The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems

Alan K. Mackworth
Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada

Eugene C. Freuder
Department of Computer Science, University of New Hampshire, Durham, NH, U.S.A.

Recommended by Judea Pearl

ABSTRACT
Constraint satisfaction problems play a central role in artificial intelligence. A class of network consistency algorithms for eliminating local inconsistencies in such problems has previously been described. We analyze the time complexity of several node, arc and path consistency algorithms and prove that arc consistency is achievable in time linear in the number of binary constraints. The Waltz filtering algorithm is a special case of the arc consistency algorithm. In the edge labelling computational vision application the constraint graph is planar and so the time complexity is linear in the number of variables.

1. Introduction

The purpose of this paper is to analyze the network consistency algorithms described previously [6].

A constraint satisfaction problem (CSP) is defined as follows: Given a set of n variables each with an associated domain and a set of constraining relations each involving a subset of the variables, find all possible n-tuples such that each n-tuple is an instantiation of the n variables satisfying the relations. In this paper we shall consider only CSPs in which the domains are discrete, finite sets and the relations are unary and binary. These restrictions are not necessary for consistency techniques to be applied [2, 6, 7].

Since graph colouring is an NP-complete CSP it is most unlikely that a

Artificial Intelligence **25** (1985) 65–74

polynomial time algorithm exists for solving general CSPs. Accordingly, the class of network consistency algorithms was invented [2, 6, 8, 9]. These algorithms do not solve a CSP completely but they eliminate once and for all local inconsistencies that cannot participate in any global solutions. These inconsistencies would otherwise have been repeatedly discovered by any backtracking solution. One role for network consistency algorithms is as a preprocessor for subsequent backtrack search.

A k-consistency algorithm removes all inconsistencies involving all subsets of size k of the n variables. For example, the node, arc and path consistency algorithms detect and eliminate inconsistencies involving $k = 1$, 2 and 3 variables, respectively. Freuder [2] generalized those algorithms for $k = 1, \ldots, n$ thereby producing the complete set of solutions to the CSP.

Node, arc and path consistency can be achieved in polynomial time. The most significant new result presented here is that arc consistency is achievable in time linear in the number of binary constraints. If the constraint graph is planar (see, for example, Waltz's system [9]), then that time bound is also linear in the number of variables.

2. The Complexity of Node and Arc Consistency

The algorithms below are reprinted from a previous paper [6] which should be consulted for a full explanation. The domain of variable i is D_i and P_{ij} is the binary constraint predicate on the variables i and j corresponding to an edge between vertices i and j in the constraint graph G. The edge between i and j may be replaced by the directed arc from i to j and the arc from j to i as they are treated separately by the algorithms. Let the number of variables be n, the number of binary constraints be e (the number of edges in the constraint graph) and the edge degree of vertex i be d_i. The time unit used for our complexity measures will be the application of a unary or binary predicate. To simplify the description of the results of the analysis we shall assume that each D_i has the same size a, and that there is no internal structure to D_i or P_{ij}, such as strict ordering, that could be exploited.

As an illustrative example of these concepts, consider the n-queens problem. The task is to place n queens on an $n \times n$ chessboard so that no queen is on the same row, column or diagonal as any other. Since each queen must be in a different column, the task can be formulated as a CSP with a set of n variables $\{v_1, v_2, \ldots, v_n\}$, one for each column. The value of v_i is the row number of the queen in column i. We have each D_i initially as the set $\{1, 2, \ldots, n\}$, and so $a = n$. Each queen constrains each other so the edge degree of each vertex $d_i = n - 1$. The constraint graph G is the fully connected complete graph on n vertices, and so $e = \frac{1}{2}n(n-1)$. The binary predicate P_{ij} with $i \neq j$ is

$$P_{ij}(x, y) = ((x \neq y) \wedge (|x - y| \neq |i - j|))$$

The unary predicates P_i are in this case vacuous.

We first make the obvious remark that node consistency is always established in linear time by the algorithm NC-1. (See Fig. 1.)

```
procedure NC(i):
D_i ← D_i ∩ {x | P_i(x)}
begin
  for i ← 1 until n do NC(i)
end
```

FIG. 1. NC-1: the node consistency algorithm.

Using NC-1, node consistency is achieved in $O(an)$ time.

Next we analyze the two arc consistency algorithms AC-1 and AC-3. (See Figs. 2 and 3.)

```
procedure REVISE((i, j)):
begin
  DELETE←false
  for each x ∈ D_i do
    if there is no y ∈ D_j such that P_ij(x, y) then
      begin
        delete x from D_i;
        DELETE←true
      end;
  return DELETE
end

1  begin
2    for i←1 until n do NC(i);
3    Q←{(i, j) | (i, j) ∈ arcs(G), i ≠ j}
4    repeat
5      begin
6        CHANGE←false;
7        for each (i, j) ∈ Q do CHANGE←(REVISE((i, j)) or CHANGE)
8      end
9    until ¬CHANGE
10 end
```

FIG. 2. AC-1: the first arc consistency algorithm.

Consider AC-1. Note that the number of arcs on Q is twice the number of edges and the length of Q does not change, that is, $|Q| = 2e$. The 'repeat' loop of lines 4–9 in Fig. 2 iterates until there is no deletion from any D_i. The maximum number of iterations occurs when only one element is deleted from one D_i on each complete iteration. There are then at most na iterations. Each iteration requires $|Q| = 2e$ calls to REVISE. Each call to REVISE requires at most a^2 evaluations of P_{ij}. Hence the worst case time complexity of AC-1 is $O(a^3ne)$.

```
1 begin
2   for i←1 until n do NC(i);
3   Q←{(i, j) | (i, j) ∈ arcs(G), i ≠ j}
4   while Q not empty do
5     begin
6       select and delete any arc (k, m) from Q;
7       if REVISE((k, m)) then Q←Q ∪ {(i, k) | (i, k) ∈ arcs(G), i ≠ k, i ≠ m}
8     end
9 end
```

FIG. 3. AC-3: the third arc consistency algorithm.

AC-3 is a simpler and more general version of AC-2, the Waltz filtering algorithm. It improves on AC-1 by only reconsidering arcs that may have become inconsistent whenever a deletion from a variable domain is performed. As with AC-1 initially the length of the queue of arcs waiting to be made consistent is $|Q| = 2e$. However Q may grow and shrink during the iterations of the 'while' loop (Fig. 3, lines 4–8) until it is finally exhausted. The worst case occurs when each element is deleted from each D_k on separate successful calls to REVISE and when, moreover, none of the arcs to be subsequently added to Q is already on it.

Entries are made in Q when a call to REVISE on an arc has succeeded. If REVISE$((k, m))$ has succeeded, then at most $(d_k - 1)$ arcs are added to Q. That number may be entered a times per vertex and so the total number of new entries made in Q is:

$$\sum_{k=1}^{n} a(d_k - 1) = a(2e - n).$$

Regardless of whether REVISE succeeds, one arc is deleted on each iteration and so the number of iterations is at most the original length of Q plus the total number of new entries.

$$2e + a(2e - n).$$

Each iteration may require a^2 binary predicate evaluations and so, the total number is at most $a^2(2e + a(2e - n))$.

If the constraint graph is not connected, each of its components may be treated independently so we may assume the graph is connected and hence $e \geq n - 1$ and so the time complexity may be written as $O(a^3e)$, that is, linear in the number of edges or binary constraints.

A lower bound on the worst case complexity can be obtained by considering the case when the network is already arc consistent. The number of predicate evaluations required to confirm that could be simply the original length of Q $(=2e)$, times a^2 or $2a^2e$. Thus the worst case running time for AC-3 is bounded below by $\Omega(a^2e)$ and above by $O(a^3e)$. Both bounds are linear in the number of edges.

For a complete graph, such as in the n-queens CSP, $e = \frac{1}{2}n(n-1)$ and so in general AC-3 is $O(a^3n^2)$. However many constraint graphs are sparse; that is, the number of edges is only linear in the number of vertices. For such graphs, e is $O(n)$ and so AC-3 is at most $O(a^3n)$ and at least $\Omega(a^2n)$. Planar graphs are, for example, sparse in that sense. Thus we have proven that the Waltz filtering algorithm *necessarily* has linear behavior on planar graphs. This behavior is not the result of any special attribute of the vision scene labelling domain in which it arose other than the sparsity of the constraint graphs. Those constraint graphs are planar because each vertex in the graph corresponds to a junction in a two-dimensional projection of a three-dimensional scene and each edge in the graph corresponds to the constraint that the three-dimensional edge corresponding to a straight line in the image must have the same interpretation at each of the two corners it connects. Those constraint graphs are thus embedded in the two-dimensional image plane.

Another intriguing question not yet answered is: when do node and arc consistency alone provide a sufficient guarantee that there is a complete solution? Freuder [3] has observed that the definition of k-consistency implies that any constraint network which is j-consistent for all $j \leq k$ can have each variable instantiated without backup due to failure on depth-first backtracking if the order of instantiation guarantees that any variable, when instantiated, is constrained directly by at most $k-1$ other previously instantiated variables. That is, under those conditions, a complete solution can be found from the network in linear time. For $k = 2$ a constraint graph that is a strict tree satisfies Freuder's requirement if one instantiates the variables in top-down order from the root of the tree to the leaves. When each variable is instantiated, only its parent directly constrains it. If the tree is node and arc consistent, then there must be at least one value for the variable consistent with the value for its parent.

Our new result above shows that AC-3 can be applied in $O(a^3n)$ time since a tree is a sparse graph. After applying AC-3 it is either the case that all the domains are empty which indicates that there is no solution or each of the domains contains at least one element and the network is node and arc consistent. In the latter case, a solution can subsequently be instantiated in at most $O(an)$ predicate evaluations. Thus we have shown that when the constraint graph is a strict tree, a solution to the CSP will be found (or failure reported if there are no solutions) in linear time.

Node and arc consistency alone may be sufficient in other cases not yet explained by these results perhaps due to domain specific attributes such as decoupling of constraint subgraphs or the degree of restrictiveness of the individual binary constraints [4].

The running time of arc consistency methods has been a matter of some controversy. Waltz [9] reported that his program required time "roughly proportional to the number of line segments in the scene" (which is the number of binary constraints). His program performed arc consistency using AC-2, a

special case of AC-3. Waltz attributed the linearity in part to a special property of polyhedral scene labelling, namely, the decoupling effect of T-junctions which limit the propagation of inconsistencies. Mackworth [6, p. 115] has some speculations on this topic which are not valid. Gaschnig [5] skeptically observed that Waltz provided only six measurements and argued that "Little can be concluded from so few datapoints". Gaschnig carried out some data analysis that cast doubt on the linear hypothesis, but cautioned that "little can be concluded with confidence from this plot".

3. The Complexity of Backtracking

The worst case of depth-first backtracking occurs when no solution exists solely because of an inconsistency between the last variable instantiated and the other variables. The number of pair tests is at most the number of leaves on the search tree, a^n, times the number of constraints, e, and so backtracking is $O(ea^n)$. This emphasizes the importance of reducing the domain size a as much as possible. This can be done beforehand by arc consistency and indeed for one class of constraint graphs, trees, an exponential algorithm can be replaced by a linear one.

4. The Complexity of Path Consistency

The analysis of the path consistency algorithms, PC-1 and PC-2, is analogous to the arc consistency analysis. PC-1 is due to Montanari [8]. We shall not repeat all the justification and explanation of the algorithms and the notation from Mackworth's paper [6]. But we quote the following paragraph [6, p. 107]:

> The arc consistency algorithms operate on an explicit data structure representation of the unary predicates, (i.e., the sets of all values that satisfy them, D_i) deleting values that cannot be part of a complete solution because of the restrictions imposed on adjacent nodes by the binary predicates. However, it is a matter of indifference to those algorithms whether the binary predicates are represented by a data structure or a procedure. The path consistency algorithms can be seen as generalizations in that although the predicate $P_{13}(x, y)$ may allow a pair of values, say, $P_{13}(a, b)$ that pair may actually be forbidden because there is an indirect constraint on v_1 and v_3 imposed by the fact that there must be a value, c, for v_2 that satisfies $P_{12}(a, c)$, $P_2(c)$ and $P_{23}(c, b)$. If there is no such value then that fact may be recorded by deleting the pair (a, b) from the set of value pairs allowed initially by P_{13}, in a fashion directly analogous to the deletion of individual values from the variable domains in the arc consistency algorithms. In order to perform that deletion it is necessary to have a data representation for the set of pairs allowed by a binary predicate. If the variable domains are finite and discrete then a relation matrix with binary entries is such a representation.

We represent the predicate P_{ij} by the relation matrix R_{ij}. The a rows of R_{ij} correspond to the a possible values of v_i and the a columns correspond to the a possible values of v_j. The algorithm deletes entries from copies of the relation matrices called Y_{ij} which are initialized to the corresponding R_{ij}. In the

n-queens CSP, for example, two queens may be placed on rows in their respective columns without violating the direct constraints. However, there may be no legal position for the queen in a third column that is compatible with both of those placements. This fact is recorded by deleting the entry in Y_{ij} that allowed that pair, since that pair of positions cannot appear in any legal solution. This example is expanded in more detail in the earlier paper [6].

The path consistency algorithms ensure that any pair of domain elements allowed by the direct relation R_{ij} between the vertices i and j is also allowed by all paths of any length from vertex i to vertex j. A theorem of Montanari's [8] states that, in a network with a complete graph, if every path of length 2 is path consistent then the network is path consistent. Both PC-1 and PC-2 thus need only examine all length-2 paths.

```
1  begin
2    Y^n ← R
3    repeat
4      begin
5        Y^0 ← Y^n
6        for k ← 1 until n do
7          for i ← 1 until n do
8            for j ← 1 until n do
9              Y_ij^k ← Y_ij^(k-1) & Y_ik^(k-1) · Y_kk^(k-1) · Y_kj^(k-1)
10     end
11   until Y^n = Y^0;
12   Y ← Y^n
13 end
```

FIG. 4. PC-1: the first path consistency algorithm.

The operation at line 9 of PC-1 (see Fig. 4) updates the relation matrix Y_{ij} by deleting any pair of values for v_i and v_j that is illegal because there is no legal value of v_k consistent with it. The operation $\cdot$ is binary matrix multiplication and & corresponds to element-by-element matrix intersection.

PC-1 is a symbolic relaxation algorithm that halts when there is no change in any of the relation matrices, at which point all paths of length 2 are consistent.

In analyzing PC-1 and PC-2, we shall use as the time complexity measure the number of binary operations. The loop of lines 3–11 in Fig. 4 is executed repeatedly until no change is observed in the total set of binary relations. The worst case is that at most one pair of elements is deleted from one relation on each iteration. There are n^2 binary relations and a^2 elements in each so the number of iterations is at most $O(a^2n^2)$. Each iteration performs line 9 of Fig. 4 n^3 times. The operation of making the path from vertex i to vertex j through vertex k consistent requires $O(a^3)$ binary operations if implemented conventionally. (This can be asymptotically improved to $O(a^{2.495548})$ [1].) Hence the worst-case time complexity of PC-1 is $O(a^5n^5)$.

PC-2 (see Fig. 5) operates in a fashion analogous to AC-3. It is based on the following observation: whenever a binary or unary relation is modified it is not necessary to re-examine, as PC-1 does, *all* the length-2 paths in the network; it suffices to re-examine only those length-2 paths containing the modified relation.

In PC-2 we represent the path from vertex i through vertex k to vertex j as the triple (i, k, j). The function RELATED PATHS$((i, k, j))$ returns the set of length-2 paths that might have their consistency affected by a change in the consistency of (i, k, j). Mackworth [6] gives the details of RELATED PATHS and shows that if $i \neq j$ the set returned has $2n - 2$ members whereas if $i = j$ it has $\frac{1}{2}n(n+1) - 2$ members.

```
procedure REVISE((i, k, j))
begin
  Z←Y_ij & Y_ik · Y_kk · Y_kj;
  if Z = Y_ij then return false
    else Y_ij←Z; return true
end

1 begin
2   Q←{(i, k, j) | (i ≤ j), —(i = k = j)};
3   while Q is not empty do
4     begin
5       select and delete a path (i, k, j) from Q;
6       if REVISE((i, k, j)) then Q←Q ∪ RELATED PATHS((i, k, j))
7     end
8 end
```

FIG. 5. PC-2: the second path consistency algorithm.

In analyzing PC-2 we use reasoning analogous to that used in analyzing AC-3 by examining the effect of successful and unsuccessful calls to REVISE on the length of the queue of paths waiting to be made consistent. On the successful calls to REVISE when $i = j$, an element has been deleted from Y_{ii} and $\frac{1}{2}n(n+1) - 2$ paths (at most) are added to Q. This can occur at most na times since there are at most na non-zero entries initially in all the Y_{ii}. When $i < j$ an element has been deleted from Y_{ij} and $2n - 2$ paths are added to Q. This can occur at most $\frac{1}{2}n(n-1)a^2$ times. And so the maximum number of new entries on Q is:

$$na(\tfrac{1}{2}n(n+1) - 2) + \tfrac{1}{2}n(n-1)a^2(2n-2) =$$
$$= (a^2 + \tfrac{1}{2}a)n^3 + (\tfrac{1}{2}a - 2a^2)n^2 + (a^2 - 2a)n$$

which is $O(a^2n^3)$.

On each iteration of lines 4–7 (Fig. 5), one path is deleted from Q. If REVISE is unsuccessful on that path, no new paths are added to Q, whereas if it is successful, a number of new paths must be added as enumerated above. Since the iteration proceeds until Q is exhausted the maximum total number of

iterations is the number of paths originally on Q, $\frac{1}{2}(n^3+n^2-2n)$, plus the maximum number of new entries computed above. The time complexity of PC-2 is then $O(a^5n^3)$. This is an improvement by a factor of n^2 over the bound on PC-1's behavior. A lower bound on the behavior of PC-2 is obtained by considering a network which is already path consistent yielding $\Omega(a^3n^3)$ so the algorithm is truly cubic in its behavior.

5. Conclusion

We have shown that arc consistency is achievable in time linear in the number of binary constraints. For a fully connected graph of n nodes the time complexity of AC-3 is $\Omega(a^2n^2)$ and $O(a^3n^2)$ but for sparse graphs, which occur in many applications, the complexity is $\Omega(a^2n)$ and $O(a^3n)$. Furthermore, if the constraint graph is a strict tree, then one can find a complete instantiated solution if one exists (or report failure if there are no solutions) in $O(a^3n)$ time. Path consistency is achievable in $\Omega(a^3n^3)$ and $O(a^5n^3)$ time. Moreover, the additional implementation complexity of AC-3 and PC-2 when compared with AC-1 and PC-1 is justified by guaranteed worst-case complexity of $O(n^2)$ and $O(n^3)$ respectively. It should be noted, however, that AC-1 and PC-1 have more inherent parallelism that AC-3 and PC-2.

Finally, we note that there are several ways to use these algorithms to achieve complete solutions to constraint satisfaction problems [2, 6, 7]. The simple approach is to use them to preprocess constraint networks before backtracking is applied. Constraint satisfaction techniques may also be interleaved with variable instantiation (backtracking) or case analysis (domain splitting). These techniques are attractive because at the cost of linear, quadratic or cubic time they may reduce the worst-case time complexity of backtracking exponentially by reducing the size of the variable domains and relations.

ACKNOWLEDGMENT

We are grateful to Raimund Seidel and David Kirkpatrick for useful discussions on this topic and to Judea Pearl and the referees for their suggestions. This material is based upon work supported by the Natural Sciences and Engineering Research Council Canada under Grant A9281 and the National Science Foundation under Grant No. MCS-8003307.

REFERENCES

1. Coppersmith, D. and Winograd, S., On the asymptotic complexity of matrix multiplication, *SIAM J. Comp.* **11** (1982) 472–492.
2. Freuder, E.C., Synthesizing constraint expressions, *Comm. ACM* **21** (1978) 958–966.
3. Freuder, E.C., A sufficient condition for backtrack-free search, *J. ACM* **29** (1982) 24–32.
4. Haralick, R.M. and Elliott, G.L., Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence* **14** (1980) 263–313.
5. Gaschnig, J., Performance measurement and analysis of certain search algorithms, CMU-CS-79-124 Tech. Rept., Carnegie-Mellon University, Pittsburgh, PA, 1979.