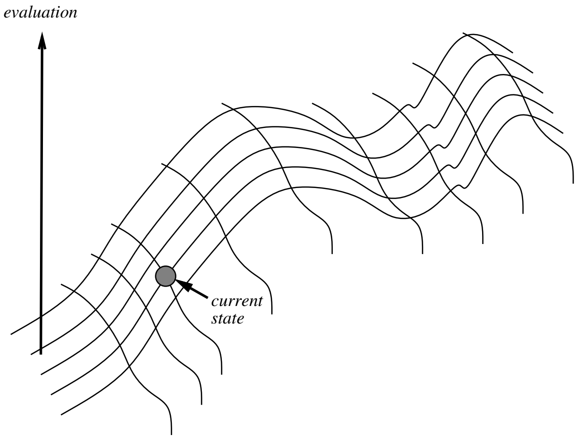

Figure 4.13 Iterative improvement algorithms try to find peaks on a surface of states where height is defined by the evaluation function.

function HILL-CLIMBING(*problem*) **returns** a solution state
  inputs: *problem*, a problem
  local variables: *current*, a node
                next, a node

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    next ← a highest-valued successor of *current*
    if VALUE[next] < VALUE[current] **then return** *current*
    current ← *next*
  end

Figure 4.14 The hill-climbing search algorithm.

◇ **Ridges**: a ridge may have steeply sloping sides, so that the search reaches the top of the ridge with ease, but the top may slope only very gently toward a peak. Unless there happen to be operators that move directly along the top of the ridge, the search may oscillate from side to side, making little progress.

RANDOM-RESTART HILL-CLIMBING

In each case, the algorithm reaches a point at which no progress is being made. If this happens, an obvious thing to do is start again from a different starting point. **Random-restart hill-climbing** does just this: it conducts a series of hill-climbing searches from randomly generated initial states, running each until it halts or makes no discernible progress. It saves the best result found so far from any of the searches. It can use a fixed number of iterations, or can continue until the best saved result has not been improved for a certain number of iterations.