Clearly, if enough iterations are allowed, random-restart hill-climbing will eventually find the optimal solution. The success of hill-climbing depends very much on the shape of the state-space "surface": if there are only a few local maxima, random-restart hill-climbing will find a good solution very quickly. A realistic problem has a surface that looks more like a porcupine. If the problem is NP-complete, then in all likelihood we cannot do better than exponential time. It follows that there must be an exponential number of local maxima to get stuck on. Usually, however, a reasonably good solution can be found after a small number of iterations.

Simulated annealing

Instead of starting again randomly when stuck on a local maximum, we could allow the search to take some downhill steps to escape the local maximum. This is roughly the idea of **simulated annealing** (Figure 4.15). The innermost loop of simulated annealing is quite similar to hill-climbing. Instead of picking the *best* move, however, it picks a *random* move. If the move actually improves the situation, it is always executed. Otherwise, the algorithm makes the move with some probability less than 1. The probability decreases exponentially with the "badness" of the move—the amount ΔE by which the evaluation is worsened.

A second parameter T is also used to determine the probability. At higher values of T, "bad" moves are more likely to be allowed. As T tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. The *schedule* input determines the value of T as a function of how many cycles already have been completed.

The reader by now may have guessed that the name "simulated annealing" and the parameter names ΔE and T were chosen for a good reason. The algorithm was developed from an explicit analogy with **annealing**—the process of gradually cooling a liquid until it freezes. The VALUE function corresponds to the total energy of the atoms in the material, and T corresponds to the

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
     T ← schedule[t]
     if T=0 then return current
     next ← a randomly selected successor of current
     ΔE ← VALUE[next] – VALUE[current]
     if ΔE > 0 then current ← next
     else current ← next only with probability e^(ΔE/T)
```

Figure 4.15 The simulated annealing search algorithm.