Local Search

CISC481/681, Lecture #5
Ben Carterette

With material adapted from Prof. Marie desJardins (UMBC) and Kathy McCoy.

Local Search

- A “lightweight” search strategy
  - Do not store any information about the path from the start state to the current node
  - (Other than what is already stored in the state)
  - Solves memory problems that Tree-Search and Graph-Search have with storing expanded nodes
- Applicable to problems where the path doesn’t matter
  - 8 queens, sudoku, many optimization problems
- Basic idea: start from some state and explore states around it looking for an improvement

Why Local Search?

Example: n-queens

- Place n queens on an n by n board such that no queen can attack any other queen

- The order the queens are placed on the board doesn’t matter—only the final configuration

Local Search Methods

- Hill climbing
- Simulated annealing
- Beam search/Local beam search
- Genetic algorithms
- Continuous search (gradient descent, Newton-Raphson)
Hill Climbing

- Define an evaluation or **objective function** $f(n)$
  - $f(n)$ takes a state as input and outputs a real number
  - The goal is to maximize (or minimize) $f(n)$
- From the current state, move to a state that increases the value of $f(n)$
- If no state increases the value of $f(n)$, stop the search
- Performance:
  - Complete? Optimal? Complexity?

State Space Landscape

2-D State Space Landscape

Example: 8-queens

$f(n) = \text{number of pairs of queens that are attacking each other; minimize } f(n)$

$f(n) = 17 \text{ for this state}$
Example: 8-queens

\[ f(n) = \text{number of pairs of queens that are attacking each other} \]
\[ f(0) = 1 \text{ for this state—which is a local minimum} \]

Hill Climbing Variations

- Allow “sideways” moves: don’t get trapped on a shoulder
- Stochastic hill climbing: if there is more than one uphill move, choose one randomly
- First-choice hill climbing: take the first uphill move found (when there are very many)
- Random restart: start the search over from a new state
  - Complete with probability approaching 1

Simulated Annealing

- Like hill climbing, but allowing occasional “bad” moves
  - Goal is to escape local maxima
  - Gradually decrease the frequency of bad moves as the search continues
- Analogous to annealing, a metallurgical process of heating a metal then allowing it to cool to temper or harden it

Simulated Annealing Algorithm

```plaintext
function SIMULATED-HELSENING( 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

1. current \( \leftarrow \) MAKE-NODE(INITIAL-STATES(problem))
2. for \( t \leftarrow 1 \) to \( \infty \) do
   3. \( T \leftarrow \text{schedule}(t) \)
   4. if \( T = 0 \) then return current
   5. next \( \leftarrow \) a randomly selected successor of current
   6. \( \Delta E \leftarrow \text{VALUE}(next) - \text{VALUE}(current) \)
   7. if \( \Delta E > 0 \) then current \( \leftarrow \) next
   8. else current \( \leftarrow \) next only with probability \( e^{\Delta E/T} \)
```

Copyright © Ben Carterette
**SA Probability of Descending**

![Graph showing the probability of descending across different values of a variable.]

**Local Beam Search**

- Beam search is like best-first search for finding a path, but it only keeps the $k$ best nodes.
- Local beam search is a local search algorithm (it does not store path information) that keeps track of $k$ nodes instead of just one.
- Algorithm:
  - Generate $k$ states randomly
  - Loop until reaching a maximum $f(n)$:
    - Generate all successors of all $k$ states
    - Take the $k$ states with greatest $f(n)$

**Genetic Algorithms**

- Like local beam search, except generate successor states by combining two parent states.
- Algorithm:
  - Start with $k$ random states ranked by “fitness”
  - Loop until done:
    - Combine pairs of states to generate children
    - “Mutate” children with some probability

**Genetic Algorithm for 8-queens**

- Fitness function = number of non-attacking pairs of queens

![Diagram showing the process of selecting parents, generating offspring, and mutating them.]

Copyright © Ben Carteret 2009
GeneTc
Algorithm
for
8-queens

Copyright © Ben Carteret

Continuously
Search

• All search strategies so far have dealt with
discrete state spaces
• The real world is mostly continuous
  — i.e. a state is represented as a vector of continuous
    variables \(x_1, x_2, \ldots, x_n\)
  — \(f(x_1, x_2, \ldots, x_n)\) is a
    continuous function
• Algorithms for searching continuous spaces go
  back a long way
  — At least to Isaac Newton in the 1670s
  — Calculus required to give them formal grounding

Copyright © Ben Carteret

Newton-Raphson (1600s)

• Find the roots of a differentiable function \(g(x)\)
  — i.e. \(x\) such that \(g(x) = 0\)
• Algorithm:
  — Set \(x := an\ initial\ guess\ x_0\)
  — Iterate until \(x\) is fixed:
    • \(x := x - g(x)/g'(x)\)
• Like hill climbing, can get stuck in local
  maxima, require multiple restarts, etc

Copyright © Ben Carteret

Copyright © Ben Carteret

Gradient Descent

• Use the rate of change in the objective
  function to determine where to go to next

Copyright © Ben Carteret
Gradient Descent

• \( f(x_1, x_2, ..., x_n) \) is a function of continuous variables \( x_1, ..., x_n \)
• The gradient of \( f \) is the vector of its partial derivatives
  \[
  \nabla f(x) = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \right)
  \]
• Algorithm:
  – Set vector \( x = x_0 \)
  – Loop until no change in \( x \)
  \[
  x := x + \alpha \nabla f(x)
  \]

Gradient Descent Intuition

• When the surface defined by \( f \) is steeper, move to a further point
• When the surface defined by \( f \) is less steep, stay nearby
• “Learning rate” \( \alpha \) is a parameter that determines how near or far to move

Search So Far

Search problems

Blind search (cost function)

Heuristic search:
  best-first and \( A^* \) (evaluation function, heuristic function)

Construction of heuristics

Variants of \( A^* \)

Local search (objective function, fitness function)