A Sufficient Condition for Backtrack-Free Search

EUGENE C. FREUDER

University of New Hampshire, Durham, New Hampshire

ABSTRACT. A constraint satisfaction problem involves finding values for a set of variables subject to a set of constraints (relations) on those variables Backtrack search is often used to solve such problems. A relationship involving the structure of the constraints is described which characterizes to some degree the extreme case of minimum backtracking (none) The relationship involves a concept called "width," which may provide some guidance in the representation of constraint satisfaction problems and the order in which they are searched The width concept is studied and applied, in particular, to constraints which form tree structures.

Categories and Subject Descriptors: F.2.2 **[Analysis of Algorithms and Problem Complexity]**· Nonnumerical Algorithms and Problems—*sorting and searching*; G 2 1 **[Discrete Mathematics]** Combinatorics—*combinatorial algorithms*, G.2 2 **[Discrete Mathematics]** Graph Theory, H.3 3 **[Information Storage and Retrieval]**. Information Search and Retrieval—*search process*; I 2.8 **[Artificial Intelligence]** Problem Solving, Control Methods and Search—*backtracking*

General Terms: Algorithms, Theory

Additional Key Words and Phrases constraint network consistency, constraint satisfaction, graph coloring, scene labeling

1. *Introduction*

Backtrack search is recognized as a basic algorithmic technique in computer science [15, 16]. It can be used to search for values which instantiate a set of variables subject to a set of constraints. Combinatorial puzzles provide easily accessible examples, for example, the eight queens problem, where the variables represent positions of the eight queens on the chessboard, and the constraints require that no two queens can attack each other. A more serious example of a constraint problem involves the assignment of semantic interpretations to lines or regions in a visual scene [13, 14].

Backtrack search provides an improvement over simple depth-first search by cutting down the search space. However, backtrack search still permits a great deal of redundant and unnecessary effort [3]. Analysis of the effort required by backtrack search is currently obtained through sampling or experiments [5, 6, 11]. Attempts to reduce that effort are largely heuristic [2, 15].

It would be desirable to have a more analytical understanding of the effort involved in backtrack search. This effort is dependent on the structure of the problem and the order in which the search is conducted. The structure involves connective structure, the pattern of constraints among the variables, and contextual structure, the actual prohibitions of various combinations of values. In terms of the usual "backtrack tree" picture, we may distinguish "vertical order," the order in which variables are chosen for instantiation, and "horizontal order," the order in which values are tested for a given variable.

This paper analyses the relationship that holds between structure and order in the special case of "backtrack-free" search, where search effort, in terms of the amount

This paper is based in part upon work supported by the National Science Foundation under Grant MCS 80-03307

Author's address Department of Computer Science, University of New Hampshire, Durham, NH 03824

of backtracking actually required, is minimal. "Backtrack-free" implies that once a value for a variable is chosen which satisfies constraints involving previously chosen values, the choice never has to be "unmade," because a dead end is reached further down the tree. I present a relationship between the structure of the problem and the order of search which provides a sufficient condition for backtrack-free search. The characterization may be used, in theory, to obtain backtrack-free search. In practice this may not be practical; however, the insight gained from this work may at least provide heuristic guidance. This guidance might be applied to the structural design of the search space or the vertical ordering of the search.

Section 2 defines "width," "consistency," and "backtrack-free," and uses width and consistency to provide a sufficient condition for backtrack-free search. Section 3 provides an alternative characterization of width and means for determining width. Section 4 provides applications in which the connective structure of the constraints is a tree structure and a characterization of width-one structures in terms of trees. Section 5 demonstrates connections to graph theory and references some related work in that area. Section 6 contains brief conclusions.

Before proceeding I introduce three examples of constraint problems that will be used later in the paper to illustrate definitions or results.

First, a toy problem. We are given three sets of numbers: $X = \{5, 2, 4, 6\}$, $Y = \{2, 4, 6, 10\}$, $Z = \{5, 2, 4, 6\}$. Choose one number from each set such that the number chosen from Z divides both the others. A naive programmer might attack such a problem with a backtrack search which takes the given order X, Y, Z as the vertical search order, and the listed order as the horizontal search order for each set. This would lead to a considerable amount of backtracking. The first choice for X, 5, can only participate in a solution with the choice of 10 for Y, yet the search algorithm would bullheadedly try each possibility for Z with first 2, then 4, then 6, for Y.

The analysis of tree-structured constraint graphs in Section 4 will provide a formal basis (and thus, incidentally, a way of teaching) the "common sense" which might direct a more experienced programmer to utilize a different search order, Z, X, Y. (Indeed, the latter search order might seem a bit counterintuitive—does it make sense to choose a Z before we know the numbers it is supposed to divide?)

Second, consider the classical graph coloring problem. This problem requires us to assign colors to the vertices of a graph in such a way that if two vertices are joined by an edge in the graph, they will not have the same color. The variables of this constraint problem are the permissible node colors, and the constraints are the requirements that neighboring vertices not have the same color. For our purposes we will restrict the problem by establishing a priori a finite set of permissible colors.

Finally, consider the scene labeling problem (see Figure 1). This is the problem of labeling the lines of a two-dimensional drawing of a group of blocks with labels that represent their three-dimensional properties, for example, as convex or concave edges. The line drawings are themselves graphs ("picture graphs"). The labeling problem can be regarded as a constraint problem in which the variables are the vertices, the potential values are legitimate labelings of the lines entering the vertex, and the constraints are the necessity that lines connecting two vertices receive the same label at both vertices.

2. *Basics*

A constraint satisfaction problem requires us to instantiate a set of variables subject to a set of constraints, that is, relations involving the variables. We instantiate each variable from a finite set of potential values for that variable.

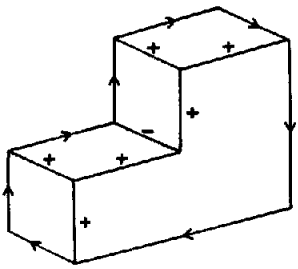

FIG 1 Scene labeling

A *constraint graph* for such a problem is a graph where the nodes represent variables and two nodes are linked to represent the existence of a constraint which involves these variables (and possibly others). At most a single edge will join two nodes, even if they are related by more than one constraint; in any case, we shall be concerned primarily with binary constraints. Graphs will be undirected in this paper. An *ordered constraint graph* arranges the nodes in a linear order. Obviously a given constraint graph admits of many orderings ($n!$, where n is the number of variables). The intention is for an ordering to correspond to a vertical order of backtrack search, the levels in the backtrack tree, that is, to the order in which variables are chosen for instantiation.

The constraint graph for a graph coloring problem has the same structure as the graph to be colored. Figure 2 shows a graph to be colored, a possible coloring, and the corresponding constraint graph. It also shows the six different ordered constraint graphs for this problem.

The *width* at a node in an ordered constraint graph is the number of links that lead back from that node to previous nodes. The width of an ordering is the maximum width at the nodes. The width of a constraint graph is the minimum width of all the orderings of that graph.

The width at node C in the first ordered constraint graph in Figure 2 is 2; at node C in the second ordered constraint graph the width is 1. The width of the first ordered graph is 2; of the second, 1. The width of the constraint graph in Figure 2 is 1, the minimum of the widths of the ordered constraint graphs.

K-consistency was developed in [4] as a generalization of low-order notions of consistency [7]. K-consistency implies the following: Choose any set of $k-1$ variables along with values for each that satisfy all the constraints among them. Now choose any kth variable. There exists a value for the kth variable such that the k values taken together satisfy all constraints among the k variables.

For example, the coloring problem in Figure 2 is not 3-consistent. If we choose red for vertex a and blue for vertex c, these choices are mutually consistent; however, there is then no color that can be chosen for vertex b that satisfies the constraints among a, b, and c together.

In [4] k-consistency was defined formally in terms of "constraint networks." It was observed that the algorithm given for synthesizing constraint expressions achieved j-consistency for all $j \leq k$ after k steps. I now define j-consistency for all $j \leq k$ as *strong k-consistency*. (It is possible to have k-consistency without strong k-consistency. A variation of the coloring problem in Figure 2 provides an example where it is specified a priori that colors red and blue are available for vertex b but vertices a and

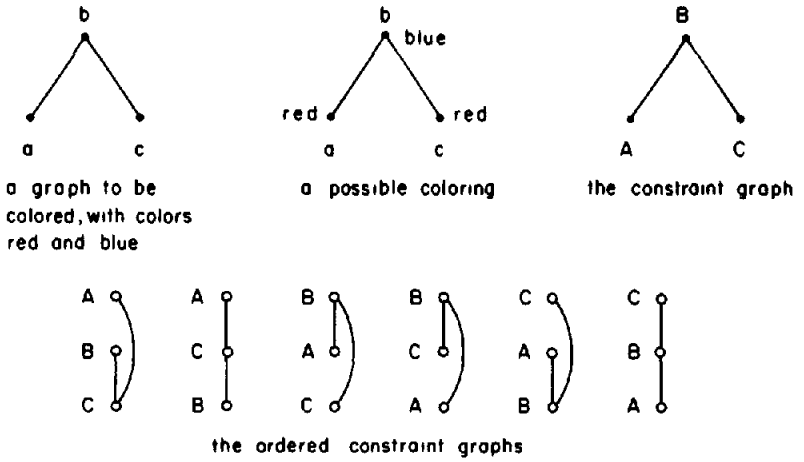

FIG 2 A graph coloring example

c can only be colored red. The resulting problem is 3-consistent but not 2-consistent. If we choose red for vertex b, there is no consistent choice for vertex a. However, if we have a consistent choice for any 2 vertices, there will always be a consistent choice of color for the third.)

Backtracking occurs when an instantiation chosen during a backtrack search, consistent with all previous choices, must be discarded later in the search when no consistent instantiation can be made for a variable at a lower level in the backtrack tree. A search may, of course, be fortuitously backtrack-free. We say that a given vertical search order is *backtrack-free* if it guarantees a backtrack-free search regardless of the horizontal order of search.

Given these definitions, the following relationships emerge among connective structure (width), contextual structure (consistency), and vertical order.

THEOREM 1. *Given a constraint satisfaction problem:*

(1) *A vertical search order is backtrack-free if the level of strong consistency is greater than the width of the corresponding ordered constraint graph.*
(2) *There exists a backtrack-free vertical search order for the problem if the level of strong consistency is greater than the width of the constraint graph.*

PROOF. In instantiating any variable v we must check consistency requirements involving at most j other variables, where j is the width at the node. If the consistency level k is at least $j + 1$, then given prior choices for the j variables, consistent among themselves, there exists a value for v consistent with the prior choices. □

3. *Width*

We can determine the width of a constraint graph and find an ordered graph with this width, utilizing an ordered-search algorithm [10]. Employ a search tree where each node at level i represents a choice for the ith node (variable) of an ordered constraint graph. Each branch represents the nodes of a (partial) ordered constraint graph. Repeatedly expand a node at the end of a minimal width branch by adding as its children the variables not yet used in that branch until a branch using all the variables is completed. This will represent a full minimal-width ordered constraint graph for the problem. A more efficient algorithm can be developed using an alternative characterization of width in terms of a concept I call "linkage."

We define the *linkage* of a subgraph as the maximum n such that every node in the subgraph is connected to at least n other nodes, that is, has degree at least n. We then have the following connection between linkage and width.

THEOREM 2. *The width of a constraint graph is equal to the maximal linkage of its subgraphs.*

PROOF. We first show that width $\geq n$ iff there is a subgraph with linkage $\geq n$. If a subgraph has linkage $\geq n$, then whichever node of the subgraph is last in the ordered constraint graph has width $\geq n$; thus the graph has width $\geq n$. If there is no such subgraph, we can demonstrate an ordered graph with width $< n$ as follows. Consider the graph as a whole. Its linkage is $< n$, so there must be at least one node that is connected to $< n$ others. Make such a node the last node of an ordered constraint graph. The width at that node will be $< n$. Now consider the remaining nodes of the graph. Again there must be at least one node that is connected to $< n$ others, as no subgraph has linkage $\geq n$. Choose such a node as the next to last node of the ordered graph. Again the width at that node will be $< n$. Continue until all nodes have been placed in an ordered graph. The width of this ordered graph will be $< n$. Thus the width of the constraint graph itself is $< n$.

If the maximal linkage of the subgraphs is n, there is at least one with linkage n, which we now see implies that the width is $\geq n$. Furthermore, since there is no subgraph with linkage $\geq n + 1$, the width is $< n + 1$. Therefore the width is also n.

If the width is n, there exists a subgraph with linkage $\geq n$; thus the maximal subgraph linkage is $\geq n$. If the maximal linkage were $> n$, there would be a subgraph with linkage $\geq n + 1$, and the width would be $\geq n + 1$. But the width is n, and thus the maximal subgraph linkage is also n. □

If we spot a highly linked subgraph, that at least provides a lower bound for the width. If we know the width, the above proof provides us with a method for finding an ordered constraint graph with that width, based on the knowledge that there is no subgraph with linkage greater than the width. I will restate that simple method explicitly as an algorithm. The outer loop has n passes; at each pass we examine at most i nodes.

For a graph with width k, to find an ordered graph with that width:
Repeat for i from n to 1 by -1.
    Find a node connected to $\leq k + 1$ others
    (Its existence is implied by a maximal subgraph linkage of k. If there is more than one, any one will do.)
    Remove the node from the graph, along with any edges connected to it. Make the node the ith node of the ordered graph.

The characterization of the width of a constraint graph in terms of subgraph linkage provides an alternative method for finding the width. The following algorithm determines the maximal subgraph linkage and finds the maximum subgraph with that linkage.

To determine maximal subgraph linkage
Remove from the graph all nodes not connected to any others. Set $k = 0$
Do while there are nodes left in the graph
    Set k to $k + 1$
    Do while there are nodes not connected to more than k others:
        Remove such nodes from the graph, along with any edges connected to them

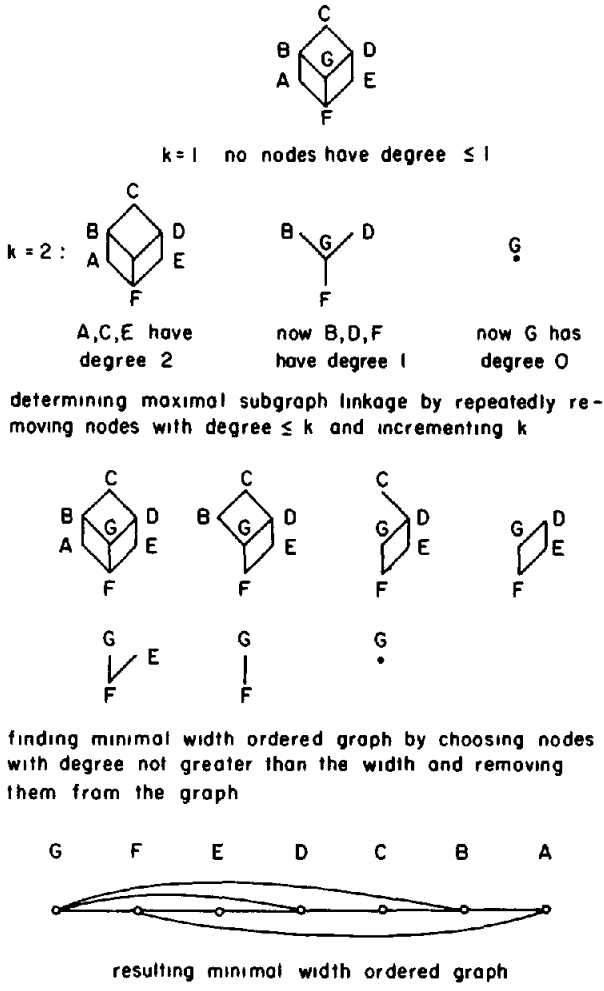

FIG 3 Maximal subgraph linkage and width.

Upon completion of the algorithm, k will be the maximal subgraph linkage. The nodes left before the last execution of the outer loop will constitute the largest subgraph with that linkage. Since the maximal subgraph linkage k must be $\leq n - 1$, the outer loop has at most $n - 1$ iterations. The inner loop examines at most n nodes on any one pass.

Taken together, the two algorithms above provide an alternative to the ordered-search algorithm sketched at the beginning of this section, while avoiding the potential $n!$ explosion of the ordered search.

As an application, consider the scene labeling problem. Waltz [14] attacked this problem with a filtering algorithm that achieves 2-consistency, eliminating many impossible labelings in the process, followed when necessary by a final search phase.

Now at first glance one might assume that picture graphs had a width of 3, as there are so many vertices which are connected to, and thereby constrained by, three others. Even in the usual view of a simple cube, over half the vertices are connected to three others. Nevertheless, using the techniques of this section, we quickly discover that the maximal subgraph linkage of the cube picture graph, and therefore the width, is 2, and we can derive an ordered graph with that width (see Figure 3). Even more complex scenes, some involving vertices connected to four others, may still only

have a width of 2: for example, two cubes lined up on top of one another, with a four line vertex where they touch in the center of the scene. (A view looking down on a pyramid would be an example of a picture graph with width three.)

Nevertheless, even width-2 demands strong 3-consistency to be able to guarantee no backtracking, and Waltz' filtering algorithm is a strong 2-consistency algorithm. Thus the consistency level is not enough by itself to guarantee a backtrack-free search order.

4. *Tree-Structured Constraint Graphs*

We will consider by way of illustration and application the case in which the constraint graph is a tree. For example, it might be an organization chart to be filled in or a hierarchical program structure. In either case the constraints are that connected components must be "compatible."

The three standard methods of tree traversal—inorder, preorder and postorder—provide a linear ordering of the nodes which can serve as a basis for an ordered constraint graph. The constraint graphs resulting from postorder or inorder traversal have width greater than 1, in general; however, preorder traversal results in a width of only 1. A breadth-first traversal also results in an ordered graph with width 1.

The fact that a tree has a width of 1, and has that width regardless of the branching factor, is perhaps a bit counterintuitive. At first glance a binary tree structure, for example, might seem to require a width of at least 2, or perhaps 3. Indeed, a randomly chosen ordering of the nodes would be likely to have higher width.

Consider now by way of illustration a graph coloring problem where the graph is a binary tree. If we allowed two colors, the problem is 2-consistent but not 3-consistent. Given a coloring for any one node, we can find a consistent coloring for any one other. However, if we color two siblings with different colors, there remains no way to color their parent. If we use a preorder traversal for our vertical search order, the constraint graph will have a width of 1, less than the consistency level, and a backtrack-free search is assured. However, a postorder traversal could lead to backtrack trouble. If, on the other hand, we allow three colors, the problem is (strongly) 3-consistent, and postorder traversal will not lead to backtracking either.

Clearly, the width of a constraint graph is 0 iff there are no constraints—the graph is a collection of unconnected points. We can characterize graphs of width 1 in terms of trees.

THEOREM 3. *A connected constraint graph (with more than one node) has width* 1 *iff it is a tree. More generally, a constraint graph has width* ≤ 1 *iff it is a forest.*

PROOF. A forest is a graph with no cycles [1]. I showed in the preceding section that the width of a graph is equal to the maximal linkage of its subgraphs. I claim that the maximal linkage of the subgraphs is ≤ 1 iff the graph has no cycles.

If the graph has cycles, the nodes of any cycle will generate a subgraph with linkage at least 2.

If there is a subgraph with linkage ≥ 2, we can use it to construct a cycle. Start with any node n in the subgraph. Choose a node n' connected to it. All nodes are connected to at least two others in the subgraph, so n' must be connected to at least one other node besides n; choose one and call it n''. Repeat the following procedure until a cycle is found, starting with n'' as the current node: The current node must be connected to another node in addition to the previous node chosen. If it is connected back to another already chosen node, we have found a cycle. If not, choose one of the remaining nodes which it is connected to as the new current node. As the number of nodes is finite this process will terminate and produce a cycle. □

Trees then not only have width 1 (aside from the trivial tree consisting merely of a root node); they are the only connected constraint graphs with width 1. Strong 2-consistency is sufficient to guarantee the existence of a backtrack-free vertical search order for constraint problems whose constraint graphs are forests.

As another example, recall the toy problem introduced in Section 1: Given three sets of numbers, $X = \{5, 2, 3, 6\}$, $Y = \{2, 4, 6, 10\}$, $Z = \{5, 2, 4, 6\}$, choose one number from each set such that the number chosen from Z divides both the others. The problem is 2-consistent. The constraint graph can be regarded as a simple tree structure with Z at the root. A preorder traversal will provide an ordering of width 1. That ordering—Z, X, Y—will thus be backtrack-free.

5. *Graph Theory Connections*

The constraint-graph concept permits us to make use of standard graph-theoretic results. For example, the following theorem comes easily.

THEOREM 4. *A simple planar constraint graph has width* ≤ 5.

PROOF. Any simple planar graph has a vertex of degree ≤ 5 [1]. Utilize this vertex as the last node of an ordered constraint graph. The width of that node is ≤ 5. The remaining vertices form a subgraph; it too must have a vertex of degree ≤ 5. Use that vertex as the next to last node of the constraint graph. Continue in this fashion to build an ordered constraint graph of width ≤ 5. □

For the graph coloring problem, the (strong) consistency level will obviously be greater than or equal to the number of colors available. Thus from Theorems 1 and 4 we can conclude that for the coloring problem on any planar graph, given six colors, there exists a backtrack-free search order.

This observation can be generalized to provide a bound on the chromatic number of a graph, the minimum number of colors required to color the graph. Since the consistency level is greater than or equal to the number of colors available, if the number of colors is greater than the width, the consistency level is greater than the width, and a coloring can be obtained (with a backtrack-free search). In particular, the graph can be colored with width + 1 colors. Thus width + 1 is an upper bound on the chromatic number of a graph.

An equivalent result was obtained by Szekeres and Wilf [12] involving the minimum vertex degree of subgraphs, a concept clearly equivalent to what I have called linkage. This result was obtained as well by Matula [8]. Matula also employed a concept called "degree decomposition sequence," closely related to what I have called width, and proved an analog of Theorem 2 in terms of degree decomposition sequence and minimum vertex degree. Both Matula and Szekeres and Wilf have methods for determining the value of the width analog for a given graph and for obtaining a corresponding ordering of the vertices and a coloring within the established bound. These results are summarized in [9], where a concept of "sequential coloring" is described which is essentially equivalent to a backtrack-free vertical search order in the graph coloring context. This paper serves to generalize this graph coloring work to the larger context of constraint satisfaction (although as it happens my work arose from concerns with constraints and backtracking, and I only belatedly became aware of these related developments in graph theory).

6. *Conclusions*

In theory, the algorithm for determining a minimal-width path in Section 3, together with the algorithm for obtaining any level of strong k-consistency found in [4], should

permit us to achieve backtrack-free search for any problem. However, not only are these algorithms themselves time consuming, they may need to be applied a number of times, as the algorithm for achieving consistency may well alter the connective structure, and the width, of the problem, as it makes explicit constraints previously only implied by other constraints. Thus we need to determine width, achieve a consistency level one greater than the width, and repeat this procedure until the width does not change.

I am therefore not treating this method as a practical approach in general. However, the insights we have gained here may provide some heuristic guidance in the choice of vertical search order. They may also guide us in the structural design of constraint problems, where we have a choice in which specific pattern of constraints we use to embody the problem.

Specific structures or problems may yield to a more complete formal analysis, as is the case for tree structures. In a larger context, my hope is that an understanding of the limiting case of backtrack-free search will abet our assault on the general problem of backtrack search analysis.

REFERENCES

1 BERGE, C *Graphs and Hypergraphs.* North Holland, London, 1973.
2 BITNER, J.R., AND REINGOLD, E M Backtrack programming techniques. *Commun ACM 18,* 11 (Nov. 1975), 651–656.
3. BOBROW, D G , AND RAPHAEL, B New programming languages for artificial intelligence research *Comput Surv. 6,* 3 (Sept. 1974), 153–174.
4 FREUDER, E.C. Synthesizing constraint expressions. *Commun. ACM 21,* 11 (Nov 1978), 958–966
5 GASCHNIG, J Experimental case studies of backtrack vs Waltz-type vs new algorithms for satisficing assignment problems. Proc. 2nd Nat Conf of the Canadian Society for Computational Studies of Intelligence, Toronto, Ontario, 1978, pp. 268–277
6 KNUTH, D E. Estimating the efficiency of backtrack programs *Math. Comput 29* (Jan 1975), 121–136.
7 MACKWORTH, A.K Consistency in networks of relations *Artif. Intell. 8* (1977), 99–118
8. MATULA, D.W A min-max theorem for graphs with application to graph coloring *SIAM Rev 10* (1968), 481–482
9 MATULA, D W , MARBLE, G , AND ISAACSON, J.D Graph coloring algorithms In *Graph Theory and Computing,* R C Read, Ed , Academic Press, New York, 1972, pp 109–122
10 NILSSON, N J. *Problem-Solving Methods in Artificial Intelligence* McGraw-Hill, New York, 1971
11. PURDOM, P W. Tree size by partial backtracking. *SIAM J. Comput 7* (1978), 481–491.
12 SZEKERES, G., AND WILF, H S. An inequality for the chromatic number of a graph *J Comb Theory 4* (1968), 1–3.
13 TENENBAUM, J M , AND BARROW, H.G IGS A paradigm for integrating image segmentation and interpretation. In *Pattern Recognition and Artificial Intelligence,* C.H. Chen, Ed , Academic Press, New York, 1976, pp 472–507
14 WALTZ, D.L Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision,* P H Winston, Ed , McGraw-Hill, New York, 1975, pp 19–91.
15 WELLS, M B. *Elements of Combinatorial Computing* Pergamon Press, New York, 1971.
16. WIRTH, N *Algorithms + Data Structures = Programs* Prentice-Hall, Englewood Cliffs, N J , 1976

RECEIVED AUGUST 1979, REVISED AUGUST 1980, ACCEPTED SEPTEMBER 1980