2. What inferences should be performed at each step in the search (INERENCE)?
3. When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

The subsections that follow answer each of these questions in turn.

### 6.3.1 Variable and value ordering

The backtracking algorithm contains the line

```
SELECT-UNASSIGNED-VARIABLE(esp).
```

The simplest strategy for SELECT-UNASSIGNED-VARIABLE is to choose the next unassigned variable in order, \(\{X_1, X_2, \ldots\}\). This static variable ordering seldom results in the most efficient search. For example, after the assignments for \(WA = \text{red}\) and \(NT = \text{green}\) in Figure 6.6, there is only one possible value for \(SA\), so it makes sense to assign \(SA = \text{blue}\) next rather than assigning \(Q\). In fact, after \(SA\) is assigned, the choices for \(Q, NSW,\) and \(V\) are all forced. This intuitive idea—choosing the variable with the fewest "legal" values—is called the **minimum-remaining-values (MRV)** heuristic. It also has been called the "most constrained variable" or "fail-first" heuristic, the latter because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree. If some variable \(X\) has no legal values left, the MRV heuristic will select \(X\) and failure will be detected immediately—avoiding pointless searches through other variables. The MRV heuristic usually performs better than a random or static ordering, sometimes by a factor of 1,000 or more, although the results vary widely depending on the problem.

The MRV heuristic doesn't help at all in choosing the first region to color in Australia, because initially every region has three legal colors. In this case, the **degree heuristic** comes in handy. It attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables. In Figure 6.1, \(SA\) is the variable with highest degree, 5; the other variables have degree 2 or 3, except for \(T\), which has degree 0. In fact, once \(SA\) is chosen, applying the degree heuristic solves the problem without any false steps—you can choose any consistent color at each choice point and still arrive at a solution with no backtracking. The minimum-remaining-
Section 6.3. Backtracking Search for CSPs

values heuristic is usually a more powerful guide, but the degree heuristic can be useful as a tie-breaker.

Once a variable has been selected, the algorithm must decide on the order in which to examine its values. For this, the least constraining-value heuristic can be effective in some cases. It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph. For example, suppose that in Figure 6.1 we have generated the partial assignment with WA = red and NT = green and that our next choice is for Q. Blue would be a bad choice because it eliminates the last legal value left for Q’s neighbor, SA. The least-constraining-value heuristic therefore prefers red to blue. In general, the heuristic is trying to leave the maximum flexibility for subsequent variable assignments. Of course, if we are trying to find all the solutions to a problem, not just the first one, then the ordering does not matter because we have to consider every value anyway. The same holds if there are no solutions to the problem.

Why should variable selection be fail-first, but value selection be fail-last? It turns out that, for a wide variety of problems, a variable ordering that chooses a variable with the minimum number of remaining values helps minimize the number of nodes in the search tree by pruning larger parts of the tree earlier. For value ordering, the trick is that we only need one solution; therefore it makes sense to look for the most likely values first. If we wanted to enumerate all solutions rather than just find one, then value ordering would be irrelevant.

6.3.2 Interleaving search and inference

So far we have seen how AC-3 and other algorithms can infer reductions in the domain of variables before we begin the search. But inference can be even more powerful in the course of a search: every time we make a choice of a value for a variable, we have a brand-new opportunity to infer new domain reductions on the neighboring variables.

One of the simplest forms of inference is called forward checking. Whenever a variable X is assigned, the forward-checking process establishes consistency for it: for each unassigned variable V that is connected to X by a constraint, delete from V’s domain any value that is inconsistent with the value chosen for X. Because forward checking only does consistency inferences, there is no reason to do forward checking if we have already done arc consistency as a preprocessing step.

Figure 6.7 shows the progress of backtracking search on the Australia CSP with forward checking. There are two important points to notice about this example. First, notice that after WA = red and Q = green are assigned, the domains of NT and SA are reduced to a single value; we have eliminated branching on these variables altogether by propagating information from WA and Q. A second point to notice is that after V = blue, the domain of SA is empty. Hence, forward checking has detected that the partial assignment {WA = red, Q = green, V = blue} is inconsistent with the constraints of the problem, and the algorithm will therefore backtrack immediately.

For many problems the search will be more effective if we combine the MRV heuristic with forward checking. Consider Figure 6.7 after assigning {WA = red}. Intuitively, it seems that that assignment constrains its neighbors, NT and SA, so we should handle those