Local Search
Toolbox so far

• Uninformed search
  – BFS, DFS, Iterative deepening DFS, Uniform cost search

• Heuristic search
  – A*
Review

• Search:
  – Use in environments that are static, discrete, 100% observable, deterministic.

• Things we care about:
  – Completeness, optimality, time/space complexity.
it's not the destination that matters, it's the journey.
New Idea: Local Search

• Can be used when path from start state to goal doesn't matter (only the goal matters).
• Process is slightly different than "normal" search:
  – Nodes/states are always complete solutions to the problem, not partial solutions.
  – One current node is maintained that has the best solution at the moment.
  – Actions generate new nodes with new complete solutions.
Local Search

• Benefits:
  – Use very little memory, often constant.
  – Can search very large state spaces quickly.

• Useful in optimization problems.
State-space landscape
State-space landscape

Graph that shows the values of the heuristic cost function or objective function in terms of the search space of possible states.
Hill climbing algorithm

• Loop that looks at all possible neighbors of the current state, and picks the one that increases the optimization function the most.

```plaintext
function HILL-CLIMBING(problem) returns a state that is a local maximum

current ← MAKE-NODE(problem.INITIAL-STATE)
loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≥ current.VALUE then return current.STATE
    current ← neighbor
```
Variants

• Stochastic hill climbing
• Random-restart hill climbing
Simulated annealing
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
         schedule, a mapping from time to "temperature"

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 $e^{ΔE/T}$
Beam search

• Variation of hill climbing
  – Use k current states
  – Generate all of their successors
  – Take k best

• Variation: stochastic beam search
  – Adds in probabilistic idea from simulated annealing.
  – Same as above, but take k best successors based on probability.
Genetic algorithms

• Variation on stochastic beam search.
• Successor states are generated using two parent states, not one. (Crossover)
• Mutation: Randomly modifies a current state.
function Genetic-Algorithm\((\text{population}, \text{Fitness-Fn})\) returns an individual

**inputs:** \(\text{population}\), a set of individuals

\(\text{Fitness-Fn}\), a function that measures the fitness of an individual

repeat

\(\text{new-population} \leftarrow \text{empty set}\)

**for** \(i = 1\) **to** \(\text{Size}(\text{population})\) **do**

\(x \leftarrow \text{Random-Selection}(\text{population}, \text{Fitness-Fn})\)

\(y \leftarrow \text{Random-Selection}(\text{population}, \text{Fitness-Fn})\)

\(\text{child} \leftarrow \text{Reproduce}(x, y)\)

**if** (small random probability) **then** \(\text{child} \leftarrow \text{Mutate} (\text{child})\)

add \(\text{child}\) to \(\text{new-population}\)

\(\text{population} \leftarrow \text{new-population}\)

until some individual is fit enough, or enough time has elapsed

**return** the best individual in \(\text{population}\), according to \(\text{Fitness-Fn}\)