Search Methods

- **BFS**
  - Tries all possible partial assignments; like brute force search
  - All solutions are on deepest level of tree

- **DFS**
  - Cannot catch obvious violated constraints in partial assignments due to atomic state representation

- **Backtracking Search**
  - Fix order of variables
  - Check constraints as you go with “incremental goal test”
Backtracking Search Improvements: Filtering

- **Forward checking**: Cross off values that violate a constraint when added to the existing assignment
  - Checks all arcs into the assigned variable

- **Arc Consistency**: constraint propagation to ensure all arcs are consistent
  - Any time a variable X loses a value, all arcs into X need to be rechecked

- **K-Consistency**: any consistent assignment to k-1 variables can extend to kth node
  - Strong K-Consistency: ensuring 1-consistency, 2-consistency, 3-consistency, ..., k-consistency
Backtracking Search Improvements: Ordering

- **Variable Ordering: Minimum remaining values (MRV)**
  - Choose the variable with the fewest legal values left in its domain
  - Tie-break using the variable involved in most constraints
  - “Fail-fast” to prune search tree

- **Value Ordering: Least Constraining Value (LCV)**
  - Choose the value that rules out the fewest values in the remaining variables
  - May require additional computation (forward-checking/AC3)
  - Leave highest flexibility for later variable assignments
Algorithm Pseudocode

function BACKTRACK(csp, assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
    for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
        if value is consistent with assignment then
            add \{var = value\} to assignment
            inferences ← INERENCE(csp, var, assignment)
            if inferences ≠ failure then
                add inferences to csp
                result ← BACKTRACK(csp, assignment)
                if result ≠ failure then return result
                remove inferences from csp
            remove \{var = value\} from assignment
    return failure
CS 188: Artificial Intelligence

Local search

Instructors: Angela Liu and Yanlai Yang

University of California, Berkeley

[These slides adapted from Stuart Russell and Dawn Song]
Local search algorithms

- In many optimization problems, *path* is irrelevant; the goal state *is* the solution.
- Then state space = set of “complete” configurations; find *configuration satisfying constraints*, e.g., n-queens problem; or, find *optimal configuration*, e.g., travelling salesperson problem.

- In such cases, can use *iterative improvement* algorithms: keep a single “current” state, try to improve it.
- Constant space, suitable for online as well as offline search.
- More or less unavoidable if the “state” is yourself (i.e., learning).
Hill Climbing

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
Heuristic for $n$-queens problem

- **Goal:** $n$ queens on board with no *conflicts*, i.e., no queen attacking another
- **States:** $n$ queens on board, one per column
- **Actions:** move a queen in its column
- **Heuristic value function:** number of conflicts

![Diagram showing transitions with heuristic values h = 5, h = 2, and h = 0.](image)
Hill-climbing algorithm

```plaintext
function HILL-CLIMBING(problem) returns a state
    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
```

“Like climbing Everest in thick fog with amnesia”
Global and local maxima

Random restarts
- find global optimum
- duh

Random sideways moves
- Escape from shoulders
- Loop forever on flat local maxima
Hill-climbing on the 8-queens problem

- **No sideways moves:**
  - Succeeds w/ prob. 0.14
  - Average number of moves per trial:
    - 4 when succeeding, 3 when getting stuck
  - Expected total number of moves needed:
    - $3(1-p)/p + 4 \approx 22$ moves

- **Allowing 100 sideways moves:**
  - Succeeds w/ prob. 0.94
  - Average number of moves per trial:
    - 21 when succeeding, 65 when getting stuck
  - Expected total number of moves needed:
    - $65(1-p)/p + 21 \approx 25$ moves

Moral: algorithms with knobs to twiddle are irritating
Simulated annealing

- Resembles the annealing process used to cool metals slowly to reach an ordered (low-energy) state

- Basic idea:
  - Allow “bad” moves occasionally, depending on “temperature”
  - High temperature => more bad moves allowed, shake the system out of its local minimum
  - Gradually reduce temperature according to some schedule
  - Sounds pretty flaky, doesn’t it?
Simulated annealing algorithm

function SIMULATED-ANNEALING(problem, schedule) returns a state

current ← 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

- **Theoretical guarantee:**
  - Stationary distribution (Boltzmann): \( P(x) \propto e^{E(x)/T} \)
  - If \( T \) decreased slowly enough, will converge to optimal state!

- **Proof sketch**
  - Consider two adjacent states \( x, y \) with \( E(y) > E(x) \) [high is good]
  - Assume \( x \rightarrow y \) and \( y \rightarrow x \) and outdegrees \( D(x) = D(y) = D \)
  - Let \( P(x), P(y) \) be the equilibrium occupancy probabilities at \( T \)
  - Let \( P(x \rightarrow y) \) be the probability that state \( x \) transitions to state \( y \)
Simulated Annealing

- Is this convergence an interesting guarantee?

- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - “Slowly enough” may mean exponentially slowly
  - Random restart hillclimbing also converges to optimal state...

- Simulated annealing and its relatives are a key workhorse in VLSI layout and other optimal configuration problems
Local beam search

Basic idea:

- $K$ copies of a local search algorithm, initialized randomly
- For each iteration
  - Generate ALL successors from $K$ current states
  - Choose best $K$ of these to be the new current states

Why is this different from $K$ local searches in parallel?

- The searches communicate! “Come over here, the grass is greener!”

What other well-known algorithm does this remind you of?

- Evolution!
Genetic algorithms use a natural selection metaphor

- Resample $K$ individuals at each step (selection) weighted by fitness function
- Combine by pairwise crossover operators, plus mutation to give variety
Example: N-Queens

- Does crossover make sense here?
- What would mutation be?
- What would a good fitness function be?
Local search in continuous spaces
Example: Siting airports in Romania

Place 3 airports to minimize the sum of squared distances from each city to its nearest airport

Airport locations
\[ x = (x_1, y_1), (x_2, y_2), (x_3, y_3) \]

City locations \((x_c, y_c)\)

\[ C_a = \text{cities closest to airport } a \]

Objective: minimize
\[ f(x) = \sum_a \sum_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2 \]
Handling a continuous state/action space

1. Discretize it!
   - Define a grid with increment $\delta$, use any of the discrete algorithms

2. Choose random perturbations to the state
   a. First-choice hill-climbing: keep trying until something improves the state
   b. Simulated annealing

3. Compute gradient of $f(x)$ analytically
Finding extrema in continuous space

- Gradient vector $\nabla f(\mathbf{x}) = (\partial f/\partial x_1, \partial f/\partial y_1, \partial f/\partial x_2, ...)^T$
- For the airports, $f(\mathbf{x}) = \sum_a \sum_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
- $\partial f/\partial x_1 = \sum_{c \in C_1} 2(x_1 - x_c)$
- At an extremum, $\nabla f(\mathbf{x}) = 0$
- Can sometimes solve in closed form: $x_1 = (\sum_{c \in C_1} x_c)/|C_1|$
- Is this a local or global minimum of $f$?
- Gradient descent: $\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$
  - Huge range of algorithms for finding extrema using gradients
Many configuration and optimization problems can be formulated as local search.

General families of algorithms:
- Hill-climbing, continuous optimization
- Simulated annealing (and other stochastic methods)
- Local beam search: multiple interaction searches
- Genetic algorithms: break and recombine states

Many machine learning algorithms are local searches.