Machine Learning

Greedy Local Search
Local search algorithms

• In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

• State space = set of "complete" configurations

• Find configuration satisfying constraints, e.g., n-queens

• In such cases, we can use local search algorithms (e.g. hill-climbing)

• keep a single "current" state, try to improve it
Hill-climbing (greedy local search)

\[
\text{find } x_{\text{max}} = \arg \max_{x \in X} (f(x))
\]
Greedy local search needs

- A “successor” function
  Says what states I can reach from the current one.
  Often implicitly a distance measure.

- An objective (error) function
  Tells me how good a state is

- Enough memory to hold
  The best state found so far
  The current state
  The state it’s considering moving to
Hill-climbing search

• "Like climbing Everest in thick fog 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
Hill-climbing (greedy local search)

"Like climbing Everest in thick fog with amnesia"

Bryan Pardo, Machine Learning: EECS 349 Fall 2009
Hill-climbing (greedy local search)

It is easy to get stuck in local maxima

Bryan Pardo, Machine Learning: EECS 349 Fall 2009
Example: $n$-queens

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.
Greedy local search needs

- A “successor” (distance?) function
  Any board position that is reachable by moving one queen in her column.

- An optimality (error?) measure
  How many queen pairs can attack each other?
Hill-climbing search: 8-queens problem

- \( h = 17 \)

- \( h \) = number of pairs of queens that are attacking each other, either directly or indirectly.
Hill-climbing search: 8-queens problem

- A local minimum with $h = 1$
Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but **gradually decrease** their frequency

```plaintext
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
        schedule, a mapping from time to "temperature"
local variables: current, a node
                next, a node
                T, a "temperature" controlling prob. of downward steps

current ← MAKE-NODE(INITIAL-STATE[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] - VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability e^ΔE/T
```
Properties of simulated annealing

• One can prove: If $T$ decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

• Widely used in VLSI layout, airline scheduling, etc
Local beam search

- Keep track of \( k \) states rather than just one
- Start with \( k \) randomly generated states
- At each iteration, all the successors of all \( k \) states are generated
- If any one is a goal state, stop; else select the \( k \) best successors from the complete list and repeat.
Let’s look at a demo

INSERT DEMO HERE