THE COMPLEXITY OF SOME POLYNOMIAL NETWORK CONSISTENCY ALGORITHMS FOR CONSTRAINT SATISFACTION PROBLEMS

by

A.K. Mackworth and E.C. Freuder

Technical Report 82-6

August 1982

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

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

E.C. Freuder
Department of Computer Science
University of New Hampshire
Durham, N.H., U.S.A.

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. In this paper we analyze the time complexity of several node, arc and path consistency algorithms. 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 that computational vision application the constraint graph is planar and so the complexity is linear in the number of variables.

Introduction

The purpose of this paper is to analyze the network consistency algorithms described in (Mackworth, 1977a).

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 CSP's in which the domains are discrete, finite sets and the relations are unary or binary. These restrictions are not necessary for consistency techniques to be applied (Mackworth, 1977a, 1977b; Freuder, 1978).

Since graph colouring is an NP-complete CSP it is most unlikely that a polynomial time algorithm exists for solving general CSP's. Accordingly, the class of network consistency algorithms was invented (Waltz, 1972; Montanari, 1974; Mackworth, 1977a; Freuder, 1978). 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 (1978) generalized those algorithms for k=1,...,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 result 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, 1972)) then that time bound is also linear in the number of variables.

The Complexity of Node and Arc Consistency

The algorithms below are reprinted from (Mackworth, 1977a) which should be consulted for a full explanation. The domain of variable i is D_i and P_{ij} is the binary constraint relation (predicate) between 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. We shall assume for simplicity of description of the results of the analysis that each D_i is 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.

```
procedure NC($i$):
$D_i \leftarrow D_i \cap \{x \mid P_i(x)\}$
begin
  for $i \leftarrow 1$ until $n$ do NC($i$)
end
```

NC-1: the node consistency algorithm

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

Next we analyze the two arc consistency algorithms AC-1 and AC-3.

```
procedure REVISE(($i, j$)):
begin
  DELETE $\leftarrow$ false
  for each $x \in D_i$ do
    if there is no $y \in D_j$ such that $P_{ij}(x, y)$ then
      begin
        delete $x$ from $D_i$;
        DELETE $\leftarrow$ true
      end;
  return DELETE
end

begin
  for $i \leftarrow 1$ until $n$ do NC($i$);
  $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arcs}(G), i \neq j\}$
  repeat
    begin
      CHANGE $\leftarrow$ false
      for each $(i, j) \in Q$ do CHANGE $\leftarrow$ (REVISE (($i, j$)) or CHANGE)
    end
  until $\neg$ CHANGE
end
```

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 <u>repeat</u> loop of lines 4-9 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 $i \leftarrow 1$ until $n$ do NC($i$);
3   $Q \leftarrow \{(i,j) \mid (i,j) \in \text{arcs}(G), i \neq j\}$
4   while $Q$ not empty do
5     begin
6       select and delete any arc $(k, m)$ from $Q$;
7       if REVISE $((k, m))$ then $Q \leftarrow Q \cup \{(i, k) \mid (i, k) \in \text{arcs}(G), i \neq k, i \neq m\}$
8     end
9 end
```

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 <u>while</u> loop (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 the Q, 2e, times $\frac{1}{2}$ 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 $e = \frac{1}{2}n(n-1)$ 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 shown that the Waltz filtering algorithm necessarily has linear behaviour on planar graphs. This behaviour is <u>not</u> the result of any special attribute of the vision scene labelling domain in which it arose other than the sparsity of the constraint graphs.

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 (1982) 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 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 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 tree satisfies the requirement; AC-3 can be applied in $O(a^3n)$ time since a tree is a sparse graph; if node and arc consistency are thereby achieved a solution can be instantiated in at most O(an) predicate evaluations.

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 (Haralick and Elliott, 1980).

The running time of arc consistency methods has been a matter of some controversy. Waltz (1972) 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 (1975) has some speculations on this topic which are not valid. Gaschnig (1979) sceptically 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".

The Complexity of Backtracking

The worst case of depth-first backtracking occurs when no solution exists solely because of a conflict 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.

The Complexity of Path Consistency

The analysis of the path consistency algorithms, PC-1 and PC-2, is analogous to the arc consistency analysis. We shall not repeat all the justification and explanation of the algorithms and the notation from (Mackworth, 1977). PC-1 is due to Montanari (1974).

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 (1974) 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.

```
begin
  $Y^n \leftarrow R$
  repeat
    begin
      $Y^0 \leftarrow Y^n$
        for $k \leftarrow 1$ until $n$ do
          for $i \leftarrow 1$ until $n$ do
            for $j \leftarrow 1$ until $n$ do
              $Y^k_{ij} \leftarrow Y^{k-1}_{ij} \,\&\, Y^{k-1}_{ik} \cdot Y^{k-1}_{kk} \cdot Y^{k-1}_{kj}$
    end
  until $Y^n = Y^0$;
  $Y \leftarrow Y^n$
end
```

PC-1: the first path consistency algorithm

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 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 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 minimally be improved to $O(a^{2.81})$ (Aho, Hopcroft and Ullman, 1974).) Hence the worst case time complexity of PC-1 is $O(a^5n^5)$.

PC-2 operates in a fashion analogous to AC-3. It is based on the following observation: whenever a binary relation is modified it not necessary to re-examine, as PC-1 does, <u>all</u> 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 include the path (i,k,j) and might have their consistency affected by a change in the consistency of (i,k,j). Mackworth (1977) 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
  $Z \leftarrow Y_{ij} \,\&\, Y_{ik} \cdot Y_{kk} \cdot Y_{kj}$
  if $Z = Y_{ij}$ then return false
    else $Y_{ij} \leftarrow Z$; return true
end
```

```
begin
  $Q \leftarrow \{(i, k, j) \mid (i \leqslant j), \neg(i = k = j)\}$
  while $Q$ is not empty do
    begin
      select and delete a path $(i, k, j)$ from $Q$;
      if REVISE$((i, k, j))$ then $Q \leftarrow Q \cup$ RELATED PATHS$((i, k, j))$
    end
end
```

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:

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

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

On each iteration of lines 4-7, one path is deleted from Q. If REVISE is unsuccessful on that path no new paths are added to Q whereas if it is sucessful 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, $(n^3+n^2-2n)/2$, plus the maximum number of new entries computed above. The worst case 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 behaviour. A lower bound on the worst case behaviour 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 behaviour.

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 $O(a^3n)$. Path consistency is achievable in $\Omega(a^3n^3)$ and $O(a^5n^3)$ time.

Moreover, the additional implementation complexity of the 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 than 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 (Mackworth, 1977a, 1977b; Freuder, 1978) but even the simple approach of using them to preprocess constraint networks before backtracking is applied is 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.

Acknowledgements

We are grateful to Raimund Seidel and David Kirkpatrick for useful discussions on this topic. 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

Aho, A.V., Hopcroft, J.E. and Ullman, J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

Freuder, E.C., Synthesizing constraint expressions. Comm. ACM 21, 11 (Nov. 1978), 958-966.

Freuder, E.C., A sufficient condition for backtrack-free search. J. ACM 29, 1 (Jan. 82), 24-32.

Haralick, R.M. and Elliott, G.L., Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14 (1980), 263-313.

Gaschnig, J. Performance measurement and analysis of certain search algorithms. CMU-CS-79-124 Tech. Report, Carnegie-Mellon Univ., 1979.

Mackworth, A.K. Consistency in networks of relations. Artificial Intelligence 8, 1977a, 99-118.

Mackworth, A.K. On reading sketch maps. Proc. IJCAI, Cambridge, MA, 1977b, pp. 598-606.

Montanari, U. Networks of constraints: fundamental properties and applications in picture processing. Information Science 7, 1974, 95-132.

Waltz, D.E. Generating semantic descriptions of scenes with shadows. Tech. Report MAC AI-TR-271, MIT, Cambridge, MA, 1972.