A Sufficient Condition for Backtrack-Bounded Search

EUGENE C. FREUDER

University of New Hampshire, Durham, New Hampshire

Abstract. Backtrack search is often used to solve constraint satisfaction problems. A relationship involving the structure of the constraints is described that provides a bound on the backtracking required to advance deeper into the backtrack tree. This analysis leads to upper bounds on the effort required for solution of a class of constraint satisfaction problems. The solutions involve a combination of relaxation preprocessing and backtrack search. The bounds are expressed in terms of the structure of the constraint connections. Specifically, the effort is shown to have a bound exponential in the size of the largest biconnected component of the constraint graph, as opposed to the size of the graph as a whole.

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*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*backtracking*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Biconnected component, constraint consistency, constraint networks, constraint satisfaction, relaxation, width

1. *Introduction*

Constraint satisfaction problems involve finding values for a set of variables that simultaneously satisfy a set of constraints. I consider here problems where there is a finite set of discrete values for each variable and the given constraints are binary relations on these sets. I also assume that at least one solution exists. Such problems are typically solved using backtrack search [1], relaxation [4], or some combination of the two.

Backtrack search is a basic control structure in computer science and finds particular application in artificial intelligence. As the search process is, in general, exponential in the number of variables, it is particularly helpful to have some means of obtaining bounds on the effort required for individual problems or classes of problems. Knuth [8] developed a sampling technique for estimating the search effort for a specific problem. Brown and Purdom [2] investigate average time behavior of search techniques over sets of random problems. Freuder [5] characterizes a class of problems that permit search to be successfully carried out without backup, the limiting case of "backtrack-free" search.

In this paper I generalize the backtrack-free result to "backtrack-bounded" search. Initially it might appear that once any backtracking is permitted, the "transitive"

This paper is based on 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.

nature of the constraints will lead to a completely general situation. However, we can distinguish degrees of backtracking. When we reach a level h in the backtrack search tree, we have chosen values for h variables that meet the constraints that affect those variables. I term a search *b-level backtrack-bounded*, or *b-bounded*, if for any level h in the search tree ($h < n$, the number of variables), we can push forward to find a value at level $h + 1$ by considering possible values at level $h + 1$ and at most reconsidering already chosen values at $b - 1$ higher levels. Backtrack-free search is then the special case of 1-level backtrack-bounded search.

In Section 2 I define measures of the constraint structure; j-width, which reflects the interconnection of constraints, and (i,j)-consistency, which reflects the extensibility of partial solutions. Though the distinction may be overused, I like to think of width as reflecting the "syntactic" structure and consistency the "semantic" structure of the constraints. A simple relationship between those two structural measures will suffice to guarantee a level of backtrack-boundedness. In Section 3 this relationship is used to provide bounds on the amount of search effort required that are based on the structure of the constraint graph. In particular it is shown that the search problem can be solved in time exponential in the size of the largest biconnected component of the constraint graph.

2. *Width, Consistency, and Backtrack-Boundedness*

The concept of j-width is a generalization of width [5] and (i,j)-consistency is a generalization of k-consistency [4], which in turn traces its roots back through the relaxation work of Mackworth [9], Montanari [10], and Waltz [14]. See also early work by Fikes [3] and Ullman [13]. It should be especially noted that (i,j)-consistency turns out to be closely related to Haralick and Shapiro's ϕ_{KP} operator [6], which in turn generalizes the ϕ_P operator of Haralick et al. [7]. In Section 2.1 j-width and (i,j)-consistency are defined and used to provide a generalization of the sufficiency condition for backtrack-free search established in [5]. In Section 2.2 methods are discussed for determining j-width and establishing (i,j)-consistency.

2.1. Relating Width and Consistency to Boundedness. A constraint satisfaction problem can be represented as a *constraint graph*, where the vertices correspond to the variables and the edges to the binary constraints. The constraints specify which pairs of values are allowed. If all pairs are allowed, we say that there is no constraint. If we place the vertices in a linear order, we obtain an *ordered constraint graph*. Such an ordering corresponds to the levels of a backtrack search tree at which the variables are instantiated. The order in which a backtrack process chooses variables to instantiate is termed the *vertical order* of search, corresponding to depth in the backtrack tree.

The *width* at a vertex v in an ordered constraint graph is the number of vertices preceding v in the ordering that are linked to v by an edge. The width of a group of j consecutive vertices in an ordered constraint graph is defined here as the number of vertices preceding the group in the ordering that are linked to any of the j vertices in the group. The *j-width* at a vertex v is the minimum, for $k = 1$ to j, of the width of the k consecutive vertices up to and including v. Thus the width at vertex v is the special case of 1-width. The j-width of an ordered constraint graph is the maximum j-width at all vertices in the ordering. The j-width of a constraint graph is the minimum j-width of all orderings of the constraint graph.

The property of *k-consistency* specifies that if we choose values for any $k - 1$ variables that satisfy the constraints on these variables, along with a kth variable, we will then be able to find a value for the kth variable such that all the constraints on the k variables will be satisfied by the k values taken together. Strong k-consistency is k'-consistency for all $k' \leq k$. (See [5] for an example of k-consistency without strong k-consistency.) I generalize now to *(i,j)-consistency*: Given values for i variables, satisfying the constraints on those variables, and given any other j (or fewer) variables, it will be possible to find values for those j variables such that the $i + j$ values taken together satisfy all constraints on the $i + j$ variables. Thus k-consistency is the special case of $(k - 1,1)$-consistency. Strong (i,j)-consistency is (i,j)-consistency for all $i' \leq i$.

THEOREM 1. *Given a constraint graph for a constraint satisfaction problem, there exists a vertical search order that guarantees j-bounded backtrack search if the graph is strongly (i,j)-consistent for i equal to the j-width of the graph.*

PROOF. If the j-width is w, there is an ordered constraint graph with that width, and we will use a vertical search order that corresponds to that ordering of the constraint graph. Suppose we wish to proceed from vertex v to vertex v' in the search. The j-width is w, so for some $j' \leq j$ the width of the j' variables up to and including v' is $w' \leq w$. In other words, those j' variables are related to only w' variables already instantiated at higher levels of the search.

Since the graph is strongly (i,j)-consistent for $i = w$, we know it is possible to find values for the j' variables consistent with those already chosen for the w' to which they are related. Thus it is possible to proceed to the level of vertex v' by finding an appropriate instantiation of the j' variables up to and including v'. Actually finding such an instantiation can at most involve the corresponding j' levels in the backtrack tree. Q.E.D.

When $j = 1$, we have the condition for backtrack-free search derived in [4].

Note that it is possible for a graph to be (i,j)-consistent but not (i',j')-consistent, where $i + j = i' + j'$. Consider, for example, three variables V_1, V_2, V_3 with domains of possible values $\{a_1, a_2\}$, $\{b_1, b_2\}$ and $\{c_1, c_2\}$, respectively, and constraints C_{13} (on V_1 and V_3) $= \{(a_1c_1), (a_2c_2)\}$, $C_{12} = \{(a_1b_1), (a_1b_2), (a_2b_1)\}$, $C_{23} = \{(b_2c_1)$ $(b_1c_2)\}$. (For example, a_1 for V_1 and c_1 for V_2 is allowed, but not a_1 for V_1 and c_2 for V_2.) This is (1,2)-consistent but not (2,1)-consistent. Given a value for any one variable, we can find consistent values for the other two ($a_1b_2c_1$ and $a_2b_1c_2$ are solutions). However, given a_1 and b_1 for V_1 and V_2, there is no consistent value for V_3 (c_1 is consistent with b_1, c_2 with a_1).

The special case of $(1,j)$-consistency (as opposed to $(i,1)$-consistency, which is equivalent to the old notion of k-consistency) will prove to be particularly interesting, as we shall see in Section 3.

2.2 DETERMINING j-WIDTH AND OBTAINING (i,j)-CONSISTENCY.

Consistency and width are related in Theorem 1. I now indicate how the j-width of a graph can be determined and a given consistency level obtained. The methods discussed are straightforward, but more efficient methods may exist, as least in special cases (see [5] for a more efficient algorithm for 1-width).

A branch-and-bound process can be used to determine j-width [11]. The aim is to build an ordered constraint graph of minimal j-width. Each branch of the branch-and-bound search tree will represent an initial portion of an ordered constraint graph or the complete graph. Conduct a depth-first search choosing an unused graph vertex at each level on the way down and backing up to try another whenever the cost of a branch reaches that of the least cost-complete solution found so far. The complexity of this process is potentially factorial in the number of variables n, but the "bounding" in branch and bound may save a great deal. Of course, the j-width of a particular ordered constraint graph can be computed in time linear in n.

In an earlier paper [4] I presented an algorithm for obtaining k-consistency. Here I will describe this algorithm informally and argue that it will also obtain strong (i,j)-consistency, for all i,j such that $i+j=k$. The algorithm builds up a "constraint network," starting with one in which nodes corresponding to each binary constraint contain every permissible pair of values and nodes corresponding to each variable domain, that is, constraint, contain all permissible variable values. Binary constraint nodes are linked to the nodes for the two variables they constrain. A relaxation "propagation" process then removes inconsistent variable values and pairs containing them from the nodes, resulting in a 2-consistent network. (The basic idea behind constraint relaxation is that if a constraint shows that a given value, V, for variable V is inconsistent with all values for another variable, then v can be eliminated. This elimination may propagate: the value eliminated may have been the only remaining value for V consistent with some value v' for variable V', eliminating v' in turn.) Now nodes are added corresponding to triples of variables and linked to the binary nodes that involve pairs from the triple. Relaxation propagates through the network. Ternary nodes that do not contain all possible triples represent ternary constraints. The process is repeated until k-ary nodes have been added and "relaxed." The resulting network is shown to be k-consistent, that is, $(k-1,1)$-consistent. The k-tuples at the k-ary nodes represent consistent instantiations of the k variables.

If $k-1$ values satisfy all constraints on $k-1$ variables including new constraints that may be added by the algorithm, then for any kth variable we can find a kth value to make up a consistent k-tuple. The k-tuples at the k-ary nodes represent consistent instantiations of the k variables.

This network will also be (i,j)-consistent for any i,j such that $i+j=k$. Consider the node corresponding to the i-ary constraint, and any i-tuple at that node. We need to show that there is a k-tuple at the k-ary node that contains that i-tuple. This i-ary node is linked to the k-ary node by a chain passing through an $(i+1)$-ary node, then an $(i+2)$-ary node, and so on until the $(i+j)$-ary node is reached, where $i+j=k$. The $(i+1)$-ary node must include an $(i+1)$-tuple that contains the desired i-tuple; otherwise the relaxation process would have removed the i-tuple from the i-ary node. In similar fashion we can proceed along the chain, retaining the original i-tuple until we reach the k-ary node. (If $i=0$, a consistent k-tuple exists, since we are assuming the problem is solvable.) The k-consistency algorithm obtains strong k-consistency and thus strong (i,j)-consistency. The k-consistency algorithm has a time bound exponential in k (R. Seidel, personal communication).

3. *Bounds on Backtrack Search Effort*

In general, the upper bound on backtrack search effort is exponential in n, the number of variables. If search is j-bounded, then we clearly have an upper bound for the search effort that is only exponential in j (times a factor of n). We can advance each level deeper into the tree by solving a backtrack problem involving at most j variables.

If the j-width of a constraint graph is i and the graph is strongly (i,j)-consistent, j-bounded search is possible by Theorem 1. This suggests that we consider determining the j-width, i, of a constraint graph or a particular ordered constraint graph and obtaining strong (i,j)-consistency as a preprocessing step to ensure j-bounded backtrack search. Unfortunately, achieving (i,j)-consistency may alter the width. (For example, suppose there is no binary constraint provided initially between variables X and Y—no pair of values is directly ruled out. A pair, a from X and b from Y, may still lead to higher level inconsistency—suppose no value from variable Z is consistent with both a and b. This impediment to (2,1)-consistency can be removed by incorporating a restriction against the a,b pair in a binary constraint on X and Y, but this new constraint may change the width.) In 3.1, however, we consider the special case of $(1,j)$-consistency, which can be obtained without altering the width, and in 3.2 we apply this case to separable graphs.

3.1 $(1,j)$-CONSISTENCY

THEOREM 2. *There is a vertical search order that permits a constraint satisfaction problem to be solved in time exponential in $b + 1$, where b is the least j such that the j-width of the constraint graph is* 1.

PROOF. We can achieve strong $(1,j)$-consistency without altering the width. Apply the algorithm referenced in Section 2.2 to obtain a $(1,j)$-consistent network. From the original constraint graph and the newly obtained network construct a new graph as follows. For each variable, take each value in the original graph that has been eliminated in the new network. Remove that value from the variable vertex in the graph, and remove all pairs containing it from the constraints associated with the edges of the graph. (While the relaxation algorithm has already eliminated the value, it has done so in the network data structure it builds and operates on; we are now reflecting that change in the original constraint graph.) The resulting constraint graph will certainly not eliminate any possible solutions. The values remaining admit consistent j'-tuples ($j' \leq j$) that will not have been eliminated. Thus we have $(1,j)$-consistency. Strong (i,j)-consistency follows immediately, since i is only 1 here. The width is unchanged: no new constraints are added. If before no constraint existed between two variables, that is, all pairs of values were possible, removing pairs involving a value for one variable, *and* that value for the variable, still leaves all pairs possible.

Achieve strong $(1,b)$-consistency maintaining the b-width as 1. Then b-bounded backtrack search is possible by Theorem 1. $(1,b)$-consistency can be achieved in time exponential in $b + 1$, b-bounded backtrack in time exponential in b. Q.E.D.

3.2 SEPARABLE GRAPHS. A vertex whose removal disconnects a connected graph is called an articulation point. A graph with an articulation point is separable; without one it is biconnected. The biconnected components of a graph are the maximal biconnected subgraphs [11].

THEOREM 3. *A constraint satisfaction problem can be solved in time exponential in b, where b is the number of vertices in the maximal biconnected component of the constraint graph.*

PROOF. We show that the $(b - 1)$-width of the constraint graph is 1, and therefore by Theorem 2, a vertical search order exists that permits search within the specified bound. Indeed, we construct a vertical search order corresponding to an ordered constraint graph with $(b - 1)$-width 1. The value of b can be determined, and this vertical search order with $(b - 1)$-width 1 constructed, in time $O(V + E)$ for V vertices and E edges in the graph. Thus a solution to the constraint satisfaction problem can be found within the specified time bound by successively obtaining this vertical search order, achieving $(1,b - 1)$-consistency and conducting $(b - 1)$-bounded backtrack search.

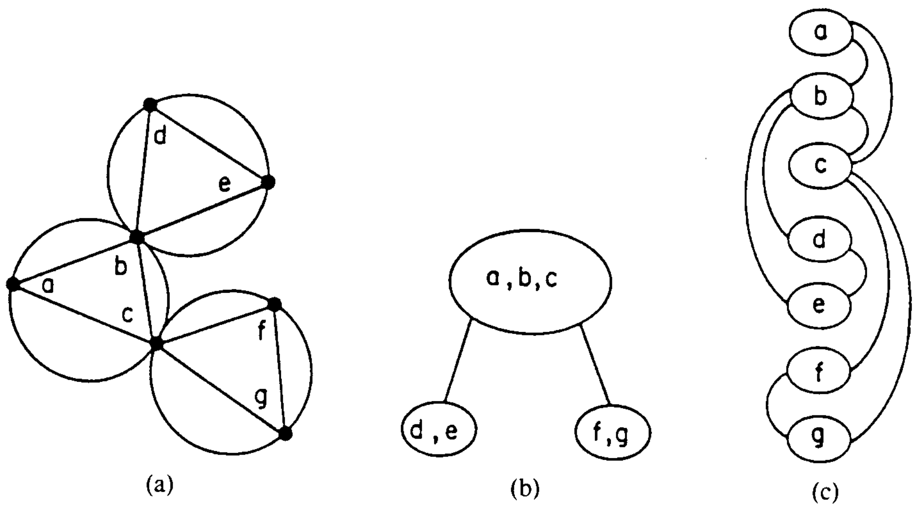

FIG. 1. Obtaining an ordered constraint graph with $(b - 1)$-width 1. (a) Find the biconnected components of the graph. (b) Form a corresponding tree structure. (c) Traverse the tree in preorder to generate an ordered constraint graph (which divides into blocks corresponding to the tree nodes).

We shall now obtain the desired vertical search order. The process is summarized in Figure 1. We assume the original constraint graph is connected; if not we can simply deal with each piece separately. We begin by finding the articulation points and biconnected components of the graph. This can be done in time $O(V + E)$ [12].

Now we can construct a tree whose nodes are the biconnected components as follows:

```
Choose one of the components to be the root of the tree and put it in a queue $Q$
While $Q$ is not empty:
  Remove a component $c$ from $Q$
  For each component $d$ which shares an articulation point with $c$ and is not already in the tree:
    Make $d$ a child of $c$ in the tree
    Put $d$ in $Q$
```

This tree can clearly be constructed in $O(V^2)$ time. Consider the vertices of the constraint graph contained at the tree nodes corresponding to biconnected components. Assign the articulation points that are common to a parent and child in this tree to the parent node only. Note that this means that there are at most $b - 1$ vertices at any node except the root, which may contain b.

The edges in the tree correspond to edges in the graph from an articulation point in the parent to vertices in the children. I claim that these are the only edges in the graph between vertices in different nodes of the tree. Adding an edge to the tree corresponding to any other edge in the graph between vertices in different tree nodes would create a cycle, and vertices at every node on that cycle could be combined to generate a biconnected subgraph, contradicting the use of maximal biconnected subgraphs in deriving the tree to begin with.

Now form a vertical search order by taking first the graph vertices at the root of our tree (in any order) and then proceeding to add the vertices we encounter at the tree nodes as we conduct a preorder traversal of the tree. Since this is a preorder traversal, by definition the parent will always be visited before the children. The resulting vertical search order can thus be divided into blocks, corresponding to the biconnected components, where vertices in one block only link outside that block to (at most) a single higher level vertex, an articulation point in a "parent" block.

If b is the size of the maximal biconnected component, the ordered constraint aph corresponding to this vertical search order has $(b - 1)$-width of 1. Recall that e number of vertices at any node in the tree we constructed is at most $b - 1$, cept for the root, which may contain b. Any vertex in the vertical search order rresponding to one from the root node of tree will be among the first b vertices the order. The $(b - 1)$-width of any such vertex can be at most 1. At any other rtex in the ordering we can include at most the preceding $b - 2$ vertices before countering the top of a block corresponding to a biconnected component. As we ve just seen, the vertices in that block link in the constraint graph to one higher rtex. Thus we have a $(b - 1)$-width of 1. Q.E.D.

If we can utilize a constraint graph that breaks up into separate unconnected mponents of maximum size k, it is obvious that the problem can be solved in ıe exponential in k. At first glance it might seem that if we allow any constraints tween the components, we risk search effort exponential in the number of riables, owing to the "transitivity" of the constraints. However, Theorem 3 ablishes that a limited form of communication, funneled through articulation ints of the graph, can be permitted while ensuring that the problem can still be ved in time exponential in the size of the largest component. If the size of the ıximal biconnected component of a constraint graph can be kept fixed at a ıstant b as the number of vertices increases, we can prevent the search effort m becoming exponential in the size of the problem.

KNOWLEDGMENT. I would like to thank Dov Harel for helpful discussion of s work.

ERENCES

BITNER, J. R., AND REINGOLD, E. M. Backtrack programming techniques. *Commun. ACM 18*, 11 (Nov. 1975), 651–656.

BROWN, C. A., AND PURDOM, P. W. JR., How to search efficiently. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence* (Vancouver, Canada), 1981, pp. 588–594.

FIKES, R. E. REF-ARF: A system for solving problems stated as procedures. *Artif. Intell. 1*, 1 (1970), 27–120.

FREUDER, E. C. Synthesizing constraint expressions. *Commun. ACM 21*, 11 (Nov. 1978), 958–966.

FREUDER, E. C. A sufficient condition for backtrack-free search. *J. ACM 29*, 1 (Jan. 1982), 24–32.

HARALICK, R. M., AND SHAPIRO, L. G. The consistent labeling problem: Part I. *IEEE Trans. Pattern Anal. Machine Intell. PAMI-1*. IEEE, New York, 2 (Apr. 1979), pp. 173–184.

HARALICK, R. M., DAVIS, L., ROSENFELD, A., AND MILGRAM, D. Reduction operations for constraint satisfaction. *Inf. Sci. 14* (1978), 199–219.

KNUTH, D. E. Estimating the efficiency of backtrack programs. *Math. Comput. 29* (Jan. 1975), 121–136.

MACKWORTH, A. K. Consistency in networks of relations. *Artif. Intell. 8* (1977), 99–118.

MONTANARI, U. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci. 7*, 2 (Apr. 1974), 95–132.

REINGOLD, E. M., NIEVERGELT, J., AND DEO, N. *Combinatorial Algorithms*. Prentice-Hall, Englewood Cliffs, N.J., 1977.

TARJAN, R. E. Depth-first search and linear graph algorithms. *SIAM J. Comput. 1* (1972), 146–60.

ULLMAN, J. R. Associating parts of patterns. *Inf. Control 9* (1966), 583–601.

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.

IVED SEPTEMBER 1982; REVISED JANUARY 1985; ACCEPTED APRIL 1985