WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
فارسی
한국어
Українська
Edit links

AC-3 algorithm

From Wikipedia, the free encyclopedia

The AC-3 algorithm (short for Arc Consistency Algorithm #3) is one of a series of algorithms used for the solution of constraint satisfaction problems (or CSP's). It was developed by Alan Mackworth in 1977. The earlier AC algorithms are often considered too inefficient, and many of the later ones are difficult to implement, and so AC-3 is the one most often taught and used in very simple constraint solvers.

The algorithm [edit]

AC-3 operates on constraints, variables, and the variables' domains (scopes). A **variable** can take any of several discrete values; the set of values for a particular variable is known as its **domain**. A **constraint** is a relation that limits or constrains the values a variable may have. The constraint may involve the values of other variables.

The current status of the CSP during the algorithm can be viewed as a directed graph, where the nodes are the variables of the problem, with edges or arcs between variables that are related by symmetric constraints, where each arc in the worklist represents a constraint that needs to be checked for consistency. AC-3 proceeds by examining the arcs between pairs of variables (*x*, *y*). It removes those values from the domain of *x* which aren't consistent with the constraints between *x* and *y*. The algorithm keeps a collection of arcs that are yet to be checked; when the domain of a variable has any values removed, all the arcs of constraints pointing to that pruned variable (except the arc of the current constraint) are added to the collection. Since the domains of the variables are finite and either one arc or at least one value are removed at each step, this algorithm is guaranteed to terminate.

For illustration, here is an example of a very simple constraint problem: *X* (a variable) has the possible values {0, 1, 2, 3, 4, 5} -- the set of these values are the domain of *X*, or D(*X*). The variable *Y* likewise has the domain D(*Y*) = {0, 1, 2, 3, 4, 5}. Together with the constraints *C1* = "*X* must be even" and *C2* = "*X* + *Y* must equal 4" we have a CSP which AC-3 can solve. Notice that the actual constraint graph representing this problem must contain two edges between *X* and *Y* since *C2* is undirected but the graph representation being used by AC-3 is directed.

It does so by first removing the non-even values out of the domain of *X* as required by *C1*, leaving D(*X*) = { 0, 2, 4 }. It then examines the arcs between *X* and *Y* implied by *C2*. Only the pairs (*X*=0, *Y*=4), (*X*=2, *Y*=2), and (*X*=4, *Y*=0) match the constraint *C2*. AC-3 then terminates, with D(*X*) = {0, 2, 4} and D(*Y*) = {0, 2, 4}.

AC-3 is expressed in pseudocode as follows:

```
 Input:
   A set of variables X
   A set of domains D(x) for each variable x in X. D(x) contains vx0, vx1... vxn,
the possible values of x
   A set of unary constraints R1(x) on variable x that must be satisfied
   A set of binary constraints R2(x, y) on variables x and y that must be satisfied

 Output:
   Arc consistent domains for each variable.

 function ac3 (X, D, R1, R2)
 // Initial domains are made consistent with unary constraints.
     for each x in X
         D(x) := { x in D(x) | R1(x) }
     // 'worklist' contains all arcs we wish to prove consistent or not.
     worklist := { (x, y) | there exists a relation R2(x, y) or a relation R2(y, x)
}

     do
         select any arc (x, y) from worklist
         worklist := worklist - (x, y)
         if arc-reduce (x, y)
             if D(x) is empty
```

```
                return failure
            else
                worklist := worklist + { (z, x) | z != y and there exists a
relation R2(x, z) or a relation R2(z, x) }
    while worklist not empty

function arc-reduce (x, y)
    bool change = false
    for each vx in D(x)
        find a value vy in D(y) such that vx and vy satisfy the constraint R2(x, y)
        if there is no such vy {
            D(x) := D(x) - vx
            change := true
        }
    return change
```

The algorithm has a worst-case time complexity of $\mathbf{O}(ed^3)$ and space complexity of $\mathbf{O}(e)$, where e is the number of arcs and d is the size of the largest domain.

References [edit]

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

Categories: Constraint programming