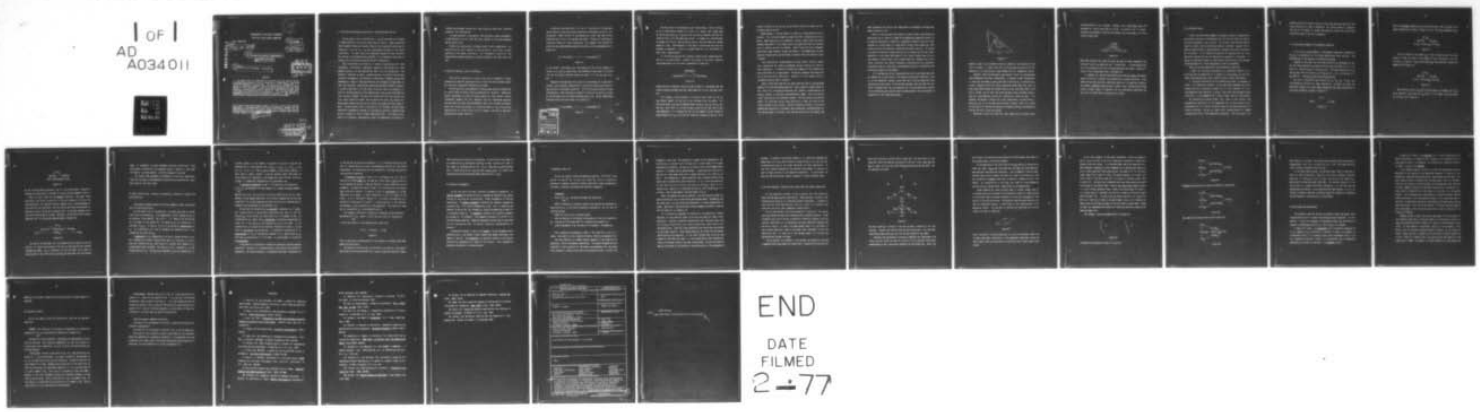
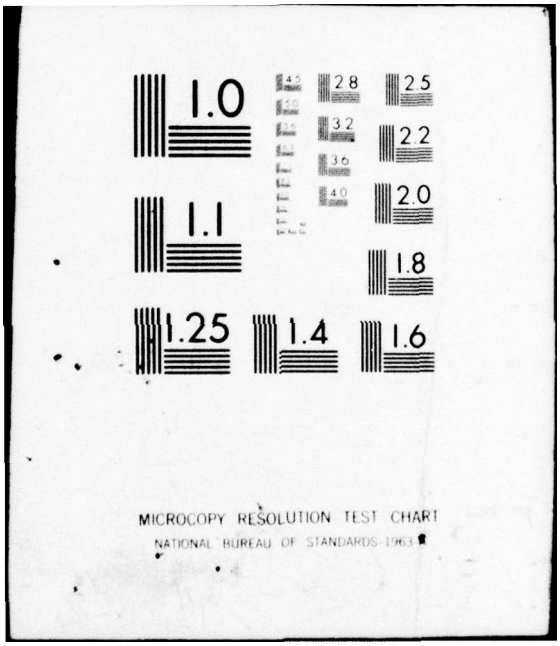
AD-A034 011 MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTE--ETC F/G 12/2
SYNTHESIZING CONSTRAINT EXPRESSIONS.(U)
JUL 76 E C FREUDER N00014-75-C-0643
UNCLASSIFIED AI-M-370 NL

1 OF 1
AD A034011



END

DATE FILMED
2-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12) B.S.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo 370

(11) July 1976

(14) AI-M-370

(12) 32p.

(6) SYNTHESIZING CONSTRAINT EXPRESSIONS.

(10) Eugene C. Freuder

(9) Memorandum rept.,

Abstract

An algorithm is presented for determining the values which simultaneously satisfy a set of relations, or constraints, involving different subsets of n variables. The relations are represented in a series of constraint networks, which ultimately contain a node for every subset of the n variables. Constraints may be propagated through such networks in (potentially) parallel fashion to determine the values which simultaneously satisfy all the constraints. The iterated constraint propagation serves to mitigate combinatorial explosion. Applications in scene analysis, graph theory, and backtrack search are provided.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

(15)

1473

407 483 1B

1. Satisfying simultaneous constraints: problem and applications

We are given a set of variables $X_1,\ldots,X_n$ and constraints on subsets of these variables limiting the values they can take on. These constraints taken together constitute a global constraint which specifies which sets of values $a_1,\ldots,a_n$ for $X_1,\ldots,X_n$ can simultaneously satisfy all the given constraints. In other words, the constraints define an n-ary relation. Our problem is to synthesize this relation, i.e. to determine those sets of values which simultaneously satisfy the set of constraints.

The simultaneous satisfaction of several constraints--call them properties, relationships, predicates, features or attributes--is a very general problem, with more applications than I can fully survey here. The essential technique we apply, iterated reduction of possibilities through constraint propagation, has analogues in many areas of computer science and mathematics. Many of these applications and analogues are described in [15], [11] and [24]. Applications range from data base retrieval (find all x, y and z such that x is a part and y is a part, and z is a supplier, x must be installed before y, and z supplies both x and y) (see also [12]) to scene analysis (segment the scene into regions such that sky regions are blue, grass regions are green, and car regions are shiny, sky regions are above grass regions and cars are not totally surrounded by either grass or sky). Of particular note is the work of J.R. Ullman, who has used constraint propagation methods in a variety of contexts, ranging from pattern recognition [19] to graph isomorphism [21]. The problem also admits of a graphical representation, where its resemblance to networks of

interacting processes conjures up a long history of other work, including recently: [8], [18] and [6].

In scene analysis, in particular, there has been a recent groundswell of applications, e.g. [1], [16], [9], [13]. Several of the latest examples can be found in [3].

Often we are only given, or choose to use, "local" constraints, i.e. constraints on small subsets of the variables, from which we must synthesize the global constraint. For fundamental results on the complementary problem, analysis of a global constraint into local ones, see [14].

2. Previous results: partial consistency

Constraints represented in network form may be propagated through (potentially) parallel algorithms which cut down the solution search space by ruling out inconsistent combinations of values.

The obvious brute force approach of testing every possible combination of values faces an equally obvious combinatorial explosion. Backtrack search techniques cut down the search space but often exhibit costly "thrashing" behavior [17] [2]. Mackworth [11] has interpreted previous work by Fikes [7], Waltz [23] and Montanari [15] as cutting down the search space and avoiding classes of thrashing behavior by eliminating combinations of values which could not appear together in any set satisfying the global constraint.

A network representation of a set of constraints is employed (restricted to unary and binary constraints, predicates on one or two variables). Each variable is represented by a node, and each binary predicate by a link or arc between two nodes. (Loops on a node may be viewed as binary or unary predicates.) For example, the problem of coloring a two node complete graph with one color can be represented as in Figure 2.1.

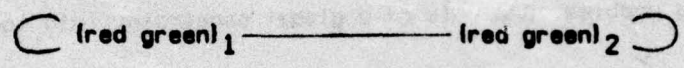


Figure 2.1

In the figure, $(red\ green)_1$ and $(red\ green)_2$ are the initial domains of values for X_1 and X_2 respectively, the predicate at each node is "colored red" and the binary predicate between the nodes is "is not the same color as".

Mackworth distinguishes three levels of inconsistency for a constraint network, which represent combinations of values which cannot participate in any solution to the global constraint. The first and most obvious is node inconsistency. Here the potential domain of values for X_1 and X_2 is given as red and green, but the unary predicates specify red. We can immediately eliminate green from both nodes, as in Figure 2.2.

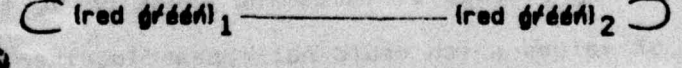
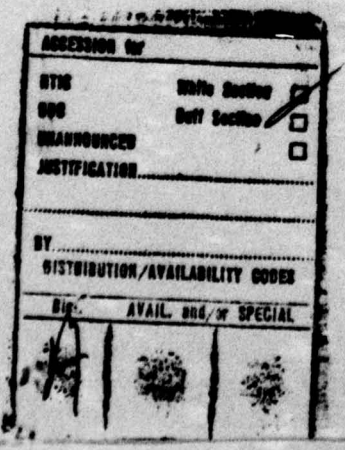


Figure 2.2



The next level of inconsistency is arc inconsistency. the arc from X_1 to X_2 is inconsistent because for a value in X_1, namely "red", there does not exist any value a_2 in X_2 such that red and a_2 together satisfy the relation "red is not the same color as a_2". To remedy this inconsistency we remove red from X_1, and similarly, from X_2. This cuts down our search space all right: unfortunately, in this case it reflects the fact that the problem is impossible. There is no global solution, i.e. the network is what I call "unsatisfiable".

It is entirely possible for a network to have no arc inconsistencies, and still be unsatisfiable. Consider the problem of coloring a complete three node graph with two colors, represented in Figure 2.3.

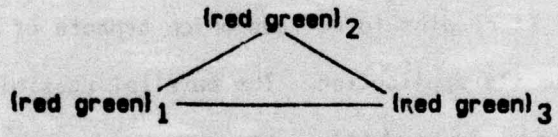


Figure 2.3

Assume the set of possible values for each variable is (red green) and the binary predicate between each pair again specifies "is not the same color as".

This network is arc consistent, e.g. given a value "red" for X_1, we can choose "green" for X_2: red is not the same color as green. Yet obviously there is no way of choosing single values a_1, a_2, a_3, for X_1, X_2, and X_3, such that all three binary constraints are satisfied simultaneously. If we choose red, for X_1, for example, we are forced to choose green for X_2 to satisfy the constraint between X_1 and X_2. This

forces a choice of red for X_3, which forces a choice of green for X_1, already picked to be red.

Nevertheless, it may be helpful to remove arc inconsistencies from a network. This involves comparing nodes with their neighbors as we did above. Each node must be so compared; however, comparisons can cause changes (deletions) in the network and so the comparisons must be iterated until a stable network is reached. These iterations can propagate constraints some distance through the network. The comparisons at each node can theoretically be performed in parallel and this parallel pass iterated.

Thus removing arc inconsistencies involves several distinct ideas: local constraints are globally propagated through iteration of parallel local operations. It remains to be seen which aspects of this process are most significant to its application. The parallel possibilities may prove to be particularly important; however, at the moment serial implementations are used in practice.

Waltz "filtering" algorithm for scene labelling [23] is the paradigm example of an arc consistency algorithm. Waltz wishes to attach labels to the lines in a line drawing indicating their semantic interpretations as convex, concave or occluding three-dimensional edges. The line drawing itself functions as the constraint network. Vertices function as network nodes. An individual vertex value consists of a label for each of the lines incident to the vertex; the set of possible values is initially constrained according to realizable three dimensional interpretations for the various types of vertices. The lines are the arcs of the network and

each represents the relation "the labellings of the adjacent vertices must agree along this line".

Waltz filtering algorithm (expecially when further constrained by specifying initial labels for edges on the background) generally results in an amazing combinatorial reduction: thousands of possibilities are often reduced to a state where all nodes have a single value remaining, thus totally solving the problem of obtaining the global solution. Of course the algorithm does not always terminate with a unique value at each node. Generally, in this case, most nodes will still have a unique value, while a few nodes will have a small set of values remaining. Normally this final state indicates that several ambiguous interpretations are possible; alternative sets of values that simultaneously satisfy all constraints can be quickly found with tree search.

It is perhaps not as well appreciated that this final state may also be reached for a figure which in fact admits no consistent labelling. This is to be suspected, however, given that the filtering algorithm only achieves arc consistency. Given the basic Huffman label set [10] [4] (not Waltz' expanded label set) and applying the filtering algorithm (without first constraining the outside lines to be occlusions), the line drawing in Figure 2.4 is left labelled as shown.

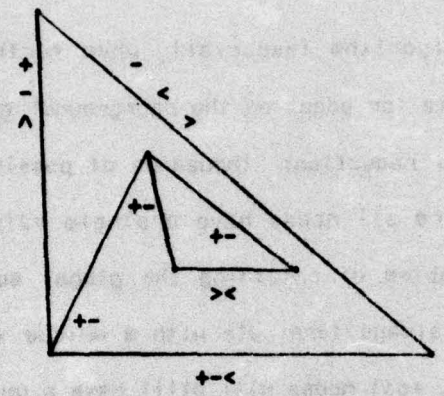


Figure 2.4

However, there is no consistent choice of labels for the vertices of the inner triangle. In other words the filtering algorithm alone will not determine if a line drawing is what Huffman calls an "impossible figure".

Montanari [15] has developed a more powerful notion of inconsistency which Mackworth calls path inconsistency. A network is path inconsistent if there are two nodes X_1 and X_2 such that a satisfies X_1, b satisfies X_2, a and b together satisfy the binary constraint between them, yet there is some other path through the network from X_1 to X_2, such that there is no set of values, one for each node along the path, which includes a and b, and can simultaneously satisfy all constraints along the path. For example, the network in Figure 2.3 is path inconsistent: red satisfies X_1, green X_3, red is not the same color as green; however, there is no value for X_2 which will satisfy the constraints between X_1 and X_2, and between X_2 and X_3, while X_1 is red, X_3 is green.

Montanari gives an algorithm that essentially removes path

inconsistencies from a network. However, path consistency does not necessarily insure satisfiability either, as powerful as it sounds. Consider the problem of coloring the complete four node graph with three colors (Figure 2.5).

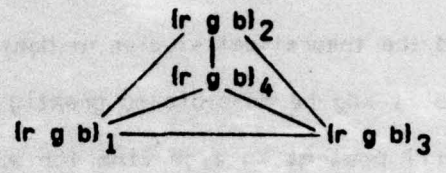


Figure 2.5

Each node contains red, green and blue, and each arc again represents the relation "is not the same color as". In particular, path consistency does not fully determine the set of values satisfying the global constraint, which in this inconsistent case is the empty set.

In summary, arc and path consistency algorithms may reduce the search space, but do not in general fully synthesize the global constraint. When there are multiple solutions, additional search will be required to specify the several acceptable combinations of values. Even a unique solution may require further search to determine, and the consistency algorithms may even fail to reveal that no solutions at all exist.

3. An extended theory

As the coloring problem suggests, the general problem of synthesizing the global constraint is NP-complete [5], and thus unlikely to have an efficient (polynomial time) solution. On the other hand the experimental results of Waltz, and the theoretical studies of Montanari, suggest that in specific applications it may be possible to greatly facilitate the search for solutions. I will present an algorithm for synthesizing the n-ary constraint defined by a set of constraints on subsets of n variables. It may be of substantial benefit in applications where pruning of arc and path inconsistencies still leaves many possibilities to be searched.

There are two key observations that motivated the algorithm.

1. Node, arc and path consistency in a constraint network for n variables can be generalized to a concept of k-consistency for any $k \leq n$, where n-consistency constitutes a natural notion of global consistency.

2. The given constraints can be represented by nodes, as opposed to links, in a constraint network; we can add nodes representing k-ary constraints to a constraint network for all $k \leq n$ (whether or not a corresponding k-ary constraint is given); and we can then propagate these constraints in this augmented net to obtain higher levels of consistency.

By successively adding higher level nodes to the network and propagating constraints in the augmented net, we can achieve k-ary consistency for all k. We do not need to restrict the given constraints to binary relations. Ruling out lower order inconsistencies in stages progressively reins in the combinatorial explosion. The final result is a

globally consistent network, where the n-ary node specifies explicitly the n-ary constraint we seek to synthesize. No further search is required. The rest of this paper will present the algorithm, along with a sufficient theoretical base to justify its operation.

4. A preliminary example of the synthesis algorithm

I will give a crude example of the synthesis algorithm in operation, by way of motivation for the formal description which follows. The presentation in this section is intentionally sketchy.

Suppose we are given the following constraints on variables X_1, X_2, X_3: The unary constraint C_1 specifies that X_1 must be either a or b, i.e. C_1=(a b). Similarly C_2=(e f) and C_3=(c d g). The binary constraint on X_1 and X_2 specifies that either X_1 is b and X_2 is e, or X_1 is b and X_2 is f: C_{12}=(be bf). Likewise C_{13}=(bc bd bg) and C_{23}=(ed fg).

We wish to determine what choices for X_1, X_2, X_3, if any, can simultaneously satisfy all these constraints. We begin building the constraint network with three nodes representing the unary constraints on the three variables, as shown in Figure 4.1.

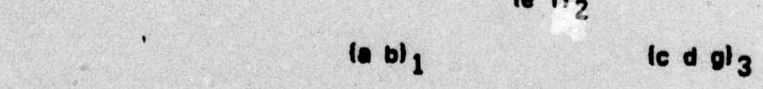


Figure 4.1

Next we add nodes representing the binary constraints, and link them to the unary constraints as shown in Figure 4.2 (e.g. (be bf)$_{12}$ represents C_{12}).

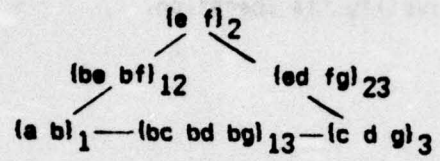


Figure 4.2

After we add and link node C_{12} we look at node C_1 and find that element a does not occur in any member of C_{12}. We delete a from C_1. Similarly, we delete c from C_3 after adding C_{23}. The constraint network now appears as in Figure 4.3.

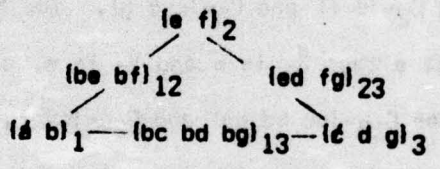


Figure 4.3

Now from C_3 we look at C_{13} and find that there is an element bc in C_{13} which requires c as a value for X_3, while c is no longer in C_3. We remove bc from C_{13}, as in Figure 4.4.

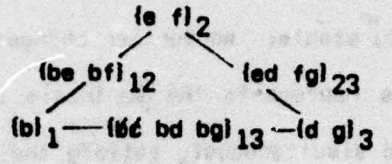


Figure 4.4

So far we have merely achieved a sort of "arc consistency" (though we indicate the restriction of the pair bc, as well as the elements a and c).

Next, we add a node for the ternary constraint. No order three constraint was given originally, so we could assume initially the "non-constraint", all possible triples. However, we will take advantage of the restrictions available from the binary and unary predicates to construct a more limited set of possibilities. C_1 and C_{23} together allow only the following set of triples: {bed bfg}. We use this as the ternary node and link it to the binary nodes as shown in Figure 4.5.

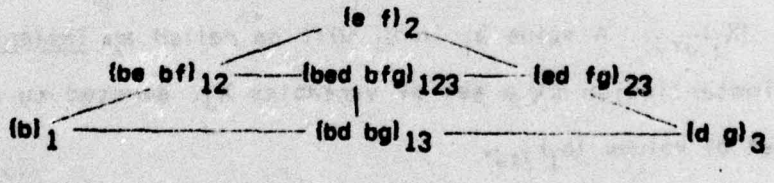


Figure 4.5

We look at the new node from its neighbors and vice versa, as we did earlier, to insure consistency of the sort we obtained earlier between neighboring nodes. C_{13} is consistent with the new node: bd is part of bed, bg part of bfg. Similarly C_{12} and C_{23} are consistent with the ternary

node. If necessary, we could propagate deletions around until local consistency is achieved on this augmented network. However, in this case, the network is already stable: no further changes are required.

The ternary node represents the synthesis of the given constraints. There are two ways to simultaneously satisfy the given constraints: X_1=b, X_2=e, X_3=d or X_1=b, X_2=f, X_3=g.

5. Basic definitions: constraint expressions, constraint networks and satisfiability

This section presents several definitions needed to state the problem and its solution precisely.

We are given a set of variables $X_1,\ldots,X_n$ which may take on values from a set of universes $U_1,\ldots,U_n$ respectively. We will assume the U_i to be discrete, finite domains. Let I=(1 2 ... n). Many of our definitions will be made for any subset $J \subseteq I$. We denote by X_J the indexed set of variables $(X_j)_{j \in J}$. A value a_i in U_i will be called an instantiation of X_i. An instantiation of a set of variables X_J, denoted by a_J, is an indexed set of values $(a_j)_{j \in J}$.

A constraint on X_J, denoted C_J, is a set of instantiations of X_J. The "indexed set" notation implies that there is a function, a, from J onto the instantiation a_J, which serves to indicate which member of a_J instantiates which variable: the value of a at j, denoted a_j, is the instantiation of X_j. We could also represent a_J as an ordered set or

m-tuple, where m is the number of elements in the set J (called the cardinality of J and denoted $|J|$): $a_J=(a_{j_1},\ldots,a_{j_m})$, a_{j_i} in U_{j_i}, $j_i<j_k$ for $i<k$, $i,k=1,\ldots,m$. Thus C_J may be thought of as an m-ary relation. I have found it useful, however, to use set notation rather than refer to cross products or predicates in the presentation which follows. Given a_J, "$a_j \subset a_J$" will denote the instantiation of X_j contained in a_J.

A <u>constraint expression</u> of order n is a conjunction of constraints $C=\bigwedge_{J\subset 2^I} C_J$, one constraint for each subset J of I (except the empty subset). Normally we will not be explicitly given constraints for all $J\subseteq I$; however, we can assume they exist, with no loss of generality, as the "non constraint" for X_J can always be specified, the set of all combinations of elements from the domains of the variables in X_J.

We say that an instantiation a_J <u>satisfies</u> a constraint C_J if $a_J \subset C_J$. The instantiation a_J satisfies a constraint C_H, $H\subseteq J$, if the set $\{a_j \subset a_J\}_{j\subset H}$, which we call a_J restricted to H, is a member of C_H. An instantiation a_J, where $|J|=k$, <u>k-satisfies</u> a constraint expression of order $n\geq k$ if a_J satisfies the constraints C_H for all $H\subseteq J$. If an instantiation a_I n-satisfies the constraint expression of order n, we say that a_I <u>satisfies</u> the expression. A constraint expression C is <u>k-satisfiable</u> if for all cardinality k subsets J of I, there exists an a_J such that a_J k-satisfies C. If C of order n is n-satisfiable it is said to be <u>satisfiable</u>.

A conjunction of constraints, a constraint expression, defines another constraint: the set of all instantiations a_I which satisfy the constraint expression. Our central problem is to synthesize the order n constraint on

X_I defined by the constraint expression, i.e. to determine explicitly the set of instantiations a_I which simultaneously satisfy all the given constraints. An instantiation a_I which satisfies C is called a solution of the constraint expression.

A constraint network of order k in n variables, $k \leq n$, is a set of constraints called nodes, N_J, for each $J \subseteq I$, $|J| \leq k$, where a link is said to exist between N_J and N_H if $H \subset J$ and $|H| = |J| - 1$. Linked nodes are called neighbors. A constraint network of order n in n variables will be called a full constraint network. A node N_J is said to correspond to a given constraint C_J if $N_J = C_J$, i.e. each instantiation of one is a member of the other. A full constraint network in n variables corresponds to a constraint expression of order n if each node N_J in the network corresponds to the constraint C_J in the expression. The order of a node N_J, or a constraint C_J, is the cardinality of J.

For example, the network in Figure 5.1 corresponds to the constraint expression $C = \bigwedge_{J \in 2^I} C_J$, where: $I = (1\ 2)$, $C_1 = (r\ g)$, $C_2 = (r\ g)$, $C_{12} = (rg\ gr)$.

(I avoid set notation in the subscripts for simplicity.)

$$(r\ g)_1 \text{ —— } (rg\ gr)_{12} \text{ —— } (r\ g)_2$$

Figure 5.1

This is obviously a representation of the problem of coloring a two node graph with two colors.

As nodes are constraints we are able to restate all the above definitions involving satisfiability in terms of nodes and networks, rather

than constraints and constraint expressions. In particular we can speak of an instantiation a_J satisfying a node N_H for $H \subseteq J$. We also will want to talk about a_J satisfying N_H for $H \supset J$. We will say that a_J satisfies N_H, $H \supset J$, if there exists an a_H in N_H such that $(a_J \langle a_H \rangle)_{J \subset J} = a_J$, i.e. there is an instantiation which satisfies N_H whose restriction to J is a_J.

6. Constraint propagation

We can now define the basic constraint propagation mechanism. To locally propagate the constraint N_J to a neighboring constraint N_H, remove from N_H all a_H which do not satisfy N_J. Global propagation is defined recursively. To globally propagate a constraint N_J through a neighboring constraint N_H: first locally propagate N_J to N_H; then, if anything was removed from N_H by the local propagation, globally propagate N_H through all its neighbors except N_J. To propagate a constraint N_J, globally propagate N_J through all its neighbors. The propagation procedure is similar to an arc consistency algorithm. Mackworth discusses efficient serial algorithms for arc consistency [11]. Of course, a parallel implementation is possible.

A constraint network is said to be relaxed if we can propagate every constraint N_J in the network without causing any change (deletions from nodes) in the net. The relaxation of a constraint network is the network obtained by propagating all nodes of the network. (The propagation obviously terminates in a relaxed network.)

7. Synthesis algorithm

We are now ready to state the synthesis algorithm. The claim, to be proven in section 10, is that this algorithm, given a constraint expression, produces a constraint network whose order n node corresponds to the order n constraint defined by the constraint expression.

ALGORITHM:

Given $C = \bigwedge_{J \subset 2^I} C_J$. We define the algorithm inductively:

STEP 1: Construct a constraint network with nodes N_J corresponding to constraints C_J in the given constraint expression, for all $J \subseteq I$ of cardinality one.

STEP k+1: For all $J \subseteq I$ of cardinality k+1:

Add the node N_J to the network corresponding to the given constraint C_J. Link N_J to all N_H such that H is a cardinality k subset of J.

Locally propagate to N_J from each of its neighbors. Propagate N_J.

For a constraint expression of order n, the algorithm is run for n steps. The result is a full constraint network, where N_I corresponds to C.

The next section will present several examples of the algorithm in operation. First a few general observations. The network produced by this algorithm is the relaxation of the network corresponding to C. We could have obtained it simply by building the corresponding order n network and

propagating each node. By proceeding in stages we take advantage of the elimination of possibilities that may occur at each stage to mitigate combinatorial explosion. We take this principle further and propagate each node as it is added, before adding another. A good heuristic would be to add earlier those nodes which exert a heavy constraint, e.g. where C_J is small. The propagation of these constraints may eliminate elements from nodes used in constructing later constraints. If C_J is the non-constraint we can construct N_J initially from some N_H and N_{J-H}, where H is a cardinality k subset of J, preferably the one for which $|N_J| \times |N_{J-H}|$ is a minimum. (Add to each member of N_H each member of N_{J-H}.)

Other refinements are clearly possible. Provision should be made for early termination, e.g. as soon as one node becomes empty. Propagation can be simplified, e.g. by noting non-constraints, or using complements of nodes. Additional links could permit direct propagation between a node N_J and the nodes for all subsets of J.

It is generally redundant to require all non-constraint nodes; basically we only need one "path" up to the n-ary node for every "real" constraint. Consider a constraint expression on four variables where only the binary constraints are really specified (the others are non-constraints). Only the binary constraints can really have any effect on the global solution. Three ternary nodes are sufficient for the network constructed by the algorithm. If the fourth ternary node rules out any element of the order four node, it is only reflecting a binary constraint, which is reflected in one of the other ternary nodes. On the other hand we may be interested in the effects on non-constraints of the propagation

process. In general the pruning process of the algorithm progressively makes explicit at N_J restrictions on instantiations of X_J that are not originally given by C_J, but rather implied by the other constraints. In the final network produced by the algorithm every member of every N_J is part of some solution of the constraint expression. (In particular, we have derived the "minimal" network, Montanari's "central problem" [15].)

8. Further examples: graph coloring, scene labelling, graph isomorphism

As the synthesis problem is such a general one, the synthesis algorithm has many potential applications. Graph problems, of course, lend themselves particularly to a constraint network formulation. I present in this section three applications which will serve to illustrate the algorithm, and are of some independent interest as well.

As we would expect from the discussion in section 2, the graph coloring problem can easily be represented as a constraint network. Given a graph G, and a set of colors, we construct a constraint network from G as follows: Each node of G is replaced by the unary constraint representing the set of colors. If there is an edge between nodes in G, we replace it by a binary constraint linked to the nodes which represents "is not the same color as". If there is no link between nodes in G, we add the non-constraint between the nodes.

Let us consider two examples. First consider the problem of coloring a complete three node graph with three colors. Figures 8.1a, 8.1b and 8.1c

show the constraint network after steps one, two and three of the algorithm, where the nodes N_1, N_2, N_3 are all the set (r g bl, N_{12}, N_{13} and N_{23} all equal (rg rb gr gb br bg) and N_{123}=(rgb rbg brg bgr grb gbr), the six possible colorings.

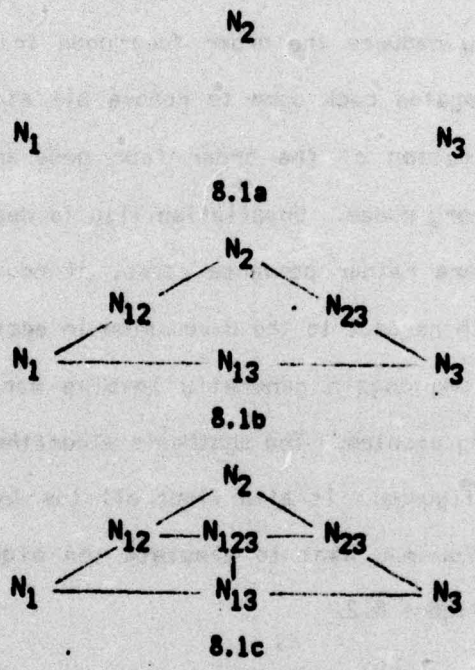


Figure 8.1

We could construct a network of the sort we used in section 2, for this problem. However the network would be path consistent; arc and path consistency algorithms would not rule out any elements at the nodes.

Consider now the problem of coloring a complete four node graph with three colors, which we used in section 2 to illustrate that path consistency is not a sufficient condition for satisfiability. After the

third step of the algorithm we would have four ternary nodes, each equal to the ternary node in the previous example.

At the beginning of the fourth step we use N_{123} and N_4 to construct an order four node: N_{1234}=(rgbr rgbg rgbb rbgr rbgg rbgb brgr brgg brgb bgrr bgrg bgrb grbr grbg grbb gbrr gbrg gbrb). Local propagation from the other ternary nodes quickly reduces the order four node to the empty set (and this constraint propagates back down to remove all elements from all the nodes). No instantiation of the order four node will simultaneously satisfy the four ternary nodes. Unsatisfiability is demonstrated.

These examples are rather perverse cases, of course, though they do illustrate points with respect to the discussion in section 2. Applications in the scene labelling domain generally involve more propagation than occurs in the coloring problem. The synthesis algorithm does function as a test for impossible figures. It also finds all the interpretations in an ambiguous figure. You may want to simulate the algorithm on a simple figure like that in figure 8.2.

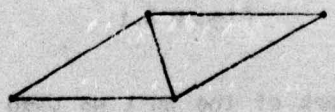


Figure 8.2

After the Waltz filtering algorithm is run on the Huffman label set (without additional constraints on the background labellings) there are three labels left at each order two vertex and two at each order three vertex.

For a final example, we take graph isomorphism. Given two graphs G and H, which we wish to test for isomorphism, construct a constraint network from G as follows. (If H has more nodes than G the algorithm will seek isomorphic mappings of G onto subgraphs of H.) Replace each node of G with a unary constraint node containing all the nodes of H. (If we allow loops, edges from a node to itself, the unary constraint on a node in G with a loop will be "has a loop in H", on a node in G without a loop, "has no loop in H". We could also incorporate additional unary constraints such as the order of the vertex [22].) Replace each edge between nodes a and b in G with a binary constraint node, linked to the unary nodes for a and b. This binary node will represent the constraint "these two (distinct) nodes share an edge in H". In other words the binary constraint will contain a pair xy if and only if there is an edge between x and y in H. Between two nodes which do not share an edge in G we also place a binary node, linked to them, but representing the constraint "these two (distinct) nodes do not share an edge in H".

For example: given the graphs G and H in Figure 8.3

Figure 8.3

we produce the constraint network in Figure 8.4.

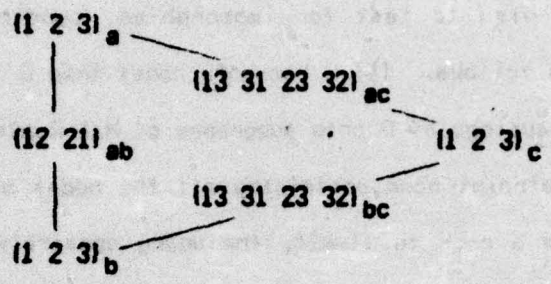


Figure 8.4

Propagating constraints, we obtain the network in Figure 8.5.

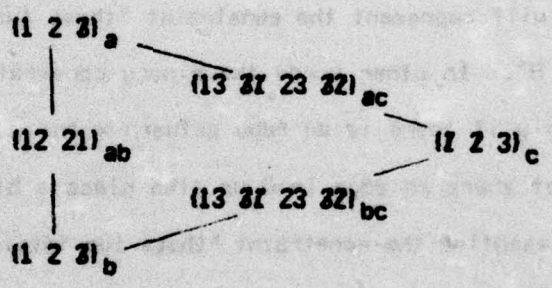


Figure 8.5

Now adding the ternary node we obtain Figure 8.6.

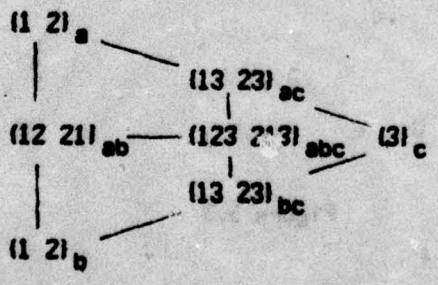


Figure 8.6

This network is relaxed. The ternary node represents the two possible isomorphisms: a->1, b->2, c->3 and a->2, b->1, c->3. (The algorithm also finds isomorphic subgraphs along the way.)

In the above applications, the desired global state is defined in terms of local constraints. Often we first face an analysis problem: choosing, or learning, a set of local constraints that specify or approximate the desired global state [20]. (An important concern will be the choice of a "good" constraint expression, i.e. one that can be synthesized efficiently.) As we explore various applications, it will, of course, be equally important to develop theoretical methods for analyzing the performance of the algorithm in a given domain.

9. Consistency and completeness

The synthesis algorithm operates by removing higher and higher level inconsistencies until a global consistency has been achieved. In this section, I define this sequence of consistency states, and also define a concept of completeness which we will want to apply to a network.

A node N_j of order k is <u>k-consistent</u> with a constraint expression C if all members of N_j k-satisfy C. A network of order k or greater is k-consistent with C if all nodes of order k are k-consistent with C. If a full constraint network of order n is n-consistent with a constraint expression C of order n we say that it is <u>consistent</u> with C.

A node N_J of order k is k-complete for C if any instantiation a_J which k-satisfies C is a member of N_J. A network N is k-complete for C if every node of order k is k-complete. An n-complete full constraint network of order n is said to be complete.

A few comments may be in order to relate the consistency notions described in this section to the background discussion in section 2. For networks of unary and binary constraints, k-consistency implies that if we pick values of any k-1 variables from the unary nodes, and a kth variable, there will be a value of the kth variable, at the unary node, such that the k values together satisfy all predicates involving the k variables (i.e. they form an instantiation of N_J where J is the set of k variables chosen). This indicates that 1-consistency of a constraint network implies node consistency of the corresponding network of the type described in section 2, 2-consistency implies arc consistency and 3-consistency implies path consistency. The first two are obvious; the latter requires reducing path consistency to the three node case, which is done by induction in a theorem of Montanari.

Suppose we seek a global solution by using depth first tree search on the elements remaining in the unary nodes of a k-consistent network of unary and binary constraints. Backup will only be initiated below the kth level. If the network is consistent, there will be no backup. Even better, and for arbitrary relaxed constraint networks, we can choose an order k node, and use its members as the alternative paths through the first k levels of a search tree, only really doing tree search on the remaining n-k nodes. Of course, if we have achieved full consistency the

members of the order n node are the solutions and no further search is required.

10. Synthesis theorem

We are now ready to state the theorem which justifies the synthesis algorithm.

THEOREM: The relaxation of the network corresponding to a constraint expression $C = \bigwedge_{J \subset 2^I} C_J$ is consistent and complete with respect to C.

The proof will be by induction. Consistency and completeness of order one are obvious. Our induction hypothesis is that the network is k-consistent and k-complete; we wish to prove k+1-consistency and k+1-completeness.

Consistency: We want to show that all N_J, for J any cardinality k+1 subset of I, are k+1-consistent. N_J, before relaxation, corresponded to C_J, so included nothing which did not satisfy C_J; relaxation does not add any elements to a node. Suppose there exists an a_J in N_J such that a_J does not satisfy C_H, for some proper subset H of J, i.e. a_J restricted to H is not a member of C_H. Pick a set G of cardinality k such that $H \subset G \subset J$. Because of the local propagation during the relaxation process, we know that a_J satisfies N_G. Thus a_J restricted to G, a_G, is a member of N_G. As the network is k-consistent a_G restricted to H is a member of C_H. But a_G restricted to H is a_J restricted to H; contradiction.

Completeness: Consider any a_J not in N_J, for J any cardinality k+1 subset of I. There are two possibilities. If a_J was not in N_J before relaxation, then a_J does not satisfy C_J. If a_J was removed during the relaxation process, then a_J does not satisfy N_H for some cardinality k subset H of J; by the induction hypothesis a_J restricted to H does not k-satisfy C. In either case, a_J does not k+1-satisfy C.

There are several immediate corollaries.

Corollary 1: N_I corresponds to the order n constraint defined by the constraint expression C.

Corollary 2: C is satisfiable if and only if N_I is not the empty set.

Corollary 3: The constraint network constructed by the synthesis algorithm operating on a constraint expression C is k-consistent with and k-complete for C after step k. The network constructed by the algorithm is consistent with and complete for C, and N_I corresponds to C.

References

1. Barrow, H.G. and Tenenbaum, J.M. MSYS: a system for reasoning about scenes. Stanford Research Institute A.I. Center Technical Note 121, Menlo Park, California, April 1976.

2. Bobrow, D.G. and Raphael B. New programming languages for A.I. research. Computing Surveys 6, 3(1974), 153-174.

3. Chen, C.H. (Ed.) Proceedings of the 1976 Joint Workshop on Pattern Recognition and Artificial Intelligence. Academic Press, New York, in preparation.

4. Clowes, M.B. On seeing things. Artificial Intelligence 2, (1971), 79-116.

5. Cook, S.A. The complexity of theorem-proving procedures. Conf. Rec. of 3rd Ann. ACM Symp. on Theory of Computing, 1971, 151-158.

6. Fahlman, S.E. Thesis progress report: a system for representing and using real-world knowledge. AI Memo 331, M.I.T. A.I. Lab., 1975.

7. Fikes, R.E. REF-ARF: a system for solving problems stated as procedures. Artificial Intelligence 1, 1(1970), 27-120.

8. Hewitt, C., Bishop P. and Steiger R. A universal modular ACTOR formalism for artificial intelligence. Proc. Third Int. Joint Conf. on A.I., 1973, pp. 235-245.

9. Horn, B.K.P. Determining lightness from an image. Computer Graphics and Image Processing 3, 4(Dec. 1974), 277-299.

10. Huffman, D.A. Impossible objects as nonsense sentences. In Meltzer, B. and Michie, D. (Eds.), Machine Intelligence 6, Edinburgh U.

Press, Edinburgh, 1971, 295-323.

11. Mackworth, A.K. Consistency in networks of relations. TR 75-3, C.S. Dept., U. of British Columbia, 1975.

12. Marr, D. Simple memory: a theory for archicortex. Phil. Trans. Roy. Soc. B. 252, (1971), 23-81.

13. Marr, D. and Poggio, T. Cooperative computation of stereo disparity. AI Memo 364, M.I.T. A.I. Lab., 1976.

14. Minsky M. and Papert S. Perceptrons. M.I.T. Press, Cambridge, Mass., 1968.

15. Montanari U. Networks of constraints: fundamental properties and applications to picture processing. Information Sciences 7, 2(April 1974), 95-132.

16. Rosenfeld, A., Hummel, R. and Zucker, S.W. Scene labelling by relaxation operations. IEEE Trans. on Systems, Man, and Cybernetics SMC-6, 6(June 1976), 420-433.

17. Sussman G.J. and McDermott, D.V. From PLANNER to CONNIVER -- a genetic approach. Proc. AFIPS 1972 FJCC, Vol. 41, AFIPS Press, Montvale, N.J., pp. 1171-1179.

18. Sussman G.J. and Stallman, R.M. Antecedent reasoning and dependency-directed backtracking in a system for computer aided circuit analysis. AI Memo in progress, M.I.T. A.I. Lab.

19. Ullman, J.R. Associating parts of patterns. Information and Control 9, 6(Dec. 1966), 583-601.

20. Ullman, J.R. Pattern Recognition Techniques. Crane Russak, New York, 1973.

21. Ullman, J.R. An algorithm for subgraph isomorphism. J.A.C.M. 23, 1(Jan. 1976), 31-42.

22. Unger, S.H. GIT--a heuristic program for testing pairs of directed line graphs for isomorphism. Comm. ACM 7, 1(Jan. 1964), 26-34.

23. Waltz, D.L. Generating semantic descriptions from drawings of scenes with shadows. AI-TR-271, M.I.T. A.I. Lab., 1972.

24. Zucker, S.W. Relaxation labelling and the reduction of local ambiguities. TR-451, C.S. Center, U. of Maryland, 1976.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Memo No. 370	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Synthesizing Constraint Expressions		5. TYPE OF REPORT & PERIOD COVERED Memo 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Eugene C. Freuder		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0643
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209		12. REPORT DATE July, 1976 13. NUMBER OF PAGES 31 pages
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

algorithm	combinatorial explosion	relaxation
artificial intelligence	scene analysis	filtering
constraints	graph theory	scene labelling
networks	backtrack search	thrashing
	machine vision	parallelism

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An algorithm is presented for determining the values which simultaneously satisfy a set of relations, or contraints, involving different subsets of n variables. The relations are represented in a series of constraint networks, which ultimately contain a node for every subset of the n variables. Constraints may be propagated through such networks in (potentially) parallel fashion to determine the values which simultaneously satisfy all the constraints. The iterated constraint propagation serves to mitigate combinatorial explosion. Applications in scene analysis, next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 cont.

cont. → graph theory, and backtrack search are provided.