6.2 CONSTRAINT PROPAGATION: INFERENCE IN CSPs

In regular state-space search, an algorithm can do only one thing: search. In CSPs there is another choice: an algorithm can search (choose a new variable assignment from several possibilities) or do a specific type of inference called constraint propagation: using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on. Constraint propagation may be intertwined with search, or it may be done as a preprocessing step, before search starts. Sometimes this preprocessing can solve the whole problem, so no search is required at all.

The key idea is local consistency. If we treat each variable as a node in a graph (see Figure 6.1(b)) and each binary constraint as an arc, then the process of enforcing local consistency in each pan of the graph causes inconsistent values to be eliminated throughout the graph. There are different types of local consistency, which we now cover in turn.

6.2.1 Node consistency

A single variable (corresponding to a node in the CSP network) is node-consistent if all the values in the variable’s domain satisfy the variable’s unary constraints. For example, in the variant of the Australia map-coloring problem (Figure 6.1) where South Australians dislike green, the variable South Australians starts with domain \{red, green, blue\}, and we can make it node consistent by eliminating green, leaving \textsc{SA} with the reduced domain \{red, blue\}. We say that a network is node-consistent if every variable in the network is node-consistent.

It is always possible to eliminate all the unary constraints in a CSP by running node consistency. It is also possible to transform all \(n\)-ary constraints into binary ones (see Exercise 6.6). Because of this, it is common to define CSP solvers that work with only binary constraints; we make that assumption for the rest of this chapter, except where noted.

6.2.2 Arc consistency

A variable in a CSP is arc-consistent if every value in its domain satisfies the variable’s binary constraints. More formally, \(X_i\) is arc-consistent with respect to another variable \(X_j\) if for every value in the current domain \(D_i\), there is some value in the domain \(D_j\) that satisfies the binary constraint on the arc \((X_i, X_j)\). A network is arc-consistent if every variable is arc-consistent with every other variable. For example, consider the constraint \(Y = X^2\) where the domain of both \(X\) and \(Y\) is the set of digits. We can write this constraint explicitly as

\[
((X, Y), \{(0, 0), (1,1), (2, 4), (3, 9)\})
\]

To make \(X\) arc-consistent with respect to \(Y\), we reduce \(X\)’s domain to \{0, 1, 2, 3\}. If we also make \(Y\) arc-consistent with respect to \(X\), then \(Y\)’s domain becomes \{0, 1, 4, 9\} and the whole CSP is arc-consistent.

On the other hand, arc consistency can do nothing for the Australia map-coloring problem. Consider the following inequality constraint on \((\textsc{SA}, \textsc{WA})\):

\[
\{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), (\text{blue, red}), (\text{blue, green})\}
\]
Section 6.2. Constraint Propagation: Inference in CSPs

**Function AC-3**

```plaintext
function AC-3(rap) returns false if an inconsistency is found and true otherwise
inputs: csp, a binary CSP with components (X, D, C)
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
  \{Xi, Xj\} ← REMOVE-FIRST(queue)
  if REVISE(csp, Xi) then
    if size of Di = 0 then return false
    for each Xk in Xi.NEIGHBORS - \{Xi, Xj\} do
      add (Xk, Xj) to queue
  return true
```

**Function REVISE**

```plaintext
function REVISE(csp, Xi, Xj) returns true iff we revise the domain of Xj.
revised 4— false
for each x in D, do
  if no value y in D allows (r, y) to satisfy the constraint between X and Xj then
    delete x from D,
    revised ← true
return revised
```

**Figure 6.3** The arc-consistency algorithm AC-3. After applying AC-3, either every arc is arc-consistent, or some variable has an empty domain, indicating that the CSP cannot be solved. The name 'AC-3' was used by the algorithm's inventor (Mackworth, 1977) because it's the third version developed in the paper.

No matter what value you choose for SA (or for WA), there is a valid value for the other variable. So applying arc consistency has no effect on the domains of either variable.

The most popular algorithm for arc consistency is called AC-3 (see Figure 6.3). To make every variable arc-consistent, the AC-3 algorithm maintains a queue of arcs to consider. (Aerially, the order of consideration is not important, so the data structure is really a set, but tradition calls it a queue.) Initially, the queue contains all the arcs in the CSP. AC-3 then pops off an arbitrary arc \( \{X_i, X_j\} \) from the queue and makes \( X_i \) arc-consistent with respect to \( X_j \). If this leaves \( D_i \) unchanged, the algorithm just moves on to the next arc. But if this revises \( D_i \) (makes the domain smaller), then we add to the queue all arcs \( \{X_k, X_j\} \) where \( X_k \) is a neighbor of \( X_i \). We need to do that because the change in \( D_i \) might enable further reductions in the domains of \( D_k \), even if we have previously considered \( X_k \). If \( D_i \) is revised down to nothing, then we know the whole CSP has no consistent solution, and AC-3 can immediately return failure. Otherwise, we keep checking, trying to remove values from the domains of variables until no more arcs are in the queue. At that point, we are left with a CSP that is equivalent to the original CSP—they both have the same solutions—but the arc-consistent CSP will in most cases be faster to search because its variables have smaller domains.

The complexity of AC-3 can be analyzed as follows. Assume a CSP with \( n \) variables, each with domain size at most \( d \), and with \( c \) binary constraints (arcs). Each arc \( \{X_k, X_l\} \) can be inserted in the queue only \( d \) times because \( X_i \) has at most \( d \) values to delete. Checking