TAKING ADVANTAGE OF STABLE SETS OF VARIABLES IN CONSTRAINT SATISFACTION PROBLEMS

Eugene C. Freuder and Michael J. Quinn*

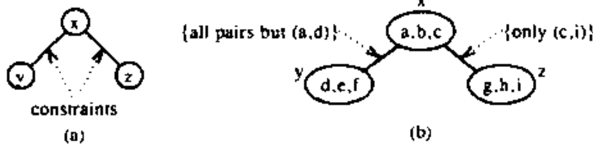
Department of Computer Science, University of New Hampshire, Durham, NH 03824

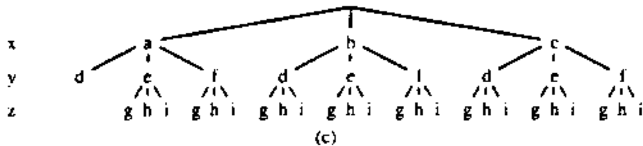
ABSTRACT

Binary constraint satisfaction problems involve finding values for variables subject to constraints between pairs of variables. Algorithms that take advantage of the structure of constraint connections can be more efficient than simple backtrack search. Some pairs of variables may have no direct constraint between them, even if they are linked indirectly through a chain of constraints involving other variables. A set of variables with no direct constraint between any pair of them forms a stable set in a constraint graph representation of a problem. We describe an algorithm designed to take advantage of stable sets of variables, and give experimental evidence that it can outperform not only simple backtracking, but also forward checking, one of the best variants of backtrack search. Potential applications to parallel processing are noted. Some light is shed on the question of how and when a constraint satisfaction problem can be advantageously divided into subproblems.

1 DIRECT INDEPENDENCE OF VARIABLES

Constraint satisfaction problems involve finding values for variables subject to constraints, or relations among the variables. Often these constraints are restricted to being binary relations between two variables, we shall consider binary constraints here. Standard back-Hack search can be used to solve such problems. In general the upper bound on the complexity of the search is exponential in the number of variables





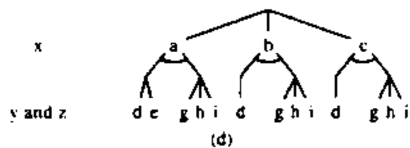


Figure 1 Taking advantage of the 'relative independence' of variables.

In this paper we propose taking advantage of the "relative independence'[1] of variables to ameliorate the search complexity. We term two variables directly independent if there is no direct constraint, between them, even if they may be indirectly related by a chain of constraints passing through intermediate variables.

The basic insight is illustrated by the example in Figure 1. The problem involves three variables, each with three possible values, and a constraint graph (Figure Ia, Ib) Links in the graph represent constraints; nodes represent variables Note that there is no direct constraint between variables *y* and *z*. If we approach the problem with a straightforward backtracking algorithm, we might have to examine almost 3 x *(3 x 3)* or 27 possible triples of values before hitting upon a solution in the rightmost branch of the search tree. More than 27 tests on pairs of values would be performed (Figure]c).

Now let. us take into account the relat ive independence of variables *y* and *z*. Having chosen a value for x, we can go ahead and choose a value for *y* and a value for *z* independently There will be at most 6 values to consider (3 for each) before we succeed or fail If we fail, we repeat the process for the next *x* value. At most we will perform 3x (3 + 3) or 18 tests on pairs of values (Figure Id).

In general we can partition constraint graphs into sets of "mutually independent" variables, where there is no direct constraint between any pair of variables in the set. Such a set is called a *stable set* in graph theory. Consider the constraint graph shown in Figure 2, for the problem of labeling the cube. The scene labeling problem Waltz., 1975 provides an application domain here, but those unfamiliar with it can regard the cube as an abstract constraint graph The graph in Figure 2b is simply a redrawing of the constraint graph in Figure 2a. Observe that the variables *F* and *C* are not joined by a constraint, nor are *B* and D, not *A* and *E*. There are no edges between variables at the same level in Figure 2b. Once a value has been chosen for (7, the values for *F* and C may be chosen mutually independently, then values for *D* and *D,* and so on. (Figure 2b is a generalized form of an "ordered constraint graph" [Freuder, 1982.)

Many interesting improvements on basic backtracking have been made Haralick and Elliott, 1980; Gaschnig, 1978]. Generally these improvements operate at a "microlevel," involving the relationships between individual values for variables, e.g., value *a* for variable x is inconsistent with value b for variable *y*. Our concerns are at a "macrolevel," involving the relationships between the variables themselves, e.g., there is no constraint between variables *x* and *y*. A more detailed experimental comparison with one of the best of the backtrack variations is made later

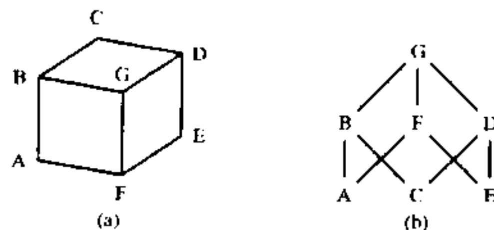


Figure 2: Labeling the cube.

* The authors' names appear in alphabetical order.

II PSEUDO-TREE SEARCH ALGORITHM

Given a constraint graph partitioned into stable .sets, we would like to take advantage of direct independence not only on one maximal stable set (or an approximation thereof), but on all the stable sets. For example, we would like to use the stable sets identified in Figure 2 as "levels" in a generalized backtrack algorithm that employed the basic "additive" insight of Section 1 at each level.

This can be done, up to a point, in a straightforward way Proceeding down through the levels, each variable at a given level can be considered independently. However, a problem arises in backtracking. If two variables at one level have a constraint path down to a single variable at a lower level (as in Figure 2b), all combinations of values for the two higher level variables may need to be tried before finding one compatible with the lower level variable Thus, we again have a "multiplicative" rather than an "additive" effect.

Definition: A *pseudo-tree* is a rooted tree *T* *(V. E)* augmented with zero or more additional edges /," such that all edges between some vertex *v* and vertices closer to the root be along the path in *T* from *v* to the root The level of a variable in a pseudo-tree is its level in the underlying rooted tree

A pseudo-tree structured constraint graph supports an algorithm that avoids the "multiplicative backtracking" problem cited above. It also makes it easy to determine what variables need to be modified during backtracking Thirdly, the pseudo-tree property makes the constraint satisfaction problem more amenable to solution on a multiprocessor, because a processor could try to assign values to *a* variable and all its descendenis without having to communicate with other processors Freuder and Quinn, 1985

Pseudo-Tree Search Algorithm

```
m -- number of levels in pseudo-tree
size(i)--number of variables at level i
a(i,j)--jth variable at level i
{1,2,...choices(a(i,j))}  potential values of a(i,j)
value(a(i,j))  current value of a(i,j)
parent(a(i,j))  index of parent of a(i,j) at level i - 1

for i := 1 to m do
   for j := 1 to size(i) do
      value(a(i,j)) := 1 (* Initialize all variables *)
   endfor
endfor,
i := 1; (* i is current search level *)
while (1 <= i) and (i <= m) do
   (* Examine level i *)
   j := 1,
   repeat
      if value(a(i,j)) violates a constraint with an ancestor then
         while value(a(i,j)) violates a constraint do
            if value(a(i,j)) = choices(a(i,j)) then
               (* No more alternatives for a(i,j)  must backtrack *)
               j := parent(a(i,j));
               i := i - 1
               if i = 0 then exit outermost while loop endif
            endif;
            (* Former value leads to constraint conflict  try another *)
            value(a(i,j)) := value(a(i,j)) + 1
         endwhile;
         set value of all descendents of a(i,j) to 1
      endif;
      j := j + 1 (* Try next variable at level i *)
   until j > size(i);
   (* All constraints satisfied through level i  deepen search *)
   i := i + 1
endwhile;
if i > m then (* Solution found  it is stored in array value *)
else (* No solution *)
endif.
```

The following lemma proves that the pseudo-tree search algorithm has a complexity bound exponential in the number of levels in the pseudo-tree, rather than in the number of variables in the problem In the next section we present an algorithm for transforming an arbitrary constraint satisfaction problem into an equivalent "metaproblem" with a pseudo-tree constraint graph structure.

Lemma: Given a pseudo-tree *T* and A variables at level *m* in T, each variable capable of taking on *b* values, the search algorithm backtracks at most *bk* times to level *rn* before backtracking to level *rn* 1.

Proof: The first time the search algorithm reaches level m, the value of every variable at that level is initialized as low as possible. The search algorithm backtracks from level *rn* - 1 in *T* when there exists at that level a variable *v* that cannot be given a value that does not violate constraints with the values of ancestor variables in *T.* By the pseudo-tree property, all of the variables constraining *v* lie along a simple path from v to the root of *T.* In order to try all combinations of values that might lead to an allowable value of v, only vaiiables along this path need to have their values changed Hence every time the search backtracks to level *rn,* exactly one of the variables *w* at level *m* must have its value modified. The value of w cannot be decremented: all lowei values were previously held by *w* and led to a backtracking lower in the tree. Because it is only necessary to reset the value of *w* when one of its ancestors has its value incremented, the value of no variable at level m will lowei until the search backtracks from level m. Since there are k variables, each with b possible values, the maximum number of values variables at level *m* can take on before one variable has no more possible values is *bk,*

Theorem: Given a pseudo-tree *T* with *rn* levels, each level / containing A; variables capable of taking on *b,* values, the worst-case time complexity of our algorithm is $\Theta(\prod_{i=1}^{m} b_i k_i)$.

Corollary; Given a pseudo-tree T our algorithm can find one assignment of values to variables that satisfies the constraints, if one exists, but it cannot be used to find all solutions that satisfy the constraints.

III DERIVING PSEUDO-TREE METAPROBLEMS

A constraint graph can be subdivided into subproblems, where each subproblem involves satisfying a subset of the original variables The problem of satisfying all the subproblems simultaneously may then be regarded as a metproblem, with the subproblems as metavariables

It has long been recognized that partitioning a constraint, satisfaction problem into subproblems may simplify the problem. However, little guidance is available as to how and when to subdivide problems. One criterion for considering a problem subdivision is the felicity of the structure of the resulting metaproblem. For example, if the metaproblem has a tree structured constraint graph on the metavariables, then it c an be solved in time linear in the number of metavariables ;Mackworth and Freuder, to appear Here we present a method lor producing a metaproblem where the metavariables are organized into stable sets forming levels in a pseudo-tree constraint structure. Recall that the previous section presented an algorithm for pseudo-tree-struct ured problems, with a complexity bound exponential in the number of pseudo-tree levels.

The original problem is partitioned into a metaproblem as follows:

1 Find a cut set in the constraint graph; i.e., a set of vertices whose removal divides the graph into two or more unconnected subgraphs. The cut set S corresponds to S variables in the metaproblem; these- *metavariable**will form the first *\S* levels of the pseudo-tree The unconnected subgraphs temporarily become metavariables and are children of the variable in S deepest in the "tree "

2 Apply this process recursively to each of the children, terminating when the- subgraph being examined cannot be split further.

The algorithm will not necessarily produce an "optimal" pseudo-tree We would like to be certain of efficiently transforming a given constraint graph into a pseudo-tree structure which takes optimal (or nearly optimal) advantage of direct independence, This remains an area for f urt her work.

IV EXPERIMENTAL RESULTS

To experimentally verify the efficiency of this algorithm, we have used it to try to color graphs with three colors The performance of our algorithm is contrasted with two others. The first algorithm performs simple backtracking, where the order in which the variables are searched is randomly chosen. The second algorithm uses forward checking Haralick and Elliott. 1980], which Haralick and Elliot showed to be superior to several other variants of backtracking under certain experimental conditions The order in which variables are considered corresponds to a level-by-level traversal of the pseudo-tree.

The form of the test graphs is shown in Figure 3. Notice that we have created constraint graphs that have the pseudo-tree property, which avoids the previously-mentioned problem of efficiently transforming constraint graphs into pseudo-trees. We have tested the three algorithms on a large number of graphs. For each problem size ranging from 4 to 36 variables, we have generated 100 random pseudo-trees. We have run the three search algorithms on all 900 graphs, measuring the number of constraint checks made by each algorithm. Each result has been put into one of two categories, depending upon whether the graph is three-colorable or not. This is because unsuccessful searches require many more constraint checks on average. Figure 4 displays the mean number of constraint checks performed by the three algorithms for successful and unsuccessful searches of the various graphs. Not only does our algorithm outperform standard backtracking, it also performs substantially fewer constraint checks than the forward checking algorithm on both colorable and uncolorable graphs. (We have also computed the median number of constraint checks performed by the three algorithms Our algorithm outperforms standard backtracking and forward checking on unsuccessful searches and standard backtracking on successful searches. However, the median number of constraint checks performed by forward checking on successful searches is a few percent lower than the number performed by our algorithm.)

Certainly our search algorithm will not outperform other algorithms, such as forward checking, in all situations. The constraints must fit a certain pattern in order to benefit from our exploitation of direct independence. For example, if our methodology were applied to the eight-queens problem (placing eight queens on a chess board so that no queen can attack another), the pseudo-tree would have eight levels, and the search algorithm would degenerate into standard backtracking.

In fact, much of the experimental work on backtrack search has been done in the context of the eight-queens problem. This is actually a very specialised type of problem: the constraint graph is complete, all constraints are the same, and all variables have the same domain. Direct independence in a constraint graph is in a sense the opposite of completeness; it involves subsets of variables, which, far from forming complete subgraphs, form stable sets, where no variable is connected to any other.

V CONCLUSIONS

We have presented an algorithm tailored to solve constraint satisfaction problems on constraint graphs that have the pseudo-tree property. We have also described an algorithm for turning any constraint graph into a constraint'graph with the pseudo-tree property Experimental evidence indicates there exist constraint graphs for which our algorithm outperforms not only standard backtracking, but also forward checking.

ACKNOWLEDGMENTS

This paper is based in part upon work supported by the National Science Foundation under Grant MCS 8003307.

REFERENCES

[1] Freuder, E.C. 1982. A sufficient condition for backtrack-free search. J. ACM 29, 1, pp. 24-32.

[2] Freuder, E.C, and Quinn, M.J. 1985. Parallelism in an algorithm that takes advantage of stable sets of variables in constraint satisfaction problems. Tech Rep. 85-21, Dept. of Computer Science, Univ of New Hampshire.

|3j Gaschnig, J. 1978. Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing-assignment problems. *Proc. 2nd National Conf. of Canadian Society for Computational Studies of Intelligence,* Toronto, Ontario, July 19-21 pp 268-277.

[4] Haralick, R , and Elliott, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14, pp. 263-313.

|5I Waltz,D 1975. Understanding line drawings of scenes with shadows In *The Psychology of Computer Vision,* P.H. Winston, ed., McGraw-Hill, New York, pp. 19-91.

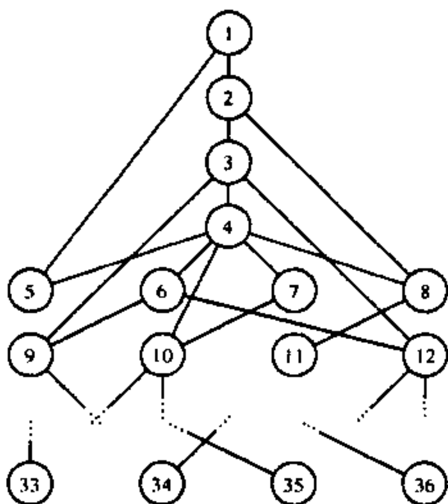


Figure 3: Form of the pseudo-trees used in the experiments.

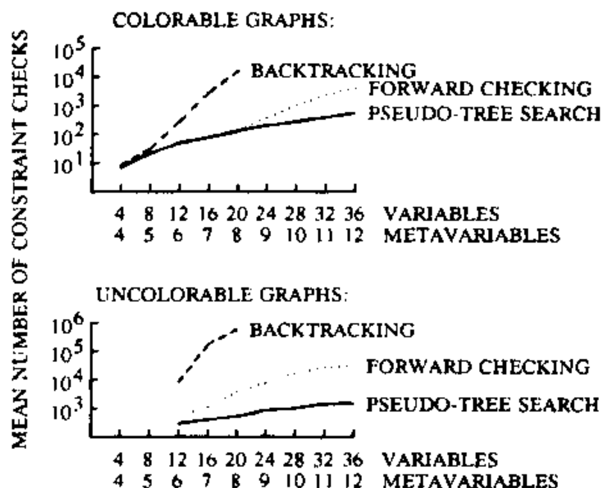


Figure 4: Experimental results.