Beyond Classical Search

- Chapter 3 covered problems that considered the whole search space and produced a sequence of actions leading to a goal.

- Chapter 4 covers techniques (some developed outside of AI) that don't try to cover the whole space and only the goal state, not the steps, are important.

- The techniques of Chapter 4 tend to use much less memory and are not guaranteed to find an optimal solution.

More Search Methods

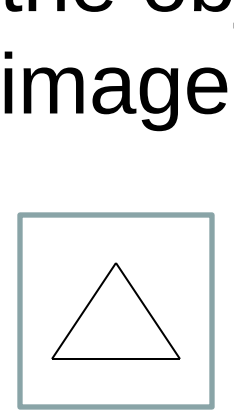
- Local Search
  - Hill Climbing
  - Simulated Annealing
  - Beam Search
  - Genetic Search
- Local Search in Continuous Spaces
- Searching with Nondeterministic Actions
- Online Search (agent is executing actions)

Local Search Algorithms and Optimization Problems

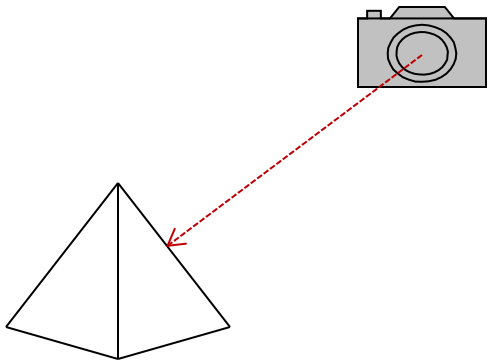
- Complete state formulation
  - For example, for the 8 queens problem, all 8 queens are on the board and need to be moved around to get to a goal state
- Equivalent to optimization problems often found in science and engineering
- Start somewhere and try to get to the solution from there
- Local search around the current state to decide where to go next

Pose Estimation Example

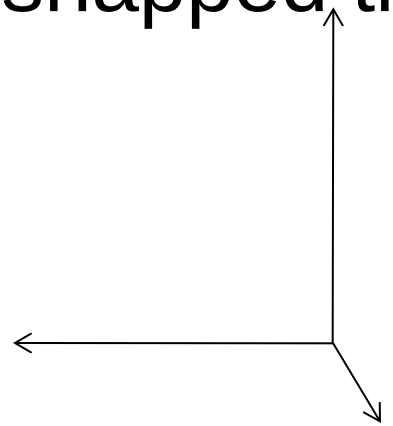
- Given a geometric model of a 3D object and a 2D image of the object.
- Determine the position and orientation of the object wrt the camera that snapped the image.



image

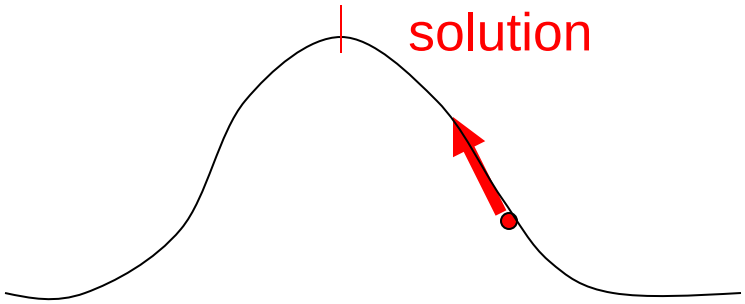


3D object



- State (x, y, z, θx, θy, θz)

Hill Climbing "Gradient ascent"



Note: solutions shown here as max not min.

Often used for numerical optimization problems.

How does it work?

In continuous space, the gradient tells you the direction in which to move uphill.

AI Hill Climbing

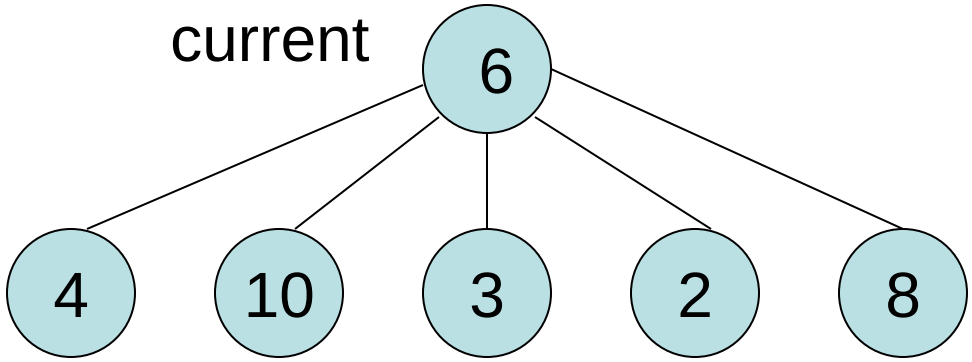
Steepest-Ascent Hill Climbing

- *current* ← start node
- loop do
  - *neighbor* ← a highest-valued successor of *current*
  - if *neighbor*.Value <= *current*.Value then return *current*.State
  - *current* ← *neighbor*
- end loop

At each step, the current node is replaced by the best (highest-valued) neighbor.

This is sometimes called greedy local search.

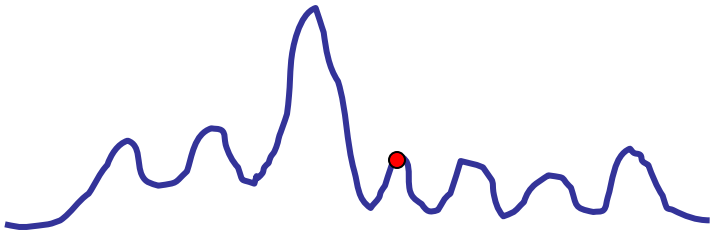
Hill Climbing Search



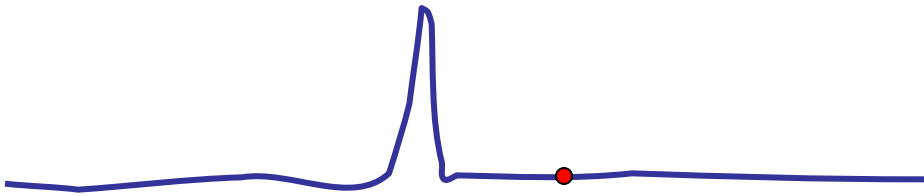
What if current had a value of 12?

Hill Climbing Problems

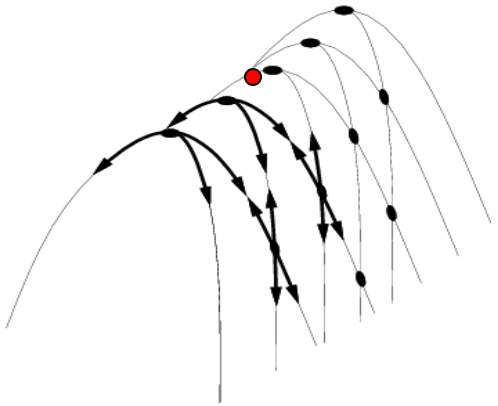
Local maxima



Plateaus



Diagonal ridges



What is it sensitive to?
Does it have any advantages?

Solving the Problems

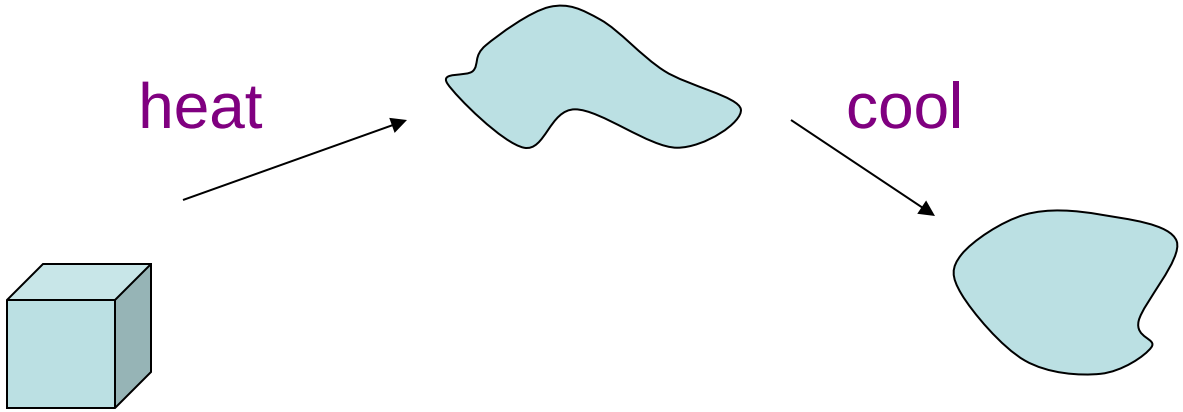
- Allow backtracking (What happens to complexity?)
- Stochastic hill climbing: choose at random from uphill moves, using steepness for a probability
- Random restarts: “If at first you don’t succeed, try, try again.”
- Several moves in each of several directions, then test
- Jump to a different part of the search space

Simulated Annealing

- Variant of hill climbing (so up is good)
- Tries to explore enough of the search space early on, so that the final solution is less sensitive to the start state
- May make some downhill moves before finding a good way to move uphill.

Simulated Annealing

- Comes from the physical process of annealing in which substances are raised to high energy levels (melted) and then cooled to solid state.



- The probability of moving to a higher energy state, instead of lower is

  $p = e^{(-\Delta E/kT)}$

  where ΔE is the positive change in energy level, T is the temperature, and k is Bolzmann's constant.

Simulated Annealing

- At the beginning, the temperature is high.
- As the temperature becomes lower
  - kT becomes lower
  - $\Delta E/kT$ gets bigger
  - $(-\Delta E/kT)$ gets smaller
  - $e^{(-\Delta E/kT)}$ gets smaller
- As the process continues, the probability of a downhill move gets smaller and smaller.

For Simulated Annealing

- ΔE represents the change in the value of the objective function.
- Since the physical relationships no longer apply, drop k. So p = e^(-ΔE/T)
- We need an annealing schedule, which is a sequence of values of T: T_0, T_1, T_2, ...

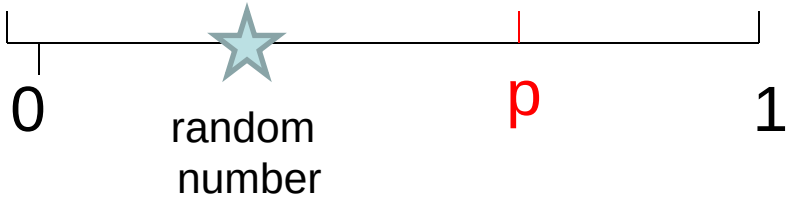
Simulated Annealing Algorithm

- *current* ← start node;
- for each T on the schedule /* need a schedule */
  - *next* ← randomly selected successor of *current*
  - evaluate next; it it's a goal, return it
  - ΔE ← *next*.Value – *current*.Value /* already negated */
  - if ΔE > 0
    - then *current* ← *next* /* better than current */
    - else *current* ← *next* with probability e^(ΔE/T)

How would you do this probabilistic selection?

Probabilistic Selection

- Select *next* with probability p



- Generate a random number
- If it's <= p, select *next*

Simulated Annealing Properties

- At a fixed “temperature” T, state occupation probability reaches the Boltzman distribution

  $p(x) = \alpha e^{(E(x)/kT)}$

- If T is decreased slowly enough (very slowly), the procedure will reach the best state.

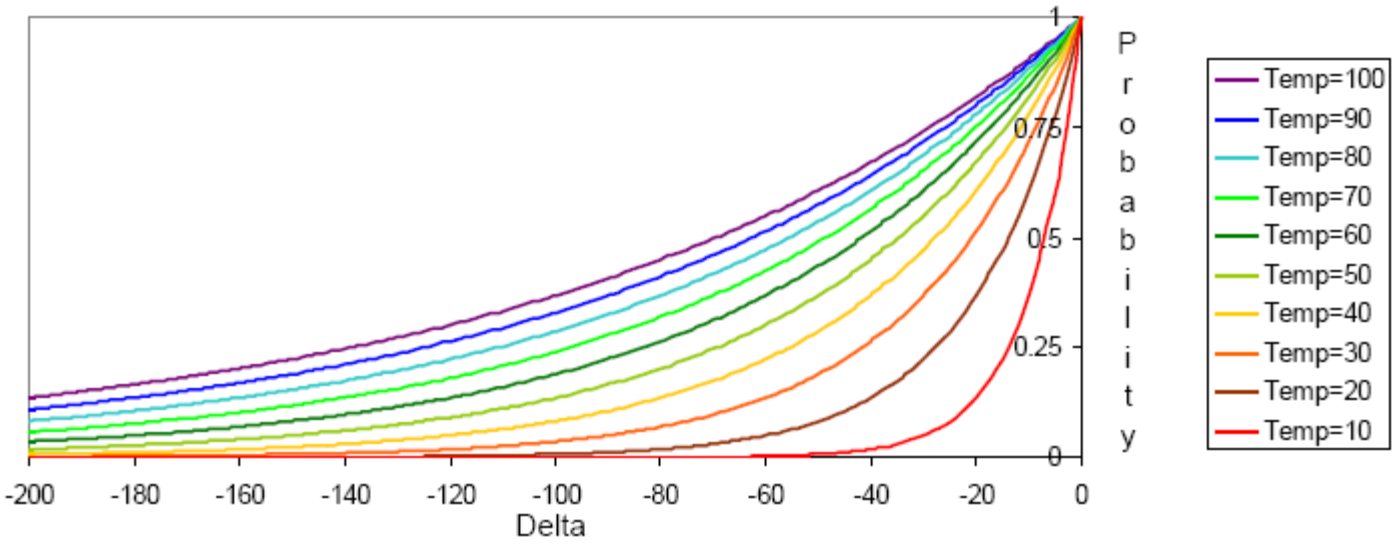
- Slowly enough has proven too slow for some researchers who have developed alternate schedules.

Simulated Annealing Schedules

- Acceptance criterion and cooling schedule

if (delta>=0) accept

else if ($random < e^{delta/Temp}$) accept, else reject /* 0<=random<=1 */



Initially temperature is very high (most bad moves accepted)

Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily “quench” the system

Simulated Annealing Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  - VLSI design
    - Placement
    - Routing
    - Array logic minimization
    - Layout
  - Facilities layout
  - Image processing
  - Code design in information theory

Local Beam Search

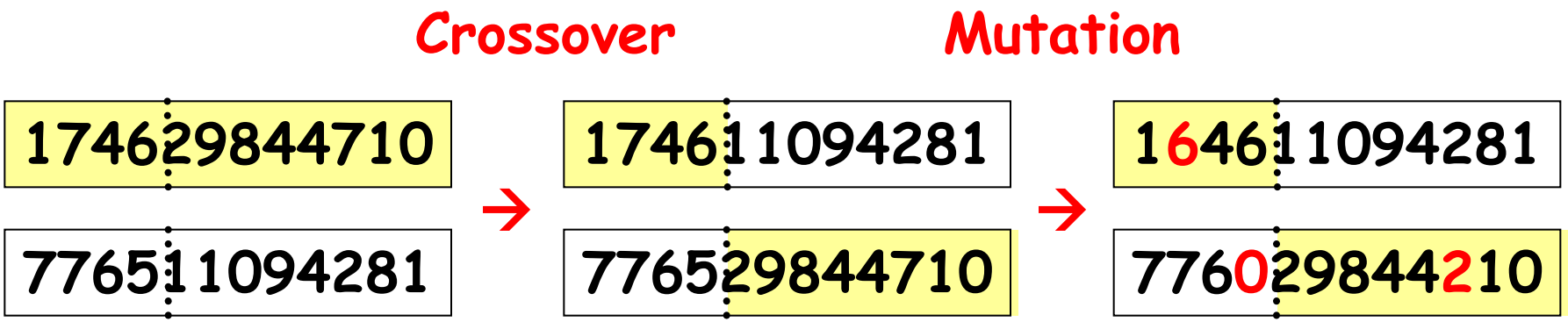
- Keeps more previous states in memory
  - Simulated annealing just kept one previous state in memory.
  - This search keeps k states in memory.

    - randomly generate k initial states
    - if any state is a goal, terminate
    - else, generate all successors and select best k
    - repeat

Local Beam Search

Genetic Algorithms

- Start with random population of states
  - Representation serialized (ie. strings of characters or bits)
  - States are ranked with “fitness function”
- Produce new generation
  - Select random pair(s) using probability:
    - probability ~ fitness
  - Randomly choose “crossover point”
    - Offspring mix halves
  - Randomly mutate bits



Genetic Algorithm

- Given: population P and fitness-function f
- repeat
  - newP ← empty set
  - for *i* = 1 to size(P)
    - *x* ← RandomSelection(P,f)
    - *y* ← RandomSelection(P,f)
    - *child* ← Reproduce(*x*,*y*)
    - if (small random probability) then child ← Mutate(child)
    - add *child* to newP
  - P ← newP
- until some individual is fit enough or enough time has elapsed
- return the best individual in P according to f

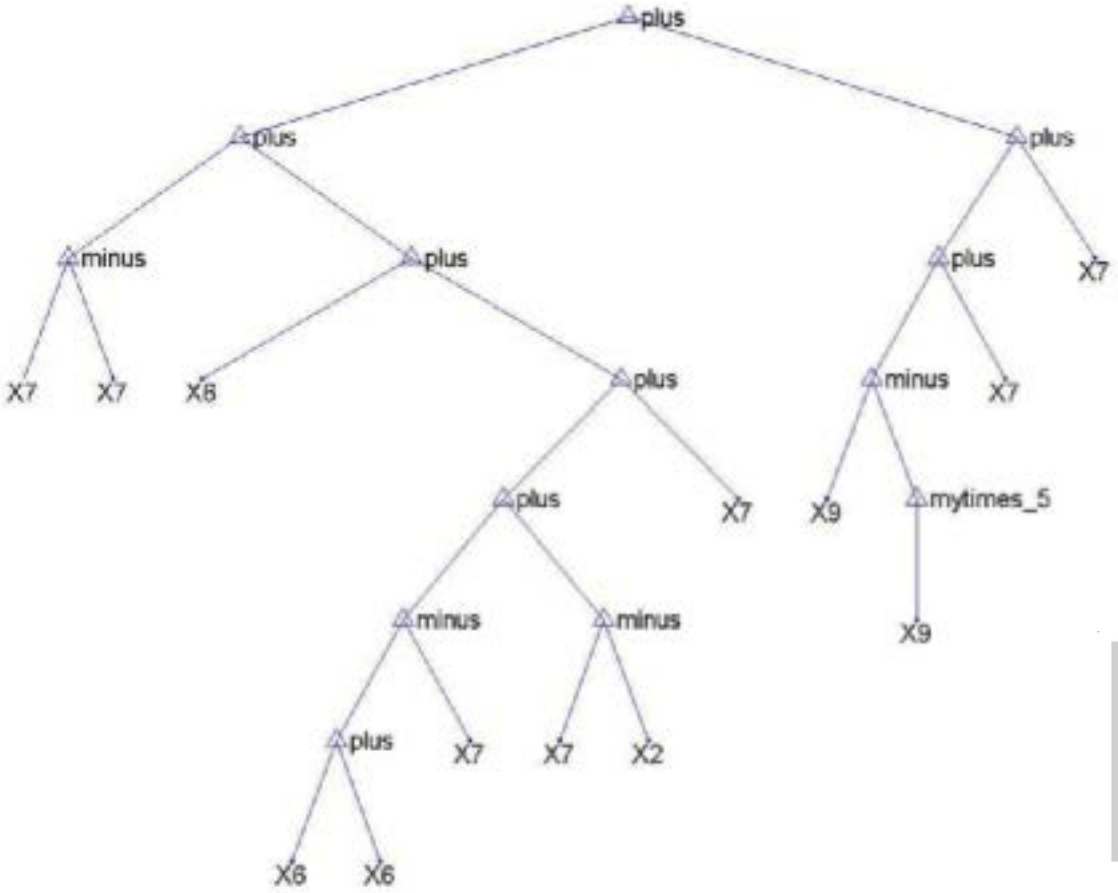
Using Genetic Algorithms

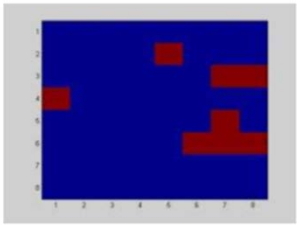
- 2 important aspects to using them
  - 1. How to encode your real-life problem
  - 2. Choice of fitness function
- Research Example
  - I have N variables V1, V2, ... VN
  - I want to produce a single number from them that best satisfies my fitness function F
  - I tried linear combinations, but that didn't work
  - A guy named Stan I met at a workshop in Italy told me to try Genetic Programming

Genetic Programming

- Like genetic algorithm, but instead of finding the best character string, we want to find the best arithmetic expression tree
- The leaves will be the variables and the non-terminals will be arithmetic operators
- It uses the same ideas of crossover and mutation to produce the arithmetic expression tree that maximizes the fitness function.

Tree structure for quantifying midface hypoplasia





Xi are the selected histogram bins from an azimuth-elevation histogram of the surface normals of the face.

Local Search in Continuous Spaces

- Given a continuous state space

  $S = \{(x_1, x_2, \ldots, x_N) \mid x_i \in \mathbb{R}\}$
- Given a continuous objective function

  $f(x_1, x_2, \ldots, x_N)$
- The gradient of the objective function is a vector $\nabla f = (\partial f/\partial x_1, \partial f/\partial x_2, \ldots, \partial f/\partial x_N)$
- The gradient gives the magnitude and direction of the steepest slope at a point.

Local Search in Continuous Spaces

- To find a maximum, the basic idea is to set $\nabla f = 0$
- Then updating of the current state becomes

  $x \leftarrow x + \alpha \nabla f(x)$

  where α is a small constant.
- Theory behind this is taught in numerical methods classes.
- Your book suggests the Newton-Raphson method. Luckily there are packages…..

Computer Vision Pose Estimation Example

I have a 3D model of an object and an image of that object.

I want to find the pose: the position and orientation of the camera.

Figure 18: The pose computed from six point correspondence using the algorithm described in Section 5.2.

Figure 19: The pose computed from an ellipse-circle correspondence using the algorithm described in Section 5.6.

and

$$R = \begin{pmatrix} s^2 + l^2 - m^2 - n^2 & 2(ln - sn) & 2(ln + sm) \\ 2(lm + sn) & s^2 - l^2 + m^2 - n^2 & 2(mn - sl) \\ 2(ln - sm) & 2(mn + sl) & s^2 - l^2 - m^2 + n^2 \end{pmatrix}. \quad (52)$$

Powell's method [19] in the seven-dimensional space of the pose solution (four quaternion parameters and the translation t) is used, along with the constraint that the sum of the squares of the quaternion parameters must equal 1, as seen in equation (51). Figure 21 shows an initial pose estimate for a single-object image as

Figure 20: The pose computed from six point correspondence and one ellipse-circle correspondence using the generalized methodology.

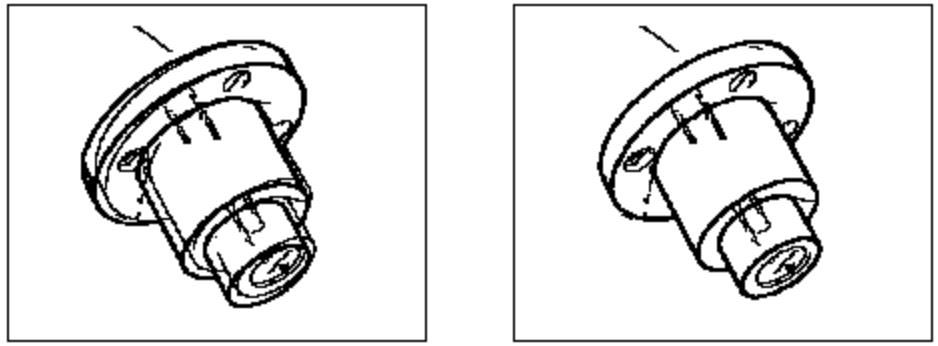
pose from 6 point correspondences

pose from ellipse-circle correspondence

pose from both 6 points and ellipse-circle correspondences

Computer Vision Pose Estimation Example

Initial pose from points/ellipses and
final pose after optimization.



(a) Initial pose (b) Final pose

Figure 21: Example pose hypothesis and final pose after constrained optimization.

- The optimization was searching a 6D space: (x,y,z,θx,θy,θz)
- The fitness function was how well the projection of the 3D object lined up with the edges on the image.

Searching with Nondeterministic Actions

- Vacuum World (actions = {left, right, suck})

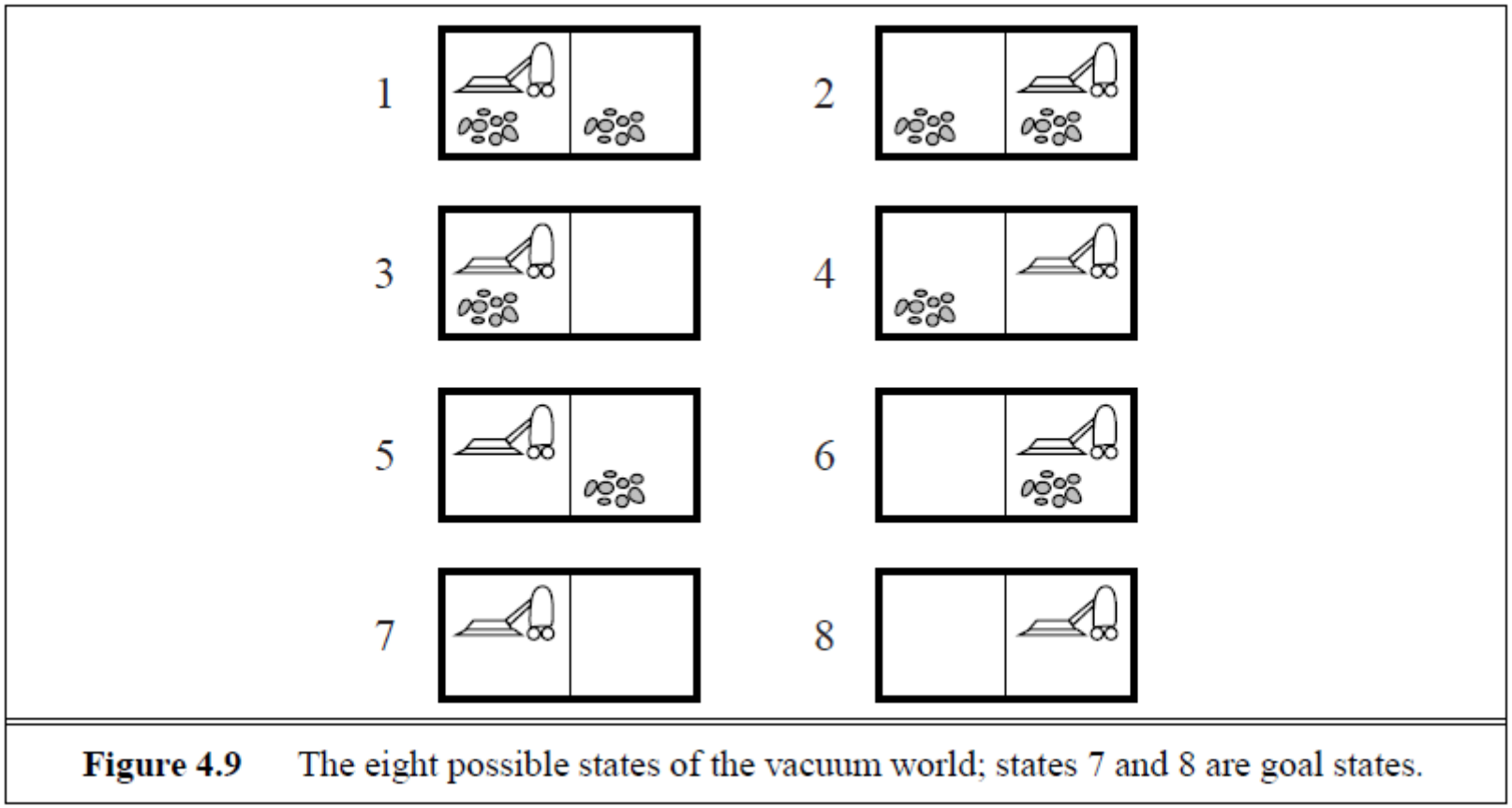


Figure 4.9 The eight possible states of the vacuum world; states 7 and 8 are goal states.

Searching with Nondeterministic Actions

In the nondeterministic case, the result of an action can vary.

Erratic Vacuum World:

- When sucking a dirty square, it cleans it and sometimes cleans up dirt in an adjacent square.
- When sucking a clean square, it sometimes deposits dirt on the carpet.

Generalization of State-Space Model

1. Generalize the transition function to return a set of possible outcomes.

   oldf: S x A -> S newf: S x A -> 2^S

2. Generalize the solution to a contingency plan.

   if state=s then action-set-1 else action-set-2

3. Generalize the search tree to an AND-OR tree.

AND-OR Search Tree

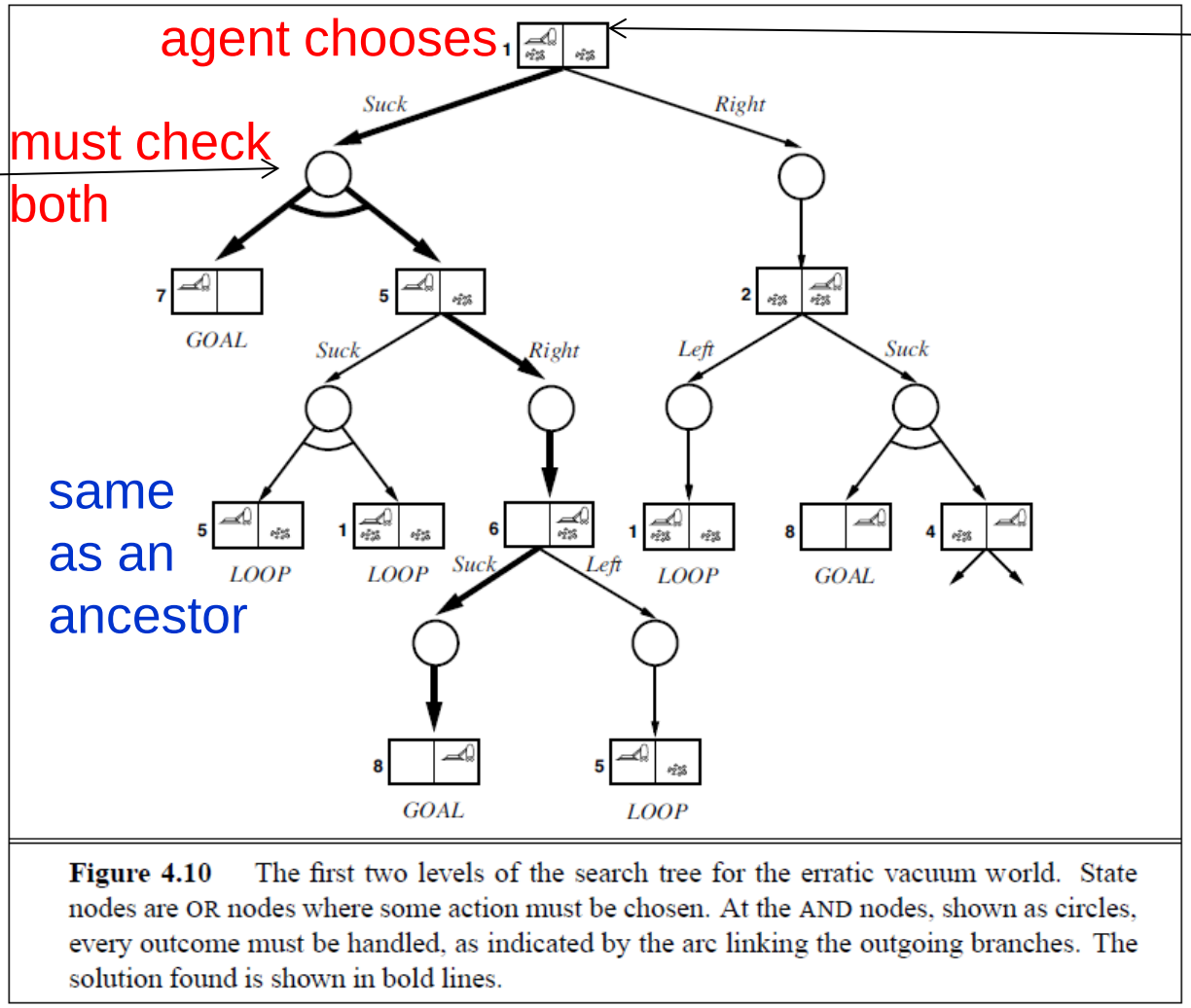


Figure 4.10 The first two levels of the search tree for the erratic vacuum world. State nodes are OR nodes where some action must be chosen. At the AND nodes, shown as circles, every outcome must be handled, as indicated by the arc linking the outgoing branches. The solution found is shown in bold lines.

Searching with Partial Observations

- The agent does not always know its state!
- Instead, it maintains a belief state: a set of possible states it might be in.
- Example: a robot can be used to build a map of a hostile environment. It will have sensors that allow it to “see” the world.

Belief State Space for Sensorless Agent

Knows it's on the left

Knows it's on the right.

Knows left side clean

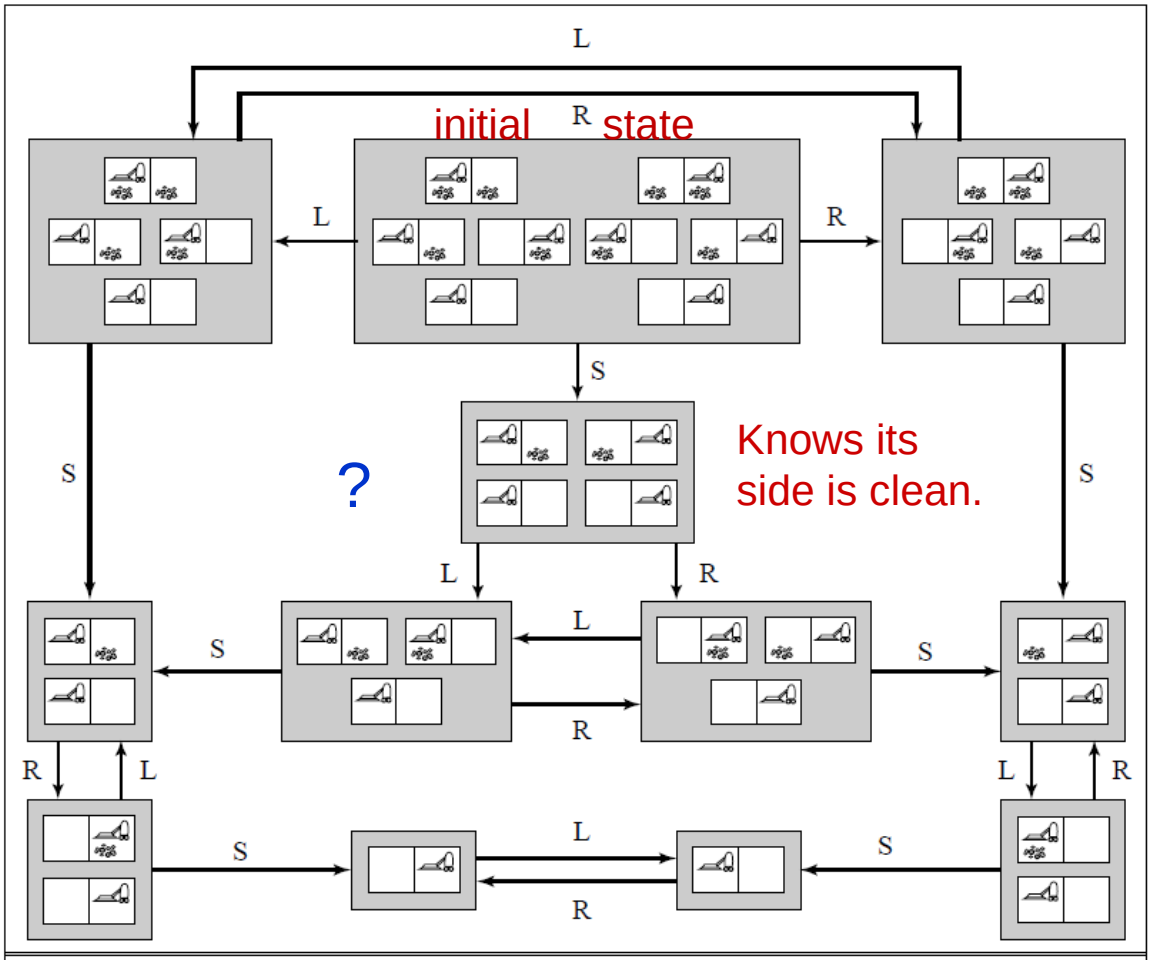


Figure 4.14 The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each shaded box corresponds to a single belief state. At any given point, the agent is in a particular belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box. Actions are represented by labeled links. Self-loops are omitted for clarity.

Online Search Problems

- Active agent
  - executes actions
  - acquires percepts from sensors
  - deterministic and fully observable
  - has to perform an action to know the outcome
- Examples
  - Web search
  - Autonomous vehicle