Evolutionary Search

Burr H. Settles
CS-540, UW-Madison
www.cs.wisc.edu/~cs540-1
Summer 2003

Announcements

- This week’s mailing list topic: think of a real-world problem where we could apply an optimization search
  - You may not repeat someone else’s answer!
  - What are the states?
  - What are the actions?
  - What is the objective function?

- Read Chapter 6 in *AI: A Modern Approach* for next time

Homework #1 Clarifications

- For problem 3, let’s say you have the initial state ABC, and are doing standard HC… what are the neighbors you need to consider?
  
  BC, AC, AB, ABCD, ABCE, ABCF

- So to do hill-climbing, you will generate all these states, score them all, and choose the best one (since we are maximizing the objective function)

Genetic Algorithms

- So far the optimization strategies we’ve discussed search for a *single* solution, one state at a time within a neighborhood

- Genetic algorithms (GAs) are a unique search approach that maintains a population of states, or individuals, which *evolves*
  - Also called evolutionary search

If you choose to do a paper instead of a programming project, 6 pages is a minimum… you may write more if you feel that there is too much material (but please no more than 10)

- Keep in mind each group will only have 15 minutes to present on the last week
- The presentations don’t have to go into as much detail as the paper, though
**Evolutionary Analogy**

- Consider a population of rabbits: some individuals are faster and smarter than others.
- Slower, dumber rabbits are likely to be caught and eaten by foxes.
- Fast, smart rabbits survive to do what rabbits to best: *make more rabbits!!*

**Evolutionary Analogy**

- The rabbits that survive breed with each other to generate offspring, which starts to mix up their genetic material.
  - Fast rabbits might breed with fast rabbits
  - Fast rabbits with slow rabbits
  - Smart with not-so-smart, etc…
- Furthermore, nature occasionally throws in a “wild hare” because genes can mutate.

**Evolutionary Analogy**

- In this analogy, an individual rabbit represents a solution to the problem (*i.e.* a single point in the state space).
  - The state description is its DNA, if you will.
- The foxes represent the problem constraints.
  - Solutions that do well are likely to survive.
- What we need to create are notions of natural selection, reproduction, and mutation.

**Core Elements of GAs**

- For selection, we use a fitness function to rank the individuals of the population.
- For reproduction, we define a crossover operator which takes state descriptions of individuals and combines them to create new ones.
  - What advantages does this present over local search?
- For mutation, we can merely choose individuals in the population and alter part of its state.

**Genetic Algorithm Example**

```
POPP = initialPopulation   // build a new population
repeat {                    // with every generation
  NEW_POP = empty
  for I = 1 to POP_SIZE {
    X = fit individual   // natural selection
    Y = fit individual   // natural selection
    CHILD = crossover(X,Y) // reproduction
    if small random probability then
      mutate(CHILD)   // mutation
    add CHILD to NEW_POP
  }
  POP = NEW_POP
} until solution found or enough time elapsed
return most fit individual in POP
```

**Genetic Algorithm Example**

- The previous algorithm completely replaces the population for each new generation… but we can allow individuals from older generations to live on.
- Reproduction here is only between two parents (as in nature), but we can allow for more!!
- The population size also is fixed… but we can have this vary from one generation to the next.
Genetic Algorithm Example

* Basically, there is no one GA, we can devise many variants of these 3 principles for nearly any problem!!

* Chapters 7 & 8 in How to Solve It: Modern Heuristics have a very thorough presentation of how to design genetic algorithms for particular problems.

Selection

- Deterministic selection
  1. Rank all the individuals using the fitness function and choose the best k to survive
  2. Replace the rest with offspring
     - Can lead fast convergence (and local optima)

- Stochastic selection
  - Instead of selecting the best k, we could select each individual in proportion to its relative fitness to the population
  - Slower to converge, but could lose good solutions

Reproduction

- The unique thing about GAs is the ability of solutions to inherit properties from other solutions in the population
- The basic way to perform a crossover operation is to splice together parts of the state description from each parent... for example, in SAT:

  \[
  \begin{array}{c}
  \text{parents} \\
  10011101 \\
  01001110 \\
  \end{array}
  \begin{array}{c}
  \text{children} \\
  10010110 \\
  01001101 \\
  \end{array}
  \]

 euroref
Reproduction for 8-Queens

- For the 8-queens problem, we could choose a crossover point with the same number of queens on either side
- What else could we do?

Reproduction for TSP

- For TSP, our individuals are \( n \)-city tours (e.g. 1-4-6-2-5-3 or 3-5-1-2-4-6 for 6 cities)
- Can we do a simple point-crossover like we could for SAT and 8-Queens?

Reproduction for TSP

- One option is to have an ordered master queue of all the cities in the problem: e.g. 1,2,3,4,5,6
- Each individual, then, is a code that corresponds to “dequeuing instructions” from this master queue
  - Numbers in the code are the relative position of the appropriate city left in the queue
  - These codes can then be mated with point-crossovers

Reproduction for TSP

- We can try something even simpler to try and conserve information from both parents
  - Pick an block of contiguous cities in the tour to pass from one parent to a child
  - Remove all the cities in block from the other parent
  - Add the remaining cities to the child in their preserved order, after the other block

Mutation

- There are also a variety of ways to mutate individuals in the population
- The first question to consider is what to mutate
  - Alter the most fit? Least fit? Random?
  - Mutate children only, or surviving parents as well?
  - How many to mutate?
- The second question is how to mutate
  - Totally arbitrarily?
  - Mutate to a better neighbor?

GAs and Creativity

- GAs can be thought of as a simultaneous, parallel hill-climbing search
  - The population as a whole is trying to converge to an optimal solution
- Because solutions can evolve from a variety of factors, without prodding from us as to “which direction to go” (as in local search), very novel problem solutions can be found discovered
GAs and Creativity

- Sensor-actuator networks (SANs) are structures that model connections between sensors and actuators (motors and muscles) in simple robots
- GAs can learn parameters for SANs that solve locomotion problems in a variety of ways

---

GAs and Emergent Intelligence

- So far, we've talked about GAs as a search strategy for a problem solving (in which case, there is an agent conducting the GA search)
- Recall from the second lecture about multi-agent environments
- Now consider a GA that evolves a population of agents!! Now, our GA population is a virtual multi-agent environment

---

GAs and Emergent Intelligence

- We could maintain a population of these agents, where each agent’s state (“DNA”) is its set of coefficients (α, β, γ, δ, etc.)
- Now let’s think about a GA for them:
  - What is our fitness function?
  - What is a good crossover?
  - How can we mutate them?

---

GAs and Emergent Intelligence

- Let’s say we want to “grow” agents to predict stock market trends
- Each agent might be some statistical function that maps a stock’s history to its predicted future performance:
  \[ (α \times \text{todaysPrice}) + (β \times \text{yesterdaysPrice}) + (γ \times \text{relativeValue}) + (δ \times 1\text{-monthStdDev}) + \ldots \]

---

GAs and Emergent Intelligence

- Over time, the population should converge on a population of individuals that reflect the current stock market trends
- But will the most fit individual become a universally good stock-picker?
- Perhaps not!
GAs and Emergent Intelligence

- It is possible that different agents in the population specialize to aspect of the task
  - Some agents predict well for the Fortune-500 stocks
  - Others predict well for sports companies
  - Still others pick non-profits well, etc...

* If there is no one universally intelligent agent in the population, perhaps we can let them all predict… or “vote” on a predictions

So our population of stock-pickers is a multi-agent environment… is it cooperative or competitive?

- This phenomenon is what we might refer to as emergent intelligence
  - No single agent in the environment is, taken individually, all that “intelligent”
  - Taken together, though, the entire population possesses a more global intelligence that emerges from its constituent agents

Genetic Programming

- Genetic programming is a field related to genetic algorithms (surprise!)

- Instead of maintaining and manipulating a population of strings, however, we use expression trees… the goal is to evolve programs

- Section 9.5 in Machine Learning provides a nice overview of this (slightly more advanced) topic

Expression trees are graphical, relational representations of functions or programs

Programming language compilers convert code to such trees before writing out machine-level instructions (CS 536)

For example:

\[
\sin(x) + \sqrt{x^2 + y}
\]

Genetic Programming

- Populations are initialized with randomized, well-formed expressions build up from:
  - Operators (e.g. +, sin, \times, etc.)
  - Terminals (x, y, 2, etc.)

- Fitness is evaluated on how well its encoded function/algorithm performs the task

- Crossover is applied by swapping subtrees in the parent expressions
Genetic Programming

J. Koza, Genetic programming: On the programming of computers by means of natural selection, MIT Press, 1992

- 300 random programs were initialized with primitives to solve block-stacking problems with the goal of spelling “UNIVERSAL”
- After 10 generations, a program evolved that solved all of 166 initial configurations

Views of Evolution

- The Lamarckian Theory
  - Popular in 1800s
  - An individual’s genetic makeup is altered as it learns through life experience
  - Today’s biological evidence contradicts this

- The Baldwin Effect
  - Learning has no effect on genetic makeup
  - However, ability to learn reduces the need for “hard-wired” functions
  - Therefore, individual learning allows for a more diverse gene pool (less hard-wiring) and more adaptable populations
  - Example: a new predator appears in an environment; individuals who can learn to avoid it live on, resulting in more adaptable gene pool

Last Thoughts on GAs

- Evolutionary algorithms are simulations of what we perceive happening in nature, but we don’t have to follow the laws of nature
  - Lamarckian GAs have been experimented with, and shown successful on some problems

  - Since we get to design the framework for the simulation, there is a wide margin for creative license in the framework we create!
    - Concepts of age/gender/politics?
    - Variety of fitness functions?

Summary of Search Strategies

- Partial Search
  - Look through state space for a goal from which a solution can be found
    - **Node**: state description
    - **Edge**: action that changes state at some cost
    - **Path**: sequence of actions that change from the start state to the goal state

- Uninformed Search: no domain information
  - Complete/optimal if costs uniform: BFS, IDS
  - Complete/optimal with costs: UCS
  - Not complete/optimal: DFS, DLS

- Informed Search: use heuristics to guide search
  - $g(n)$: cost from start to $n$
  - $h(n)$: estimates cost from $n$ to goal (heuristic)
  - $f(n) = g(n) + h(n)$: estimated cost of searching through $n$
  - Complete/optimal: $A^*$ = $h(n)$ is admissible
  - Not complete/optimal: Greedy, $A$
Summary of Search Strategies

- Optimization Search
  Look through solution space for better solutions to the problem
  - **Node**: complete solution
  - **Edge**: operator changes to a new solution
  - Can stop anytime
  - Well-suited for NP-Complete problems, optimization problems

Summary of Search Strategies

- Local Search
  Focus on a *local* part of the search space rather than exploring it all
  - Beam search limits the list of candidate states
  - Hill-climbing follows a single path of promising successor states
  - Solution heavily dependent on the initial state
  - Can get stuck in “local optima”

Summary of Search Strategies

- Escaping Local Optima
  Ways to avoid the traps into which local search methods tend to fall
  - Random Restarting
  - Simulated Annealing
  - Tabu Search

- Evolutionary Search
  Unique, non-local, parallel optimization search