Evolutionary Computation: A Novel Bioinspired Approach to Searching for Solutions

Blase B. Cindric
Mount Union College
November 14, 2006

Problem Solving Strategies

- For many problems, multiple strategies exist for finding a solution
- Some of these strategies are more efficient than others
  - faster
  - use fewer computing resources
- Goal: find the correct solution as efficiently as possible

Example: High/Low Guessing

- Think of a number between 1 and 1000….
- I'll try to guess your number
- You tell me if my guesses are too high or too low

Example: High/Low Guessing

- Strategy 1: sequential values starting with one
  - Will this eventually guess the correct value?
  - Is there a better strategy?
- Strategy 2: random guesses with no repeated guesses
  - Will this eventually work?
  - Better or worse than Strategy 1?

Example: High/Low Guessing

- Strategy 3: pick middle value of the range of values we haven't eliminated yet
  - Will this work every time?
  - Is there a better strategy?
- It can be proven mathematically that no strategy is better than strategy 3 overall
- How fast is this?
  - 1000 values → max. 10 guesses
  - 1,000,000 values → max. 20 guesses!

Some Problems Don't Have a Best Strategy (that is known)

- Consider a group of locations that must be visited all in one day
  - Delivery addresses for a bunch of packages that a UPS driver must drop off in a day, say
- What is the best order in which to visit all the locations (once) and return to the starting point?
    - "best" means smallest distance traveled to visit all the locations
- Called the *Traveling Salesman Problem*

No Known Best Strategy for Solving the TSP

- One solution strategy: write down all possible orders in which the locations could be visited
  - There are a huge number of these!!!
  - For 7 locations ➔ 720 orderings
  - For 10 locations ➔ 362,880 orderings
  - For n locations ➔ (n – 1)! Orderings
- Is there a faster strategy that guarantees a solution? NO!

How Can Problems be Solved?

- There are many real-world problems that are similar to the TSP, in that there is no known “best” solution strategy
- How are problems in the real world solved by living creatures?
- One method can be discovered by observing populations of organisms in nature….

Evolving Populations of Creatures

- Consider a population of some kind of living creature (a worm, an ant, a bear, …)
- The creatures must solve several problems in order to survive (find food, avoid predators, build nests, etc.)
- Some individuals in the population are better at these tasks than others
  - solutions are innate (instincts)
    - built into the animal's genetic code

Evolving Populations of Problem Solutions

- Represent each individual in a population by a genetic code sequence
- Rate each individual according to how well that creature solves the problem at hand
- We can say that certain genetic code sequences are “better” at solving the problem than others
- This is the basic idea behind Evolutionary Computation

Some Terminology

- Genome - an answer to a problem represented as a string of symbols
- Gene - one symbol in a genome
- Allele - the value of a gene
- Phenotype - the organism that is constructed from the genome
- Generation - all of the genomes in the population at any one instant of time

Some More Terminology

- Selection - choosing which genomes to use when reproducing via crossover
- Fitness - evaluation of a genome as to how good a solution it is for the problem
- Crossover Recombination - taking some genes from one parent genome and other genes from another parent in producing an offspring genome
- Mutation - changing the allele in one or more genes in an offspring genome

Evolutionary Algorithm

- Choose a representation of the answer to a problem as a string of symbols
  - For TSP, a list of location numbers, signifying the order in which the locations are visited
- Form a population of initial genomes, at random (guess at some possible answers)
- Evolve several generations of genomes, using fitter genomes as parents

Evolutionary Algorithm

```
genNum = 1;  // generation number
initialize the population;
evaluate and rank population's fitness;
repeat until (done) {
   genNum = genNum + 1;
   select parents for reproduction;
   crossover/recombine to form offspring;
   (possibly) mutate offspring;
   evaluate and rank population's fitness;
   decide which individuals survive;
}
```

A Simple Example

- A problem that doesn't need to be solved with the EC approach, but is simple enough for an explanatory example, is finding the square root of a number
- How would you find the square root of 2305? (without a calculator!)
- Guessing approach….
- Let's try the Evolutionary Algorithm here….

Genetic Representation of Problem

- Represent each candidate solution (answer) as a binary number
  - Possible alleles in each gene position → 0 or 1 only
- Examples:
  - $9_{10} = 8 + 1 = 000000001001_2$
  - $31_{10} = 16 + 8 + 4 + 2 + 1 = 000000011111_2$
  - $67_{10} = 64 + 2 + 1 = 000001000011_2$
  - $342_{10} = 000101010110_2$

Fitness Function

- Fitness of genome for this problem: how close is the value to the square root of 2305?
- Idea: if the genome represents the actual answer, if we square it and subtract from 2305, what should we obtain?
- Zero!
- Fitness = $2305 - (\text{genomeValue})^2$
- Smaller values are better (more fit)

Choosing Parents for Recombination

- The individuals that are most fit in the population are more likely to be chosen for reproduction
  - Think of lions in the wild, for example….
- But we won't always choose the two most fit individuals
- Mixing different genomes is key to the success of the EC strategy
  - Variety in the population

Fitness Proportional Recombination

- The chance of an individual being chosen for reproduction is proportional to its fitness
- Example: Population of 10 individuals
- Prob(most fit chosen) = 10 / 55 = 18.2 %
- Prob(second ranked chosen) = 9/55 = 16.4%
- …
- Prob(least fit chosen) = 1/55 = 1.8%
- Probabilities must add to 1
    - 1 + 2 + 3 + … + 10 = 55

Simulation of Crossover Recombination Operation

- Here is an applet that will allow us to try this out:

- http://www.mountunion.edu/~cindricbb/ev_prog/stepbystep/build/MUCJApplet.html

The Need for Mutation

- Crossover recombination is not enough in all cases
- Remember that the initial population is formed by completely random guessing
  - What if all initial genomes start with 0, but the correct answer starts with 1?
- Mutation introduces the possibility of varying the initial population
- Most mutations are detrimental
- A few can improve the fitness of an individual

Complete Applet for an EC Approach to the Square Root Problem

- http://www.mountunion.edu/~cindricbb/ev_prog/sqroot/build/MUCJApplet.html

EC Approach to Traveling Salesman Problem

- Representation: an ordered list of location numbers
  - Sequence of values represents the order in which locations will be visited
- Fitness: How close is total distance traveled to the best (smallest) possible distance?
  - We don't know the actual answer, but we know that smaller is better
  - Add up all distances between locations in the genome; smaller totals are better (more fit)
- Crossover and mutation are trickier here
  - Offspring must not contain the same allele in two different genes
  - No location number can be missing, or the offspring doesn't represent a problem solution

EC Approach to Traveling Salesman Problem

- Let's try it out!
- http://www.mountunion.edu/~cindricbb/ev_prog/TSPDemo/build/MUCJApplet.html

Example: Monkeys Randomly Typing the Works of Shakespeare

- Given enough time, a roomfull of monkeys randomly hitting keys on computer keyboards would eventually generate the complete works of Shakespeare, with no typos!
- How much time?
  - an almost uncountably large amount
- Let's examine a much smaller problem….

Example: Randomly Generating a Target Phrase

- How long would it take to randomly generate the 14-letter phrase “Purple Raiders”?
- Define the alphabet to be capital letters and a blank space
  - 27 possible characters for each letter position
- Think: how likely is this to occur all at once, at random?

Example: Randomly Generating a Target Phrase

- 1st letter of phrase
  - 27 possible choices
  - chance of getting a ‘P’: 1 in 27
- 2nd letter
  - 27 possible choices
  - chance of getting ‘PU’: 1 in 27x27 = 1 in 27^2
- 3rd letter
  - chance of getting ‘PUR’: 1 in 27^3

Example: Randomly Generating a Target Phrase

- Our target phrase has 14 letters
  - chance of generating ‘PURPLE RAIDERS’ is:
    1 in 27^{14} > 1 in 10^{20} = 1 in 100 billion billion
- This is very unlikely (!)
  - One would expect to hit the 6-number lottery almost 3 times before generating that phrase at random!
- What can an evolutionary approach do with this problem?

EC Approach to Generating a Target Phrase

- Let's try an approach with no crossover, and a 10% chance of a random mutation in any letter position:

http://www.mountunion.edu/~cindricbb/ev_prog/monkeytyping/build/MUCJApplet.html

Some EC Information on the World-Wide Web

- http://www.aaai.org/AITopics/html/genalg.html

References

- Mitchell, Melanie. An Introduction to Genetic Algorithms. MIT Press, 1998.
- Olariu, Stephan and Albert Zomaya, ed. Handbook of Bioinspired Algorithms and Applications. Chapman & Hall, 2006.
- Michalewicz, Zbigniew and David B. Fogel. How To Solve It: Modern Heuristics. Springer, 2002.

Evolutionary Computation: A Novel Bioinspired Approach to Searching for Solutions

Blase B. Cindric
Mount Union College
November 14, 2006