Randomized Optimization

These slides were assembled by Byron Boots, with grateful acknowledgement of the many others who made their course materials freely available online. Feel free to reuse or adapt these slides for your own academic purposes, provided that you include proper attribution. Please send comments and corrections to Eric.
Outline

- Local Search: Hill Climbing
- Escaping Local Maxima: Simulated Annealing
- Genetic Algorithms
Local search and optimization

• Local search:
  • Use single current state and move to neighboring states.

• Idea: start with an initial guess at a solution and incrementally improve it until it *is* a solution

• Advantages:
  • Uses very little memory
  • Find often reasonable (if not optimal) solutions in large or infinite state spaces.

• Useful for pure optimization problems.
  • Find or approximate best state according to some objective function
  • Optimal if the space to be searched is convex
Hill climbing on a surface of states

f(s): Estimate of distance from a peak (smaller is better)

OR: Height Defined by Evaluation Function (greater is better)
Hill-climbing search

I. While (∃ uphill points):
   • Move in the direction of increasing evaluation function $f$

II. Let $s_{next} = \arg \max_s f(s)$, $s$ a successor state to the current state $n$
   • If $f(n) < f(s)$ then move to $s$
   • Otherwise halt at $n$

• Properties:
  • Terminates when a peak is reached.
  • Does not look ahead of the immediate neighbors of the current state.
  • Chooses randomly among the set of best successors, if there is more than one.
  • Doesn’t backtrack, since it doesn’t remember where it’s been

• a.k.a. greedy local search

"Like climbing Everest in thick fog with amnesia"
Hill-climbing Example: $n$-queens

- $n$-queens problem: Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.

- Good heuristic: $h = \text{number of pairs of queens that are attacking each other}$

![Diagram showing the $n$-queens problem with different values of $h$.]
Hill-climbing example: 8-queens

A state with $h=17$ and the $h$-value for each possible successor

A local minimum of $h$ in the 8-queens state space ($h=1$).

$h = \text{number of pairs of queens that are attacking each other}$
Gradient ascent/descent

- Gradient descent procedure for finding the $\arg_x \min f(x)$
  - choose initial $x_0$ randomly
  - repeat
    - $x_{i+1} \leftarrow x_i - \eta f'(x_i)$
    - until the sequence $x_0, x_1, \ldots, x_i, x_{i+1}$ converges
- Step size $\eta$ (eta) is small (perhaps 0.1 or 0.05)
Gradient methods vs. Newton’s method

• A reminder of Newton’s method from Calculus:
  \[ x_{i+1} \leftarrow x_i - \eta f'(x_i) / f''(x_i) \]

• Newton’s method uses 2\textsuperscript{nd} order information (the second derivative, or, curvature) to take a more direct route to the minimum.

• The second-order information is more expensive to compute, but converges quicker.

Contour lines of a function
Gradient descent (green)
Newton’s method (red)


(this and previous slide from Eric Eaton)
Search Space features

- Objective function
- Global maximum
- Shoulder
- Local maximum
- "Flat" local maximum
- Current state
- State space
Drawbacks of hill climbing

- **Local Maxima**: peaks that aren’t the highest point in the space
- **Plateaus**: the space has a broad flat region that gives the search algorithm no direction (random walk)
- **Ridges**: dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.
The Shape of an Easy Problem (Convex)
The Shape of a Harder Problem
The Shape of a Yet Harder Problem
One Remedy to Drawbacks of Hill Climbing:  *Random Restart*
Simulated annealing (SA)

- **Annealing**: the process by which a metal cools and freezes into a minimum-energy crystalline structure (the annealing process)
- Conceptually SA exploits an analogy between annealing and the search for a **minimum** in a more general system.
Aside: Analogy-based Algorithms

Fany natural phenomenon you can think of, there will be at least one AI research group that will have a combinatorial optimization algorithm “based” on “analogies” and “similarities” with the phenomenon. Here’s the beginning of the list…

- Metal cooling annealing
- Evolution / Co-evolution / Sexual Reproduction
- Thermodynamics
- Societal Markets
- Management Hierarchies
- Ant/Insect Colonies
- Immune System
- Animal Behavior Conditioning
- Neuron / Brain Models
- Hill-climbing (okay, that’s a stretch…)
- Particle Physics
- Inability of Elephants to Play Chess
Simulated annealing (SA)

- **Annealing**: the process by which a metal cools and freezes into a minimum-energy crystalline structure (the annealing process)
- Conceptually SA exploits an analogy between annealing and the search for a minimum in a more general system.
- SA hill-climbing can avoid becoming trapped at local maxima.
- SA uses a random search that occasionally accepts changes that decrease objective function $f$.
- SA uses a control parameter $T$, which by analogy with the original application is known as the system "temperature."
- $T$ starts out high and gradually decreases toward 0.
Search using Simulated Annealing

• Simulated Annealing = hill-climbing with non-deterministic search

• 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)
More Details on Simulated Annealing

- Let's say there are 3 moves available, with changes in the objective function of $d_1 = -0.1$, $d_2 = 0.5$, $d_3 = -5$. (Let $T = 1$).
- Pick a move randomly:
  - If $d_2$ is picked, move there.
  - If $d_1$ or $d_3$ are picked, probability of move = $\exp(d/T)$
    - Move 1: $\text{prob}_1 = \exp(-0.1) = 0.9$, i.e., 90% of the time we will accept this move
    - Move 3: $\text{prob}_3 = \exp(-5) = 0.05$, i.e., 5% of the time we will accept this move

- $T$ = “temperature” parameter
  - 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 in Practice

  — theoretically will always find the global optimum (the best solution)

• useful for some problems, but can be very slow
  - slowness comes about because $T$ must be decreased very gradually to retain optimality
  — In practice how do we decide the rate at which to decrease $T$? (this is a practical problem with this method)
Local *beam search*

- **Keep track of $k$ states instead of one**
  - Initially: $k$ random states
  - Next: determine all successors of $k$ states
  - If any of successors is goal → finished
  - Else, select $k$ best from successors and repeat.

- **Major difference with random-restart search**
  - Information is shared among $k$ search threads.

- **Can suffer from lack of diversity.**
  - Stochastic variant: choose $k$ successors proportionally to state cost/fitness.
Genetic algorithms

1. Start with k random states (the *initial population*).

2. New states are generated by either
   1. “*Mutation*” of a single state or
   2. “*Sexual Reproduction*”: (combining) two *parent states* (selected proportionally to their *fitness*).

- Encoding used for the “*genome*” of an individual strongly affects the behavior of the search.
- Similar (in some ways) to stochastic beam search.


Representation: Strings of genes

- Each *chromosome* represents a possible solution
  - made up of a string of genes
- Each *gene* encodes some property of the solution
- There is a *fitness metric* on phenotypes of chromosomes
  - Evaluation of how well a solution with that set of properties solves the problem.
- **New generations are formed by**
  - Crossover: sexual reproduction
  - Mutation: asexual reproduction
Representation

Chromosomes could be:

- Bit strings $(0101 \ldots 1100)$
- Real numbers $(43.2 -33.1 \ldots 0.0 \ 89.2)$
- Permutations of elements $(E_{11} E_{3} E_{7} \ldots E_{1} E_{15})$
- Lists of rules $(R_{1} R_{2} R_{3} \ldots R_{22} R_{23})$
- Program elements (genetic programming)
- ... any data structure ...
Encoding of a Chromosome

- The chromosome encodes characteristics of the solution which it represents, often as a string of binary digits.
  
  - Chromosome 1: 1101100100110110
  - Chromosome 2: 1101111000011110

- Each set of bits represents some dimension of the solution.
Reproduction

- Reproduction by *crossover* selects genes from two parent chromosomes and creates two new offspring.
- To do this, randomly choose a crossover point (perhaps none).
- For child 1, everything before this point comes from the first parent and everything after from the second parent.
- Crossover looks like this ( | is the crossover point):

<table>
<thead>
<tr>
<th>Chromosome 1</th>
<th>11001</th>
<th>00100110110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>10011</td>
<td>11000011110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offspring 1</th>
<th>11001</th>
<th>11000011110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offspring 2</td>
<td>10011</td>
<td>00100110110</td>
</tr>
</tbody>
</table>
Mutation

- Mutation randomly changes genes in the new offspring.
- For binary encoding we can switch randomly chosen bits from 1 to 0 or from 0 to 1.

Original offspring  1101111000011110
Mutated offspring  1100111000001110
The Basic Genetic Algorithm

1. Generate random population of chromosomes

2. Until the end condition is met, create a new population by repeating following steps
   1. Evaluate the **fitness** of each chromosome
   2. Randomly select two parent chromosomes from a population with replacement, weighed by their fitness
   3. With probability $p_c$ cross over the parents to form a new offspring.
   4. With probability $p_m$ mutate new offspring at each position on the chromosome.
   5. Place new offspring in the new population

3. Return the best solution in current population
Genetic algorithms: 8-queens

Has the effect of "jumping" to a completely different new part of the search space (quite non-local)
Comments on genetic algorithms

- **Positive points**
  - Random exploration can find solutions that local search can’t
  - Appealing connection to evolution
  - Don’t need to know fitness function derivative
  - Good when evaluating a function is not that hard
  - **Highly parallelizable!**

- **Negative points**
  - Large number of “tunable” parameters
    - Difficult to replicate performance from one problem to another
  - Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general
    - Recently have come back again, e.g. to optimize neural network parameters!
  - Need to represent more (e.g. dependencies among variables) when function evaluation is expensive
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

We've discovered that evolution strategies (ES), an optimization technique that's been known for decades, rivals the performance of standard reinforcement learning (RL) techniques on modern RL benchmarks (e.g. Atari/MuJoCo), while overcoming many of RL's inconveniences.

In particular, ES is simpler to implement (there is no need for backpropagation), it is easier to scale in a distributed setting, it does not suffer in settings with sparse rewards, and has fewer hyperparameters. This outcome is surprising because ES resembles simple hill-climbing in a high-dimensional space based only on finite differences along a few random directions at each step.