Expensive Constraints and HyperArc Consistency

Rajiv Chopra, Rohini Srihari and Anthony Ralston
Dept. of Computer Science, SUNY at Buffalo

April 11, 1995

Abstract

We present an arc-consistency algorithm NAC4 to account for some types of non-binary constraints. It differs from the generalized algorithm GAC4[Mohr and Masini, 1988] as the constraints need not be modeled as explicit relations but can be retained as implicit n-ary relations. The advantage of this algorithm over GAC4 is pronounced when the checking of constraints is expensive. By an ordering of the constraints according to cost, expensive constraints can be applied after simpler ones have hopefully reduced the domains of some of the nodes. The particular non-binary constraints for which this algorithm is suited include constraints between a node and a set of other nodes.

1 Introduction

Consistency algorithms, also called discrete relaxation algorithms, were first introduced by Waltz for interpreting polyhedral scenes with shadows [Waltz, 1975]. The central idea was the propagation of constraints through a network of hypotheses. Each node of the network represents a given fact with several interpretations. Constraints on these nodes allow one to discard some of them. Hence, relaxation is a tool for solving the labeling problem, which includes the graph-coloring problem and therefore is a NP-hard problem [Garey *et al.*, 1974].

Backtracking with forward checking [Haralick and Elliot, 1980] prunes the search tree of nodes based upon incompatibility with the partial labeling constructed so far. Consistency algorithms go one step further in removing incompatible future labelings. These inconsistencies would have otherwise been discovered repeatedly by most backtracking procedures. Discrete relaxation runs in polynomial time and only produces a locally consistent solution. In the case of a tightly constrained Constraint Satisfaction Problem (CSP), an iterative application of the algorithm prior to backtracking can greatly reduce or totally eliminate the tree search. Such techniques have found wide application in artificial intelligence, pattern recognition and image analysis.

The labeling problem can also be solved using continuous relaxation techniques [Faugeras and Berthod, 1981] which, although capable of handling fuzzy constraints, are difficult

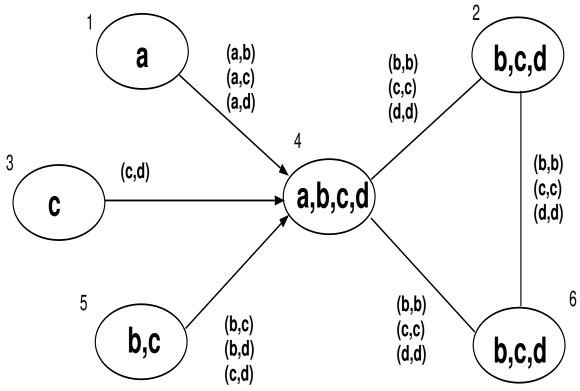

Figure 1: An example of a binary CSP with constraints modeled as explicit relations. The ovals are nodes, values in them are labels and the lists of pairs are constraints.

to control. The user has to juggle various coefficients of the system to obtain the correct solution. Discrete relaxation handles only boolean constraints and is thus easier to manipulate. A series of algorithms, notably **AC3** [Mackworth and Freuder, 1985], **AC4** [Mohr and Henderson, 1986] culminated in a generic arc-consistency algorithm **AC5** [Hentenryck *et al.*, 1992] which was similar to AC4 but better for a number of important classes of constraints (functional, monotonic etc.). An excellent reference on constraint satisfaction problems and the various approaches to solving them can be found in [Tsang, 1993]. A generalization of AC4 to handle n-ary constraints, called **GAC4** was presented in [Mohr and Masini, 1988].

Section 2 provides the motivation for exploring solutions for expensive constraints. Definitions and related work are presented in the following sections. Section 6 presents a new algorithm for n-ary constraints. Expensive constraints are discussed in Section 8.

2 Motivation

Consistency algorithms for finite domains uniformly assume that constraints can be enumerated (or provided as input)as explicit relations with little or no cost, and that verifying the existence of a relation is a unit cost operation. However, there exist applications of CSP where computing the extension of a constraint relation is neither inexpensive nor desirable. Such an application domain is computer vision[Mohr and Masini, 1988] [Srihari *et al.*, 1994]. When dealing with expensive constraints, reducing the domains of the relation's nodes prior to computing its extension seems an obvious improvement. Ordering the constraints according to their application-domain specified cost would also be desirable.

3 Definitions

A *static* constraint satisfaction problem [Mohr and Henderson, 1986][Mackworth, 1977] (V, D, C, R) involves a set of m **nodes**, $V = \{v_1, v_2, \dots v_m\}$, each with an associated

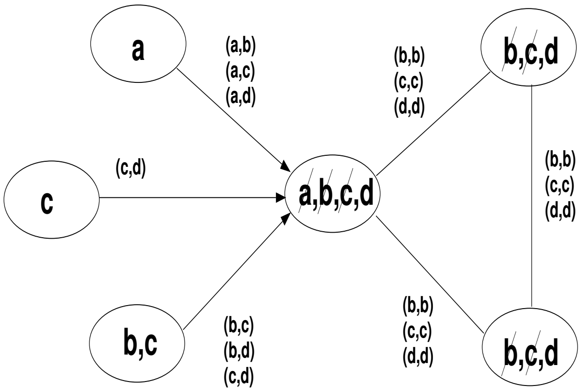

Figure 2: The CSP of fig.1 after application of an arc-consistency algorithm

domain D_i of possible **labels**. (In Fig.1, adapted from [Bessiere, 1991], $D = \{a, b, c, d\}$ and the domain of node 2 is {b,c,d})
The search space consists of the Cartesian product of the nodes' domains, $D = D_1 \times D_2 \times \ldots \times D_n$.
A set of **constraints** C is specified over some subsets of these nodes, where each constraint C_p involves a subset $\{i_1, \ldots, i_q\}$ of V. C_p is labeled by a relation R_p of R, a subset of the Cartesian product $D_{i_1} \times \ldots \times D_{i_q}$ that specifies those labels of the nodes that are compatible with each other.
A **binary** constraint involves only two nodes. This notion is extensible to an **n-ary** constraint over n nodes. The task in a CSP is to assign labels to nodes such that all the constraints on the nodes are satisfied simultaneously. Depending upon the application requirements, you might need to find any *one*, *all* or *optimal* solutions. Problem reduction is a class of techniques for transforming a CSP into problems which are hopefully easier to solve. Although problem reduction alone does not normally produce solutions, it can be extremely useful when used in conjunction with search methods. We define:

- **Node** consistency : A node is node consistent iff all labels in its domain satisfy all unary constraints on that node.

- **Arc** consistency : Arc (i, j) is arc consistent iff nodes i and j are node consistent and for any label in the domain of node i, there exists a label in the domain of node j such that all binary constraints on the two nodes are satisfied.

The unary constraints on a node can be equivalently represented (as a conjunction) by a single unary constraint on the same node. The same can be applied to group all binary constraints on two nodes into a single binary constraint between those same nodes. A graph is said to be node or arc consistent iff every node or arc (resp.) is consistent. The graph in Fig1 is assumed node consistent as no unary constraints have been specified. It is however, not arc-consistent.

4 Optimal Arc Consistency: AC4

Arc consistency is intuitively based on the notion of *support*. A label b for a node i is viable if there exists at-least one label at each of the other nodes j (connected to it by a constraint) that supports it. A label at a node is supported by a label at another node if they satisfy the constraint between the two nodes. In Fig1, label **a** at node 1 is supported by labels **b,c and d** at node 4 over the constraint between nodes 1 and 4. However, **a,b and c** at node 4 do not have any support over the constraint with node 3. Only **d** at node 4 is supported (by **c**) over that constraint. If there exists a node j where no label satisfies the required relation with b at node i, then b can be removed as a potential label for node i. In Fig.1, **a** at node 4 is such a label since there is a node (any of nodes 1,3,5) with no labels satisfying the constraint with the center node. Label **b** at the center node is also un-viable since there is a node (node 3) which has no labels satisfying the constraint between the two nodes. Fig.2 shows all the un-viable labels removed. AC4 makes this support explicit by assigning a counter to each arc-label pair. These pairs are denoted by $[(i,j),b]$ and indicate the arc from i to j with label b at node i. The counters are designated by $counter[(i,j),b]$ and indicate the number of j labels that support the label b for i in the constraint (i,j). In addition, for each label c at node j, the set S_{jc} is constructed, where $S_{jc} = \{(i,b) \mid c \text{ at node } j \text{ supports } b \text{ at node } i\ \}$. If c can no longer be a viable label for node j, then $counter[(i,j),b]$ must be decremented for each b supported by c. A table, M, to keep track of deleted labels and a list $List$ to keep track of constraint propagation are the other structures used. $List$ is initialized with all values (i,b) having at-least one counter equal to zero. During the propagation phase, the algorithm takes values (j,c) in $List$, decrements each counter $counter[(i,j)b] \forall (i,b) \in S_{jc}$ and when a $counter[(i,j),b]$ becomes equal to zero, it deletes (i,b) from M and puts it in $List$. The algorithm stops when $List$ is empty, which means that all values in M have non-empty supports on all the constraints. The upper bound time complexity for AC4 is $O(ea^2)$ where e is the number of arcs and a is the cardinality of the biggest label set.[Mohr and Henderson, 1986]

5 Generalized Arc Consistency

So far we have seen arc-consistency which handles binary constraints. We now extend the notion of arc-consistency for non-binary constraints. Instead of having binary constraints on pairs of nodes defining edges in a constraint graph, we have constraints over n nodes which define the *hyperedges* in a *hypergraph* of constraints. We choose to call such constraints *n-ary* constraints. Unlike AC4, where all constraints were binary, constraints can be of different arities.

5.1 Definitions

A constraint now becomes a relation $R_{i,j,\ldots k}$, specifying admissible labels for the nodes $i, j, \ldots k$ over this constraint (ref. Section 2).

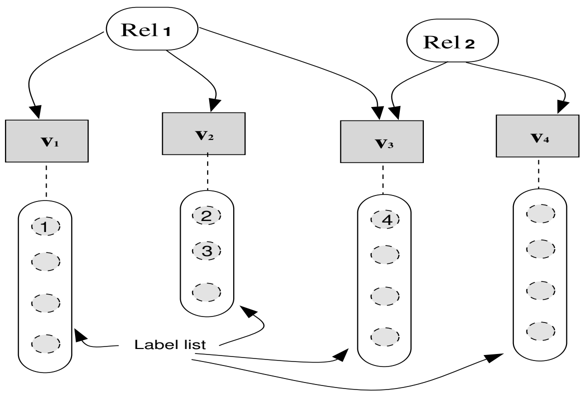

Figure 3: An example of a non-binary CSP

A labeling is *arc-consistent* iff :

$$\forall v \in V, \forall d_v \in D_v, \forall R_{i,j,\ldots k} \text{ constraining } v, \forall i, j, \ldots k,$$
$$\exists d_i, d_j, \ldots d_k \text{ such that } R_{i,j,\ldots k}(d_i, d_j, \ldots d_k) \text{ is true}$$

Which means, that for every label in the domain of every node i, there exist labels in the domains of nodes $j, \ldots k$ such that every *n-ary* constraint on these nodes is satisfied.

5.2 Previous work: GAC4

GAC4 [Mohr and Masini, 1988] works on the same principle as AC4: recursive label pruning. A relation $R_{i,j,\ldots k}$ is identified with the subset of n-tuples of labels from $i, j, \ldots k$ admissible with respect to $R_{i,j,\ldots k}$. Therefore, a constraint C is a set of n-tuples $((i, a), (j, b), \ldots (k, c))$.
When a label a has to be removed from the set D_i of possible labels for i, all the n-tuples including the label have to be discarded. If the discarded n-tuple happened to be the last in the constraint C supporting a particular label, then that label has to go and its effect propagated. GAC4 [Mohr and Masini, 1988] is shown to be an optimal algorithm and elegant data structures are presented for achieving optimality.

6 A New HyperArc Consistency Algorithm: NAC4

We have seen that constraints in GAC4 are specified as sets of tuples of admissible labels, implying that the relation is extensional (explicit). This may not be feasible in certain problem domains (for example,high level computer vision) where the constraint relations could be intensional (implicit) and computing the extension could be very expensive and/or unnecessary. Intuitively, the algorithm should take advantage of prior label pruning to minimize the computation of the relation's extension.
It has been shown ([Tsang, 1993] pp.14) that any n-ary constraint can be represented as a set of binary constraints *provided* the constraints are represented as a list of acceptable tuples of labels (as in GAC4). A constraint expressed implicitly like: node v_1 is

"brighter-than" node v_2 or node v_1 is "brighter-than" node v_3 cannot be broken into a set of binary constraints unless we compute the brightness values of all the labels in the domains of nodes v_1 to v_3.

NAC4 is designed for constraints which are defined by implicit relations. The notion of support, as in AC4, is extended here. A label at a node is now supported by a set of labels at each of the other nodes involved in the constraint, and must have a minimum support from these label sets over all the constraints involving it.
We now present the algorithm NAC4 using the same notations as in AC4 :

Algorithm NAC4

Initialization

```
M := 0; S_ib := ∅;
for r ≡ (i, j, ...k) ∈ E do
    for b ∈ A_i do
        begin
            Total := 0;
            for (c, ...d) ∈ (A_j × ... × A_k) do
                if R(i, b, j, c, ...k, d) then
                    begin
                        Total := Total + 1;
                        Tuple := {(j, c), ...(k, d)};
                        for (j', c') ∈ Tuple do
                            Append(S_j'c', ((i, b), r, Tuple));
                    end;
            if Total = 0 then M[i, b] := 1; A_i := A_i - {b};
            else Counter[r, b] := Total;
        end;
Initialize List with { (i, b) | M[i, b] = 1 };
```

Propagation phase

```
while List not Empty do
    begin
        choose and remove (j, c) from List;
        for (i, b, r, T) ∈ S_jc do
            begin
                Counter[r, b] := Counter[r, b] - 1;
                for (k, d) ∈ T do S_k,d := S_k,d − (i, b, r, T);
                if Counter[r, b] = 0 and M[i, b] = 0 then
                    begin
                        Append(List,(i, b));
                        M[i, b] := 1; A_i := A_i - {b};
                    end;
            end;
    end
```

In the initialization stage, we make a list of inconsistent labels stored in M, by calculating support for labels for each node over each n-ary relation r. So, in line 6, we need all possible tuples for the rest of the nodes involved in relation r, as opposed to all labels for the second node in a binary constraint. $S_{jc} = (i, b)$ was used in AC4 to store the fact that c at j supports b at i. We now augment the information stored in S_{jc} to include the rest of nodes involved in the relation. This is achieved by storing the relation index and a pointer to the whole tuple. We therefore append the support list of each node in r whenever it supports label b at i . If a label c were to become incompatible for such a node j, then we would have a link to re-assess the support for b at i. Counters now imply support for the label at the $first$ node of the relation.

The propagation phase must also be suitably altered (line 24) to ensure that "partner" nodes in the same relation r update their support lists. Since (j, c) no longer supports (i, b) over relation r, every other (k, d) involved should also remove the corresponding entry from their support list. However since the whole tuple had incremented the counter for b at i over r only once, multiple decrements are not necessary.

For example, in Fig.3 let us suppose that label 1 at node v_1 is supported by labels 2 and 3 at v_2 and label 4 at v_3 over constraint Rel_1.(In the explicit relation notation, Rel_1 would be $\{(1,2,4),(1,3,4)\}$) Now suppose label 4 cannot be assigned to v_3 due to some other constraint (possibly Rel_2) and we delete it from the label list of v_3, then we have to update the support of label 1 at v_1. We decrease the counter and delete the corresponding tuple from the support lists of every other node-label pair involved in this tuple. Since there is no other label at v_3 supporting the assignment of label 1 to v_1 over Rel_1, label 1 cannot be a valid label for v_1 and must be deleted. The node-label pairs which it supports will now be examined.

7 Analysis

A formal correctness proof is not presented here but can be worked out along the lines of the proof for AC4.

7.1 Correctness of NAC4

We present the steps for a complete proof of the correctness of NAC4.

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

2. The result of NAC4 is *hyperarc*-consistent: for all hyperarcs r, for all labels b for i, we have $Counter[r, b] > 0$ so b has corresponding label nodes $j, \dots k$; therefore, NAC4 builds a hyperarc-consistent solution.

3. From the above two steps, we conclude that NAC4 builds the largest hyperarc-consistent solution.

7.2 Complexity

Since the algorithm has been designed for expensive implicit constraints, the domain independent assumption that the constraint verification is of unit cost is invalid. Hence the running time is decided by the application dependent cost of constraint verification. If we were to assume (wrongly) that constraint verification in line 7 of the algorithm is a unit cost operation, the worst case time complexity of the algorithm would be $O(\sum_{n=1}^{m} C_n^m \, na^n)$, [1] where m is the number of nodes and a is the cardinality of the largest domain. However this complexity is not of much practical importance here since this algorithm is designed for constraints whose cost of evaluation is orders of magnitude greater than a unit step of the algorithm. A precise complexity analysis would estimate the cost of each verifying each constraint relation and weigh the cost summation correspondingly.

8 Expensive Constraints

An expensive constraint is one for which checking whether a tuple belongs to the corresponding relation is expensive. This also implies that computing the full extension of that relation is proportionally expensive to the sizes of the domains of the participating nodes. A preprocessing step to order the constraints according to their costs (calculated using the methods mentioned in the previous section) can be added to any consistency algorithm which represents the constraints implicitly.

For example, the initialization phase of AC4 can be modified to select the next least expensive edge. The same modification is needed for NAC4 in the initialization phase. Such a modification does not seem possible in the case of GAC4 since the initialization builds the elegant data structures based on the enumerated constraint input.

The standard way to achieve arc-consistency normally requires that we first achieve node consistency. However, unary constraints can be treated as n-ary constraints (with n = 1) and hence ordered according to cost. This implies that all unary constraints might not have been enforced before the binary/n-ary constraints. The advantage of this might be apparent in scenarios where (say) binary constraints are much cheaper to evaluate and achieve substantial problem reduction for a unique solution, or no solution.

9 Application

The algorithm has been prototyped in Allegro Common Lisp and works well for an application in high-level computer vision. The nodes are people names, the labels are rectangular sub-images denoting human faces. The constraints are extracted from a caption accompanying the image.[Srihari *et al.*, 1994]. Most of the constraints are binary spatial relations but some are also n-ary spatial relations. Another set of constraints include unary constraints which are orders of magnitude more expensive to compute

[1] For each of the $n \times C_n^m$ n-ary constraint, at most a^n counters are incremented. Since the counters never become negative, at most $(\sum_{n=1}^{m} C_n^m \, na^n)$ decrements are performed.

than the spatial constraints. Other constraints are image processing based and both binary and n-ary.
No extensive experimentation on randomly generated data has been conducted using this method for n-ary constraints. The focus has been on providing a version of arc-consistency capable of handling n-ary constraints specified through implicit relations.

10 Conclusion

We have provided an algorithm which handles certain types of n-ary constraints and which doesn't make the assumptions of GAC4 regarding representation of the constraints. We believe the assumption of unit time complexity assigned to checking a constraint is invalid and in some cases the constraint check time easily outweighs the time spent in the rest of the algorithm.

11 Acknowledgments

This work was supported in part by ARPA Contract 93-F148900-000. We would like to thank Paul Thadikaran and Hans Chalupsky for comments and help in writing this paper.

References

[Bessiere, 1991] C. Bessiere. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings of AAAI-91*, pages 221–226, 1991.

[Faugeras and Berthod, 1981] O. Faugeras and M. Berthod. Improving consistency and reducing ambiguity in stochastic labelling: an optimization approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:412–423, 1981.

[Garey *et al.*, 1974] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proc. 6th Annual ACM Symp. Theory Comput.*, pages 47–63, 1974.

[Haralick and Elliot, 1980] R.M. Haralick and G.L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[Hentenryck *et al.*, 1992] P. Van Hentenryck, Y. Deville, and C. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[Mackworth and Freuder, 1985] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for the constraint satisfaction problem. *Artificial Intelligence*, 25:65–74, 1985.

[Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mohr and Henderson, 1986] R. Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Mohr and Masini, 1988] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings of ECAI-88*, pages 651–656, 1988.

[Srihari *et al.*, 1994] Rohini Srihari, Rajiv Chopra, Debra Burhans, Venugopal Govindaraju, and Mahesh Venkatraman. Use of collateral text in image interpretation. In *Proc. of the ARPA Image Understanding Workshop*, pages 897–908, November 1994.

[Tsang, 1993] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[Waltz, 1975] D. Waltz. Understanding line drawings of scenes with shadows. In P.H. Winston, editor, *The Psychology of Computer Vision.* McGraw-Hill, New York, 1975.