Proc. of the 1991 IEEE
Int. Conf. on Tools for AI
San Jose, CA-Nov. 1991

Improving Han and Lee's Path Consistency Algorithm

Yangjun Chen

Technical Institute of Changsha, Hunan, China
current address: Department of Computer Science, University of Kaiserslautern
P. O. Box 3049, 6750 Kaiserslautern, Germany

Abstract

Han and Lee have presented a path consistency algorithm [4]. Here we describe a new algorithm for path consistency and show that the algorithm requires less time and space than Han and Lee's. The key idea of the algorithm is the arranging of the edges which are checked in the first part of Han and Lee's algorithm and the interlacing of the second part of the algorithm in the first part by using the symmetry of the triangle.

1 Introduction

Many problems in artificial intelligence can be seen as special cases of a general NP-complete problem [5] that has been called the "consistent-labeling problem" by Haralick [6, 8], the "satisfying assignment problem" by Gaschnig [3] and the "constraint satisfaction problem" by Fikes and others [1].

A constraint satisfaction problem can be defined as follows [9]

$N = \{i, j, ...\}$ *is the set of nodes, with* $|N| = n$,
$A = \{b, c, ...\}$ *is the set of labels, with* $|A| = a$,
$E = \{(i, j) | (i, j)$ *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* $R_2(i, b, j, c)$.

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

By the node consistency, only the unary relations on the different nodes are checked and the values satisfying these unary constraints are kept in the domain of each nodes. The arc consistency consists in checking the consistency of labels for each couple of nodes linked by a binary constraint and removing the labels that cannot satisfy this local condition. Path consistency algorithms ensure that any pair of labeling $(i, b) - (j, c)$ allowed by a direct relation is also allowed by all paths from i to j. A theorem of Montanari's [11] states 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 idea of the arc consistency algorithm introduced by Mohr and Henderson [10]is based on the notion of support. (The notion of support was first defined by Mackworth [9].) Han and Lee further use this notion for path consistency checking [4]. Their algorithm consists of two steps. In the first step of their algorithm, the inconsistency is checked and recorded in some data structure. In the second step, the inconsistency is propagated and eliminated. We can improve the efficiency of the algorithm by arranging the edges which are checked in the first step and interlacing the two steps in some way.

In the next section we introduce the refined path consistency algorithm. In Section 3 we prove the correctness of the algorithm. In Section 4 we analyse the computational complexity. Section 5 is a short conclusion.

2 Refined Path Consistency Algorithm

Below is the refined algorithm for path consistency. In

0-8186-2300-4/91 $01.00 © 1991 IEEE

the algorithm, we use a list, *LIST*, to store labelings (i, b) - (j, c) which have been found to be illegal but which have not yet been propagated, and a matrix, M, to keep track of which labelings have been deleted from which edges. For each edge (i, j), for each node k, and each label b for i and c for j, we introduce Counter$[(i, j); (b, c); k]$ 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, b) - (j, c) that are admissible with the label d for the node k. In addition, for each domain of A_i $(i = 1, \dots n)$, the array B_i $(i = 1, \dots n)$ is constructed, where B_{ib} is the number of A_j $(i \neq j)$, for which the arc-label pairs $[(i, j), b] \neq 0$. ($[(i, j), b]$ indicates the number of pairs of labelings for the arc from i to j with label b at node i). f is used to assign values to B_i $(i = 1, \dots n)$ and M.

procedure PC

```
begin
    $M := 0$;  $S_{kd}$ := Empty_set; $LIST$ := Empty; $f := 0$; $B_{ib} = 0$;
    for $i = 1$ to n-1 do
        for $j = i+1$ to n do
            for $k \neq i, j$ do
                for $b \in A_i$ and $B_{ib} = f$ do
                    for $c \in A_j$ and $B_{jc} = f$ such that $R(i, b, j, c)$ do
                        begin
                            $B_{jc} := f+1$;
                            for  $d \in A_k$ do
                                if $R(i, b, k, d)$ and $R(k, d, j, c)$ then
                                    begin
                                        Counter$[(i, j); (b, c); k]$ := Counter$[(i, j); (b, c); k]$ +1;
                                        Append($S_{kd}$, $(i, b)$ - $(j, c)$);
                                        Counter$[(i, k); (b, d); j]$ := Counter$[(i, k); (b, d); j]$ +1;
                                        Append($S_{jc}$, $(i, b)$ - $(k, d)$);
                                        Counter$[(j, k); (c, d); i]$ := Counter$[(j, k); (c, d); i]$ +1;
                                        Append($S_{ib}$, $(j, c)$ - $(k, d)$);
                                    end
                                if  Counter$[(i, j); (b, c); k]$ =0
                                  then
                                    begin
                                        $M[i, j, b, c] := f+1$;
                                        eliminate  $(i, b)$ - $(j, c)$ form $R_2$;
                                    end
                                 if  Counter$[(i, k); (b, d); j]$ =0
                                  then
                                    begin
                                        $M[i, k, b, d] := f+1$;
                                        eliminate $(i, b)$ - $(k, d)$ from $R_2$;
                                    end
                                if  Counter$[(j, k); (c, d); i]$ =0
                                  then
                                    begin
                                        $M[i, k, c, d] := f+1$;
                                        eliminate $(j, c)$ - $(k, d)$ form $R_2$;
                                    end
                        end
        eliminate all labelings $(l, u)$ - $(m, v)$ $(l, m \neq i)$ whose corresponding $B_{lu}$ or $B_{mv}$ is $f$;
        initialize $LIST$ with  $\{(l, u) - (m, v) \mid M[l, m, u, v] = f+1]$ ;
```

```
41          while LIST not Empty do
42              begin
43                  choose (s, u) - (t, v) from LIST and remove it from LIST;
44                    for (l, m) = (s, t), (t, s); (x, y) = (u, v), (v, u) do
45                      for (l, x) - (r, e) ∈ $S_{my}$ do
46                          begin
47                              Counter[(l, r); (x, e); m] := Counter[(l, r); (x, e); m] -1;
48                              if Counter[(l, r); (x, e); m] = 0 then
49                                  begin
50                                      if M[l, r, x, e] = 0 then
51                                          begin
52                                              M[l, r, x, e] := f+1; Append[LIST, (l, x) - (r, e)];
53                                              eliminate R(l, x, r, e) from $R_2$;
54                                          end
55                                  end
56                          end
57              end
58          f := f + 1;
59  end
```

Note that the second part (lines 41-59) of the algorithm, which propagates and eliminates the inconsistency, is interlaced in the first part (lines 1-40) of the algorithm. It makes the algorithm check fewer labels for each couple of nodes linked by a binary constraint since many inconsistent arc-label pairs are removed before they are checked in the first part. From line 7 and line 9 we can see that the value of B_{jc} is determined by $R(i, b, j, c)$ and from line 39 we see that the cardinality of R will be reduced, owing to the interlacing of the second part of the algorithm in the first part, by using the values of B_i. Therefore, the number of checks performed in the first part of the algorithm is decreased. It lowers the computational complexity of the algorithm.

3 Correctness of PC

Montanari [11] 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. Now we prove that for each edge (i, j) PC checks all length-2 paths from vertex i to vertex j. On the first iteration of PC only the consistency of edges $(1, 2), (1, 3), \dots, (1, n)$ is checked (see Fig. 1), that is, all the triangles with one of these edges as an arc are checked. Since for each triangle the counter for each arc-label pair and the set S_{ib} are simultaneously constructed, the propagation and elimination of the inconsistency can be immediately executed in the second part of PC. For example, for the arc from vertex 1 to vertex 2 all the triangles Δ_{123}, Δ_{124}, ..., Δ_{12n} will be checked and the illegal labelings will be immediately propagated and eliminated (see Fig. 2). The same analysis can be applied for the arc from vertex 1 to vertex 3. However, for the arc from vertex 2 to vertex 3 the propagation of the inconsistency is constrained only to the triangle Δ_{231}. From line 7 and line 39 we can see that if an arc-label pair is removed from R_2, the arc-label pairs supported by the removed arc-label pair will be ignored and deleted on the consequent iterations. This guarantees that every path of length 2 in the solution binary relations of PC is path consistent. From the analysis and the theorem of Montanari's [11] we have our first proposition.

Proposition 3.1 Let R_2' be the solution binary relation found by PC. Then R_2' is path consistent.

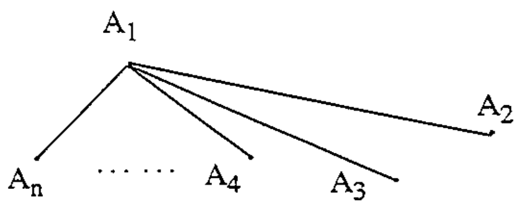

Fig. 1

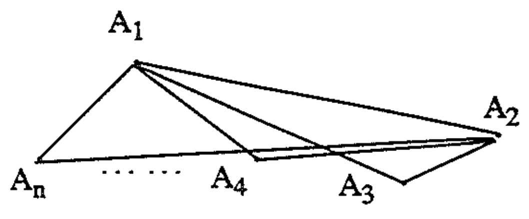

Fig. 2

The next question is: is any solution lost by using the array B_i ($i = 1, \ldots n$) to remove the inconsistent arc-label pairs earlier? The following analysis shows that the set of solutions found by PC is complete (see [2] for the exact definition of completeness).

At the beginning of the algorithm the array B_i ($i = 1, \ldots n$) is made equal to 0, which indicates that on the entering the first iteration of the outermost **for** loop from line 1 to line 59, each label in the domain will be checked.

After the first iteration of this outermost **for** loop the labels at node 2, 3, ..., n which have no support from the labels at node 1 are marked. The pairs of labelings with these labels at one node will be deleted and these labels will also be ignored on the subsequent iterations because any pair of labelings with these labels at one node has no length-2 path through node 1. On the second iteration of the outermost **for** loop the consistency of edges (2, 3), (2, 4), ..., (2, n) will be checked (see Fig.3). In the similar way, the labels at node 3, 4, ..., n which have no support from the labels at node 2 will be marked and will be ignored. This process can be viewed as a "forward-checking" [8, 12].

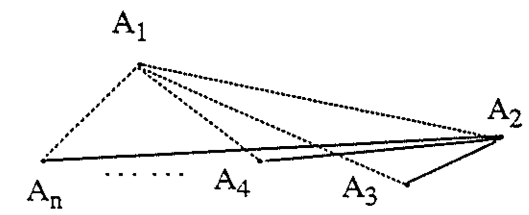

Fig. 3

From the above analysis we have the second proposition.

Proposition 3.2 Let R_2' be the solution binary relation found by PC. Then R_2' is complete.

4 Computational Complexity of PC

In this section we consider the computational complexity of PC. The constraint satisfaction problems can be represented by their *relations matrix* $[T^{ij}_{kl}]$ (undefined for $i = j$), a bit-matrix such that element $T^{ij}_{kl} = 1$ iff the kth value for node i is consistent with the lth value for node j. Otherwise bit $T^{ij}_{kl} = 0$. This is essentially a truth-table representation of all relations between pairs of variables.To simplify the description of the results of the analysis, we shall assume that A_i has the same size a and Prob $(T^{ij}_{kl} = 1) = p$ for all i, j, k and l. That is, the probability of compatibility of any two values for any two nodes equals p.

On the first iteration of the outermost **for** loop the maximum number of times lines 12-38 will be executed is on the order of $(n-1) \cdot (n-2) \cdot (p^0 a)^3$. On the second iteration of the outermost **for** loop the maximum number of times is on the order of $(n-2) \cdot (n-3) \cdot (p^1 a)^3$. In general, on the ith iteration of the outermost **for** loop the maximum number of times is on the order $(n-i) \cdot (n-i-1) \cdot (p^{i-1} a)^3$. Therefore, the time complexity of the first part of PC is

$$\sum_{i=1}^{n-1} (n-i) \cdot (n-i-1) \cdot (p^{i-1} a)^3$$
$$< n^2 a^3 \sum_{i=1}^{n-1} p^{3(i-1)}$$

Note that $\sum_{i=1}^{n-1} p^{3(i-1)}$ is less than the infinite series

$$p^{3\cdot 0} + p^{3\cdot 1} + p^{3\cdot 2} + p^{3\cdot 3} + \ldots,$$

which converges to

$$\frac{1}{p-1} \quad \text{for } p < 1.$$

Since 1/(1-p) is a constant that is independent of n, then the time complexity of the first part of PC is $O(n^2 a^3)$. In the ith iteration of the outermost **for** loop, the total number of the corresponding counter is of the order $(n-i) \cdot (n-i-1) \cdot (p^{i-1} a)^2$; furthermore, since the labels which are checked in the *i*th iteration of the outermost **for** loop is at most $p^{i-1}a$, the maximum value for a counter is $p^{i-1}a$. Therefore, for the counters there are at most

$$\sum_{i=1}^{n-1} (n-i) \cdot (n-i-1) \cdot (p^{i-1}a)^3$$
$$< n^2a^3 \sum_{i=1}^{n-1} p^{3(i-1)} = O(n^2a^3)$$

decrementations.

The space complexity of PC is: number of counters

$$|counter| \le O\left(\sum_{i=1}^{n-1} (n-i) \cdot (n-i-1) \cdot (p^{i-1}a)^2\right)$$
$$< O\left(n^2a^2 \sum_{i=1}^{n-1} p^{2(i-1)}\right) = O(n^2a^2).$$

and sum of the size of the different sets S_{ib}

$$|S| \le O\left(\sum_{i-1}^{n-1} (n-i) \cdot (n-i-1) \cdot (p^{i-1}a)^3\right)$$
$$< O\left(n^2a^3 \sum_{i=1}^{n-1} p^{3(i-1)}\right) = O(n^2a^3).$$

So the space complexity of PC is

$$O\left(\sum_{i=1}^{n-1} (n-i) \cdot (n-i-1) \cdot (p^{i-1}a)^3\right)$$
$$< O\left(n^2a^3 \sum_{i=1}^{n-1} p^{3(i-1)}\right) = O(n^2a^3).$$

5 Conclusion

In this paper a path consistency algorithm PC has been presented which requires less time and space than Han and Lee's algorithm. Both the time and space complexity are

$$O\left(\sum_{i=1}^{n-1} (n-i) \cdot (n-i-1) \cdot (p^{i-1}a)^3\right) = O(n^2a^3).$$

The key idea of the algorithm is the arranging of the edges which are checked in the first part of the algorithm and the interlacing of the second part of the algorithm in the first part, which makes the first part of the algorithm check fewer labels for each couple of nodes linked by a binary constraint; Furthermore, since the first part of the algorithm makes fewer checkings, the maximum value for a counter is smaller. Our measure of time complexity for the second part of the algorithm is the decrementing of a counter. Thus the second part will be executed more quickly. In order to interlace the two parts of the algorithm, we have to construct simultaneously counters for each edge of a triangle and sets S_{ib} for each vertex of a triangle instead of only for an edge and a vertex (see lines 12-38). We achieve the optimization in time and space on the basis of the symmetry of the triangle.

REFERENCES

1. Fikes, R.E., REF-ARF: A system for solving problems stated as procedures, *Artificial Intelligence* **1** (1970) 27-120.

2. Freunder, E. C., Synthesizing constraint expressions, *Comm. ACM* **29** (1978) 24-32.

3. Gaschnig, J., Performance measurement and analysis of certain search algorithms, Ph.D. Thesis, Dept. Computer Science, Carnegie-Mellon University, 1979.

4. Han, C.-C. and Lee, C.-H., Comments on Mohr and Henderson's path consistency algorithm, *Artificial Intelligence* **36** (1988) 125-130.

5. Haralick, R.M., Davis, L.S. and Rosenfeld, A., Reduction operations for constraint satisfaction, *Information Sci.* **14** (1978) 199-219.

6. Haralick, R.M. and Shapiro, L.G., The consistent labeling problem: Part I, *IEEE Trans. pattern Anal. Machine Intelligence* 1(2) (1979) 173-184.

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

8. Haralick, R.M. and Shapiro, L.G., The consistent labeling problem: Part II, *IEEE Trans. pattern Anal. Machine Intelligence* 2(3) (1980) 193-203.

9. Mackworth, A. K., Consistency in networks of relations, *artificial Intelligence* **8** (1977) 99-118.

10. Mohr, R. and Henderson, T.C., Arc and path consistency revisited, *Artificial Intelligence* **28** (1986) 225-233.

11. Montanari, U., Networks of constraints: Fundamental properties and applications to picture processing, *Inform. Sci.* 7 (1974) 95-132.

12. Nudel, B.A., Consistent-labeling problems and their algorithms: expected-complexities and theory-based heuristics, *Artificial Intelligence* 21 (1983) 135-178.