ARC AND PATH CONSISTENCY REVISITED[1]

Roger Mohr and Thomas C. Henderson[2]

CRIN
BP 239
54506 Vandoeuvre (France)

UUCS-85-101

6 August 1985

Abstract

Mackworth and Freuder have analyzed the time complexity of several constraint satisfaction algorithms [4]. We present here new algorithms for arc and path consistency and show that the arc consistency algorithm is optimal in time complexity and of the same order space complexity as the earlier algorithms. A refined solution for the path consistency problem is proposed. However, the space complexity of the path consistency algorithm makes it practicable only for small problems. These algorithms are the result of the synthesis techniques used in ALICE (a general constraint satisfaction system) and local consistency methods.

---

[1]This work was partially supported under an ADI contract.

[2]This work was done while the author was visiting professor at the University of Nancy I. Permanent address: Department of Computer Science, University of Utah, Salt Lake City, Utah 84112

1. Introduction

We define a constraint satisfaction problem as follows:

$N = \{i,j,...\}$ is the set of nodes, with $|N| = n$,

$A = \{b,c,...\}$ is the set of labels, with $|A| = a$,

$E = \{(i,j) \mid (i,j,)$ is an edge in NxN$\}$, with $|E| = e$,

$A_i = \{b \mid b \in A$ and (i,b) is admissable$\}$,

R_1 is a unary relation, and (i,b) is admissable if $R_1(i,b)$,

R_2 is a binary relation, and $(i,b)-(j,c)$ is admissable if $R_2(i,b,j,c)$.

The constraint satisfaction problem is to find all n-tuples in A^n which satisfy the given relations.

Several authors have presented algorithms to solve this problem. However, since the problem is NP-complete, it has been suggested by others that a preprocessing or filtering step be applied before the backtracking or search procedures [4, 5, 6, 1, 2]. Although node, arc and path consistency algorithms do not usually result in a solution, they do eliminate any labels failing to satisfy a minimum of consistency constraints. Such techniques have found wide application in artificial intelligence, pattern recognition and image analysis.

It has been shown by Mackworth and Freuder [4] that the worst case running time for their algorithms for arc and path consistency are bounded above by $O(ea^3)$ and $O(n^3a^5)$, respectively. We give arc and path consistency algorithms which are bounded above by $O(ea^2)$ and $O(n^3a^3)$, respectively. Moreover, the space requirements, although not negligible, are of the same order as Mackworth's algorithms.

The node consistency condition consists only in checking the unary relations on the differents nodes and keeping in the domain of each node value satisfying this unary constraint. Arc consistency checks the consistency of labels for each couple of nodes linked by a binary constraint and removes the labels that cannot satisfy this local condition.

Path consistency algorithms ensure that any pair of labelings $(i,b)-(j,c)$ allowed by a

direct relation is also allowed by all paths from i to j. It has been proven that for complete graphs, path consistency is equivalent to consistency of every path of length 2; therefore, this is equivalent to checking the consistency of every triple. Each graph can always be replaced by an equivalent graph by adding the **true** constraint between the nodes which are not connected.

The key idea of algorithm AC-3 given by Freuder and Mackworth is, when a label is removed from node i, to consider only the edges (i,j) because they are the only ones whose arc consistency may be effected by the change. The same idea applies for path consistency: when a pair of labelings is removed, the algorithm PC-2 considers only the length-2 paths that are related to the nodes of this pair. Therefore, algorithm AC-3 has complexity $O(ea^3)$ instead of $O(ena^3)$ for the brute force algorithm AC-1. PC-2 is of complexity $O(a^5n^3)$ whereas PC-1 is $O(a^5n^5)$.

Our improvement is based on the technical aspect of the ALICE system [Lauriere78]. ALICE was designed to solve most combinatorial problems using a unified and general strategy. However, it is not possible to express in this system that we want only to eliminate labels that are locally inconsistent. Carefully looking at how ALICE runs on this problem shows that it automatically applies the algorithm AC-4 we describe in section 2. Then it starts to find a solution to the complete problem by using - loosely speaking - backtracking. In fact, it applies AC-4 at each stage of backtracking.

2. Arc Consistency

If we consider arc consistency intuitively, we find that it is based on the notion of support. Suppose we are considering label b at node i. As long as b has a minimum of support from the labels at each of the other nodes j (j not equal to i), b is considered a viable label for node i. But once there exists a node at which no remaining label satisfies the required relation with b, then b can be eliminated as a possible label for node i.

The algorithm that we propose makes this support evident by assigning a counter to each arc-label pair. Such pairs are denoted by [(i,j),b] and indicate the arc from i to j with label b at node i. In addition, for each label c at node j, the set S_{jc} is constructed, where $S_{jc} = \{(i,b) \mid$ c at node j supports b at node i$\}$; that is, if c is eliminated at node j, then counters at [(i,j),b] must be decremented for each b supported by c. Finally, we use a

table, M, to keep track of which labels have been deleted from which objects, and a list, List, to control the propagation of constraints. The algorithm for arc consistency is given in Figure 1. Assume node consistency has already been assured.

Step 1

```
1  M := 0;  S_ib := Empty_set; List := Empty_set;
2  for (i,j) ∈ E do
3    for b ∈ A_i do
4      begin
5      total := 0;
6      for c ∈ A_j do
7        if R(i,b,j,c) then begin
8                           total := total+1;
9                           Append(S_jc,(i,b));
10                          end
11     if total = 0 then M[i,b] := 1;
12                  else counter[(i,j),b] := total;
13     end;

14 initialize List from M;
```

Step 2

```
15 while List not Empty_set do
16   begin
17   choose (j,c) from List and remove (j,c) from List;
18   for (i,b) ∈ S_jc do
19     begin
20     counter[(i,j),b] := counter[(i,j),b]-1;
21     if counter[(i,j),b] = 0 and M[i,b] = 0
22       then begin
23            Append(List, (i,b));
24            M[i,b] := 1;
25            end;
26     end
27   end.
```

Figure 1. Optimal Complexity Arc Consistency Algorithm AC-4

It is easy to see that line 7 of the innermost loop of the data structure initialization part of the algorithm can be executed at most ea^2 times. Thus, the number of elements in the sets S_{jc} is on the order of ea^2. Since line 12 is executed at most ea times, the total

number of counters is of the order ea; furthermore, since the value of total is bounded by a, then the maximum value for a counter is a. Line 14 simply puts the unique (i,b) pairs into list form; this requires order na time. Our measure of time complexity for the algorithm is the decrementing of a counter; note that the counters and decrement lists encode in a fixed way the binary relations. Thus, this measure is consistent with that of Freuder and Mackworth.

Now consider lines 15–27. A global consideration of the counters shows that if they never go negative, then there are at most

$$\sum_{i=1}^{ea} a = ea^2$$

decrementations. Another way to see this is to consider the bounds on the two loops at lines 15 and 18. First, we remark that a label is eliminated at most once from an object because the matrix M "remembers" that fact. Now, given that label c has been eliminated from node j, the only labels that can be affected are those at nodes i which have an edge to j. Let d_j be the vertex degree at node j; then since j can appear at most a times at line 17, and since there are at most $d_j a$ elements of S_{jc} for given j, we have that line 20 can be executed at most

$$\sum_{j=1}^{n} ad_j a = a^2 \sum_{j=1}^{n} d_j = a^2 e$$

Since the lower bound time complexity for arc consistency is $\Omega(ea^2)$ and the upper bound time complexity for AC–4 is $O(ea^2)$, we have an optimal algorithm. We have already shown that the space required is on the same order as that required to define the relations.

We do not claim that there are no faster algorithms; the one we suggest here can be obviously improved: in step 1 we can remove from A_i each b for which we have found that there is no more consistent labelling; this reduces the size of the $S_{j,c}$ and therefore reduces the complexity of steps 1 and 2. However this is not a major improvement for the worst case. For planar graphs e is of O(n). AC–4 will run in $O(na^2)$ and AC–3 in $O(na^3)$ and both are linear in the number of nodes.

2.1. Correctness of AC-4

We outline here the key steps for a complete proof of the correctness of AC-4. The same approach can be used to prove AC-3.

1. each label deleted from A_i is not admissible for arc consistency;
2. algorithm AC-1 is correct;
3. let us suppose that AC-1 has already removed the labeling $\{(i_p,b_p) \mid p=1,...,m\}$ and that AC-4 also removes them; let (i,b) be the next labeling removed by AC-1. Then prove that (i,b) will also be removed by AC-4. Then by induction AC-4 removes at least as many labelings as AC-1.

2.2. Space Complexity of AC-4

The sets S_{ib} can be represented as linked lists and therefore use a space proportional to their number of elements: $O(ea^2)$. The set M has to be represented by an array of bits and therefore its size is O(na). We have at most O(ea) counters. Therefore, the total space required is $O(ea^2)$. On real problems our algorithm never reaches its upper bound in space.

It should be noted that each algorithm must represent the graph and the possible labels for each of its nodes. This leads us to a minimal space requirement bounded by O(e + na). Algorithm AC-3 needs exactly this minimum upper bound.

3. Path Consistency

Montanari [5] proved that, for a complete graph, path consistency is equivalent to path consistency for all length-2 paths. If a graph is not complete it can be completed by adding edges with the always **true** relation. Therefore, the PC-1 algorithm examines only these short paths. We need to use the notation of PC-1: the relation between i and j is a Boolean matrix $R_{i,j}$ whose rows correspond to the possible labels for i and the columns to the possible labels for j.

```
 Algorithm PC-1
begin
  Y^n= R;              /* copy of the different matrixes */
  repeat
    begin
      Y^0 = Y^n
      for k in N do
```

```
        for i in N do
          for j in N do
          Y_{i,j}^k = Y_{i,j}^{k-1} and Y_{i,k}^{k-1}.Y_{k,k}^{k-1}.Y_{k,j}^{k-1}
    end
  until Y^n = Y^0;
  Y = Y^n
end
```

The body of the inner loop of PC-1 updates the relation matrixes $Y_{i,j}$ by deleting the pair of labels that are illegal because no legal label for k is consistent with them.

A similar approach can be used to find a lower complexity path consistency algorithm (see Figure 2). The maximum number of times line 12 will be executed is on the order of n^3a^3. (Remember that for this path consistency algorithm to work requires that the graph be complete; i.e., $e = n(n-1)/2$.) Likewise, a global consideration shows that if the counters never go negative, then since there are at most order n^3a^3 counters and each has a maximum value of a, then line 26 can be executed at most order n^3a^3 times. On the other hand, if we examine the loop bounds, we see that the **while** loop is executed at most na times since a given node can only appear once for each label. Finally, the **for** loop is bounded by the size of each S_{kd} which is of order n^2a^2. Taking the product, we have that line 26 is executed at most order $nan^2a^2 = n^3a^3$ times.

The space complexity is however very large: the number of counters is

$$\sum_{(i,j) \text{ in } NxN} |A_i| x |A_j| \; x \; |\{k \text{ in } N |\; k\text{-}i, k\text{-}j\}| \quad < \quad n^3a^2$$

The sum of the size of the different sets $S_{k,d}$ is bounded by:

$$\sum_{(i,j,k) \text{ in } NxNxN, k-i, k-j} |A_i| \; x \; |A_j| \; x \; |A_k| \quad < \quad n^3a^3$$

The space complexity of the whole algorithm is $O(n^3a^3)$. Because step 2 runs exactly in $O(n^3a^3)$ for a consistent network, this algorithm is truly cubic in its behavior.

Some optimization in space and time can be achieved. First, as was already mentioned in [3], when exploring the length-2 paths, we can limit ourselves to the paths (i,k,j) with $i<j$. This divides space and time by 2. Secondly, in step 2 we can update A_i and A_j each time one of (i,b) or (j,c) is put into M.

Step 1

```
1  M := 0;  S_kb := Empty_set;  List := Empty_set;
2  for (i,j) ε E do
3    for k=1,n do
4      for b ε A_i do
5        for c ε A_j such that R(i,b,j,c) do
6          begin
7          total := 0;
8          for d ε A_k do
9            if R(i,b,k,d) and R(k,d,j,c)
10             then begin
11                  total := total+1;
12                  Append(S_kd, ((i,j),b,c));
13                  end;
14         if total = 0 then begin
15                          M[i,b] := 1;
16                          M[j,c] := 1;
17                          end;
18                         else counter[(i,j),k,b,c] := total;
19         end;

20 initialize List from M;
```

Step 2

```
21 while List not Empty_set do
22   begin
23   choose (k,d) from List and remove (k,d) from List;
24   for ((i,j),b,c) ε S_kd do
25     begin
26     counter[(i,j),k,b,c] := counter[(i,j),b,c]-1;
27     if counter[(i,j),b,c] = 0
28       then begin
29            if M[i,b] = 0 then begin
30            end;              M[i,b] := 1;  Append(List, (i,b));
31                              end;
32            if M[j,c] = 0 then begin
33                              M[j,c] := 1;  Append(List, (j,c));


36     end;
37   end.
```

Figure 2. Reduced Complexity Path Consistency Algorithm PC-3

3.1. Improvement for "Empty" Graphs

Usually graphs, used in image applications for instance, are far from complete graphs and have their number of edges linear in the size of the node number. So let us suppose here that we have O(n) edges. Introducing the **true** relation between the not connected edges we are therefore increasing heavily the complexity. For instance, the result of $True_{i,k} \cdot R_{k,k} \cdot R_{k,j}$ can be computed in the obvious way in $O(n^2)$:

```
/* True_{i,k}(b,c)  <=>  b in A_i  and  c in  A_k  */

result = false;
for c in A_j do
  if there exists d in A_k such that R_{k,j}(d,c) then
    for b in A_i do  result(b,c) =  true
```

This algorithm runs in $O(a^2)$ instead of $O(a^3)$. The same can be stated for the product where the last term is a True matrix. If we have two True matrixes, i.e., we have to compute $True_{i,j} \cdot R_{k,k} \cdot True_{k,j}$, the computation is reduced to test if A_k is empty or not: this is performed in O(1). In fact this "computation" does not have to be performed. If A_k is empty the algorithm can stop: there is no solution! For this reason the length-2 paths using only True relations can be discarded in PC-1 and PC-2. Thus, we reduce the number of the length-2 edges from $O(n^2)$ to O(n); this reduces the complexity of PC-1 and PC-2 by a factor n.

For PC-3 this approach discards in step 2 all the k which are chosen and have to be connected at least to i or j. Therefore, only $O(n^2)$ triples (i,j,k) will be considered. The complexity is reduced here also by a factor n.

4. Conclusion

We have provided an optimal algorithm for arc consistency. We reduced the complexity of path consistency, but it still remains open whether the algorithm PC-3 is optimal. It is not obvious that any path consistency algorithm has to examine for each triple of nodes all possible labels in the worst case; if the answer is yes, then PC-3 is optimal.

For practical cases, AC-4 is easy to implement; however, it uses more space that AC-3. PC-3 is also easy to implement, however it may use a huge amount of space and therefore has to be run carefully. From our point of view, having a network consistency problem to solve, we prefer to run the ALICE system; using an AC-4 like algorithm at

each level of decision, it will run very fast on "common world" network problem providing the complete solution.

References

[1] Gaschnig, John.
Performance Measurement and Analysis of Certain Search Algorithms.
Technical Report CMU-CS-79-124, Carnegie-Mellon University, May, 1979.

[2] Haralick, R., Davis, L., Rosenfeld, A. and Milgram, D.
Reduction Operations for Constraint Satisfaction.
Information Sciences 14:199-219, 1978.

[3] Mackworth, A.K.
Consistency in Networks of Relations.
Artificial Intelligence 8:99-118, 1977.

[4] Mackworth, A.K. and E.C. Freuder.
The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Artificial Intelligence 25:65-74, 1985.

[5] Montanari, U.
Networks of Constraints: Fundamental Properties and Applications to Picture Processing.
Information Sciences 7:95-132, 1974.

[6] Rosenfeld, A., R. Hummel and S. Zucker.
Scene Labeling by Relaxation Operations.
IEEE Transactions on Systems, Man, and Cybernetics SMC-6(6):420-433, June, 1976.