 = [q_j]$ are σ subgraphs. Since G is acyclic, we may assume without loss of generality that there exists no path from q_1 to p_1, and that $p_i = q_j$. We can show that in the general case $H_1 \cup H_2$ is the totally ordered subgraph $[p_1, \cdots, p_i, q_{j+1}, \cdots, q_n]$, where $q_{j-k} = p_{i-k}$ for $0 \leq k \leq j-1$ and $p_{i+k} = q_{j+k}$ for $0 \leq k \leq m-i$. To see this last assertion, suppose, for instance, that $q_{j-1} \neq p_{i-1}$; then $q_{j-1} \in L(p_i)$ implying $q_{j-1} \in L(H_1)$. Since H_1 is an *m-M* subgraph, $p_i \in m_{H_1}$, so $p_i = p_1$. But now $\rho(q_1, q_{j-1})$ and $(q_{j-1}, p_1) \in E$ implies $\rho(q_1, p_1)$, a contradiction to our original assumption. The full proof requires consideration of several cases, all of which may be handled by this kind of argument. It is then easy to show that $L(H_1 \cup H_2) = L(p_1) = L(H_1)$ and $R(H_1 \cup H_2) = R(q_n) = R(H_2)$, so that $H_1 \cup H_2$ is a σ subgraph. ▌

COROLLARY. *If φ_{H_1} and φ_{H_2} are τ (or σ) contractions and $H_1 \cap H_2 \neq \emptyset$, then $\varphi_{H_1}[\varphi_{H_2}(G)] = \varphi_{H_1 \cup H_2}(G)$ is a τ (or σ) contraction.*

3. *Implementation of Graph Structures*

A *graph structure* S is an acyclic graph G_β on a set of elements, each of which is either (1) a point, or (2) a graph structure G_α which is a τ (or σ) subgraph of G_β. Generally, if a graph structure is derived from or represents a simple acyclic graph G, we denote it by S_G. For example, the graph G of Figure 2 is represented by the

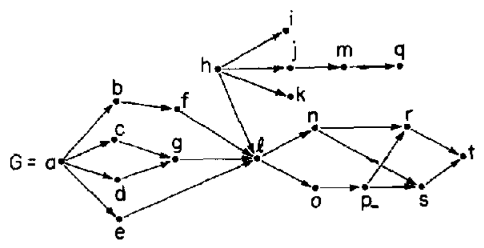


FIG. 2. Original graph corresponding to the graph structures of Figures 3 and 4

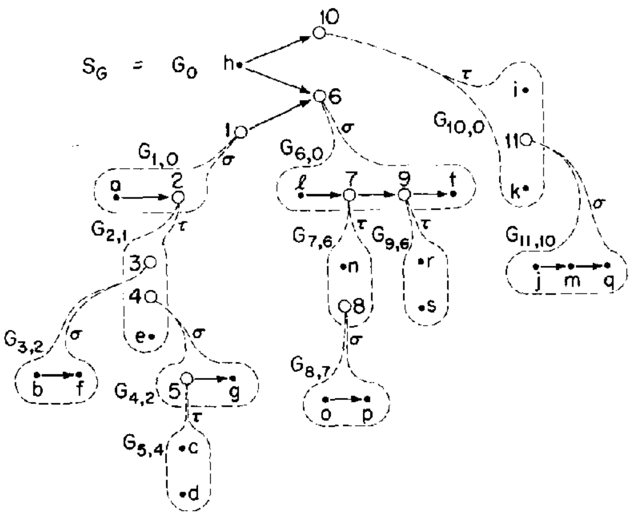


FIG. 3. Graph structure S_G of the graph G of Figure 2

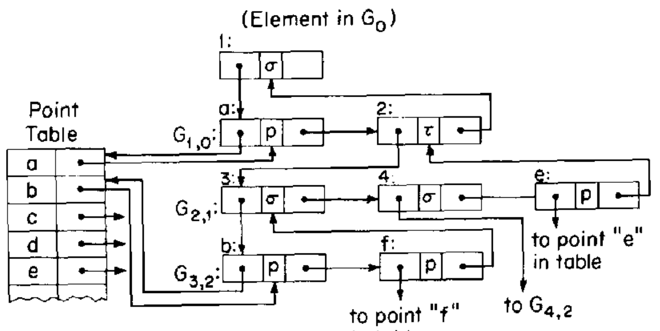


FIG. 4. Computer representation of a portion of Figure 3

graph structure S_G of Figure 3. In both figures we use lowercase letters as arbitrary point identifiers; integers are used to identify other elements.

Notice that the graph structure S_G is tree-like, with the irreducible graph G_0 serving as its root. The integer identifier "1" denotes an element (or point) in G_0, but it also identifies the entire σ subgraph on the two elements "a" and "2". This σ subgraph we might call G_1, but since G_1 also serves as a single element of G_0, we use the symbolism $G_{1,0}$. In a similar manner, the τ subgraph on the elements "c" and "d" which is a leaf of the tree structure we denote by $G_{5,4}$. In general we will use the symbolism $G_{\alpha,\beta}$ to denote subgraph structures where α is both the element and subgraph identifier and β identifies the graph of which this is an element. Thus depending on context, $G_{\alpha,\beta}$ may be used to denote either a single element α of the graph G_β, or the entire graph G_α itself.

Graph structures are particularly easy to represent in computer storage. In Figure 4 we illustrate one such representation of a small portion of Figure 3 using what is essentially a ring structure. Here each subgraph $G_{\alpha,\beta}$ is represented by linearly linking cells that denote all of its elements. The ring is completed by linking this list circularly through a cell α that denotes the entire subgraph and which in turn belongs to the ring of elements of G_β. Here each primitive point has been treated as a singleton subgraph; and these cells have been arranged as a linear symbol table to permit easy entry into S_G using an external (or user) identifier for these points. Other techniques of access, via hash coding or a directory, are equally possible.

Notice that it is unnecessary to explicitly represent any edges in any of the reduced portions of the graph, since these are implicitly indicated by the type of subgraph α (τ or σ) and its position in G_β. Elements that belong to the irreducible graph G_0 must be handled differently, since here edges must be explicitly represented. Standard techniques include the use of a binary adjacency matrix or linked chains of edge pointers. The method of representing G_0 is omitted from Figure 4.

The ring structure of Figure 4 appears to be the most economical representation; however, many variants are possible (see Knuth [2] for such alternatives). For example, a header cell may be added to precede the list of elements in $G_{\alpha,\beta}$ and the extra left field used to point directly to the header of G_β. The inclusion of such minimal redundancy greatly simplifies the design of algorithms to operate on the graph structure, as well as improving their efficiency.

Whatever actual computer representation is used, it is now a fairly straightforward procedure to write algorithms that find the left (or right) contexts of any specified point (or higher level element), or that step forward (or backward) along paths in the original graph G, even though no edges in any reducible portion of G are explicitly represented.

We have shown that, given a graph structure S_G, we can effectively and economically represent it in computer storage. But it is seldom the case that S_G is given. In most computer applications we are given a collection of points (data, if you will) and the relation between them; that is, we are given only the graph G. We require an effective procedure for deriving and thereby constructing in computer storage the corresponding graph structure S_G.

Intuitively, given a graph $G = (P, E)$, one must examine all possible subsets $P_i \subseteq P$ in order to discover reducible τ and σ subgraphs H_i—a nasty combinatorial problem. However, Proposition 4 assures us that it is sufficient to consider only those subgraphs on exactly two points. We next present an algorithm which exhaustively examines G for a two-point τ or σ subgraph H. If found, the points are identified by the appropriate contraction φ_H. The procedure is then iterated on the root G_0 of the resultant partial graph structure. If none are found, G_0 is irreducible and the reduction is complete.

In this and following algorithms we assume a programming language with basic set manipulation capability. Thus, for example, $R(q)$ denotes a primitive set valued function, which given an element in G_0, returns the set of all elements to its immediate right. We use the replacement operator, as in $\{r_k\}_{k=1,n} \leftarrow R(q)$, to denote an idexed version of this set; and we assume an iterative control statement of the form "for each $r_k \in R(q)$ do . . . ."

Procedure Reduce [This algorithm examines the root G_0 of a graph structure S_G for the existence of a two-point τ or σ subgraph H. If found, the elements of H are contracted to a single element q', while the subgraph H, now called $G_{\alpha,0}$, is stored as a list of elements.]

```
Step 0. [Initialize] $G_0 \leftarrow G$;
Step 1. for each element $q \in G_0$ do until step 4
Step 2. $\{r_k\}_{k=1,n} \leftarrow R(q)$;
        if $n = 1$ and $L(r_1) = q$ then
          [$H = \{q, r\}$ is a $\sigma$ subgraph]
          contract $(\sigma, q, r_1)$; go to step 1:
        if $n = 0$ and $L(q) = \varnothing$ then go to step 5;
Step 3. [$n \geq 2$] for each $r_k \in R(q)$ do until step 4
        $\{p_j\}_{j=1,m} \leftarrow L(r_k)$;
        for each $p_j \in L(r_k)$ do until step 4
          if $R(q) \neq R(p_j)$ then go to step 4;
          if $L(q) = L(p_j)$ then
            [$H = \{q, p_j\}$ is a $\tau$ subgraph]
            contract $(\tau, q, p_j)$; go to step 1;
Step 4. continue;
        [$G_0$ is completely reduced] return;
Step 5. [$R(q)$ and $L(q)$ are empty, so $q$ is an isolated point. Check for the existence of another
        isolated point] for each element $q^* \neq q$ in $G_0$ do until step 6
          if $L(q^*) = R(q^*) = \varnothing$ then
            contract $(\tau, q, q^*)$; go to step 1;
Step 6. [continue initial search] go to step 2;
```

Notice that "contract (type e_1, e_2)" merely involves replacing the elements e_1 and e_2 in G_0 by a new one of the correct "type" and placing them on a ring through it. Restructuring the basic search loop beginning with step 1, so that after a contraction it continues the search "where it left off" instead of blindly starting all over again, significantly improves the performance of the preceding algorithm. Other modifications in a similar vein can be made as well.

If G is known to be connected, as is often the case, then steps 5 and 6 can be omitted. If, further, G is known to be a two-terminal parallel series network (TTSPN), then for all q_1, q_2 belonging to a common τ or σ subgraph, we have $|L(q_1)| = |L(q_2)| = |R(q_1)| = |R(q_2)| = 1$, which can be used to simplify the procedure. Berkus [5] used this trick in developing an automata that would accept (construct a "parse tree" of) TTSPN's.

More importantly, every reduced subgraph in the graph structure produced by the algorithm contains only two points. However, in the graph structure S_G of

Figure 3, there are several subgraphs consisting of three or more elements. It is clearly inefficient to have, say, a τ subgraph $G_{\alpha,\beta}$ which is an element of a τ subgraph $G_{\beta,\gamma}$. The two should be combined into a single τ subgraph, thus simplifying S_G. Proposition 5 asserts that such simplification is legitimate. The procedure can be illustrated by the following deceptively short algorithm in which type (G_α) has value 0, 1, or 2 depending on whether G_α is a singleton primitive point, τ subgraph, or σ subgraph, respectively.

Procedure Simplify [This algorithm examines all the elements of a graph structure and combines pairs of the same type where possible.]

```
for each $G_{\alpha,\beta} \in S_G$ do
if type $(G_{\alpha,\beta})$ = type $(G_{\beta,\gamma})$ then
  combine $(G_{\alpha,\beta}, G_{\beta,\gamma})$;
return;
```

The subprocedure "combine" is straightforward if we assume a list-like representation such as Figure 4. The list of elements of $G_{\alpha,\beta}$ is simply inserted into the list of elements of $G_{\beta,\gamma}$ so as to replace the original cell e_α which had pointed to it. The first statement "for each $G_{\alpha,\beta} \in S_G \cdots$" indicates simply that each element of S_G must be examined. Since S_G is essentially tree structured, a simple post-order (or pre-order) traversal [2] of S_G will suffice. Use of the latter requires some additional care since we are altering portions of the tree as we are traversing them.

With these two basic procedures we have implemented a system that will accept any basic acyclic graph as input and store it as a graph structure. We would now expect to design procedures which use this new kind of information structure S_G. We illustrate with two typical examples.

Procedure Leftcontext $(G_{\alpha,\beta})$ [This procedure, given the identifier (i.e. pointer to) graph structure element $G_{\alpha,\beta} \neq G_0$, delivers the elements immediately to the left.]

```
Step 0.  get $G_{\beta,\gamma}$ [using the pointer $\beta$];
Step 1.  if $G_{\beta,\gamma} = G_0$ then leftcontext ← $L(G_{\alpha,\beta})$;  return;
         if type $(G_{\beta,\gamma})$ = 2 then
           [$G_{\beta,\gamma}$ is a $\sigma$ subgraph, hence a list of elements $e_1, \cdots, e_n$ where $e_i = e_\alpha$ (that s
           points to $G_{\alpha,\beta}$) for some $1 \leq i \leq n$]
           if $e_\alpha = e_1$ then go to step 2
             else leftcontext ← $e_{i-1}$ ; return;
Step 2.  [$G_{\beta,\gamma}$ is a $\tau$ subgraph. (note that $G_{\beta,\gamma}$ cannot be a primitive point with type = 0
         it contains $G_{\alpha,\beta}$ as a subelement)]
         get $G_{\gamma,\delta}$ [using the pointer $\gamma$];
         since $G_{\beta,\gamma}$ does not contain elements to the left of $G_{\alpha,\beta}$, we must work up in $S_G$ .]
         $G_{\beta,\gamma} \leftarrow G_{\gamma,\delta}$ ;
         go to step 1;
```

Notice that the procedure "Leftcontext $(G_{\alpha,\beta})$" differs from the primitive procedure "L$(G_{\alpha,\beta})$" in that the latter operates only on elements of G_0. Since we have not specified the computer representation of G_0, the design of the primitive procedures L and R are left to the reader. Also observe that both L and Leftcontext deliver the set of elements (not necessarily points) immediately to the left of the specified element. If we take Figure 3 as an example, then Leftcontext $(G_{5,4}) = \{a\}$ and Leftcontext $(l) = \{h, G_{1,0}\}$, where a, l, and h denote primitive points. If we wish the left context of the point l in the sense of primitive points immediately to the left (e.g. Figure 2), we must provide an additional procedure that delivers the "rightmost" points of $G_{1,0}$, namely $\{f, g, e\}$.

A frequently used procedure is one which, given two points p_1, p_2 of G (or elements of S_G), determines whether there is a path $\rho(p_1, p_2)$ from p_1 to p_2. An efficient implementation of this procedure is essential in information retrieval systems where the path represents a transitive chain of implications or associations. In a dynamic system where new points or edges are being added to a basic model G, repeated and rapid applications of this procedure are required to test that the acyclicity and basicness of G are preserved with each new addition.

Boolean Procedure Path (G_1, G_2) [This procedure tests for the existence of a path from G_1 to G_2 which are elements (usually primitive points) in S_G. It returns true if one exists, otherwise false.]

```
Step 1. [Find the smallest subgraph $G_\beta$ such that $G_1 \subseteq G_{\alpha1,\beta} \subset G_\beta$, and $G_2 \subseteq G_{\alpha2,\beta} \subset G_\beta$. If
        either $G_1$ or $G_2$ is contained in the other then the path concept is meaningless.] (This
        is a simple search in the tree $S_G$; the details of following links from $G_1$ and $G_2$ back
        to the root element $G_\beta$ and testing are left to the reader.)
Step 2. if $G_\beta = G_0$ then go to step 3;
        if type$(G_\beta)$ = 1 then
          [$G_\beta$ is a $\tau$ subgraph]
          path ← false; return
          else
          [$G_\beta$ is a $\sigma$ subgraph]
          find $G_{\alpha1,\beta}$, $G_{\alpha2,\beta}$ in the element list of $G_\beta$;
          if $G_{\alpha2,\beta}$ follows $G_{\alpha1,\beta}$
            then path ← true; return
            else path ← false; return;
Step 3. [$G_{\alpha1,\beta}$, $G_{\alpha2,\beta}$ are elements of $G_0$] (Here several standard procedures can be used.
        Iterated application of the left context operator $L$ in $G_0$ to form $L^k(G_{\alpha2,\beta})$ followed by
        testing for $G_{\alpha1,\beta}$ as an element of these sets will yield the path if it exists. A dynamic
        search, in the manner of Bellman [1], which builds and tests $R^k(G_{\alpha1,\beta})$ and $L^k(G_{\alpha2,\beta})$
        alternately is a more sophisticated and generally more efficient algorithm.)
```

The essential advantage of applying path to a graph structure S_G lies in step 2 where the existence or nonexistence of a path can be determined by examining a single subgraph G_β of known characteristics; and in the fact that if step 3 must be invoked, the graph G_0 has been considerably reduced. For example, in Figure 3 we can immediately verify that there is no path between points d and f since $G_{2,1}$ (the smallest subgraph containing them) is a τ subgraph. Similarly, the existence of a path from c to s is quickly verified by noting that the elements $G_{1,0}$ and $G_{6,0}$ (containing them respectively) are joined by a single edge in G_0.

4. *Generalizations of Graph Structures*

In defining a graph structure S_G of a graph G, we required that G be both acyclic and that S_G be determined by only σ and τ contractions. As a consequence we have been assured that (1) G can be reconstructed from S_G, since each subgraph H is an *m-M* subgraph; and (2) S_G can be derived from G by the iterated application of local procedures. Basic acyclic graphs occur as a natural model in many computer applications; nevertheless, one would like to generalize the procedure if possible.

It has been shown [4] that σ and τ contractions applied in this environment are

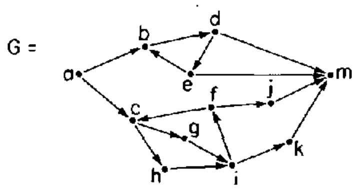


FIG. 5. Source graph for graph structure of Figure 6

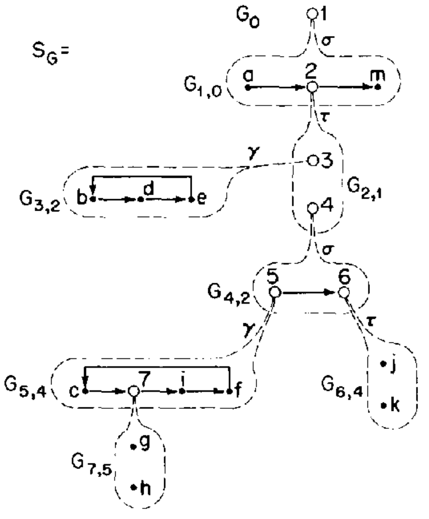


FIG. 6. Graph structure S_G of the graph G of Figure 5

strong convex homomorphisms which in some sense preserve the essential subgraph structure of G. Suppose we relax the first requirement of acyclicity and let G contain cycles (strongly connected components). We may then introduce a third kind of contraction—call it a γ contraction—which identifies all points on a cycle as a single element. γ contractions are also strong convex homomorphisms and γ-subgraphs induce ρ-congruences. We could thus let the generalized graph structure S_G of Figure 6 represent the graph of Figure 5. Unfortunately we have lost both the properties of unique reconstructibility and local reduction. For example, from the graph structure of Figure 6 we can deduce that there exists at least one edge from the strongly connected γ subgraph $[b, d, e]$ to the point m, but we cannot determine the number of such edges nor the actual points involved. Similarly, reduction of G to S_G may require nonlocal searches unless there is a bound on the length of the smallest cycle in any strongly connected component.

On the other hand, Proposition 5 (or its equivalent) still holds true, since strong connectivity is an equivalence relation on G, so that if H_1 and H_2 are both γ subgraphs of G and $H_1 \cap H_2 \neq \emptyset$, then $H_1 \cup H_2$ is a γ subgraph. Further, in some applications the sacrifice of properties (1) and (2) may not be serious. Consider an information retrieval system where certain assertions (or terms) are considered equivalent (or synonymous) and it is only the transitive relationships, or path structures, that are essential. Clearly the procedure "path" with appropriate modifications is still applicable. And if the system is a dynamic one, where edges, or relations, are slowly being added to the structure, the "path" procedure may be used to easily detect the presence of cycles and reduce them as they occur.

$$\begin{tikzcd}
\varphi\colon p_1 \arrow[r, bend left, "a_1"] \arrow[r, bend right, "a_2"'] & p_2 & \Longrightarrow & p_1' \arrow[r, "a'"] & p_2'
\end{tikzcd}$$

$$\begin{tikzcd}
\varphi\colon p_1 \arrow[r, "a_1"] & q \arrow[r, "a_2"] & p_2 & \Longrightarrow & p_1' \arrow[r, "a'"] & p_2'
\end{tikzcd}$$

FIG. 7. Edge contractions

In a similar manner the requirement of basicness can be relaxed. Many of the preceding remarks are still applicable.

One last generalization is of importance in certain network applications. We have regarded the set of points as the essentially primitive set in the definition of the graph G. Subgraphs and paths were defined in terms of points, and contractions identified collections of points (or elements) which had a common structural significance. But in PERT networks, for example, it is the edges (called activities) which are important and the points (called events) serve primarily as connectors. One is tempted to simply consider the corresponding line graph, but this doesn't quite work. Instead we must begin afresh and define contractions that identify two edges (activities) which originate and terminate at the points (events) p_1 and p_2. Again we have two fundamental cases, where the edges are disjoint or linearly related, yielding the contractions schematically illustrated in Figure 7. These intuitively correspond to our natural idea that two or more activities, conducted either serially or in parallel, may constitute a single overall activity. The latter kind of contraction has also been called a homoeomorphism (e.g. Harary [3]). The resultant tree-like structure is easily visualized, and the reader is left to work out the formal details.

REFERENCES

1. BELLMAN, R., COOKE, K., AND LOCKETT, J. *Algorithms, Graphs, and Computers.* Stevens and Co., 1970.
2. KNUTH, D. E. *The Art of Computer Programming.* Addison-Wesley, Reading, Mass., 1968.
3. HARARY, F. *Graph Theory,* Addison-Wesley, Reading, Mass., 1969.
4. PFALTZ, J. Convexity in graphs. *J. Comb. Theory. 10,* 2 (Apr. 1971), 143–162.
5. PFALTZ, J., AND BERKUS, M. Web grammars and picture description. CSC Tech. Rep., U. of Maryland, College Park, Md., Sept. 1970, pp. 70–138.

RECEIVED FEBRUARY 1971; REVISED SEPTEMBER 1971