J. DE KLEER

telligence Laboratory, AIM-551,

rtificial Intelligence Laboratory,

ublished, 1985.
is, *IEEE Trans. Pattern Anal.*

13 (1980) 81–132.
er programming language based
", Cambridge, MA, 1979.
verview, Inference Corporation,

unpublished, 1985.

er 1985

RESEARCH NOTE

Arc and Path Consistency Revisited*

Roger Mohr and Thomas C. Henderson**
CRIN, BP 239, 54506 Vandoeuvre, France

Recommended by Alan Mackworth

ABSTRACT

Mackworth and Freuder have analyzed the time complexity of several constraint satisfaction algorithms [5]. *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* [3].

1. Introduction

We define a constraint satisfaction problem (after Mackworth [4]) as follows

$N = \{i, j, \ldots\}$ is the set of nodes, with $|N| = n$,
$A = \{b, c, \ldots\}$ is the set of labels, with $|A| = a$,
$E = \{(i, j) | (i, j)$ is an edge in $N \times N\}$, with $|E| = e$,
$A_i = \{b | b \in A$ and (i, b) is admissible$\}$,
R_1 is a unary relation, and (i, b) is admissible if $R_1(i, b)$,
R_2 is a binary relation, and (i, b)–(j, c) is admissible if $R_2(i, b, j, c)$.

*This work was partially supported under an ADI contract.

**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, UT 84112, U.S.A.

Artificial Intelligence **28** (1986) 225–233

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 [1, 2, 5, 7, 8]. 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 [5] that the worst-case running times 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 different nodes and keeping in the domain of each node values 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 complete 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 affected 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 a technical aspect of the ALICE system [3]. 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. (See also [6].)

2. Arc

If we consider arc consis
support. Suppose we a
minimum of support fro
i), b is considered a viab
no remaining label satisf
as a possible label for i

The algorithm that w
counter to each arc-labe
the arc from i to j with
the set S_{jc} is constructe
that is, if c is elimina
decremented for each b
of which labels have bee
the propagation of const
1. Assume node consiste
of the innermost loop of
be executed at most ea^2
the order of ea^2. Since l
counters is of the orde
by a, the maximum val
(i, b) pairs into list for
complexity for the algo
counters and decrement
this measure is consiste

Now consider lines 15
they never go negative,

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

decrementations. Anoth
loops at lines 15 and 18.
from an object because
label c has been elimina
those at nodes i which h
then since j can appear a
elements of S_{jc} for a giv

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

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 $S_{jc} = \{(i, b) | c \text{ at node } j \text{ supports } b \text{ at node } i\}$; 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 Fig. 1. Assume node consistency has already been assured. 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, 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 a given j, we have that line 20 can be executed at most

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

Step 1. *Construction of the data structures*

```
1  M := 0; S_ib := 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
8            begin
9              Total := Total + 1;
               Append(S_jc, (i, b));
10           end;
11       if Total = 0 then M[i, b] := 1; A_i := A_i − {b};
12                    else Counter[(i, j), b] := Total;
13     end;
14 initialize List with {(i, b) | M(i, b) = 1};
```

Step 2. *Pruning the inconsistent labels*

```
15 while List not Empty 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 then
22           begin
23             Append(List, (i, b));
24             M[i, b] := 1; A_i := A_i − {b};
25           end;
26       end;
27   end
```

FIG. 1. Optimal complexity arc consistent algorithm AC-4.

Since the lower bound time complexity for arc consistency is $O(ea^2)$ 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 labeling; this reduces the size of the S_{jc} and therefore reduces the complexity of Steps 1 and 2. However, this is not a major improvement for the worst case. But it is very easy to add and may in some cases divide the complexity by a factor of 2. 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
The same approach can

Step 1. By induction, e
consistency solution: the l
no more corresponding la
labels could not belong to

Step 2. The result of A
i, we have Counter[(i, j),
AC-4 builds an arc consis

Step 3. From Steps 1
consistent solution.

2.2. Space complexity of

The sets S_{ib} can be rep
proportional to their n
represented by an array c
$O(ea)$ counters. Therefor
our algorithm never reac

It should be noted th
possible labels for each
requirement bounded b
minimum upper bound.

3. Path

Montanari [7] proved tha
to path consistency for a
completed by adding edg

```
begi
Y'
rep
b
e
unt
Y =
end
```

FIG. 2. The path consistency a

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.

Step 1. By induction, each label delected from A_i is not admissible for any arc consistency solution: the label is removed if one of its counters goes to zero; so it has no more corresponding labels at one edge; by induction all the previously removed labels could not belong to any solution, so this one cannot belong to any solution.

Step 2. The result of AC-4 is arc consistent: for all edges (i, j), for all labels b for i, we have Counter$[(i, j), b] > 0$ so b has a corresponding label node j; therefore, AC-4 builds an arc consistent solution.

Step 3. From Steps 1 and 2 we conclude that AC-4 builds the largest arc consistent solution.

2.2. Space complexity of AC-4

The sets $S_{i,b}$ 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 [7] 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

```
begin
  Y^n = R;   /* copy of the different matrices */
  repeat
    begin
      Y^0 = Y^n
      for k in N do
        for i in N do
          for j in N do
            Y^k_{i,j} = Y^{k-1}_{i,j} and Y^{k-1}_{i,k} · Y^{k-1}_{k,k} · Y^{k-1}_{k,j}
    end
  until Y^n = Y^0;
  Y = Y^n
end
```

FIG. 2. The path consistency algorithm PC-1.

algorithm (Fig. 2) 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.

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

4. Algorithm PC-3

A similar approach can be used to find a lower complexity path consistency algorithm (see Fig. 3). For each edge (i, j), for each node k, and each label b for i and c for j, we introduce Counter$[(i, j), k, b, c]$ which counts the number of labels for node k that are still consistent with the assignment of label b to i and c to j. The sets S_{kd} provide all the $((i, j), b, c)$ that are admissible with the label d for node k. 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 it is required that the graph be complete; i.e., $e = \frac{1}{2}n(n-1)$.) Likewise, a global consideration shows that if the counters never go negative, then since there are at most order n^3a^2 counters and each has a maximum value of a, 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

$$\sum_{(i,\,j)\in N\times N} |A_i| \times |A_j| \times |\{k \in N \mid k \text{ neighbors } i \text{ and } j\}| < n^3a^2 .$$

The sum of the size of the different sets S_{kd} is bounded by:

$$\sum_{(i,\,j,\,k)\in N\times N\times N} |A_i| \times |A_j| \times A_k| < n^3a^3 .$$

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 [4], 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.

4.1. 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

Step 1

```
1  M := 0; S_kd
2  for (i, j) ∈
3    for k = 1
4      for b ∈
5        for
6          be
7
8
9
10
11
12
13
14
15
16
17
17
18
19           en
20 initialize Li
```

Step 2

```
21 while List
22   begin
23     choos
24     for ((i,
25       beg
26         C
27         if
28
29
30
31
31
32
33
34
35
36           en
37     end
```

FIG. 3. Reduced complexity

Step 1

```
1  M := 0; S_kb := Empty_set; List := Empty;
2  for (i, j) ∈ E do
3    for k = 1, n do
4      for b ∈ A_i do
5        for c ∈ A_j such that R(i, b, j, c) do
6          begin
7            Total := 0;
8            for d ∈ A_k do
9              if R(i, b, k, d) and R(k, d, j, c) then
10               begin
11                 Total := Total + 1;
12                 Append(S_kd, ((i, j), b, c));
13               end;
14           if Total = 0
15             then
                 begin
16                 M[i, b] := 1; A_i := A_i − {b};
17                 M[j, c] := 1; A_j := A_j − {c};
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 do
22   begin
23     choose (k, d) from List and remove (k, d) from List;
24     for ((i, j), b, c) ∈ 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 then
28           begin
29             if M[i, b] = 0 then
30               begin
31                 M[i, b] := 1; Append(List, (i, b)); A_i := A_i − {b};
31               end;
32             if M[j, c] = 0 then
33               begin
                   M[j, c] := 1; Append(List, (j, c)); A_j := A_j − {c};
34               end;
35         end;
36       end;
37   end
```

FIG. 3. Reduced complexity path consistency algorithm PC-3.

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 $\text{True}_{i,k} \cdot R_{k,k} \cdot R_{k,j}$ 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 matrices, i.e., we have to compute $\text{True}_{i,j} \cdot R_{k,k} \cdot \text{True}_{k,j}$, 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.

5. 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 than 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 algorithm like AC-4 at each level of decision, it will run very fast on "common world" network problem providing the complete solution. ALICE is running in PL/1 under VM.

REFERENCES

1. Gaschnig, J., Performance measurement and analysis of certain search algorithms. Tech. Rept. CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh, PA, 1979.
2. Haralick, R., Davis, L., Rosenfeld, A. and Milgram, D., Reduction operations for constraint satisfaction, *Inform. Sci.* **14** (1978) 199–219.
3. Lauriere, J.-L., A language and a program for stating and solving combinatorial problems, *Artificial Intelligence* **10** (1978) 29–127.

4. Mackworth, A.K., Consist
5. Mackworth, A.K. and Fre
algorithms for constraint s
6. Mohr, R. and Masini, G., R
France, 1986.
7. Montanari, U., Networks
processing, *Inform. Sci.* 7
8. Rosenfeld, A., Hummel,
Trans. Systems Man Cyber

Received October 1985

ducing the *true* relation
e increasing heavily the
$R_{k,j}$ can be computed in

me can be stated for the
e two True matrices, i.e.,
tation is reduced to test if
this "computation" does
hm can stop: there is no
ly True relations can be
ber of the length-2 edges
1 and PC-2 by a factor n.
hich are chosen and have
^{3}) triples (i, j, k) will be
factor n.

stency. We reduced the
n whether the algorithm
stency algorithm has to
n the worst case; if the

ever, it uses more space
ıay use a huge amount of
point of view, having a
the ALICE system; using
t will run very fast on
ıplete solution. ALICE is

arch algorithms, Tech. Rept.
79.
tion operations for constraint

ing combinatorial problems.

4. Mackworth, A.K., Consistency in networks of relations, *Artificial Intelligence* **8** (1977) 99–118.
5. Mackworth, A.K. and Freuder, E.C., The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artificial Intelligence* **25** (1985) 65–74.
6. Mohr, R. and Masini, G., Running efficiently arc consistency, Tech. Rept. 86-R-001, CRIN, Nancy, France, 1986.
7. Montanari, U., Networks of constraints: Fundamental properties and applications to picture processing, *Inform. Sci.* **7** (1974) 95–132.
8. Rosenfeld, A., Hummel, R. and Zucker, S., Scene labeling by relaxation operations, *IEEE Trans. Systems Man Cybernet.* **6** (1976) 420–433.

Received October 1985