INFORMATION SCIENCES **19**, 229–250 (1979)

Relational Consistency Algorithms and Their Application in Finding Subgraph and Graph Isomorphisms

J. J. McGREGOR

Department of Applied Mathematics and Computing Science, University of Sheffield, Sheffield, S10 2TN, England

Communicated by Azriel Rosenfeld

ABSTRACT

The determination of subgraph and graph isomorphisms is an important application for the algebraic manipulation of networks of binary constraints. Simplified and streamlined arc consistency and tree search algorithms are introduced, and experimental results show substantial reduction in timings compared with previous algorithms for determining isomorphisms. Several path consistency algorithms, including a new one, have been timed experimentally on isomorphism problems, and found not to be cost effective despite their theoretical appeal. The importance of this result is enhanced by the absence of previously published experimentation with path consistency. A theoretical study of the new path consistency algorithm provides insight into the experimental results.

1. INTRODUCTION

The paper of Cherry and Vaswani [2] on relational constraints was ahead of its time in 1961. At last, relational consistency is being recognized as a basic problem that underlies many practical problems, for instance in picture processing [14]. Another possible applications area is automatic correction of errors in text [17], which a few years ago might not have been viewed as a relational consistency problem. Kowalski [9] suggests a mechanism for selecting clauses for resolution in a theorem prover which uses a relational consistency algorithm for eliminating unprofitable lines of search. Mackworth [11] mentions other application areas.

The present paper will show that relational consistency algorithms can readily be specialized to find subgraph or graph isomorphisms. Graph isomorphism or subgraph isomorphism problems arise in, for example, chemical information retrieval [10, 15] and scene analysis [5]. The subgraph isomorphism

0020-0255/79/09229-22$01.75

problem is known to be NP complete [1, 3], which implies that the general relational consistency problem is also NP complete. Karp [8] has conjectured that no algorithms exist which solve NP complete problems in polynomial time. Nevertheless the present work aims to achieve a significant and useful flattening of the early part of the time growth curve for consistency algorithms.

Now that more applications are emerging for consistency algorithms, it becomes more important to perfect them in detail. It also becomes important to appreciate that existing algorithms in diverse problem areas are in fact members of the same family, thus enabling general improvements in consistency algorithms to be particularized to such individual problem areas. This task of consolidation has been started by Mackworth [11], and the present paper continues it by enhancing the efficiency of arc consistency algorithms, particularly when used during backtracking. A new path consistency algorithm is also described.

Because of its generality and intuitive appeal, path consistency has been subject to several recent theoretical investigations [12, 11] but without publication of experimental results. We remedy this situation by reporting experimental timings obtained by applying all our algorithms to graph and subgraph isomorphism problems.

2. CONSTRAINT SATISFACTION PROBLEM

Our problem is to determine the induced relation, that is, the subset of the Cartesian product $D_1 \times D_2 \times \cdots \times D_n$ whose elements satisfy a given collection of unary and binary constraints. Although it may be sufficient, in some practical applications, to select just one value from each of $D_1, D_2, \ldots, D_n$ so as to satisfy the constraints, we shall be concerned with the general problem of enumerating the induced relation.

A unary constraint is a predicate P_i such that $P_i(x)$ is *true* iff $x \in D_i$ satisfies the constraint. Thus P_i specifies a subset of D_i. A binary constraint P_{ij} is a predicate such that $P_{ij}(x,y)$ is *true* iff $x \in D_i$ and $y \in D_j$ satisfy the constraint. P_{ij} defines a relation on D_i and D_j. There may not be constraints defined for all domains or pairs of domains. The set of defined binary constraints constitute a network G such that $(i,j) \in \text{arcs}(G)$ iff a binary constraint P_{ij} is defined. We assume that

$$(\forall i,j,x,y)_{(i,j)\in \text{arcs}(G)} \{P_{ij}(x,y) \equiv P_{ji}(y,x)\}.$$

Occasionally it will be convenient to assume the existence of a P_i for *all* domains D_i and of a P_{ij} for *all* pairs of domains D_i and D_j. Where no

constraint P_i is defined we take $P_i(x)$ = true for all $x \in D_i$, and where no binary constraint P_{ij} is defined we take $P_{ij}(x,y)$ = true for all values $x \in D_i$ and $y \in D_j$.

In an appendix we present a simple constraint satisfaction problem which can be used to illustrate the behavior of the algorithms discussed in the ensuing sections.

3. MACKWORTH'S DEFINITIONS OF CONSISTENCY

Figure 1 shows a backtrack search algorithm which outputs all combinations of values, one from each of $D_1, D_2, \ldots, D_n$ that satisfy a given set of unary and binary constraints. Despite the elimination from consideration of large proportions of the search space by the test at line 6, such a search can be extremely inefficient. Mackworth [11] illustrates this and defines three types of consistency which can hold within a network of relations. Values which are inconsistent according to these definitions can be eliminated before being explored by a backtrack search. Mackworth's definitions are

(A) *Node consistency*. Node i is node consistent iff for any value $x \in D_i$, $P_i(x)$ holds.

(B) *Arc consistency*. Arc (i,j) is arc consistent iff for any value $x \in D_i$ such that $P_i(x)$, there is a value $y \in D_j$ such that $P_j(y)$ and $P_{ij}(x,y)$.

```
 1. k := 1; mark all elements of D1 as untried;
 2. repeat
 3.   if there are any untried elements in Dk then
 4.     begin
 5.       set xk to one of them and mark this value in Dk as tried;
 6.       if Pk(xk) ∧ (∧i<k Pik(xi, xk)) then
 7.         begin
 8.           if k = n then output (x1, x2, ··· xn)
 9.           else
10.             begin
11.               k := k + 1; mark all elements of Dk as untried
12.             end
13.         end
14.     end
15.   else
16.     k := k − 1
17. until k = 0
```

Fig. 1. The basic backtracking algorithm.

(C) *Path consistency*. A path of length m through the nodes $(i_0, i_1, \ldots, i_m)$ is path consistent iff for any values $x \in D_{i_0}$ and $y \in D_{i_m}$ such that $P_{i_0}(x)$ and $P_{i_m}(y)$ and $P_{i_0 i_m}(x,y)$, there is a sequence of values $z_1 \in D_{i_1}, \ldots, z_{m-1} \in D_{i_{m-1}}$ such that

(i) $P_{i_1}(z_1)$ and ... and $P_{i_{m-1}}(z_{m-1})$,
(ii) $P_{i_0 i_1}(x, z_1)$ and $P_{i_1 i_2}(z_1, z_2)$ and ... and $P_{i_{m-1} i_m}(z_{m-1}, y)$.

A network can easily be made node consistent by removing from each D_i any element x such that $P_i(x)$ is false, and we will assume throughout the rest of the paper that this is done before any further processing takes place. We defer consideration of path consistency until Sec. 10 and will at first deal only with arc consistency.

4. IMPROVED ARC CONSISTENCY ALGORITHMS

Figure 2 shows a basic procedure which achieves arc consistency in a network which is already node consistent. This is Mackworth's procedure AC1 [11] reworded to make explicit its nested loop structure. It is easy to see that AC1 deletes from D_i any x that does not belong to arc consistent arcs.

The use of bit vectors to represent sets is essential for an efficient implementation of the algorithm of Fig. 2. For example, the domain D_j can be represented by a vector of bits in which there is a bit position for each possible value in D_j, a one indicating that D_j still contains the corresponding value and a zero indicating that the corresponding value has been deleted from D_j. This is precisely the representation which the designers of the programming language PASCAL had in mind when they introduced the *set* type [7]. Similarly, in the relationship P_{ij}, for each x we can represent the set of values $\{y | P_{ij}(x,y)\}$ by a bit vector whose bit positions correspond to those in D_j. Thus

```
1.  repeat changed:=false;
2.    for i:=1 to n do
3.      for j:=1 to n do
4.        if (i,j)ϵ arcs (G) and i j then
5.          for each x∈D_i do
6.            if there is no y∈D_j s.t. P_ij(x,y) then
7.              begin
8.                delete x from D_i;
9.                changed:=true
10.             end
11.   until not changed.
```

Fig. 2. Mackworth's basic arc consistency algorithm AC1.

```
1.  repeat changed := false;
2.    for j := 1 to n do
3.      for i := 1 to n do
4.        if (i,j)∈arcs(G) and i≠j then
5.          begin
6.            D_i' := ∅;
7.            for each y∈D_j do D_i' := D_i' ∪ {x|P_ij(x,y)};
8.            if D_i≠D_i ∩ D_i' then
9.              begin D_i := D_i ∩ D_i'; changed := true end
10.         end
11.   until not changed
```

Fig. 3. An alternative basic arc consistency algorithm.

the test at line 6 of Fig. 2 can be implemented by *and*ing two bit vectors and testing for zero result [16]. Deleting an element from D_i (line 8) can also be implemented by a single *and* operation.

In Fig. 3, we present an alternative basic arc consistency algorithm which is an improvement on Fig. 2 in that the test for zero result of *and*ing and the further *and* to delete a one from D_i have been omitted from the innermost part of the procedure. At line 7, we construct the set of values $\{x|\exists y\ (P_{ij}(x,y))\}$. The set intersection and union operations used in lines 7–9 can all be implemented by *and*ing and *or*ing bit vectors. [Since $P_{ij}(x,y)=P_{ji}(y,x)$, $\{x|P_{ij}(x,y)\}=\{x|P_{ji}(y,x)\}$.] The order of nesting of the iterations over i and j has been reversed in order to facilitate further improvements in efficiency which can be made when algorithms developed from that of Fig. 3 are used during backtracking (Sec. 5) and in the subgraph isomorphism problem (Sec. 8).

Mackworth makes improvements to AC1 based on the observation that if during an iteration of AC1 an element is deleted from a single domain D_i, only domains connected to D_i in the network can possibly be affected on the next iteration. Similar improvements can be made to the algorithm of Fig. 3 without losing its advantages. In the algorithm of Fig. 4, Q is used to hold values of j for which it is worth applying the arc consistency algorithm. When a domain D_i is changed, i is added to Q. The two outer loops of the algorithm are equivalent to the outer loop of Mackworth's AC3. These two outer loops could be merged by making Q hold pairs of values (i,j) as in Mackworth's AC3, thus allowing further improvements in efficiency. We maintain the separation of the iterations over j and i for reasons connected with the applications of the algorithm discussed in subsequent sections.

5. USE OF ARC CONSISTENCY ALGORITHMS DURING BACKTRACKING

It is likely that after an initial application of consistency algorithms, there will still be a large number of alternatives to be considered by a backtrack search such as that in Fig. 1. In the simple example described in the Appendix, after the application of an arc consistency algorithm, each domain still contains two elements. Even after the application of a path consistency algorithm [11, 12], the domains corresponding to suspects in Boston, London and Paris still contain two suspects each.

An alternative way of looking at the process of giving a tentative value to x_k (Fig. 1, line 5) is to think of D_k as being tentatively restricted to a single value. When viewed in this light, it is clear that application of an arc consistency algorithm after each such decision could now cause further reduction in the sizes of the other domains, thus reducing the number of combinations of values which have to be considered in conjunction with the value selected for x_k. One of the other domains may even be reduced to the empty set by the arc consistency algorithm, thus enabling the selected value for x_k to be rejected immediately without further variable instantiation.

In fact the test at line 6 in Fig. 1 can be replaced by an application of any of the arc consistency algorithms discussed so far. The effect of this is that each time a domain is restricted to a single value, all elements which are not arc consistent with this value are removed from other domains. For a given value of k, only domains D_i such that $i>k$ can be changed in this way. At depth k in the backtrack search, x_k must be arc consistent with x_i for $i<k$, since otherwise x_k would have been previously removed from D_k. Thus if algorithm 2, 3 or 4 is used to refine the search space during backtracking as suggested, the iteration over i in each case can be reduced to **for** $i := k+1$ **to** n **do**. The cases where previously the test at line 6 would have failed are now eliminated from consideration by an earlier application of the arc consistency algorithm.

Against the saving of time resulting from reduction in the size of the search space must be balanced the extra work involved, not only in applying the arc consistency algorithm but also in stacking and unstacking copies of domains D_k, $D_{k+1},\ldots,D_n$ each time a variable x_k is instantiated. This is necessary because any of these domains could change as a result of selecting a value for x_k and applying the arc consistency algorithm, and when that value x_k is later rejected, these changes to the other domains will have to be undone.

It seems likely that in many problems, a full scale application of an arc refinement algorithm at every step during backtracking will be inappropriate. This is certainly the case when these techniques are applied to subgraph or

```
1.  Q := {j|1 ≤ j ≤ n};
2.  while Q isn't empty do
3.    begin
4.      select and delete any node j from Q;
5.      for i := 1 to n do
6.        if (i,j) ∈ arcs (G) and i ≠ j then
7.          begin
8.            D_i' := ∅;
9.            for each y ∈ D_j do D_i' := D_i' ∪ {x|P_ij(x,y)};
10.           if D_i ≠ D_i ∩ D_i' then
11.             begin D_i := D_i ∩ D_i';
12.               Q := Q ∪ {i}
13.             end
14.         end
15.   end
```

Fig. 4. An improved version of the algorithm of Fig. 3.

graph isomorphism problems (Secs. 8 and 9). In our experiments with these problems we have noticed that the effectiveness of the arc refinement algorithm falls off extremely rapidly after the first iteration of the main loop (Fig. 2 or 3). The version of the arc consistency algorithm of Fig. 4 is in a form particularly appropriate for conversion into a reduced version for use during backtracking. When domain D_k is tentatively reduced to a single value, x_k, the only value of j for which it is initially worth applying the algorithm of Fig. 4 is $j=k$. Also, for this value of j, D_j contains only a single element and the statement at line 9 need no longer be a loop. The only value for y to be considered is $y=x_k$. Thus, during backtracking, we could use the reduced version of the arc consistency algorithm presented in Fig. 5. This produces an effect comparable to that of a single iteration of the main loop in Fig. 2 or Fig. 3, but involves the execution of only a single loop whose extent decreases as the depth of search k increases.

```
1.  j := k; y := x_k;
2.  for i := k+1 to n do
3.    if (i,j) ∈ arcs(G) then
4.      D_i := {x|P_ij(x,y)} ∩ D_i;
```

Fig. 5. A restricted arc consistency algorithm for use during backtracking.

6. ORDER OF INSTANTIATION OF VARIABLES DURING BACKTRACKING

The order in which variables are selected for instantiation during backtracking is also of importance, particularly when using refinement techniques as discussed in the previous section. It would be sensible in any case, before applying the Fig. 1 algorithm, to reorder the domains so that the variables with the smallest domains are instantiated first [16]. Thus when a subtree is eliminated from the search space at line 6, more alternatives will be eliminated than would be the case if the larger domains had been considered first and pruning had taken place at the same depth.

When arc consistency is being used to refine the domains during backtracking, ideally we would like to select for instantiation the variable with the smallest domain at each stage. Such a dynamic reordering of the domains is likely to be computationally expensive, but a useful compromise can be adopted, particularly when not all pairs of domains in the network are affected by constraints. Having selected a variable x_k for instantiation and restricted its domain D_k to a single value, the domains most likely to be reduced in size by the application of an arc consistency algorithm (particularly the limited algorithm of Fig. 5) are those which are connected to D_k by constraints in the network. In the example given in the Appendix it would be foolish to instantiate the variable representing the Boston suspect, apply a limited arc consistency algorithm and then choose the variable corresponding to the London suspect for instantiation. Applying the limited arc consistency algorithm with the Boston domain reduced to a single value will leave the London domain unchanged, as there are no constraints between London and Boston. One of the three variables whose domains are connected to Boston by constraints should be instantiated next. Extending this idea, we propose that prior to a backtrack search with limited arc refinement, the domains should be reordered so that the domain whose variable is chosen for instantiation at each stage will be the one which has most constraints connecting it with domains whose variables have already been instantiated. This should tend to be the domain which has had most elements removed from it by prior applications of the arc consistency algorithm. The domain whose variable is instantiated first should of course be the one which is initially smallest. A straightforward algorithm can be written which will reorder the domains in this way, and this has been done in the programs used to produce the experimental results reported later.

7. SUBGRAPH ISOMORPHISM AS A CONSTRAINT SATISFACTION PROBLEM

An isomorphism of a graph G_α with a subgraph of a further graph G_β can be viewed as a labeling of each node of G_α with a different node of G_β. Accordingly we associate with each node i in G_α a variable x_i which can be assigned as a value any one of the nodes in G_β. A subgraph isomorphism is a set of values for $x_1, x_2, \ldots x_{p_\alpha}$, such that

$$(\forall i,j)_{i \neq j} \left\{ (a_{ij} = 1 \Rightarrow b_{x_i x_j} = 1) \text{ and } x_i \neq x_j \right\}$$

where $A = [a_{ij}]$ and $B = [b_{ij}]$ are the adjacency matrices of G_α and G_β respectively, and p_α is the number of nodes in G_α.

Finding a subgraph isomorphism is equivalent to finding $x_1, x_2, \ldots, x_{p_\alpha}$ that satisfy the following constraints:

UNARY CONSTRAINTS

$P_i(x_i)$ is *false* if there is any *a priori* reason why node i in G_α cannot correspond to node x_i in G_β (e.g., the degree of node x_i in G_β is less than that of node i in G_α).

BINARY CONSTRAINTS

Strong constraints: if $a_{ij} = 1$, then $p_{ij}(x_i, x_j) \equiv (b_{x_i x_j} = 1)$.
Weak constraints: if $a_{ij} = 0$, then $P_{ij}(x_i, x_j) \equiv (x_i \neq x_j)$.

The definition of P_{ij} in the case $a_{ij} = 0$ corresponds to the fact that no two nodes in G_α can be mapped to the same node in G_β in a subgraph isomorphism.

8. IMPROVED SUBGRAPH ISOMORPHISM ALGORITHMS

Ullmann's algorithm for subgraph isomorphism [16] used (apart from unimportant differences) the tree search algorithm of Fig. 1, with the arc consistency procedure of Fig. 2 and the constraint definitions that we have just formulated in Sec. 7.

The execution time for Ullmann's algorithm can be reduced by reordering the domains along the lines discussed in Sec. 6, and by using refinement procedures based on the arc consistency algorithm of Fig. 3.

In the subgraph isomorphism problem, constraints were defined in the last section between all pairs of domains in the network. However, the constraints marked as being strong constraints will predominate in the process of refining the domains during the backtrack search. In ordering the nodes for instantiation as discussed in Sec. 6 we have therefore considered only these strong constraints.

Ullmann found the use of the arc consistency algorithm of Fig. 2 during backtracking worth while in terms of reduced overall program execution time. However, he reports later [17] that a version of his refinement algorithm in which the outer iteration (see Fig. 2) is performed only once each time it is used during backtracking resulted in a further overall increase in speed. We found that still further restrictions on the amount of work done by the Fig. 2 algorithm during backtracking gave further savings. For example, the loop starting on line 2 can be executed for values of i up to just beyond the current depth of the backtrack search, thus ensuring that the domain of the next variable to be instantiated is reduced in size at the appropriate moment.

Use of the restricted refinement algorithm of Fig. 5 during backtracking resulted in significant further improvements to the best timings obtained with reduced versions of the Fig. 2 algorithm. Table 1 presents the results obtained on sets of random graphs whose adjacency matrices were generated using a pseudorandom number generator [13]. Pairs of graphs were constructed in the same way as in [16], and the results are presented in the same way as in that paper. Each result was obtained over 50 trails with different pairs of graphs.

TABLE 1

Results of Using Arc Consistency Algorithms in Finding Subgraph Isomorphisms

Sizes of graphs involved		Number of isomorphisms		Best times using restricted versions of Fig. 2 algorithm[a] (sec)		Times using Fig. 5 algorithm[a] (sec)	
p_α	p_β	av.	s.d.	av.	s.d.	av.	s.d.
6	12	1042	1187	0.481	0.353	0.199	0.161
7	14	6470	9915	2.405	2.716	1.029	1.232
8	10	858	1237	1.098	1.029	0.428	0.424
8	12	3050	5094	2.143	2.408	0.845	1.020
8	14	6735	12635	3.474	4.674	1.428	2.095
8	16	22579	31303	9.156	9.944	3.778	4.440
10	12	1184	2053	3.233	4.160	1.130	1.522
10	14	7592	17793	7.727	11.828	2.932	4.846

[a]For refinement during backtracking.

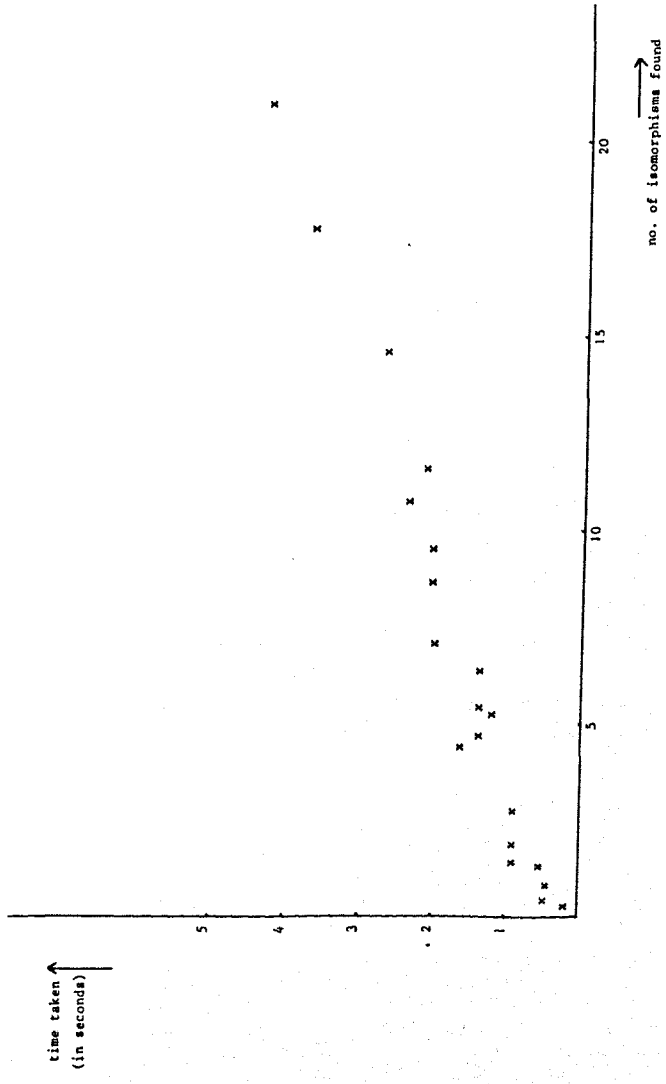

Fig. 6. Time taken versus number of subgraph isomorphisms found (in thousands) for $p_a = 8$.

The results presented in Table 1 were obtained from programs written in ALGOL 68 and run on an ICL 1906S computer.

One interesting feature of these results is that, for a given size of graph G_α, the time taken to find all isomorphisms of G_α into G_β appears to be linearly related to the number of isomorphisms found. In Fig. 6 we have plotted the number of isomorphisms found against the time taken for every 10th pair of graphs from each set of 50 pairs for which $p_\alpha = 8$.

9. APPLICATIONS TO FINDING GRAPH ISOMORPHISMS

Some further experiments were carried out on the detection of isomorphisms between graphs. The relationships P_{ij} were defined for this problem by

$$\text{if} \quad a_{ij}=1 \quad \text{then} \quad P_{ij}(x,y) \equiv (b_{xy}=1),$$

$$\text{if} \quad a_{ij}=0 \quad \text{then} \quad P_{ij}(x,y) \equiv (x \neq y \wedge b_{xy}=0).$$

Using this network of relationships, some experiments were made on the problem of finding all isomorphisms of a graph onto itself. Algorithm 5 was used as a refinement procedure during backtracking. For these experiments we used some strongly regular 25-node graphs which were described and used by Ullmann in [16]. These probably represent pathological worst case examples of this type of problem. Table 2 presents the times required to process the first 5 of these graphs. The results of Table 2 were obtained by using a program written in PASCAL and run on an ICL 1906S computer. (The change from ALGOL 68 to PASCAL was necessary as our ALGOL 68 implementation does not permit the manipulation of bit patterns of more than 24 bits.)

TABLE 2

Times Required to Find All Isomorphisms of a Strongly Regular 25-Node Graph onto Itself

	Time (sec)	
Graph	Using restricted refinement algorithm of Fig. 5	Using alternative restricted refinement algorithm discussed in Sec. 9
1	10.296	4.967
2	3.692	1.212
3	5.774	3.265
4	4.949	1.460
5	4.363	1.551

Some further experiments were carried out using an alternative restricted arc consistency algorithm during backtracking as follows:

for $j := 1$ **to** k **do**
  if $(k+1,j) \in \text{arcs}(G)$ **then**
    $D_{k+1} := \{z | P_{k+1}(z, x_j)\} \cap D_{k+1}$

After instantiating x_k, this algorithm was used to delete from D_{k+1} all elements

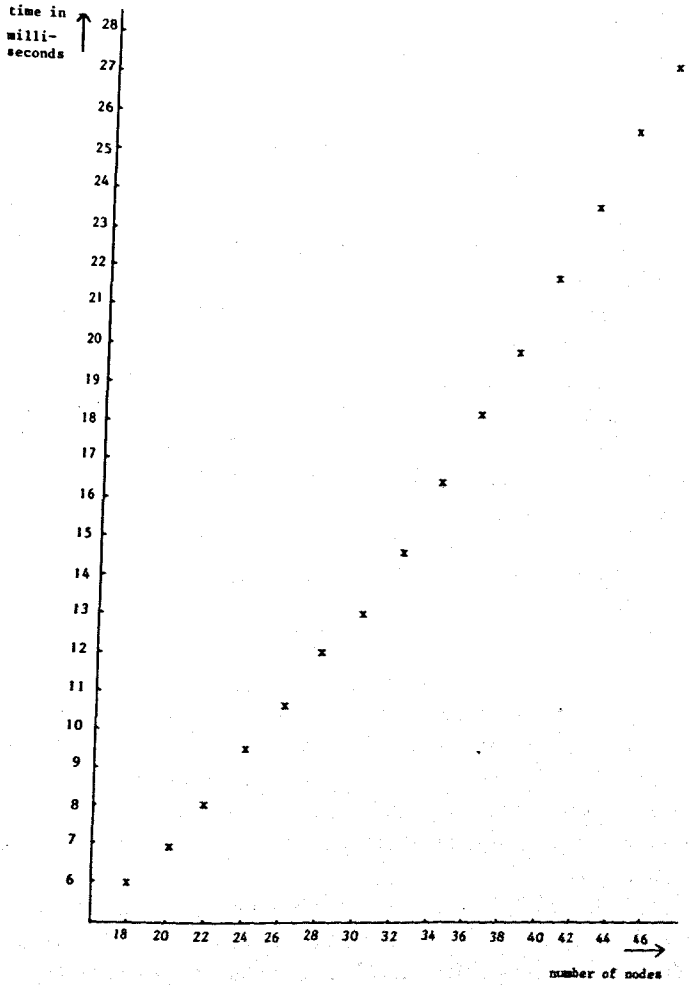

Fig. 7. Backtrack search time versus number of nodes for determination of isomorphism of pseudorandomly generated graphs.

which are inconsistent with the values selected so far for x_1, x_2,...,x_k. The advantage of this is that copies of only two domains, D_k and D_{k+1}, need be stacked at each step in the backtrack search, as D_{k+1} is the only domain which is refined immediately after instantiating x_k. This gave considerably improved timings in the case of our graph isomorphism problems as indicated in the last column of Table 2, but in the case of the subgraph isomorphism problems of the previous section, somewhat poorer timings were obtained.

Each application of the above refinement technique requires the execution of a single loop, but the extent of the loop increases as the depth of search increases. Whether or not this is compensated for by the reduction in the amount of domain copying clearly depends on the structure of the backtrack search tree explored.

The well-known graph isomorphism algorithm of Corneil and Gotlieb [4], although founded on a conjecture since proved to be false, provides a benchmark against which subsequent algorithms have been compared. From Fig. 7, we see that for isomorphic random graphs with an average edge density of 0.5, the timing of a backtrack search incorporating the above refinement algorithm depended on p_α^x, where $x \simeq 1.7$ for the range of values of p_α which we were able to consider. The time taken to set up the necessary data structures before the backtrack search depends on p_α^2 for any type of graph. Corniel and Gotlieb report a theoretical time dependence of p_α^2 for their algorithm when applied to similar random graphs. They also report an observed time of $0.00447 = 268$ msec for $p_\alpha = 20$. Our algorithm took an average backtrack search time of 6.96 msec for this size of graph together with an average preliminary time of 46.50 msec to set up the data structures. Our results therefore compare favorably with theirs for random graphs, as well as producing unusually good results on strongly regular graphs.

10. PATH CONSISTENCY ALGORITHMS

We now turn our attention to path consistency, which is mathematically more general than arc consistency. Elements can be deleted from domains on grounds of path consistency as well as arc consistency; and the search might be speeded up still further by deleting from each relationship P_{ij} any element that connects a pair of values which are not path consistent along some path from node i to node j.

To facilitate such refinement of relationships P_{ij}, we require and therefore describe the full bit matrix representation of the P_{ij} used by Montanari [12] and Mackworth [11], in which each P_{ij} is explicitly represented. For all pairs of values i,j $(i \neq j)$, P_{ij} is represented by the bit matrix R_{ij}, where $R_{ijxy} = 1$ if $P_{ij}(x,y)$ is true and $R_{ijxy} = 0$ otherwise. It is convenient to think in terms of

$R_{ijx} = \{y | P_{ij}(x,y)\}$, the sets R_{ijx} being represented by bit vectors as discussed in Sec. 4. It should be noted that the use of such an explicit representation of the constraints may immediately introduce additional storage and computational overheads: in some problems, for many pairs of values i, j, there may not be constraints which have to be satisfied by the values x_i, x_j; or there may be a convenient reduced representation for the P_{ij}, as is the case in the subgraph isomorphism problem, where all that has to be stored and accessed for the algorithms discussed in previous sections is the pair of bit matrices A and B representing the two graphs involved.

The path consistency algorithm of Montanari [12], improved by Mackworth [11], works by repeatedly examining all paths of length two and deleting elements from the domains and relationships until all paths of length two are path consistent. (Montanari proved that a network which is path consistent along all paths of length two is path consistent along paths of any length.) This algorithm requires setting up the bit matrices R_{ij} before the first iteration, and all subsequent processing carries the overhead of accessing the multidimensional array containing the R_{ij}.

Figure 8 presents an algorithm that has the effect of exploring paths of all lengths and that can work on the first iteration by using any appropriate reduced representation of the P_{ij}. The (partially refined) R_{ij} can be constructed

```
1.  for 1 := 1 to n do DCOPY1 := D1;
2.  repeat changed := false;
3.    for i := 1 to n do
4.      for each x ∈ Di do
5.        begin
6.          Di := {x};
7.          apply a full arc consistency algorithm;
8.          if any domain has been reduced to the empty set then
9.            begin DCOPYi := DCOPYi − {x};
10.             changed := true
11.           end
12.         else for j := 1 to n do
13.                 if j ≠ i then
14.                   if Rijx ≠ Dj then
15.                     begin changed := true; Rijx := Dj end
16.         for l := 1 to n do Dl := DCOPYl
17.       end
18. until not changed
```

Fig. 8. A new path consistency algorithm.

as a side effect of this first iteration. Furthermore, this path consistency algorithm uses an *arc* consistency algorithm as its basic tool. Thus any improvements to arc consistency algorithms or special purpose hardware implementation of the arc consistency algorithm, similar to that discussed by Ullmann [16], can be used to advantage in the *path* consistency algorithm of Fig. 8. It should be noticed in this respect that all applications of arc refinement done during one iteration of the main loop of Fig. 8 could be performed in parallel. Haralick [6] has recently proposed an algorithm for finding graph homomorphisms which in fact uses a specialization of the path consistency algorithm of Fig. 8.

It may not be obvious that the algorithm of Fig. 8 does establish path consistency. To confirm that it does, we call upon the following two theorems.

THEOREM 1. *If a network is* arc *consistent and a domain D_i contains only a single value x (and all other domains are nonempty), then all paths in the network which have i as a terminal node are path consistent.*

Proof (by induction on path length). The theorem is true for all paths of length 1 with i as a terminal node, by the definition of arc consistency. We assume that it is true for all paths of length $<n$, and let i and i_n be the terminal nodes of any path of length n: $i, i_1, i_2, \cdots i_n$. For any $x_n \in D_{i_n}$, $\exists x_{n-1} \in D_{i_{n-1}}$ such that $P_{i_{n-1}i_n}(x_{n-1}, x_n)$, since the network is arc consistent. Also, $P(x, x_{n-1})$, since the network is arc consistent and x is the only element of D_i, and so $\exists x_1 \in D_{i_1}$, $x_2 \in D_{i_2}, \ldots, x_{n-2} \in D_{n-2}$ such that $P_{ii_1}(x, x_1) \wedge P_{i_1 i_2}(x_1, x_2) \wedge \cdots \wedge P_{i_{n-2}i_{n-1}}(x_{n-2}, x_{n-1})$ by the induction hypothesis. Thus a path of length n with i as a terminal node is path consistent. ■

THEOREM 2. *If a network is not* arc *consistent and a domain D_i contains a value x, then if an element y is deleted from domain D_j by an arc consistency algorithm (Fig. 7, line 7), then element (x,y) should be deleted from P_{ij} (if present) on path consistency grounds.*

Proof. Say element y such that $P_{ij}(x,y)$ is deleted from D_j by the arc consistency algorithm. This happens if there is a domain D_k such that there is no $z \in D_k$ such that $P_{jk}(y,z)$. Consider any path $i,\ldots,k,j$. There can be no value for $z \in D_k$ which forms part of a consistent sequence of values $x,\ldots,z,y$ along this path. The pair (x,y) should therefore be deleted from P_{ij} on path consistency grounds. ■

The algorithm of Fig. 8 considers each element in each domain in turn. For each element $x \in D_i$ the remaining elements in D_i are temporarily deleted, and a full *arc* consistency algorithm (Fig. 4 say) is applied to the modified network. (Note that Q in Fig. 4 can be initialized to contain only the i currently selected in Fig. 8.) This has the effect of examining all paths emanating from node i

and deleting from each domain D_j any element y which is not path consistent with x along all paths from node i to node j. Thus lines 4 to 17 in Fig. 8 have the effect of checking path consistency along all paths in the network emanating from node i.

Montanari and Mackworth examine all elements of the R_{ij} on every iteration of their path consistency algorithm. Many elements of the R_{ij} which would be examined and changed by their algorithm would not be processed by the algorithm of Fig. 8. These elements would in fact never be used subsequently by any of the backtracking or arc consistency algorithms previously discussed (there being no elements in the domains which would cause such elements in R to be used). This selective refinement of R is a natural consequence of the way in which the operation of the Fig. 8 algorithm is controlled by the presence of elements in the domains.

A number of improvements can be made to the given version of the algorithm of Fig. 8. Montanari has shown that only paths of length two need be considered by a path consistency algorithm, and we have also observed (Sec. 5) that the effectiveness of an arc consistency algorithm falls off rapidly after its first iteration. It may therefore be sensible in the algorithm of Fig. 8 that the arc consistency algorithm used at line 7 should be restricted to performing only two main iterations, thus examining only paths of length two from node i. Improvements analogous to those made by Mackworth to Montanari's algorithms can also be made to the algorithm of Fig. 8: the efficiency of the arc refinement algorithm at line 7 can be improved on second and subsequent iterations by keeping a note of changes made to the R_{ij} and the D_i since the corresponding execution of line 7 during the previous main iteration, and using this information to avoid unnecessary work being done by the arc refinement algorithm.

11. EXPERIMENTS WITH PATH CONSISTENCY IN ISOMORPHISM PROBLEMS.

Previously published discussions of path consistency algorithms have been entirely theoretical. In order to remedy this situation, we have experimented with path refinement algorithms on subgraph and graph isomorphism problems using the same data as were used for the experiments reported in Secs. 8 and 9.

The first column of timings in Table 3 are the average times taken for the subgraph isomorphism problems using exactly the same algorithm as was used for the experiments reported in Sec. 8, except that in this case a full bit matrix representation of the P_{ij} was used by that algorithm. Comparison with Table 1 indicates the slight overheads involved in such a representation of the P_{ij}.

TABLE 3

Results of Experiments with Path Consistency Algorithms in Finding Subgraph Isomorphisms[a]

Graph sizes		Av. time, no p.r. (sec)	Av. time, 1 pass, Montanari-Mackworth p.r. algorithm (sec)	Av. search time after full p.r. algorithm (sec)	Av. time, 1st pass, Fig. 8 p.r. algorithm (sec)	Av. search time after 1 pass, Fig. 8 p.r. algorithm (sec)
P_α	P_β					
6	12	0.200	0.140	0.199	0.115	0.199
7	14	1.034	0.303	1.030	0.198	1.030
8	10	0.430	0.261	0.423	0.157	0.424
8	12	0.849	0.366	0.841	0.203	0.841
8	14	1.435	0.478	1.422	0.259	1.422
8	16	3.795	0.583	3.782	0.326	3.783
10	12	1.136	0.749	1.102	0.318	1.103
10	14	2.949	1.011	2.904	0.389	2.905

[a]p.r. = path refinement.

Montanari's and Mackworth's path consistency algorithms are functionally equivalent in the first iteration. Mackworth's improvements were intended to speed up the second and subsequent iterations. The second column of timings in Table 3 are average times taken in the subgraph isomorphism problems by the *first* iteration of a Montanari-Mackworth path consistency algorithm. These can be treated as lower bounds for the times taken by a full path refinement algorithm of this family, the additional time required depending on how effective Mackworth's improvements are. The next column of timings in Table 3 are average times taken by the previously used backtracking algorithm after an initial application of a *full* path refinement algorithm for each problem. These results show very little improvement in backtracking times compared with the previous algorithm. Any improvements certainly do not compensate for the additional time overhead required by the initial application of path refinement.

In common with arc consistency algorithms, path consistency algorithms appear to be considerably less effective on second and subsequent iterations. For this reason, it might be expected that maximum cost effectiveness would be attained by applying only a single iteration of a path consistency algorithm. The new path consistency algorithm introduced in Sec. 10 permits an efficient implementation of its first iteration when there is a good reduced representation for the P_{ij}, as is the case with the isomorphism problems. The fourth column of timings in Table 3 are the average times taken by the first iteration

of this algorithm, and the next column gives the times taken by the subsequent application of the previously used backtracking algorithm. In fact the algorithm for finding graph homomorphisms proposed by Haralick [6] is equivalent to this specialized use of path consistency. The path refinement process is now considerably quicker (despite the fact that paths of all lengths are being examined), but it is still clearly not cost effective.

Equivalent experiments were carried out on the graph isomorphism problems for which results were previously presented in Sec. 9. The path refinement algorithms of Montanari and Mackworth took approximately 11 seconds on each of these problems, while that of Fig. 8 took approximately 4 seconds. However, no changes were made by the path consistency algorithms to the network for these problems, and there was therefore no resulting improvement in search times.

In view of the above results, we did not consider it appropriate to investigate the application of path refinement during backtracking, as this would have involved further large overheads arising from the use of the path consistency algorithm itself, as well as from the need to copy the R_{ij} whenever the path consistency algorithm was used.

12. DISCUSSION

One of the main contributions of this paper is the demonstration that the cost effectiveness of arc consistency in finding subgraph and graph isomorphisms is increased by using only a one pass arc consistency algorithm as a refinement technique during backtracking. The reason for this is that the cumulative effect of the repeated application of the restricted arc consistency algorithm will approximate more and more closely to the effect of a full arc consistency algorithm as the depth of search increases. Thus there is very little advantage to be gained by doing the additional work involved in applying a full arc consistency algorithm at each stage in the search.

The theorems of Sec. 10 provide some insight as to why path consistency is found experimentally not to be cost effective in graph matching. To appreciate this it is only necessary to recognize that the path consistency algorithm of Fig. 8 is closely related to a backtrack search algorithm in which a full *arc* consistency procedure is used as a refinement technique. Each step forward in the backtrack search involves temporarily deleting all but one element from a domain D_k and applying the arc consistency algorithm. This amounts to performing a partial path consistency check with respect to all paths emanating from node k, thus ensuring that no elements remain in the other domains which are path inconsistent with the element selected from D_k (see Theorem 1,

Sec. 10). The extent to which the network becomes path consistent will increase as the depth of search increases. In view of the remarks made in the preceding paragraph, path consistency will be attained to some extent even when only a one pass arc consistency algorithm is used for refinement during backtracking.

The conclusion to which we are inescapably led is that, despite its theoretical appeal, path consistency algorithms of the type discussed merely duplicate to a large extent work which is more efficiently and conveniently done by using a limited arc consistency algorithm during backtracking in the way we have described.

APPENDIX—SIMPLE EXAMPLE OF A CONSTRAINT SATISFACTION PROBLEM

The problem is to discover a spyring consisting of one spy in each of Boston, London, Houston, Paris and Sheffield subject to the following constraints.

UNARY CONSTRAINTS

The spy in the ith city is one of the suspects whose code name is listed in the ith column of Table 4.

BINARY CONSTRAINTS

The spy in Boston must be a suspect who has met a spy in Houston, and vice versa. A similar constraint must hold between the following pairs of cities: Boston-Paris, Boston-Sheffield, London-Houston, London-Paris, London-Sheffield. Table 5 shows which suspects have met which other suspects where a 1 signifies "has met".

In the notation of Sec. 2, $D_1 = \{B_1, B2, B3, B4\}, D_2 = \{L1, L2, L3\}, D_3 = \{H1, H2, H3, H4\}$, $D_4 = \{P1, P2, P3\}$, $D_5 = \{S1, S2, S3, S4\}$. P_{13} is defined by the

TABLE 4

Boston	London	Houston	Paris	Sheffield
B1	L1	H1	P1	S1
B2	L2	H2	P2	S2
B3	L3	H3	P3	S3
B4		H4		S4

TABLE 5

	H1	H2	H3	H4	P1	P2	P3	S1	S2	S3	S4
B1	1	1	1	1	1	1	0	0	1	0	1
B2	0	0	1	1	1	0	0	1	0	0	0
B3	0	0	0	1	0	1	0	1	0	1	0
B4	1	1	0	0	1	0	0	0	1	0	1
L1	0	0	1	1	0	1	0	1	0	0	0
L2	0	0	0	1	1	0	0	1	0	1	0
L3	1	1	0	0	0	0	1	0	1	0	1

subset of Table 5 which indicates which pairs of suspects, one from Boston and one from Houston, have met. The other P_{ij} are defined similarly.

Thanks are due to Professor J. R. Ullmann, with whom helpful discussions were held at all stages of the work.

REFERENCES

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
2. C. Cherry and P. K. T. Vaswani, A new type of computer for problems in propositional logic, with greatly reduced scanning procedures, *Information and Control* 4:155–168 (Sept. 1961).
3. S. A. Cook, The complexity of theorem proving procedures, in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, 1971, pp. 151–158.
4. D. G. Corneil and C. C. Gotlieb, An efficient algorithm for graph isomorphism, *J. Assoc. Comput. Mach.* 17(1):51–64 (Jan. 1970).
5. E. C. Freuder, Structural isomorphism of picture graphs, in *Pattern Recognition and Artificial Intelligence* (C. H. Chen, Ed.), Academic, New York, 1976.
6. R. M. Haralick, The characterisation of binary relation homomorphisms, *Internat. J. General Systems* 4:113–121 (1978).
7. K. Jensen and N. Wirth, *Pascal: User Manual and Report* (corrected printing), Springer, New York, 1978.
8. R. M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, Eds.), Plenum, New York, 1972.
9. R. Kowalski, A proof procedure using connection graphs, *J. Assoc. Comput. Mach.* 22(4):572–595 (Oct. 1975).
10. M. F. Lynch, Storage and retrieval of information on chemical structures by computer, *Endeavour* 27 (101):68–73 (May 1968).
11. A. K. Mackworth, Consistency in networks of relations, *Artificial Intelligence* 8(1):99–118 (1977).
12. U. Montanari, Networks of constraints: fundamental properties and applications to picture processing, *Information Sci.* 7(2):95–132 (1974).