CMU 15-381
Lecture 4: Local Search

Teachers: Alex Waibel
Path Search vs. Local Search

- The algorithms discussed so far are designed to find a goal state from a start state: the path to the goal constitutes a solution to the search problem.
- In many problems the path doesn’t matter: the goal state itself is the solution.

State space = set of “complete” configurations
- **Optimization problems**: Find optimal configuration (objective or cost function).
- **Constraint Satisfaction Problems**: Find configurations satisfying (all or the highest number of) constraints.
Path Search vs. Local Search

- **Local search algorithms** at each step consider a single "current" state, and try to improve it by moving to one of its neighbors → Iterative improvement algorithms

- **Pros and cons**
  - No complete (no optimal), except with *random restarts*
  - Space complexity $O(b)$
  - Time complexity $O(d)$, $d$ can be $\infty$
  - Can perform well also in large (infinite, continuous) spaces
  - Relatively easy to implement
HILL-CLIMBING SEARCH

• Move in the direction of increasing value of goodness (up the hill)
  ○ (later: down-hill if it’s a cost or error function)
• Terminate when no neighbor has higher value
HILL-CLIMBING SEARCH
Like Climbing Everest in thick for 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

• Move in the direction of increasing value (up the hill)
• Terminate when no neighbor has higher value

Greedy (myopic) local search
State space landscape

Objective function

- Global maximum
- Shoulder
- Local maximum
- "Flat" local maximum
- Current state

State space
CSP Example: N-Queens

Put \( n \) queens on an \( n \times n \) board with no two queens on the same row, column, or diagonal.

**State:** Position of the \( n \) queens, one per column (or row)

**Successor states:** generated by moving a single queen to another square in its column \( (n(n-1)) \)

**Cost of a state:** the number of constraint violations
State with 17 conflicts, showing the #conflicts by moving a queen within its column, with best moves in red.

Local optimum: state that has only one conflict, but every move leads to larger #conflicts.

HILL-CLIMBING SEARCH
Variants on Hill-Climbing

• Random-restart hill climbing: Conducts a series of hill-climbing searches from random states
  o Probabilistically complete
  o Expected number of iterations is roughly 7, with roughly 22 steps overall
**Variants on Hill-Climbing**

- **Random-restart hill climbing:** Conducts a series of hill-climbing searches from random states
  - Probabilistically complete
  - Expected number of iterations roughly $7$, with roughly $22$ steps overall

*If at first you don’t succeed, try, try again!"
VARIANTS ON HILL-CLIMBING

- Random-restart hill climbing: Conducts a series of hill-climbing searches from random states
  - Probabilistically complete
  - Expected number of iterations is roughly 7, with roughly 22 steps overall

- Sideways moves: if no uphill moves, allow moving to a state with the same value as the current one (escape shoulders)
Hill-Climbing Performance

• Hill-Climbing can solve large instances of n-queens (n=10^6) in a few milliseconds

• 8 queens statistics:
  o State space of size \( \approx 17 \text{ million} \)
  o Starting from random state, steepest-ascent hill climbing solves 14% of problem instances
  o It takes 4 steps on average when it succeeds and 3 when it gets stuck
  o When “sideways” moves are allowed, solves 94% of instances, but with 21 steps for success and 64 for failure
Hill-Climbing Can Get Stuck!

- Objective function
- Global maximum
- Shoulder
- Local maximum
- "Flat" local maximum
- Current state
- State space

**sideways moves** ($M$):
- $M=100 \rightarrow 94\%$ solved instances for the 8-queens!
- 21 steps avg. on success
- 64 steps avg. on "failure"

**random restarts:**
- 100\% solved instances
- 28 steps avg.
Variants on Hill-Climbing

• Random-restart hill climbing: Conducts a series of hill-climbing searches from random states
  o Probabilistically complete
  o Expected number of iterations is roughly 7, with roughly 22 steps overall

• Sideways moves: if no uphill moves, allow moving to a state with the same value as the current one (escape shoulders)

• Stochastic hill climbing: Selection among the available uphill moves is done randomly (uniform, proportional, soft-max, e-greedy,…) depending on the improvement, to be “less” greedy
Hill-Climb can get stuck!

Diagonal ridges:
From each local maximum all the available actions point downhill, but there is an uphill path!

Zig-zag motion, very long ascent time!

Gradient ascent doesn’t have this issue: all state vector components are (potentially) changed when moving to a successor state, climbing can follow the direction of the ridge.
Local Optima

• But random restarts is not guaranteed to improve: Do I feel lucky?
SIMULATED ANNEALING

• *Escape from local optima* by accepting, with a probability that decreases during the search, also moves that are worse than the current solution (going downhill!)

• Stochastic, solution-improvement *metaheuristic* for global *optimization*

• Inspired by the process of *annealing* of solids in metallurgy:
  • The temperature of the solid is increased until it melts
  • The temperature is *slowly decreased through a quasi-static process* until the solid reaches a minimal energy state in which a regular crystal structure appears
Simulated Annealing

```python
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 $e^{ΔE/T}$
```
Simulated Annealing

procedure Simulated_Annealing()
    \[ S' = \{ \text{set of all feasible solutions} \}; \]
    \[ \mathcal{N} = \text{neighborhood structure defined over } S; \]
    \[ s \leftarrow \text{Generate a starting feasible solution}; \quad \text{\texttt{// e.g., with a construction heuristic}} \]
    \[ s_{\text{best}} \leftarrow s; \]
    \[ T \leftarrow \text{Determine a starting value for temperature}; \]
    \[ \text{\texttt{while (NOT YET frozen)} \quad \text{\texttt{// termination criterion}} \]
        \[ \text{\texttt{while (NOT YET AT equilibrium FOR THIS TEMPERATURE)} \]
            \[ s' \leftarrow \text{Choose a random solution from neighborhood } \mathcal{N}(s); \quad \text{\texttt{// e.g., select a random 2-opt move}} \]
            \[ \Delta E \leftarrow f(s') - f(s); \]
            \[ \text{if } (\Delta E \leq 0) \quad \text{\texttt{// downhill, locally improving move}} \]
            \[ s \leftarrow s'; \]
            \[ \text{if } (f(s) < f(s_{\text{best}})) \]
            \[ s_{\text{best}} \leftarrow s; \]
        \[ \text{else} \quad \text{\texttt{// uphill move}} \]
            \[ r \leftarrow \text{Choose a random number uniformly from } [0,1]; \]
            \[ \text{if } (r < e^{-\Delta E/T}) \quad \text{\texttt{// accept the uphill, not improving, move}} \]
            \[ s \leftarrow s'; \]
        \[ \text{end if} \]
    \[ \text{end while} \]
    \[ T \leftarrow \text{Lower the temperature according to the selected cooling schedule}; \]
    \[ \text{end while} \]
    \[ \text{return } s_{\text{best}}; \]
end procedure
Effect of Temperature

![Graph showing the effect of temperature on the acceptance probability. The graph plots the change in probability against the change in energy (ΔE = f(s') - f(s)) for different temperatures (T = 1, 10, 50, 100).]
Beam Search

• Keeping only one (the best) note is a bit extreme attempt to minimize memory

• How about retaining $k$ states rather than just one?

• Beam Search:
  
  Start with random set of $k$ states
  
  Loop:
  
  Generate all successors of all $k$ states
  
  If anyone is a goal, terminate
  
  Else select $k$-best successors from complete list
Beam Search

• Is it just k parallel Hill-Climbing searches?
• k best successes can switch track and proceed on more promising threads, thereby abandoning unfruitful searches
• Possible Problem: Reduce Diversity
• Extension: Stochastic Beam Search
  o Choose successor stochastically with a probability that is function of promise (value)...

Variant: Genetic Algorithm 15-811 Spring 2017: Lecture 4
Search in Speech

- Find Best Association between Word and Signal
- Efficiently Compose Words from Phones Using Dictionary
Search - continuous speech

- Compose Sentences from Words Using Language Model

Word 1

Word 2

Word 3
Viterbi Alignment
Viterbi Alignment
Genetic Algorithm

- Like Stochastic Beam Search, choose successor stochastically based on a function of value.
- This time, however, it combines two parent nodes to generate successor.
- Terminology in analogy to genetics.
**Genetic Algorithm**

- Start with $k$ randomly generated states
- Each *individual* state is represented by a string over a finite alphabet
- Production of *next generation* of states:
  - Each State is rated by a *fitness function*.
  - Probability of being chosen for reproduction is proportional to fitness score
  - Two pairs are chosen according to probability
  - Choose random cross-over point in string
  - Recombine
GA on 8-Queens Problem

24748552 → 32752411 → 32748552
32752411 → 24748552 → 24752411
24415124 → 32752411 → 32752124
32543213 → 24415124 → 24415411

Chessboard configurations:
Local Search – Issues

• Puzzles are great, But the real world is more complicated!

• Continuous Spaces:
  o Discretize States and Neighborhoods, or
  o Formulate “Improvement” by continuous values
  o Compute Gradient according to Objective
  o → Neural Networks

• States, Actions, Percepts.. may not be certain:
  → Partially Observable problems