Local Search and Optimization
CSE 473
Autumn 2014
Dan Weld
(Based on slides of Mausam, Padhraic Smyth, Stuart Russell, Rao Kambhampati, Raj Rao, ...)

Search thru State Space

What if Robot is Blind?
Moving into wall → noop

Conformant Planning

Moving into wall

Search thru State Space

- States
  - SETS of states
  - “Belief state”
- Operators
  - Move actions
- Initial State
  - Set of all states
- Goal State
  - Set of just goal state(s)

Move Right

- States
  - SETS of states
  - “Belief state”
- Operators
  - Move actions
- Initial State
  - Set of all states
- Goal State
  - Set of just goal states

Conformant Planning
Sterilizing surgical gear
Bowl feeder
<table>
<thead>
<tr>
<th>Move Down</th>
<th>Move Down</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>States</strong></td>
<td><strong>States</strong></td>
</tr>
<tr>
<td>- SETS of states</td>
<td>- SETS of states</td>
</tr>
<tr>
<td>- “Belief state”</td>
<td>- “Belief state”</td>
</tr>
<tr>
<td><strong>Operators</strong></td>
<td><strong>Operators</strong></td>
</tr>
<tr>
<td>- Move actions</td>
<td>- Move actions</td>
</tr>
<tr>
<td><strong>Initial State</strong></td>
<td><strong>Initial State</strong></td>
</tr>
<tr>
<td>- Set of all states</td>
<td>- Set of all states</td>
</tr>
<tr>
<td><strong>Goal State</strong></td>
<td><strong>Goal State</strong></td>
</tr>
<tr>
<td>- Set of just goal states</td>
<td>- Set of just goal states</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Move Right</th>
<th>Move Right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>States</strong></td>
<td><strong>States</strong></td>
</tr>
<tr>
<td>- SETS of states</td>
<td>- SETS of states</td>
</tr>
<tr>
<td>- “Belief state”</td>
<td>- “Belief state”</td>
</tr>
<tr>
<td><strong>Operators</strong></td>
<td><strong>Operators</strong></td>
</tr>
<tr>
<td>- Move actions</td>
<td>- Move actions</td>
</tr>
<tr>
<td><strong>Initial State</strong></td>
<td><strong>Initial State</strong></td>
</tr>
<tr>
<td>- Set of all states</td>
<td>- Set of all states</td>
</tr>
<tr>
<td><strong>Goal State</strong></td>
<td><strong>Goal State</strong></td>
</tr>
<tr>
<td>- Set of just goal states</td>
<td>- Set of just goal states</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Move Up</th>
<th>Move Up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>States</strong></td>
<td><strong>States</strong></td>
</tr>
<tr>
<td>- SETS of states</td>
<td>- SETS of states</td>
</tr>
<tr>
<td>- “Belief state”</td>
<td>- “Belief state”</td>
</tr>
<tr>
<td><strong>Operators</strong></td>
<td><strong>Operators</strong></td>
</tr>
<tr>
<td>- Move actions</td>
<td>- Move actions</td>
</tr>
<tr>
<td><strong>Initial State</strong></td>
<td><strong>Initial State</strong></td>
</tr>
<tr>
<td>- Set of all states</td>
<td>- Set of all states</td>
</tr>
<tr>
<td><strong>Goal State</strong></td>
<td><strong>Goal State</strong></td>
</tr>
<tr>
<td>- Set of just goal states</td>
<td>- Set of just goal states</td>
</tr>
</tbody>
</table>

Yay!
• States
  - SETS of states
  - “Belief state”

• Goal State
  - Set of just goal state(s)

Heuristics?
Relaxed Problem?
  - What if weren’t blind?
  - Max # moves from any state in belief state
Also… nonadmissible
  - Number of states in belief state

Outline
• Blind Search
• Heuristic Search
• Local search techniques and optimization
  - Hill-climbing++
  - Simulated annealing
  - Genetic algorithms
  - Gradient methods
• Constraint Satisfaction
• Adversarial Search

Goal State vs Path
• Previously: Search to find best path to goal
  - Systematic exploration of search space.

• Today: a state is solution to problem
  - for some problems path is irrelevant.
    - E.g., 8-queens

• Different algorithms can be used
  - Search
  - Local Search
  - Constraint Satisfaction

Local Search and Optimization
• Local search
  - Keep track of single current state
  - Move only to neighboring states
  - Ignore previous states, path taken

• Advantages:
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces

• “Pure optimization” problems
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
  - Does not quite fit into path-cost/goal-state formulation
  - Local search can do quite well on these problems.

Trivial Algorithms
• Random Sampling
  - Generate a state randomly

• Random Walk
  - Randomly pick a neighbor of the current state

• Why even mention these?
  - Both algorithms asymptotically complete.
**Hill-climbing (Greedy Local Search)**
review from last time

**function** HILL-CLIMBING{problem} return a state that is a local maximum

**input:** 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 search**

- "a loop that continuously moves towards increasing value"
  - terminates when a peak is reached
  - Aka greedy local search
- Value can be either
  - Objective function value
  - Heuristic function value (minimized)
- Hill climbing does not look ahead of the immediate neighbors
- Can randomly choose among the set of best successors
  - if multiple have the best value
- "climbing Mount Everest in a thick fog with amnesia"

---

**Example: n-queens**

- Put n queens on an n x n board with no two queens on the same row, column, or diagonal
  - Note different search space... all states have N queens

  ![n-queens example](image)

- Is it a satisfaction problem or optimization?

---

**Search Space Recap**

- State
  - All N queens on the board in some configuration
- Successor function
  - Move single queen to another square in same column.
- Example of a heuristic function h(n):
  - the # of queens-pairs that are attacking each other
  - (we want to minimize this)

---

**Hill-climbing search: 8-queens problem**

- Need heuristic function
  - Convert to an optimization problem
- h = number of pairs of queens attacking each other
- h = 17 for the above state

---

**Hill-climbing search: 8-queens problem**

- Is this a solution?
- What is h?
- Is any successor better?
Hill-climbing on 8-queens

- Randomly generated 8-queens starting states...
- 14% the time it solves the problem
- 86% of the time it get stuck at a local minimum
- However...
  - Takes only 4 steps on average when it succeeds
  - And 3 on average when it gets stuck
  - (for a state space with $8^8 \approx 17$ million states)

Escaping Shoulders: Sideways Move

- If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
  - Must limit the number of possible sideways moves to avoid infinite loops
- For 8-queens
  - Allow sideways moves with limit of 100
  - Raises percentage of problems solved from 14 to 94%
- However...
  - 21 steps for every successful solution
  - 64 for each failure

Escaping Local Optima - Enforced Hill Climbing

- Perform breadth first search from a local optima
  - to find the next state with better h function
- Typically,
  - prolonged periods of exhaustive search
  - bridged by relatively quick periods of hill-climbing
- Middle ground b/w local and systematic search

Tabu Search

- Prevent returning quickly to the same state
- Keep fixed length queue (“tabu list”)
- Add most recent state to queue; drop oldest
- Never make a step that is currently “tabu”
- Properties:
  - As the size of the tabu list grows, hill-climbing will asymptotically become “non-redundant” (won’t look at the same state twice)
  - In practice, a reasonable sized tabu list (say 100 or so) improves the performance of hill climbing in many problems

Hill Climbing: stochastic variations

→When the state-space landscape has local minima, any search that moves only in the greedy direction cannot be complete

→Random walk, on the other hand, is asymptotically complete

Idea: Combine random walk & greedy hill-climbing
Hill-climbing with random restarts

• If at first you don’t succeed, try, try again!
• Different variations
  – For each restart: run until termination vs. run for a fixed time
  – Run a fixed number of restarts or run indefinitely
• Analysis
  – Say each search has probability $p$ of success
    • E.g., for 8-queens, $p = 0.14$ with no sideways moves
  – Expected number of restarts?
  – Expected number of steps taken?

Restarts

<table>
<thead>
<tr>
<th>Restarts</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success?</td>
<td>14%</td>
<td>36%</td>
<td>53%</td>
<td>74%</td>
<td>92%</td>
<td>99%</td>
<td>99.994%</td>
</tr>
</tbody>
</table>

Hill-climbing with random walk

• At each step do one of the two
  – Greedy: With prob $p$ move to the neighbor with largest value
  – Random: With prob $1-p$ move to a random neighbor

Hill-climbing with both

• At each step do one of the three
  – Greedy: move to the neighbor with largest value
  – Random Walk: move to a random neighbor
  – Random Restart: Resample a new current state

Simulated Annealing

• Simulated Annealing = physics inspired twist on random walk
• Basic ideas:
  – like hill-climbing identify the quality of the local improvements
  – instead of picking the best move, pick one randomly
  – say the change in objective function is $\delta$
  – if $\delta$ is positive, then move to that state
  – otherwise:
    • move to this state with probability proportional to $\delta$
    • thus: worse moves (very large negative $\delta$) are executed less often
    • however, there is always a chance of escaping from local maxima
    – over time, make it less likely to accept locally bad moves
    – (Can also make the size of the move random as well, i.e., allow “large” steps in state space)

Physical Interpretation of Simulated Annealing

• A Physical Analogy:
  • Imagine letting a ball roll downhill on the function surface
  • Now shake the surface, while the ball rolls,
  • Gradually reducing the amount of shaking
Physical Interpretation of Simulated Annealing

• A Physical Analogy:
  • Imagine letting a ball roll downhill on the function surface
  • Now shake the surface, while the ball rolls,
  • Gradually reducing the amount of shaking

Temperature T

• high T: probability of “locally bad” move is higher
• low T: probability of “locally bad” move is lower
• typically, T is decreased as the algorithm runs longer
• i.e., there is a “temperature schedule”

Simulated annealing

function SIMULATED-ANNEALING(problem, schedule) return a solution state
input problem, a problem
  schedule, a mapping from time to temperature
local variables: current, a node.
  next, a node.
  T, a “temperature” controlling the 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
  ∆E ← VALUE(next) - VALUE(current)
  if ∆E > 0 then current ← next
  else current ← next only with probability e^(-∆E/T)

Simulated Annealing in Practice

  • theoretically will always find the global optimum
– Other applications: Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, ...

  – useful for some problems, but can be very slow
  • slowness comes about because T must be decreased very gradually to retain optimality

Local beam search

• Idea: Keeping only one node in memory is an extreme reaction to memory problems.

  • Keep track of k states instead of one
  – Initially: k randomly selected states
  – Next: determine all successors of k states
  – If any of successors is goal → finished
  – Else select k best from successors and repeat
Local Beam Search (contd)

- Not the same as $k$ random-start searches run in parallel!
- Searches that find good states recruit other searches to join them
- Problem: quite often, all $k$ states end up on same local hill
- Idea: Stochastic beam search
  - Choose $k$ successors randomly, biased towards good ones
- Observe the close analogy to natural selection!

Genetic algorithms

- Twist on Local Search: successor is generated by combining two parent states
- A state is represented as a string over a finite alphabet (e.g., binary)
  - 8-queens
    - State = position of 8 queens each in a column
- Start with $k$ randomly generated states (population)
- Evaluation function (fitness function):
  - Higher values for better states.
  - Opposite to heuristic function, e.g., # non-attacking pairs in 8-queens
- Produce the next generation of states by "simulated evolution"
  - Random selection
  - Crossover
  - Random mutation

Can we evolve 8-queens through genetic algorithms?

Evolving 8-queens

- String representation 16257483
- Can we evolve 8-queens through genetic algorithms?
- Sorry! Wrong queens
Gene/calgorithms

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29%
  etc

4 states for 8-queens problem
2 pairs of 2 states randomly selected based on fitness. Random crossover points selected

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29%
  etc

Initial Population Fitness Factor Selective Crossover Mutation
4 states for 8-queens problem New states after crossover Random mutation applied

5.1

Comments on Genetic Algorithms

- Genetic algorithm is a variant of "stochastic beam search"

- Positive points
  - Random exploration can find solutions that local search can’t
  - (via crossover primarily)
  - Appealing connection to human evolution
  - "Neural" networks, and "genetic" algorithms are metaphors!

- Negative points
  - Large number of "tunable" parameters
    - Difficult to replicate performance from one problem to another
  - Lack of good empirical studies comparing to simpler methods
  - Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general

5.2

Optimization of Continuous Functions

- Discretization
  - use hill-climbing

- Gradient descent
  - make a move in the direction of the gradient
  - gradients: closed form or empirical

5.3

Gradient Descent

Assume we have a continuous function: \( f(x_1, x_2, \ldots, x_n) \)
and we want minimize over continuous variables \( X_1, X_2, \ldots, X_n \)

1. Compute the gradients for all \( \frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_i} \)
2. Take a small step downhill in the direction of the gradient:
   \[ x_i \leftarrow x_i - \lambda \frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_i} \]
3. Repeat.
   - How to select \( \lambda \)
     - Line search: successively double
     - until \( f \) starts to increase again

5.4