Recap

- A* uses both backward costs and (estimates of) forward costs
- Heuristic design is key: often use relaxed problems
- A* is optimal with admissible (tree search) / consistent heuristic (graph search)

A* Optimality

TREE SEARCH

- A* is optimal if heuristic is admissible
- UCS is a special case ($h(n) = 0$ for all n)

GRAPH SEARCH

- A* optimal if heuristic is consistent
- UCS optimal ($h = 0$ is consistent)

Consistency $\Rightarrow$ admissibility

In general most admissible heuristics tend to be consistent.

SEARCH AND MODELS

- Search operates over **models of the world**
- The agent does not actually try all the plans out in the real world
- Planning is all "in simulation"
- Search is only as good as the model (Ouch! Bottlenecked!)

In many problems, **path is irrelevant**, the goal state itself is the only thing we care about.

Sometimes the **goal test itself is unclear**, in reality we are solving an optimization problem.

TRAVELING SALESMAN

- Given a list of cities and distances between each pair of cities, what is the **shortest possible route** that visits each city exactly once and returns to the origin city?

DISCRETE SPACE

Local Search

- Useful when path to goal state does not matter/solving pure optimization problem

BASIC IDEA:

- Only keep a single "current state"
- Heuristic function to evaluate the "goodness" of the current state
- Improve by iterations
- Don't save paths followed

CHARACTERISTICS:

- Low memory requirements - usually constant
- Effective -> find good solutions in extremely large state spaces

HILL-CLIMBING SEARCH

- State space -> landscape
- Location = state
- Elevation = evaluation of state (objective function value)
- Continually move in direction of increasing value (try to maximize obj. function)

```
function HillClimbing(problem) return a state that is a local maximum
current <- initial state
repeat:
    best neighbor <- current
    for state in Neighbors(current):
        if state.value > best Neighbor.value
            best Neighbor <- state
    if best Neighbor.value > current.value:
        current <- best Neighbor
    else:
        return current
```

CHALLENGES FOR HILL CLIMBING

- Local maxima; when it is reached, there is no way to backtrack or move out of that maximum
- Plateau; can save difficult time finding its way off a flat portion of the state space landing
- Ridges; can produce a series of local maxima that are difficult to navigate out of

SOME VARIANTS OF LOCAL HILL-CLIMBING

- Stochastic hill-climbing; select randomly from all moves that improve the value of the objective function

- Random-restart hill-climbing; conducts a series of hill-climbing searches, starting from random positions, very frequently used in general AI

SIMULATED ANNEALING

- Idea: mostly goes "uphill" but occasionally travels "downhill" to escape local optimum
- Likelihood to go downhill is controlled by a "temperature schedule"
- more and more "conservative" as the search progresses (less likely to go downhill)

```
function SimulatedAnnealing(problem, schedule) return a solution state
current <- initial state
for t = 1 to infty
    T <- schedule(t)
    if T = 0 then return current
    next <- a randomly selected neighbor of current
    DELTAE = next.value - current.value
    if DELTAE > 0:
        current <- next
    else:
        current <- next with probability e^{DELTAE/T}
```

Note $\Delta E < 0$ and $T -> 0 \Rightarrow e^{\Delta E/T} -> 0$

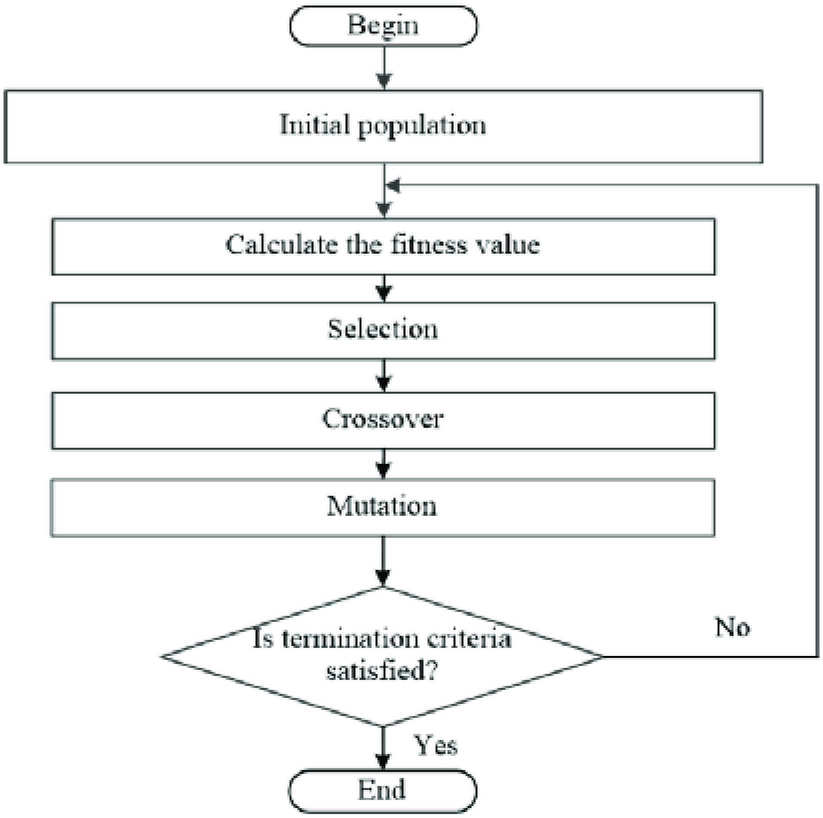
LOCAL BEAM SEARCH

- Similar to hill-climbing, but
- keep track of k current states rather than just a single current state
- select the k best neighbors among all neighbors of the k current states

ANALOGY

- Hill-climbing: " trying to find the top of Mt. Everest in a thick fog while suffering amnesia"
- Local beam search: "Doing this with several friends, each of whom has a short-range radio and an altimeter"
- Stochastic beam search: "select successors at random weighted by value"

GENETIC ALGORITHM



- Inspired by evolutionary biology
- Mimics the evolution of a population under natural selection

DISCRETE STATE SPACE

Ex: bounding box prediction (as search)

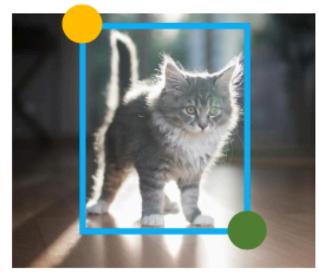
- Object detection: single object

- Idea: check all possible boxes, find the box that contains "the object"

Bounding Box Prediction (as Search)

Hill Climbing Search

$$\max_{\text{box}} f(\text{Image}, \text{box})$$



How to represent a state?

- Bounding box:
  - Top-left and bottom right corners
  - (x1,y1,x2,y2)
- How to define a neighborhood?
  - (x1+1,y1,x2,y2)
  - (x1-1,y1,x2,y2)
  - (x1,y1+1,x2,y2)
  - (x1,y1-1,x2,y2)
  - …

Search formulation used in computer vision research (Li’22, Yeh’17, Yeh’18):