Local Search

School of EECS
Washington State University
Local Search

- Focused on finding a goal state
  - Less focused on solution path or cost
- Choose a state and search nearby (local) states
  - Not a systematic search of the state space
- Advantages
  - Use little memory
  - Can find solutions in large or infinite state spaces
- Can also solve optimization problems
  - Maximize some objective function
Water Jug Problem

- **States**: Water jugs of various sizes with some amount of water in them
  - Jug j has capacity c(j) and contains w(i) gallons of water
- **Initial state**: Water jugs all empty: w(j) = 0
- **Actions**:
  - Fill a jug to the top with water from water source
  - Pour water from one jug into another until second jug is full or first jug is empty
  - Empty all water from a jug
- **Transition model**:
  - Fill(j): w(j) = c(j)
  - Pour(j1,j2):
    - w(j1) = max(0,w(j1)−c(j2)+w(j2))
    - w(j2) = min(c(j2),w(j1)+w(j2))
  - Empty(j): w(j) = 0
- **Goal test**: Some w(j) = X
- **Path cost**: Number of actions

Die Hard with a Vengeance (1995)
c(1)=3, c(2)=5, Goal: w(2)=4
State–Space Landscape
Local Search Techniques

- Hill climbing
- Simulated annealing
- Beam search
- Genetic algorithms
Hill–Climbing Search

function **HILL-CLIMBING** *(problem)* **returns** a state which is a local maximum

\[ current ← MAKE-NODE(problem.\text{INITIAL-STATE}) \]

loop do

    \[ neighbor ← \text{a highest-valued successor of } current \]

    if \[ \text{neighbor.VALUE} ≤ current.VALUE \] then return \[ current.\text{STATE} \]

\[ current ← neighbor \]

Also called “steepest ascent” or “greedy local search”

- Gets stuck in local maxima, ridges and plateaux
  - Some number of “sideways” moves may help
Ridges in State Space Objective Function

- Only way to get up ridge is to first go down
Hill–Climbing

- Complete?
- Optimal?
- Time complexity?
- Space complexity?
Hill–Climbing Search Variants

- **Stochastic hill climbing**
  - Randomly selects from among uphill moves
  - Selection weighted by move steepness

- **First–choice hill climbing**
  - Randomly generates successors and chooses first uphill move generated

- **Random–restart hill climbing**
  - Performs multiple hill–climbing searches from different random initial states
Simulated Annealing

Annealing is the process of heating and then cooling materials to improve certain properties (e.g., strength)

Simulated annealing
- Randomly pick a move
- If positive improvement, then make move
- If negative improvement, then make move with some probability $P$
  - $P$ proportional to improvement
  - $P$ decreases over time (i.e., cooling)

Hill–climbing with some chance of descending
function **SIMULATED-ANNEALING** (*problem, schedule*) returns a solution state

`current ← MAKE-NODE(*problem*.INITIAL-STATE)`

for t = 1 to ∞ do

`T ← schedule(t)`

if T = 0 then return `current`

`next ←` a randomly selected successor of `current`

`ΔE ← next.VALUE – current.VALUE`

if `ΔE > 0` then `current ← next`

else `current ← next` only with probability $e^{ΔE/T}$

- **Schedule** is a mapping from time to temperature
- If schedule lowers T slowly enough, algorithm will find global maximum
Beam Search

- Keeps track of \( k \) states rather than just one
- Each iteration
  - All successors of \( k \) states are generated
  - Keeps \( k \) best successor states
- Problem: \( k \) states may become too similar (lack diversity)
- Solution: Stochastic beam search
  - Choose \( k \) successors at random with probability based on value
Genetic Algorithm (GA)

- Variant of beam search
  - Successor states generated by combining pairs of k states
- GA begins with k randomly generated states, called the population
- Each state, or individual, is represented by a string over a finite alphabet
- Pairs of population selected as parents based on their value (fitness function)
- Parents “mated” using crossover to produce offspring (another k individuals)
- Offspring subjected to mutation
Genetic Algorithm

(a) Initial Population
(b) Fitness Function
(c) Selection
(d) Crossover
(e) Mutation
function **GENETIC-ALGORITHM** *(population, FITNESS-FN)* returns an individual
repeat

    *new_population* ← empty set
    for i = 1 to *SIZE*(population) do
        x ← **RANDOM-SELECTION**(population, FITNESS-FN)
        y ← **RANDOM-SELECTION**(population, FITNESS-FN)
        child ← **REPRODUCE**(x,y)
        if (small random probability) then child ← **MUTATE**(child)
        add child to *new_population*
    population ← *new_population*
until some individual is fit enough, or enough time has elapsed
return the best individual in *population*, according to FITNESS-FN

Much of success depends on representation of individuals and clever crossover
Summary

- Local search
- Select one or more random initial states and search among nearby states for goal
- Good for finding reasonable solutions in large state spaces