Beyond Classical Search

Chapter 4

LOCAL SEARCH ALGORITHMS AND OPTIMIZATION PROBLEMS

- The search algorithms that we have seen so far are designed to explore search spaces systematically.

- This systematicity is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each paint along the path.

- When a goal is found, the path to that goal also constitutes a solution to the problem. In many problems, however, the path to the goal is irrelevant

- Local search algorithms operate using a single current node

- local search algorithms are not systematic, they have two key advantages: (1)they use very little memory—usually a constant amount; and (2) they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

- In addition to finding goals, local search algorithms are useful for solving pure optimization problems, in which the aim is to find the best state according to an objective function

- To understand local search, we find it useful to consider the state-space landscape (as in Figure . A landscape has both "location" (defined by the state) and "elevation" (defined by the value of the heuristic cost function or objective function).

- If elevation corresponds to cost, then the aim is to find the lowest valley—a global minimum; if elevation corresponds to an objective function, then the aim is to find the highest peak—a global maximum

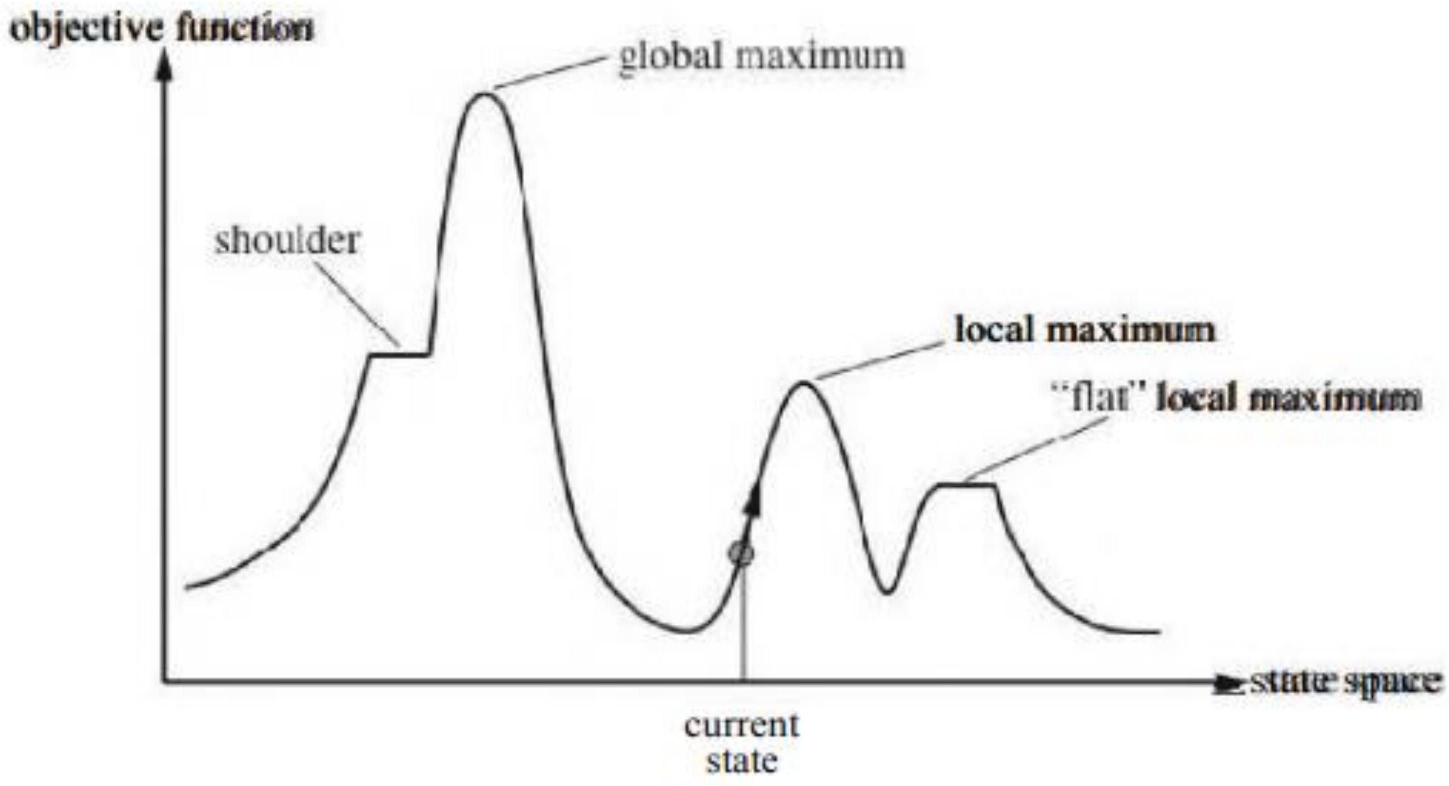


Figure · A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

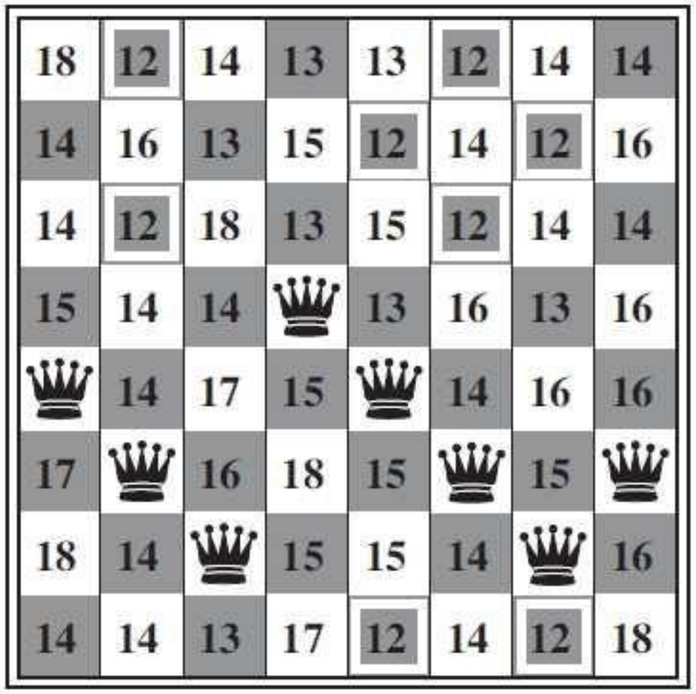
If elevation corresponds to cost, then the aim is to find the lowest valley—a global minimum; if elevation corresponds to an objective function, then the aim is to find the highest peak—a global maximum

Local search algorithms explore this landscape. A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a global minimum/maximum.

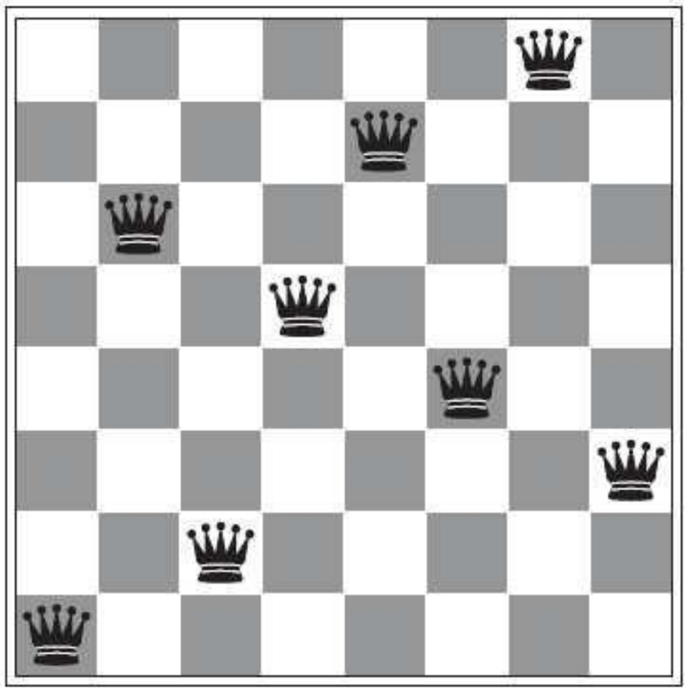
1. Hill-climbing search

Hill climbing algorithm is a **local search algorithm** which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

Hill-climbing contd.



a

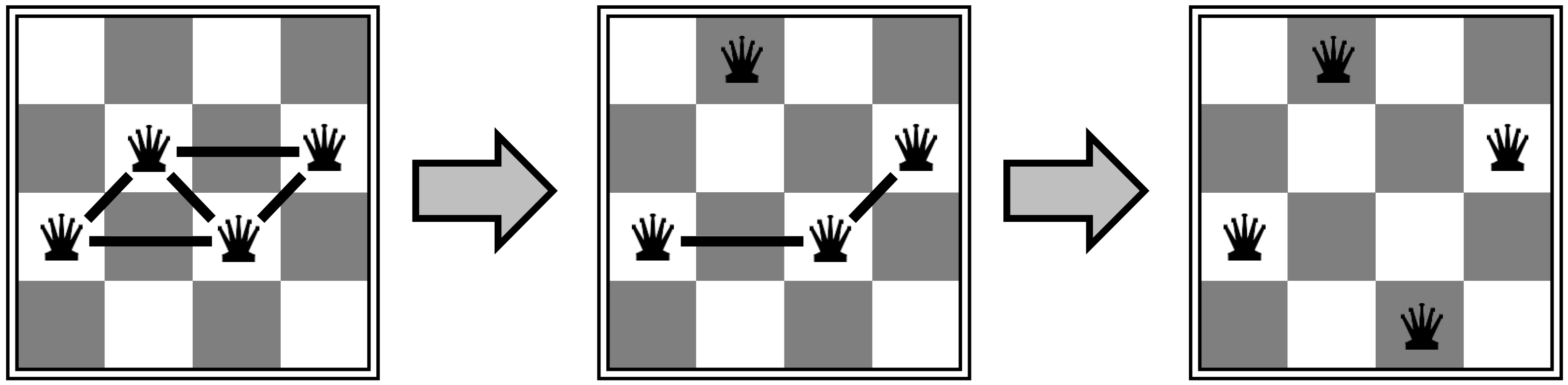


b

Example: *n*-queens

Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts



h = 5 **h = 2** **h = 0**

Almost always solves *n*-queens problems almost instantaneously for very large *n*, e.g., *n* = 1*million*

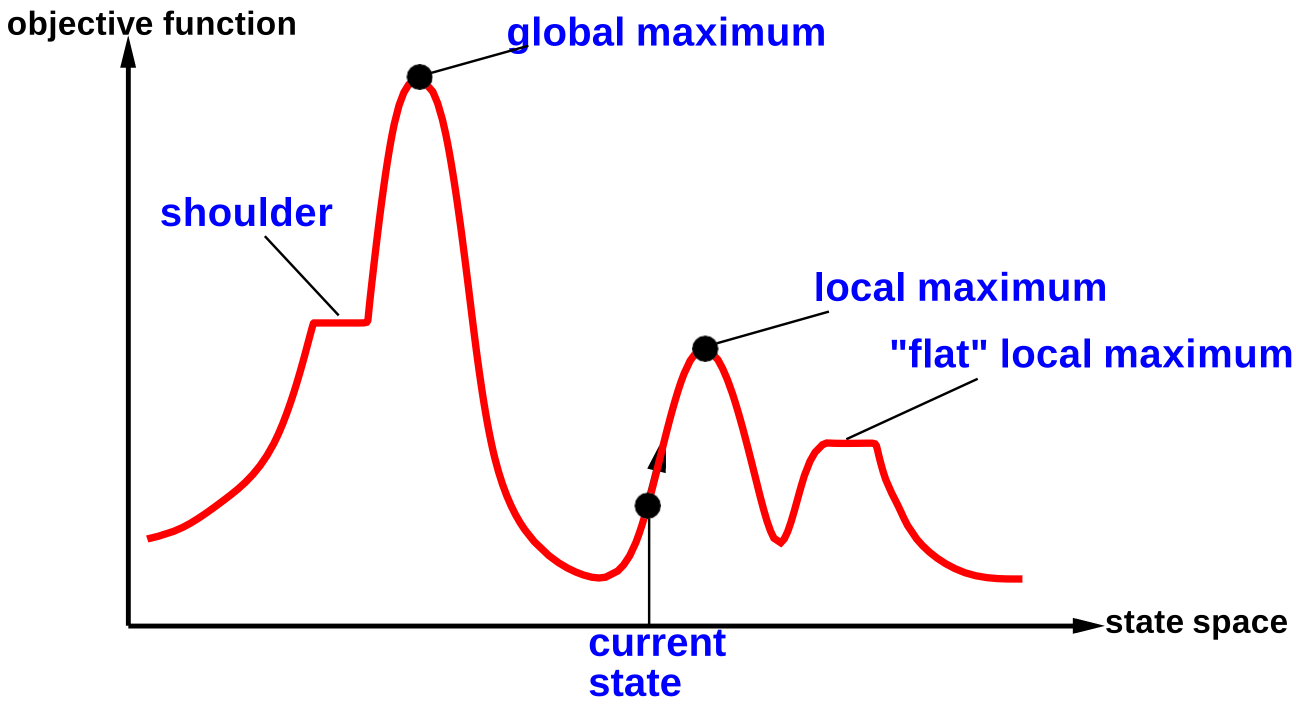
Hill-climbing (or gradient ascent/descent)

"Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node
    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

Hill-climbing contd. (greedy local search)

Useful to consider state space landscape



Unfortunately, hill climbing often gets stuck for the following reasons:

- **Local maxima**: Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go. move of a single queen makes the situation worse.

- **Ridges**: A ridge is shown in figure. Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate

- **Plateaux**: A plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau.

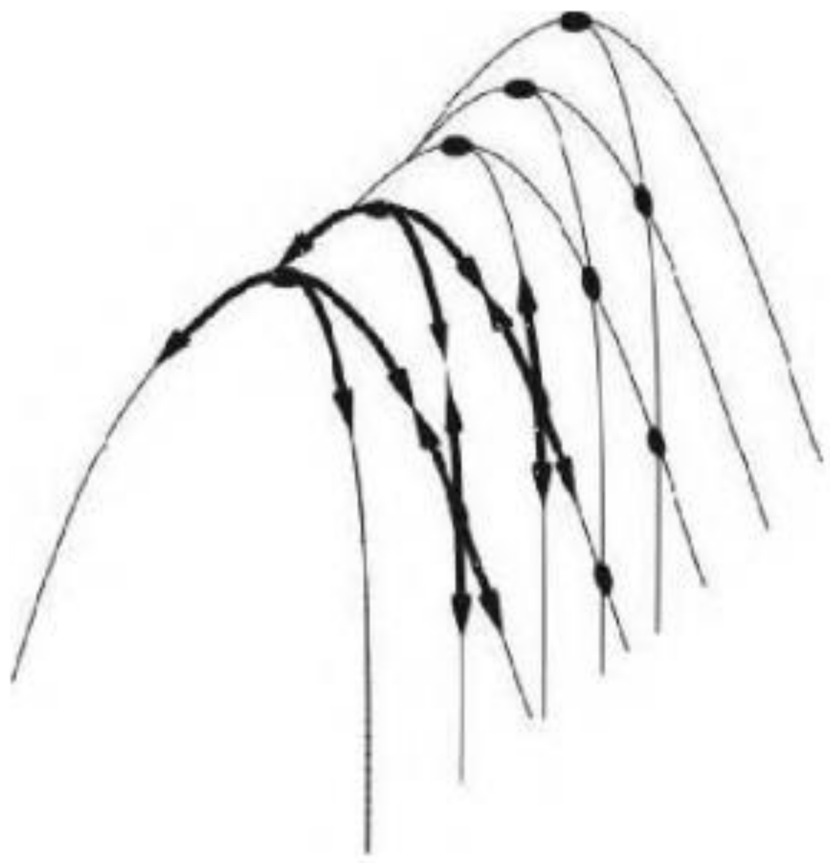


Figure Illustration of why ridges cause difficulties for hill climbing. The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima that are not directly connected to each other. From each local maximum, all the available actions point downhill.

2.Simulated annealing

- A hill-climbing algorithm that never makes "downhill" moves toward states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum.

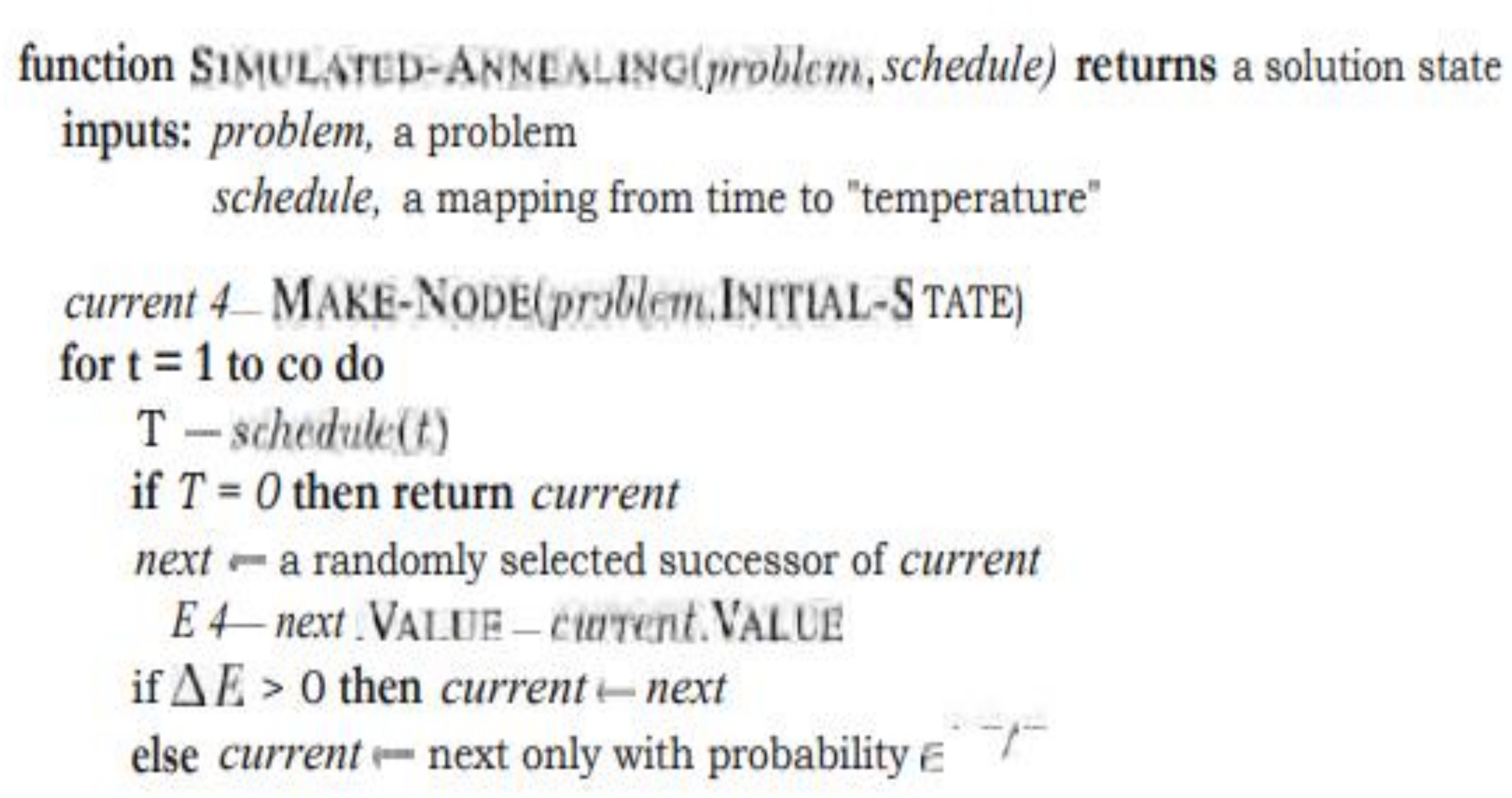
- In contrast, a purely random walk—that is, moving to a successor chosen uniformly at random from the set of successors—is complete but extremely inefficient.

- Therefore, it seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness.

- Simulated annealing is such an algorithm. In metallurgy, annealing is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low energy crystalline state.

Simulated annealing

Idea: escape local maxima by allowing some “bad” moves but gradually decrease their size and frequency



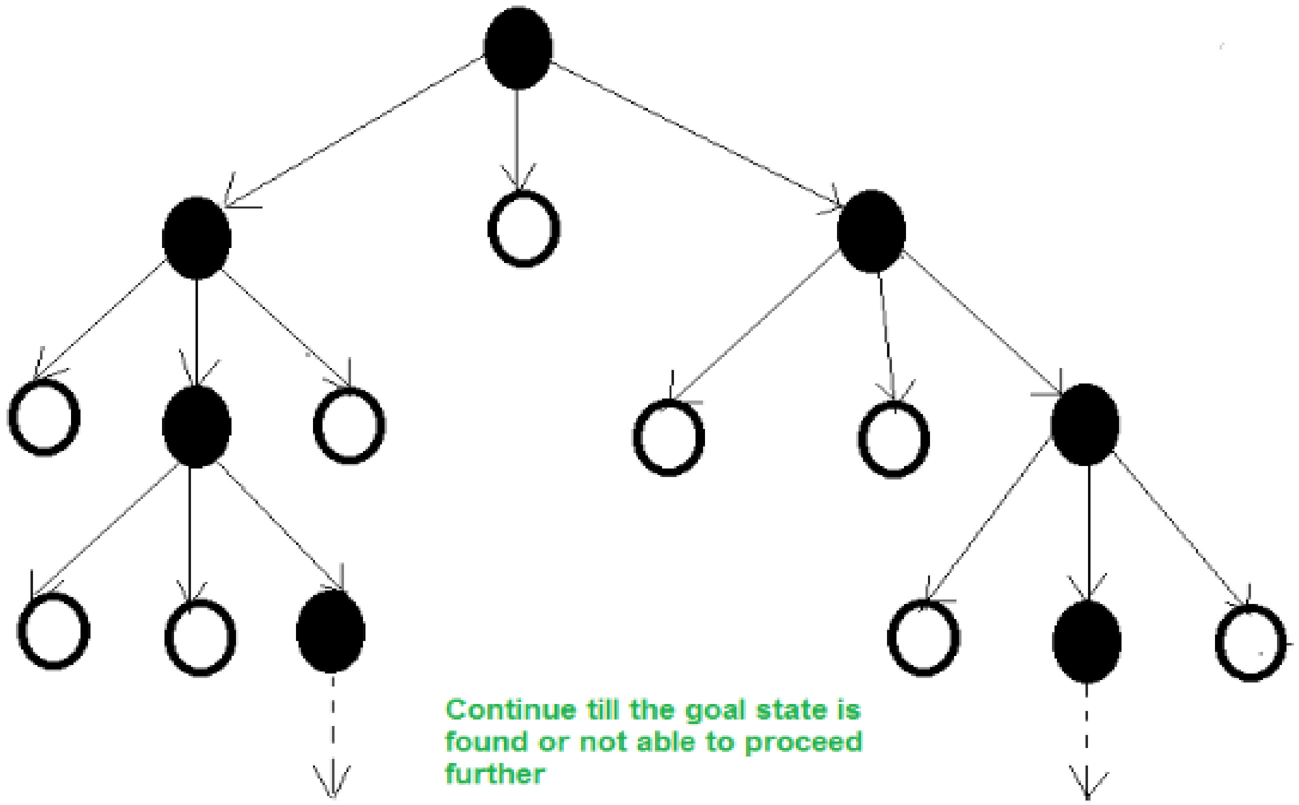
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current 4_ MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to co do
      T — schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
        E 4— next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability ε [illegible]
```

3.Local beam search

- The local beam search algorithm keeps track of k states rather than just one. It begins with k randomly generated states.

- At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts.

- Otherwise, it selects the k best successors from the complete list and repeats.

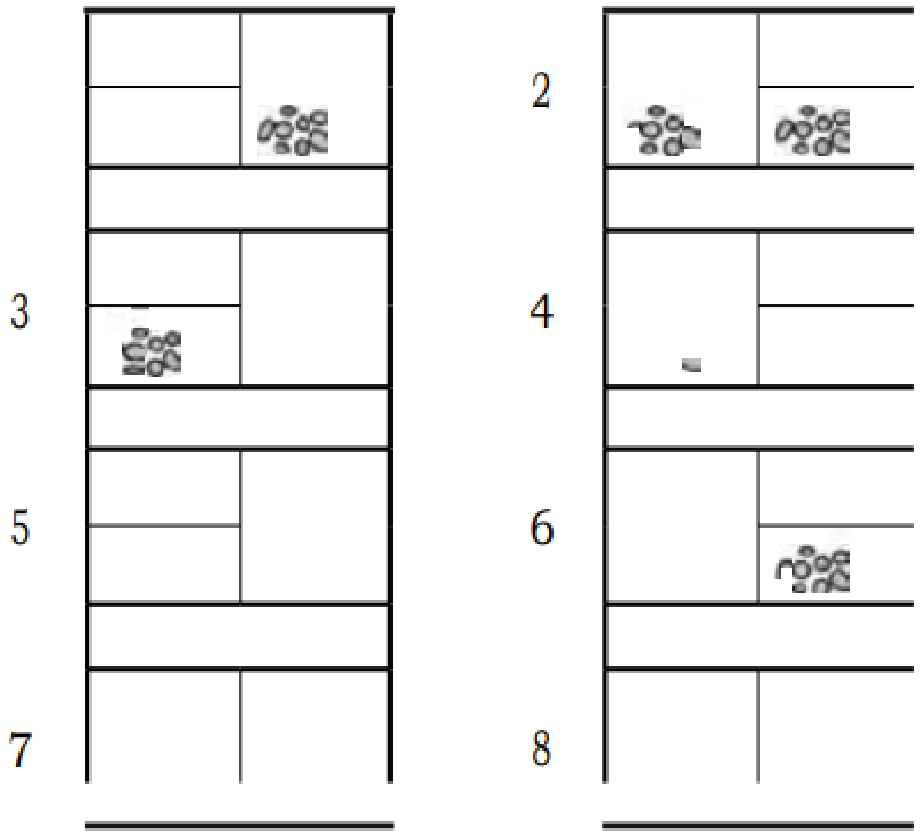

Continue till the goal state is found or not able to proceed further

SEARCHING WITH NONDETERMINISTIC ACTIONS

The nondeterministic algorithms are often used to find an approximation to a solution, when the exact solution would be too costly to obtain using a deterministic one

1. The erratic vacuum world

we use the vacuum world, There are three actions—Left, Right, and Suck—and the goal is to clean up all the dirt (states 7 and 8). If the environment is observable, deterministic, and completely known, then the problem is trivially solvable by any of the algorithms and the solution is an action sequence. For example, if the initial state is 1, then the action sequence [Suck, Right, Suck] will reach a goal state, 8



◆When applied to a dirty square the action cleans the square and sometimes cleans up dirt in an adjacent square. too.

◆ When applied to a clean square the action sometimes deposits dirt on the carpet

• To provide a precise formulation of this problem, we need to generalize the notion of a transition model .

• Instead of defining the transition model by a RESULT function that returns a single state, we use a RESULTS function that returns a set of possible outcome states.

• For example, in the erratic vacuum world, the Suck action in state 1 leads to a state in the set {5, 7}—the dirt in the right-hand square may or may not be vacuumed up.

• We also need to generalize the notion of a solution to the problem. For example, if we start in state 1, there is no single sequence of actions that solves the problem.

• Instead, we need a contingency plan such as the following:

[Stick, if State = 5 then [Right, Suck] else

Thus, solutions for non deterministic problems can contain nested if—then—else statements; this means that they are trees rather than sequences.

2. AND—OR search trees

- In a deterministic environment, the only branching is introduced by the agent's own choices in each state. We call these nodes OR nodes.

- In the vacuum world, for example, at an OR node the agent chooses Left or Right or Suck.

- In a nondeterministic environment, branching is also introduced by the envimninent's choice of outcome for each action. We call these nodes AND nodes.

- For example, the Suck action in state l leads to a state in the set {5, 7}, so the agent would need to find a plan for state 5 and for state 7.

- These two kinds of nodes alternate, leading to an AND—OR tree

AND-OR search trees

Deterministic env.- branching is introduced by the agent's own choices in each state (Left or Right or Suck) - OR-node

Standard search tree is all OR-nodes Agent chooses action;

At least one branch must be solved

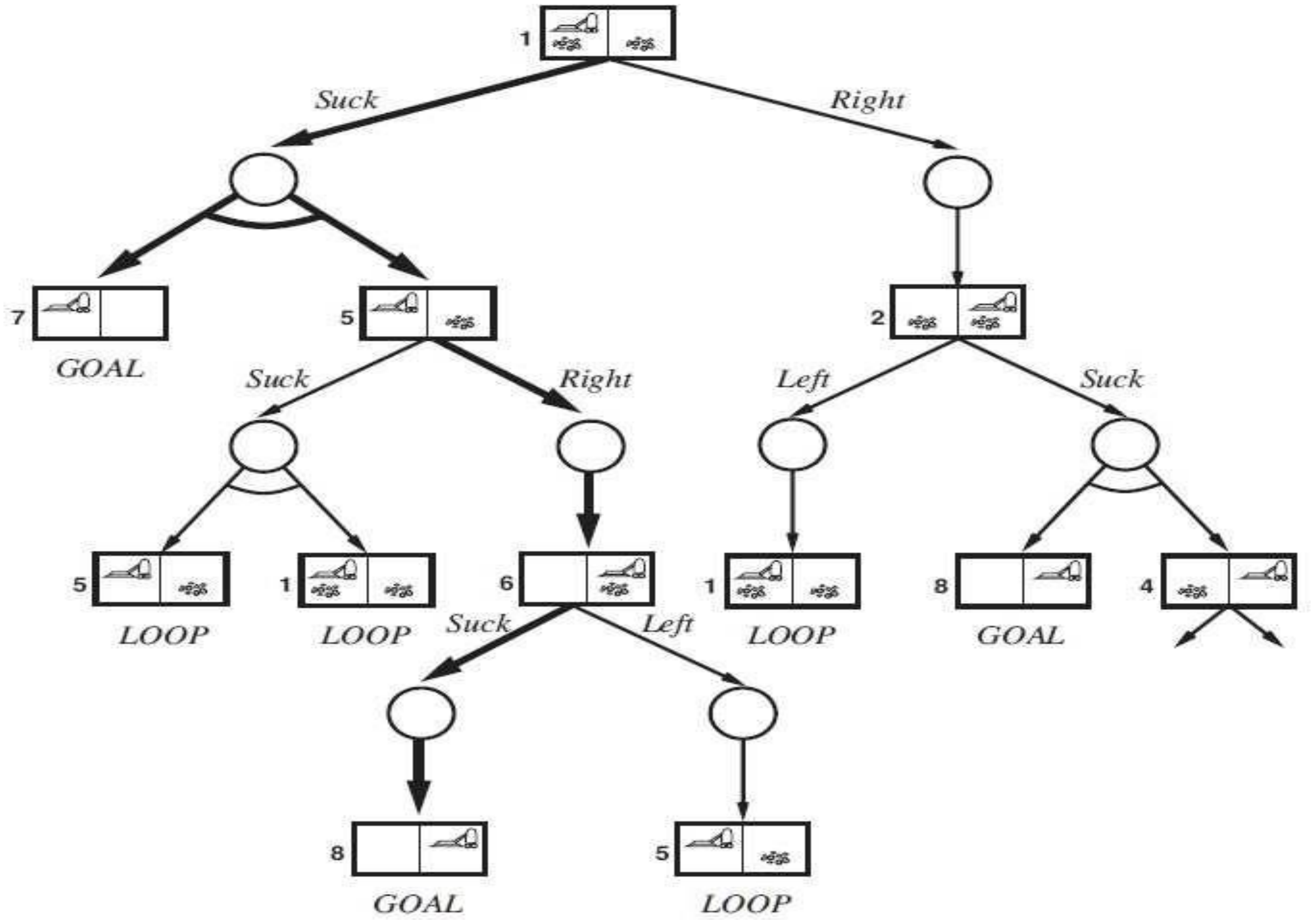
Nondeterministic env. - branching is introduced by the environment's choice of outcome of each action (Suck in state 1 leads to {5,7})- AND-node

All branches must be solved Results(s,a) returns a set of states

Results(1,Suck) = 5,7

Results(4,Suck) = 2,4

AND-OR search trees



A solution is a sub-tree that has a goal node at every leaf

ONLINE SEARCH AGENTS AND UNKNOWN ENVIRONMENTS

- The agents use offline search algorithms. They compute a complete solution before setting foot in the real world and then execute the solution.

- In ONLINE SEARCH contrast, an online search agent interleaves computation and action: first it takes an action, then it observes the environment and computes the next action.

- Online search is a good idea in dynamic or semi dynamic domains—domains where there is a penalty for sitting around and computing too long.

- Online search is a necessary idea for unknown environments, where the agent does not know what states exist or what its actions do.

- In this state of ignorance, the agent faces an EXPLORATEN PROBLEM exploration problem and must use its actions as experiments in order to learn enough to make deliberation worthwhile

1.Online search problems

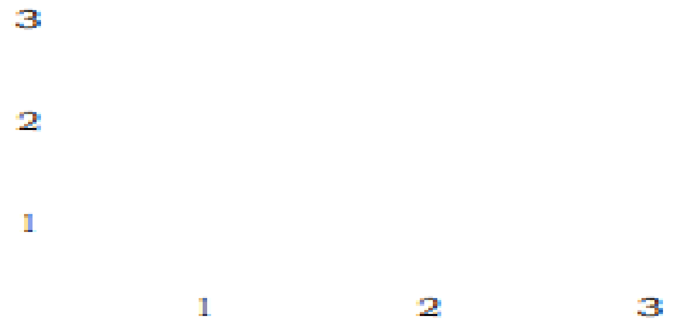
An online search problem must be solved by an agent executing actions, rather than by pure computation. We assume a deterministic and fully observable environment, but we stipulate that the agent knows only the following:

• ACTIONS(s),which returns a list of actions allowed in state s;

• The step-cost function c(s, a, s')—note that this cannot be used until the agent knows that s' is the outcome; and

• GOAL-TEST(s).

Note in particular that the agent cannot determine Result,a) except by actually being in s and doing a. For example, in the maze problem shown in Figure, the agent does not know that going Up from (1,1) leads to (1,2); nor, having done that, does it know that going Down will take it back to (1,1). This degree of ignorance can be reduced in some applications



for example, a robot explorer might know how its movement actions work and be ignorant only of the locations of obstacles.

.

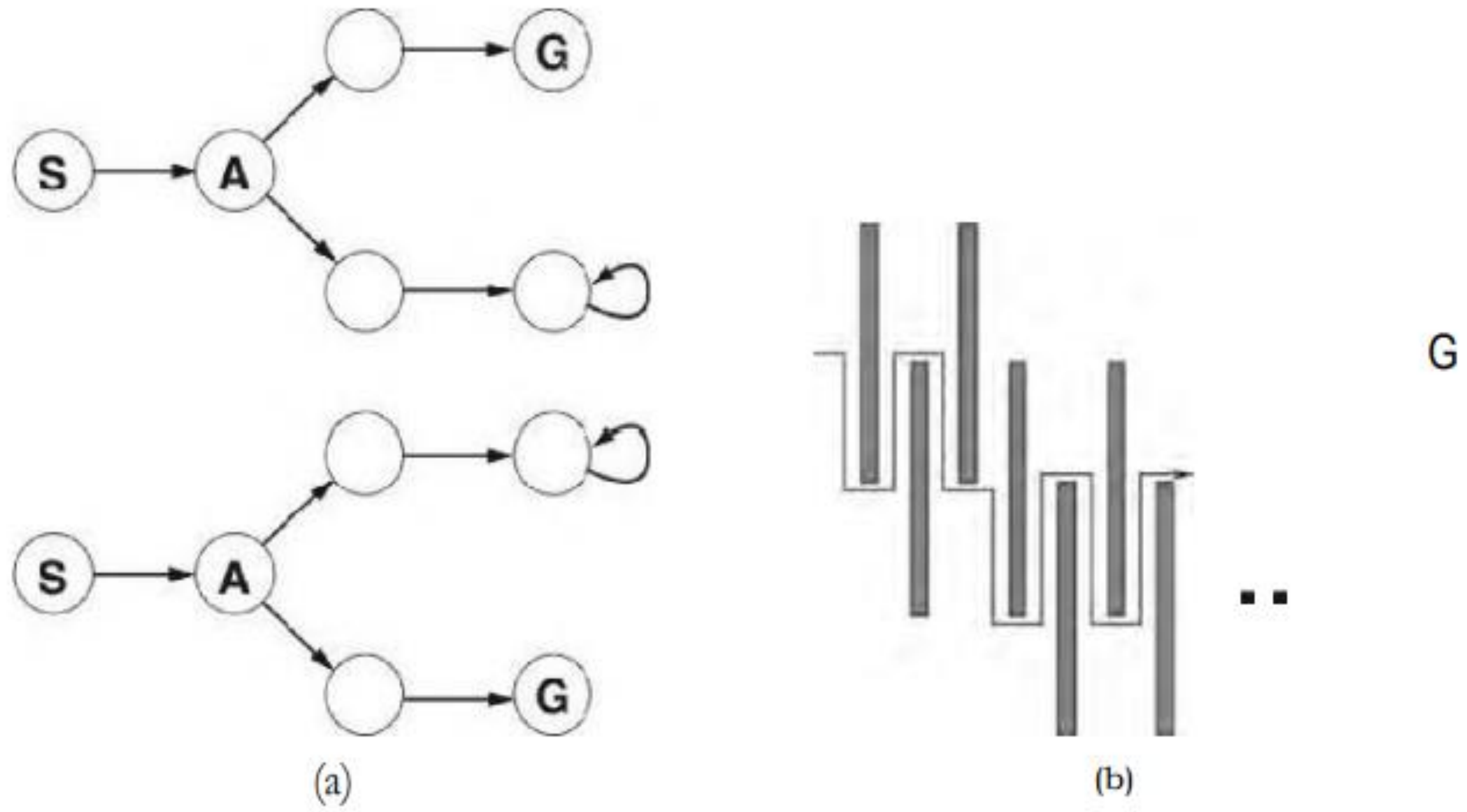


Figure (a) Two state spaces that might lead an online search agent into a dead end. Any given agent will fail in at least one of these spaces. (b) A two-dimensional environment that can cause an online search agent to follow an arbitrarily inefficient route to the goal.

- Typically, the agent's objective is to reach a goal state while minimizing cost. (Another possible objective is simply to explore the entire environment.)

- The cost is the total path cost of the path that the agent actually travels. It is common to compare this cost with the path cost of the path the agent would follow if it knew the search space in advance—that is, the actual shortest path (or shortest complete exploration). In the language of online algorithms, this is called the competitive ratio; we would like it to be as small as possible.

- Dead ends are a real difficulty for robot exploration—staircases.ramps, cliffs, unc-way streets, and all kinds of natural terrain present opportunities for irreversible actions

2.Online search agents

• After each action, an online agent receives a percept telling it what state it has reached; from this information, it can augment its map of the environment.

• The current map is used to decide where to go next. This interleaving of planning and action means that online search algorithms are quite different from the offline search algorithms we have seen previously.

• For example, offline algorithms such as A* can expand a node in one part of the space and then immediately expand a node in another part of the space, because node expansion involves simulated rather than real actions.

•

• An online algorithm, on the other hand, can discover successors only for a node that it physically occupies. To avoid traveling all the way across the tree to expand the next node, it seems better to expand nodes in a Inca! order.

.

• Depth-first search has exactly this property because (except when backtracking) the next node expanded is a child of the previous node expanded.

Sensorless vacuum world

▪

Specifying sensorless problems

$P : ACTIONS_P, RESULT_P, GOAL - TEST_P$, and $STEP - COST_P$

◆**Belief states**: every possible set of physical states. If P has N states, then the sensorless problem has up to 2^N

◆**Initial state**: Typically the set of all states in P, although in some cases the agent will have more knowledge than this.

◆**Actions**: Suppose the agent is in belief state $b = s_1, s_2$, but $ACTIONS_P(s_1)$ $ACTIONS_P(s_2)$. What are possible actions in current (union vs. state? intersection): $ACTIONS(b) = \bigcup_{s \in b} ACTIONS_P(s)$

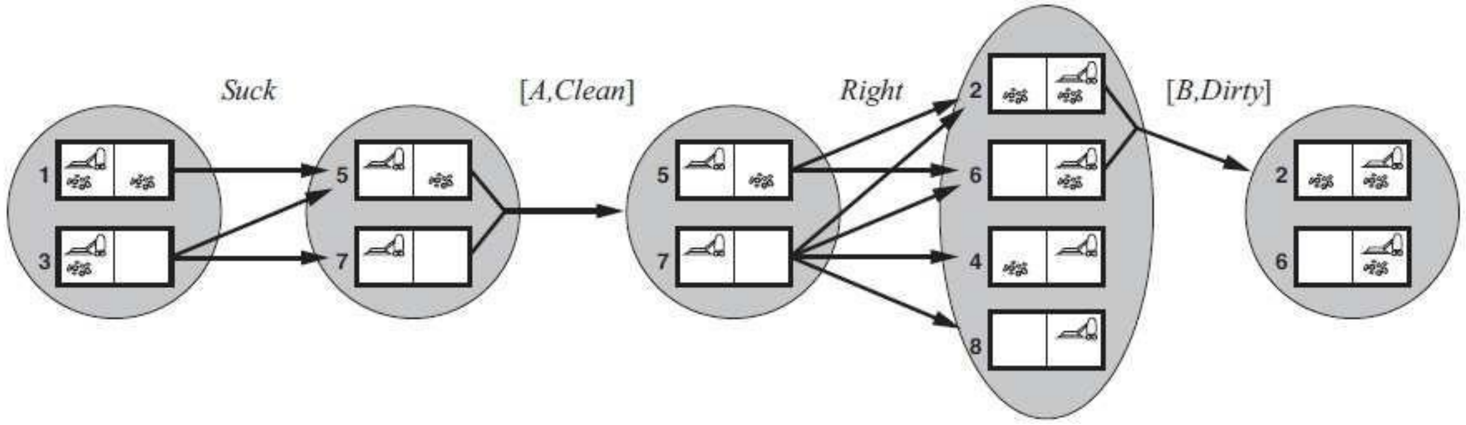
◆**Transition model**: deterministic vs. nonderministic

$$b' = RESULT(b, a) = \{s' : s' = RESULT_P(s, a) \text{ and } s \in b\}$$

$$b' = RESULT(b, a) = \{s' : s' = RESULTS_P(s, a) \text{ and } s \in b\}$$

Specifying sensorless problems

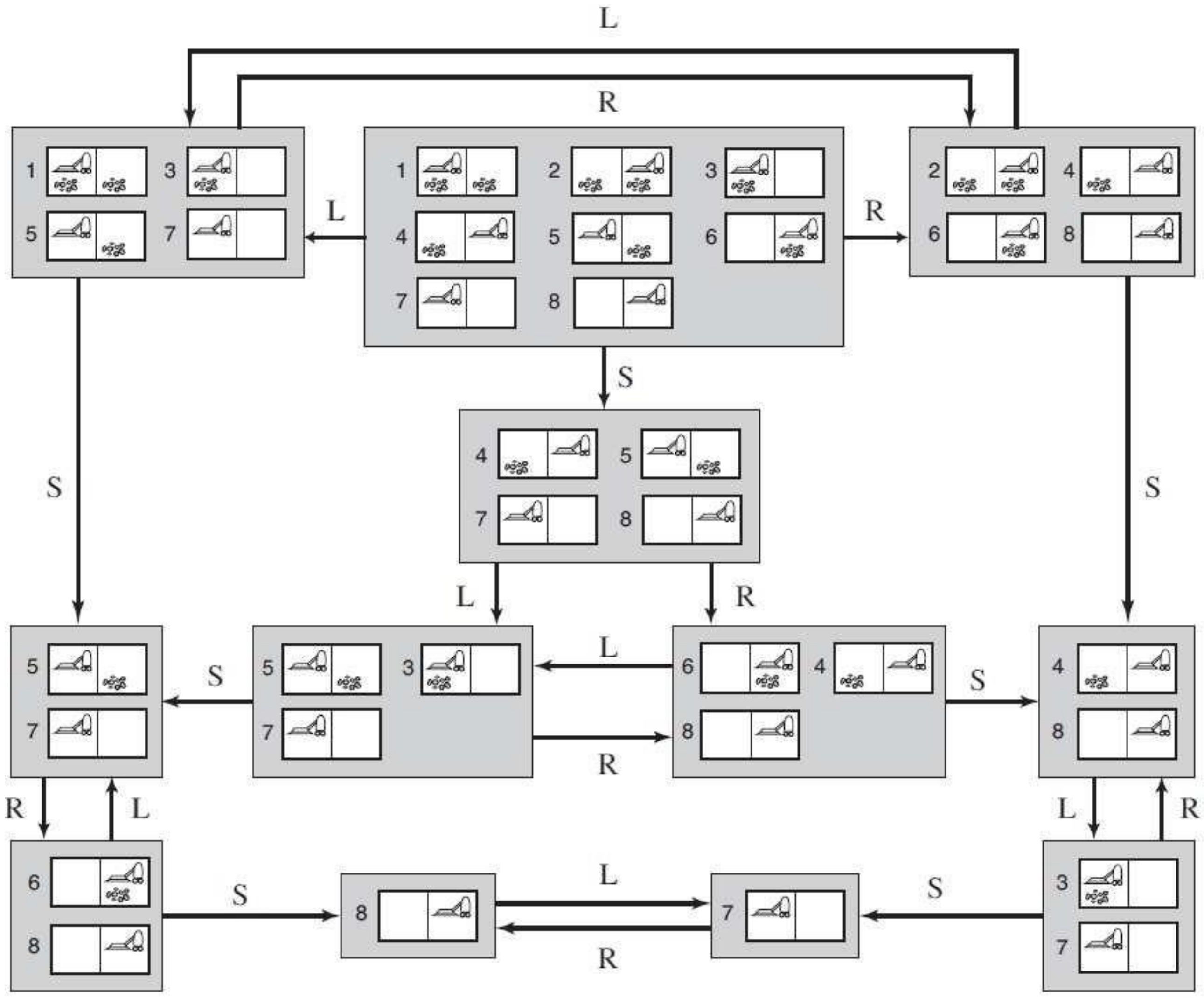
Predicting the next belief state



◆Goal test: a belief state satisfies the goal only if all the physical states in it satisfy $GOAL - TEST_P$. The agent may accidentally achieve the goal earlier, but it wont know that it has done so.

◆Path cost: If the same action can have different costs in different states, then the cost of taking an action in a given belief state could be one of several values.

Reachable belief state space for deterministic



Reachable belief states vs. possible belief states

Incremental belief state search

Builds up the solution one physical state at a time

E.g., initial belief state is {1, 2, 3, 4, 5, 6, 7, 8}, and we have to find an action sequence that works in all 8 states.

finding a solution that works for state 1 then we check if it works for

state 2

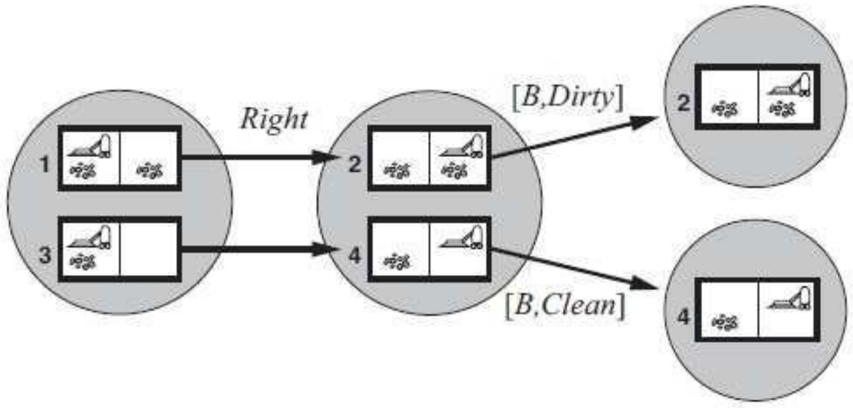
if not, go back and find a different solution for state 1,

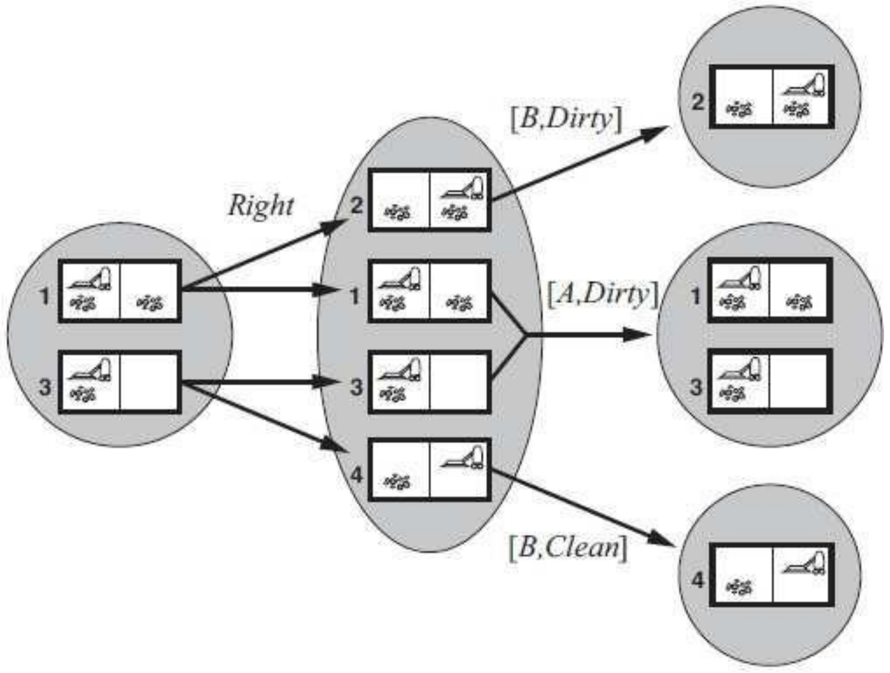
Searching with partial observations

Local-sensing vacuum world: has a position sensor and a local dirt sensor but has no sensor capable of detecting dirt in other squares.

The percept [A, dirt] produces in states ...

Transition model in local sensing vacuum





Transition model in local sensing vacuum

3 stages

1.Prediction stage is the same as for sensorless problems: given the action
a in belief state b, the predicted belief state is $\hat{b} = PREDICT(b, a)$

2.Observation prediction stage determines the set of percepts o that could be observed in the predicted belief state:

$$POSSIBLE-PERCEPTS(\hat{b}) = \{o : o = PERCEPT(s) \text{ and } s \in \hat{b}\}$$

3.Update stage determines, for each possible percept, the belief state that would result from the percept. The new belief state bo is just the set of states in $\hat{b}$ that could have produced the percept:

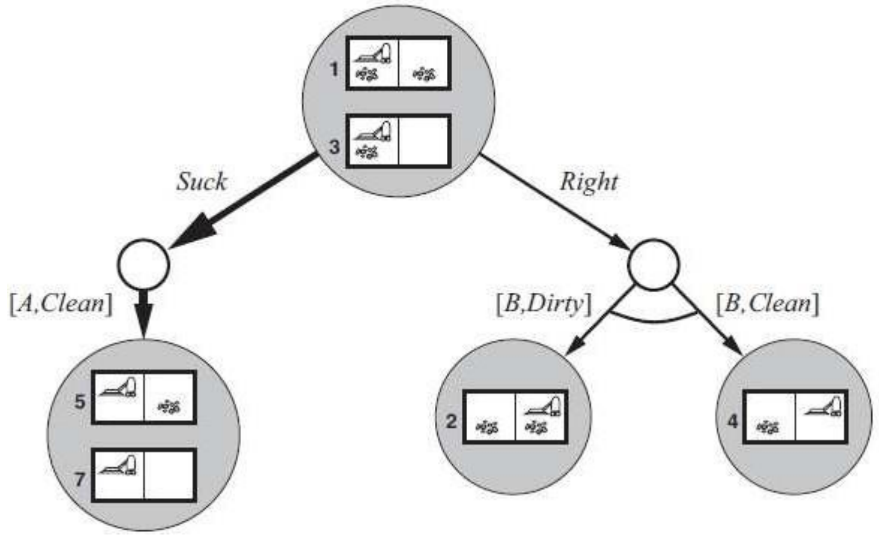
$$b_o = UPDATE(\hat{b}, o) = \{s : o = PERCEPT(s) \text{ and } s \in \hat{b}\}$$

Note:

◆the updated belief state b_o can be no larger than the predicted belief state $\hat{b}$

◆the belief states for the different possible percepts will be disjoint, forming a partition of the original predicted belief state (for deterministic sensing).

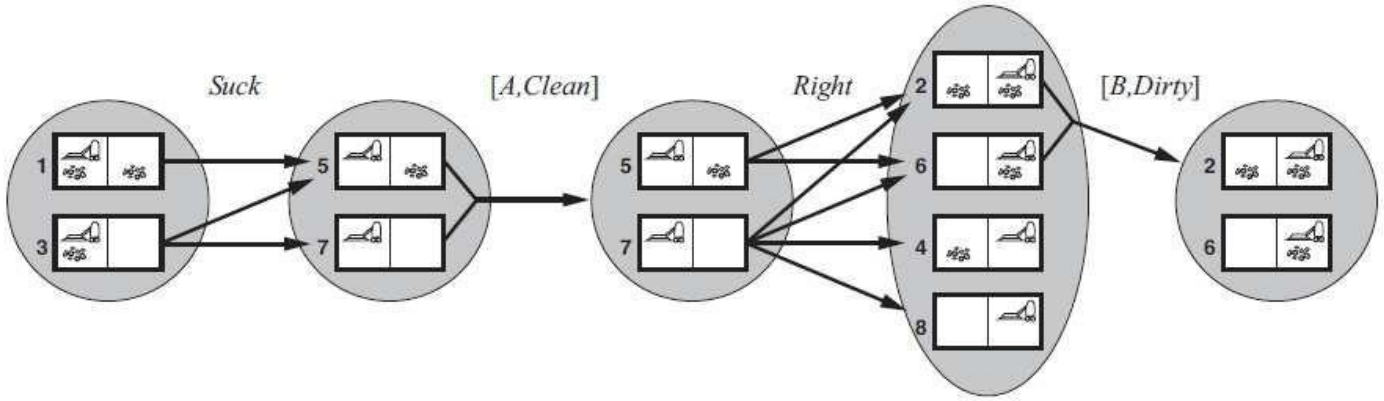
Solving partially observable problems



Assuming initial percept [A, dirty] the solution is conditional plan ...

Solving partially observable problems

[Suck, Right, if Bstate ={6} then Suck else [ ]] .



Summary

◆Local search methods such as hill climbing operate on complete-state formulations, keeping only a small number of nodes in memory. Several stochastic algorithms have been developed, including simulated annealing, which returns optimal solutions when given an appropriate cooling schedule.

◆A genetic algorithm is a stochastic hill-climbing search in which a large population of states is maintained. New states are generated by mutation and by crossover, which combines pairs of states from the population

◆In nondeterministic environments, agents can apply ANDOR search to generate contingent plans that reach the goal regardless of which outcomes occur during execution.

◆When the environment is partially observable, the belief state represents the set of possible states that the agent might be in.