CS 771
Artificial Intelligence

Local Search Algorithms
Outline

• Hill-climbing search
  – Gradient Ascent/Descent in continuous spaces
• Simulated annealing search
• Local beam search
• Genetic algorithms
• Linear Programming
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
• keep a single "current" state, try to improve it.
• Very memory efficient (only remember current state)
Examples

• Integrated circuit design
• Factory floor layout
• Job-shop scheduling
• Telecommunication network optimization
• Portfolio management
Local search algorithms

- Operates using a single current node (rather than multiple paths)
  - In each step moves to a suitable neighbor node
  - Typically path followed by search are not maintained

- Advantages
  - Requires very little memory (usually a constant amount)
  - Can often find reasonable solutions in large or infinite state space where systematic algorithms are unsuitable
Local search algorithms

- In addition to finding goals, local search algorithms are useful in solving pure optimization problems
  - Here aim is to find the best state according to an objective function
  - Each state is associated with an objective function value
  - Goal is to find, for example, the state corresponding to maximum objective function value (global maximum)
Local search algorithms

• Completeness
  – A complete local search algorithm always finds a goal if one exists

• Optimality
  – An optimal local search algorithm always finds a global maximum/minimum
Example: $n$-queens

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
Hill-climbing search

- Requires an **objective function**
  - keeps track of how far from goal
- Algorithm **does not** maintain a search tree
  - data structure for current node need to remember only state and value of the objective function
  - doesn’t look ahead beyond immediate neighbors of current state
- "Like climbing Everest in thick fog with amnesia"

```plaintext
function HILL-CLIMBING(problem) returns a state that is a local maximum

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

- Let $h$ be the number of pairs of queens that are attacking each other.
- Current state/configuration consists of placement of 8 queens as shown.
- Current configuration has $h=17$.
- A successor of a state are possible states generated by moving a single queen to another square in the same column.
  - $7 \times 8=56$ successors.
Hill-climbing search: 8-queens problem

• Let $h$ be the number of pairs of queens that are attacking each other
  • Each number indicates $h$ if we move a queen in its corresponding column
  • Current configuration has $h=17$
  • Local maxima: is a peak that is higher than each of its neighboring states but lower than global maximum
Hill-climbing search: 8-queens problem

- A local minima with $h=1$
- What can be done to get out of this minima?
Hill-climbing Difficulties

- Problem: depending on initial state, can get stuck in local maxima
Hill-climbing statistics for 8-queen

• Starting from a randomly generated 8-queen state
  – hill climbing gets stuck 86% of the time (solves only 14% of the problem)
  – works quickly: takes 4 steps on average when succeeds and 3 steps when it gets stuck

• If we allow bounded number of consecutive sideways moves when there is no uphill move
  – % of problem instances solved raises from 14% to 94%
  – success comes at a cost: takes 21 steps on average when succeeds and 64 for each failure
Variants of hill climbing

- **Stochastic hill climbing**
  - chooses at random from the among the uphill moves
  - Probability of selection varies with steepness of uphill move
  - usually converges more slowly than steepest descent

- **First-choice hill climbing**
  - Generate successor randomly until one is generated that is better than the current state

- **Random restart hill climbing**
  - Conducts a series of hill climbing searches from randomly generated initial states until a goal is found
Simulated annealing

• **Hill climbing**
  – Never moves downhill moves towards states with lower value
  – Guaranteed to be incomplete as can stuck to a local maximum

• **Purely random walk**
  – Chooses the successor state uniformly at random
  – Complete but extremely inefficient

• **Simulated annealing**
  – Combines both the above techniques in such a way to ensure completeness and efficiency
Simulated annealing

• Imagine the following task of getting a ping pong ball into a deepest crevice of a bumpy surface
  – If we let the ball roll it will come to rest at a local minima
  – If we shake the surface we can bounce the ball out of local minima
  – The trick is to shake just hard enough to bounce the ball out of local minima but not hard enough to dislodge it from global minima

• Simulated annealing solution
  – Starts by shaking hard (high temperature parameter $T$)
  – Then gradually reduce the intensity of shaking (lower temperature $T$)
Simulated annealing search

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

**inputs:** *problem*, a problem

    *schedule*, a mapping from time to “temperature”

**local variables:** *T*, a “temperature” controlling the probability of downward steps

1. current ← MAKE-NODE(*problem*.INITIAL-STATE)
2. for *t* ← 1 to ∞ do
   1. *T* ← schedule(*t*)
   2. if *T* = 0 then return current
   3. next ← a randomly selected successor of current
   4. Δ*E* ← next.VALUE − current.VALUE
   5. if Δ*E* > 0 then current ← next
   6. else current ← next only with probability *e*^{Δ*E/Τ}
Local beam search

- **Keeps k states as opposed to one state**
  - Begins with k randomly generated states
  - At each step all successors of all k states are generated
  - If any one is a goal, then algorithm stops
  - Otherwise, it selects the k best successors from the complete list and repeats

- **This is different from k random restart hill climbing**
  - In random restart search, each search process runs independently of others
  - In local beam search, useful information is passed among parallel search threads
  - Local beam search algorithm quickly abandons unfruitful searches and moves its resources to where the most progress is being made
Stochastic beam search

• Analogous to stochastic hill climbing
  – Instead of choosing the best $k$ from the pool of candidate successors
    stochastic beam search chooses $k$ successors at random
  – Probability of choosing a given successor being an increasing function of its value
  – Bears some resemblance to the process of natural selection
    • Successors (offsprings) of a state (organism) populate the next generation
      according to its value (fitness)
Genetic Algorithm

- A variant of stochastic beam search
  - In which successor states are generated by combining two parent states rather than by modifying a single state
- Solutions are encoded as a population of individuals (a string over a finite alphabet)
  - Bit strings or integers
- New generations are created from one or two individuals from previous generation
  - Selection
  - Mutations
  - Crossover (combination of two parents)
- Mutations represent random exploration
- A fitness function is used to evaluate individuals
Ranking by fitness
Two parents are selected with probability proportional to individual fitness values
Cross over

Exchange information
Mutation

Explore unknown
Genetic Algorithm Cycle
Crossover vs Mutation

• **Crossover**
  – Early states are diverse (explores state broadly)
  – Later stages are more similar (fine tunes in small region)
  – Similar to Simulated Annealing

• **Mutation**
  – Diverting away from a good solution is possible
  – Explores untapped part of search space

• To hit the optimum you often need a ‘lucky’ mutation
Fitness function: number of non-attacking pairs of queens (min = 0, max = $8 \times 7/2 = 28$)

- P(child) = $24/(24+23+20+11) = 31\%$
- P(child) = $23/(24+23+20+11) = 29\%$ etc
Gradient descent

- This is a technique for finding the local minimum of a multivariate function

- Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$
  - suppose for any $x \in \mathbb{R}^n$, the function value $f(x)$ at $x = (x_1, \ldots, x_n)$ is given
  - in which direction $\nu$ should we go next from $x$ so that at $x + \nu$, $f(x + \nu) \leq f(x)$?

- Consider Taylor series expansion of $f(x)$ around $x$:
  $$f(x + \nu) \approx f(x) + \nabla f(x)^T \nu$$
  - where, $\nabla f(x) = \left( \frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n} \right)$ is the gradient of $f$ at $x$
  - then, $f(x) - f(x + \nu) \approx -\nabla f(x)^T \nu$
  - therefore, $f$ will reduce maximally locally at $x$ in the direction $\nu = -\nabla f(x)$
Gradient descent

**Input**: stepsize $\eta > 0$, tolerance parameter $\epsilon > 0$

1. Start at some point $x_0 \in \mathbb{R}^n$
2. For $i \geq 1$, repeat until $\|x_{i+1} - x_i\| \leq \epsilon$
   1. $x_{i+1} = x_i - \eta \nabla f(x_i)$

How do we find the best value of $\eta$?
Line Search

• Gradient descent requires to choose a step size

• Line search picks a direction, $v$, (say the gradient direction) and searches along that direction for the optimal step:

$$\eta^* = \arg\min \ f(x + \eta v)$$

• Repeated doubling can be used to effectively search for the optimal step

$\eta \rightarrow 2\eta \rightarrow 4\eta \rightarrow 8\eta$ (until cost increases)
Gradient ascent

- Used for finding local **maximum** of a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$
- $f$ will increase maximally locally at $x$ in the direction $v = \nabla f(x)$

**Input:** stepsise $\eta > 0$, tolerance parameter $\epsilon > 0$

1. Start at some point $x_0 \in \mathbb{R}^n$
2. For $i \geq 1$, repeat until $\|x_{i+1} - x_i\| \leq \epsilon$
   - $x_{i+1} = x_i + \eta \nabla f(x_i)$