Day 23A: Evolutionary Algorithms

Background

- Evolution (natural selection of variations of types) is a very powerful form of learning
    - Optimization (speed, energy usage, vision)
    - Discovery of novelty (vision, flight, cognition)
    - Organization of complexity
- Biology is a consistent source of analogies and ideas for weak method and theories:
    - Hill climbing, generate and test, neural networks, etc.
- Genetic algorithms aren't exactly like biology but are designed to emulate the key principles

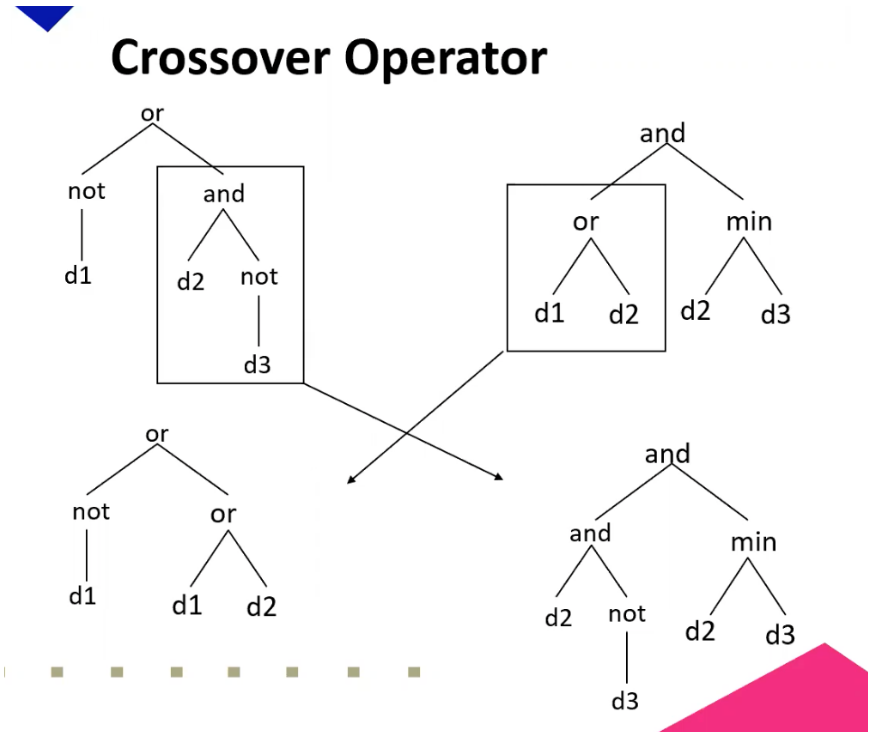
Implementation

- Basics of Evolution:
    1. Start with random population
    2. Evaluate fitness of each member, stop when:
        1. Some member meets a criteria (if we have a good criteria), or
        2. when no more progress is being made
    3. Normalize the fitness of each individual to get their relative fitness
    4. Generate a new population, with more fit individuals reproducing more often
    5. Go to Step 2

- Steady-state genetic algorithms: for each round, add a new individual and remove a different one, instead of modifying the entire population
- How to represent the population?
    - Set of genotypes, each of which has a value (commonly bits or real numbers), stored in some data structure
        - Bitstrings are traditionally popular, but really anything works
            - Gray Codes are often used so that small changes to the bitstring result in small changes to the output, so that small mutations actually are small mutation
        - Need to establish range and granularity
- Fitness function: a computable function applicable to all members of the population, which evaluates a member based on its genotypes and returns a number representing how fit that individual is
    - Needs to be tuned precisely to the task domain
        - Lots of human intuition and inductive bias
    - Slowest part of a genetic algorithm
    - Possibilities:
        - Mathematical function
        - Calculation over a large dataset
        - Built and test a model based on parameters
        - Construct a machine and test it
- How to do proportional selection?
    - The fittest members of the population should increase in number, while the least fit should decrease in number or die out.
    - "Roulette Wheel:" Each individual's likelihood of being a parent is equal to their fitness over the sum of all fitness values. Then, randomly pick parents (number depends on your strategy) from that distribution to create the next generation.

    - The average fitness of the population should grow over time.
- Loss of Diversity: when the population converges to one individual
- Mutation: a small change (like a bit of random noise or a grey code bit flip)
    - Don't want to mutate every individual every time
    - This results in exploring new possibilities
    - Bad mutation → won't reproduce, so no worries
    - Can do a gaussian mutation (where it usually mutates a small amount but sometimes mutates a lot)
- Would OnlyMutation work?
    - Like Parallel Hill Climbing (but slightly better when using gaussian mutation)
    - Mutation helps to break past local maxima, but isn't always a silver bullet (again, gaussian mutation helps)
- Cross-breeding: if two parents age each good at different things, maybe their kids will be good at both!
    - Variants:
        - One point crossover: First part of genome from one, the rest from the other
        - Two point crossover: Prefix and suffix from one, middle from another
        - Uniform crossover: Flip a coin for each bit
        - Knowledge-based crossover: Use knowledge of the task to intelligently breed viable children
- When doing evolutionary programming, a bunch of factors need to be ballanced:
    - Too much proportional reproduction raises average fitness at the expense of diversity and convergence
    - Introducing new ideas is needed to reach maximum fitness

    - Adding diversity with mutation and crossover can help (you can still get stuck on a local maximum), but too much can make the system noisy and lead to a drop in average fitness.
- Elitism: keep parents as-is some proportion of the time
- Premature convergence: any time the population converges before you want it to, or converges to the wrong answer
- Competing forces: Exploration vs. Exploitation
    - Need to be balanced by parameters and fiddling with representations
        - This is a form of inductive bias
- Genetic Programming (1992): using genetic algorithms to evolve code (in this case, LISP expressions)
    - Expressions can be of variable length
    - Crossover and evaluation are more complex
        - Running the program determines its fitness
    - Koza's attempt:
        - Genotypes are formal lisp expressions, from a constrained set of primitives
        - Fitness is done by running the program on a desired task
        - Fitness is done by slicing subtrees
    - Hacks to make GP simpler:
        - Protect functions from overflow: don't divide by 0, do abs before square root
    - Initial function generator biases for diversity
    - Limit to how complex and slow programs can get



- What's needed for GP?
  - Set of functions and their arity
    - + - * / sin cos etc.
    - if(x, then, else)
    - iflte(x, y, then, else)
  - Set of terminals:
    - Problem variables: x, y, z
    - R: ephemeral random number (generated when its first used and persisted)
  - Fitness function
- Past GP applications:
  - Pendulum control
  - Logical problems
  - Symbolic integration

    - Induction of sequences
    - Game learning
    - Classification
    - ...and more!
- GP was parallelized by Juille (grad student) & Pollack in 1995
    - Also included niche preservation
    - Hard problem for NNs that genetic programming can solve: intertwined spiral problem
- Big question: are genetic algorithms (or other bio-inspired things, like NNs) just fancy weak algorithms or does biology have deep secrets.