Evolutionary Computation

Evolutionary algorithms
Genetic algorithms
Genetic programming
Artificial life ("alife")

Some References

Zbigniew Michalewicz
Genetic Algorithms + Data Structures = Evolution Programs
3rd edition, Springer-Verlag, 1993

David Goldberg

Historical

1966, Fogel, Owens, and Walsh, Artificial Intelligence through Simulated Evolution, John Wiley & Sons

Looked at derivation of
- finite-state machines
- controllers
- data reduction through successive mutations

Among Mr. Holland’s Opuses

1975, John H. Holland
Adaptation in natural and artificial systems, MIT Press
- Focus was on natural systems, simulation
- Introduced current genetic algorithm idea
- Mostly theory, some applications to:
  - game-playing
  - search programs


See http://www.pscs.umich.edu/jhhfest/jhh-pub.html for more.
Mr. Holland’s Progeny

- David Goldberg
- Stephanie Forrest
- Kenneth DeJong
- Melanie Mitchell
- John Koza
- many others
- See [http://www.pscs.umich.edu/jhhfest/schedule-closed.html](http://www.pscs.umich.edu/jhhfest/schedule-closed.html) for Festschrift papers.

Today

Genetic Algorithms

- An approach to difficult optimization problems (TSP, etc.)
- Heuristic, not guaranteed to find true optima
- Finds good approximations fast

GA Applications

- Hundreds, if not thousands
- All manner of optimization problems in engineering, science, finance, etc.
- Application to neural networks:
  - Evolve the structure and/or weights of a neural network, rather than train it.

Principles of Natural Selection

- Concentrate on population rather than a single individual
- Individuals that are fit enough to survive will reproduce
- Create new individuals from existing ones
  - Crossover
  - Mutation

Genetic Flow Diagram

Population
Selection criterion
Subset of Population
Operators

until an individual exists satisfying performance criterion or resources are exhausted
Common Operators

- **Copy (aka Reproduce):** An individual from the current generation is copied into the next generation.
- **Crossover:** Two (or more) individuals from the current generation are used to form an individual in the next generation.
- **Mutate:** A single individual from the current generation is mutated to form an individual in the next generation.

Individuals

- Individuals are represented by their genome, or **genotype**.
- The genotype may be the “program” for the actual individual, or **phenotype**.

Genetic Algorithm Example

- Consider the “subset sum” problem:
  - Given a set of integers \( S \) and a target value \( T \), find a subset of \( S \) that the maximum sum without exceeding \( T \).

Genetic Approach

- As the genome, use a bit-vector.
- There is one bit for each element in the set \( S \).
- The bit is 1 iff the element is in the subset.

Example

- \( S = \{19, 23, 35, 52, 61, 68, 76, 84, 92\} \)
- \( T = 200 \)
- Genome is an element of \( \{0, 1\}^9 \)
- Possible individuals:
  
  \[
  \begin{align*}
  &010001001, \text{ sum}(23, 68, 92) = 183 \\
  &101000010, \text{ sum}(19, 35, 84) = 138
  \end{align*}
  \]
What can we try to produce more fit individuals?

- **Mutation**: change a random bit:
  
  \[ 101000010, \text{sum}(19, 35, 84) = 138 \]
  
  \[ 11100010, \text{sum}(19, 23, 35, 84) = 161 \]

---

Fatals Mutations

- Note that a mutation could be “fatal”, resulting in a totally unfit individual. The “carcass” of this individual could still be in the next generation, however.

---

What can we try to produce more fit individuals?

- **Crossover**: Combine two individuals
  
  \[ 001110000, \text{sum}(35, 52, 61) = 148 \]
  
  \[ 100001100, \text{sum}(19, 76, 84) = 179 \]

  - New genomes:
    
    \[ 001000110, \text{sum}(35, 76, 84) = 195 \]
    
    \[ 100110000, \text{sum}(19, 52, 61) = 132 \]

  - Crossover point selected at random

---

Crossover Variations

- Sometimes two crossover points are chosen, rather than one, and the subsequences between them are swapped.

  - Just as with mutation, crossover could produce one or more individuals that are totally unfit.

---

Sample GA Program

- The program
  
  `/cs/cs152/ga/subsum/subsum.java`
  
  carries out the genetic algorithm on this problem.

- Examples:
  
  go ss1.in
  
  go ss2.in

---

Main Loop of the Subset Sum GA Program (1)

```java
public void evolve(int generations)
{
  for( generation = 0; generation < generations; generation++ )
  {  
    retain(); // retain the more fit individuals
    crossover(); // perform crossover on those retained
    mutate();  // mutate the resulting population
    sort();   // sort by fitness
  }
}
```
Sample Run of subsum (1)

<table>
<thead>
<tr>
<th>Subset sum problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>generations = 100</td>
</tr>
<tr>
<td>population size = 10</td>
</tr>
<tr>
<td>retain size = 5</td>
</tr>
</tbody>
</table>

| immutable = 5 |
| mutation rate = 0.1 |
| target = 200.0 |
| values = (19 23 35 52 61 68 76 84 92) |

Sample Run of subsum (2)

generation 1, average fitness = 64.4:
181/200 (r 19) 000111000 52 61 68
164/200 (r 36) 100100110 19 61 84
160/200 (r 40) 0000000110 76 84
145/200 (r 55) 000010010 61 84
-1/200 (r 201) 001100101 35 52 76 92
-1/200 (r 201) 000011100 61 68 76 84
-1/200 (r 201) 010111000 23 52 61 68
-1/200 (r 201) 000011110 61 68 76 84
-1/200 (r 201) 100111110 19 61 68 76 84
-1/200 (r 201) 000010110 61 76 84

Sample Run of subsum (3)

generation 2, average fitness = 114.3:
197/200 (r 3) 000110010 52 61 68
181/200 (r 19) 000110000 52 61 68
164/200 (r 36) 100010000 19 61 84
160/200 (r 40) 000000110 76 84
148/200 (r 52) 100010000 19 61 68
145/200 (r 55) 000010000 19 61 84
96/200 (r 104) 001010000 35 61
54/200 (r 146) 101000000 19 35
-1/200 (r 201) 000110011 52 61 84 92
-1/200 (r 201) 001100101 35 52 76 92

Sample Run of subsum (4)

generation 5, average fitness = 157.7:
200/200 (r 0) 100111000 19 52 61 68
197/200 (r 3) 000110010 52 61 84
197/200 (r 3) 000110010 52 61 68
181/200 (r 19) 000111000 52 61 68
181/200 (r 19) 000111000 52 61 68
179/200 (r 21) 001100001 35 52 92
164/200 (r 36) 100010000 19 61 84
164/200 (r 36) 100010000 19 61 84
115/200 (r 85) 101010000 19 35 61
-1/200 (r 201) 000111010 52 61 68 84

Complexity Considerations

- All possible subsets of n values can be enumerated in \(2^n\) steps.
- For low n (say in the 20's or less), this might be feasible.
- For larger n, it is not \(2^{2^n} = \text{about 5 billion}\).
- For large n, the genetic algorithm can produce good, if not optimal, answers in much less time than enumeration.
Other Genomes

- A bit vectors is not the only way, or necessarily the best way, to represent a genome.
- Other possibilities:
  - A list or matrix of integers or floats

Challenges in representing Genomes: Traveling Salesperson Problem

- Not every optimization problem has a genome encoding that will allow naive mutations and crossovers.
- Consider the TSP:
  - An instinctive way to represent a genome is as a permutation of the cities on a tour.

Crossing two Permutations

- [1, 3, 2, 6, 5, 4]
  [2, 3, 4, 1, 6, 5]

  - Result of naive crossing:
    [1, 3, 4, 1, 6, 5]
    [2, 3, 2, 6, 5, 4]
  - Unfortunately, these sequences are not permutations.

Reinterpreting permutation crossings

- [1, 3, 2, 6, 5, 4]
  [2, 3, 4, 1, 6, 5]

  - Interpret as:
    - Insert 2,6,5 in the second genome at the first crossover point and remove those elements from wherever they occurred in the second genome.

Similarly, to produce the second new genome

- [1, 3, 2, 6, 5, 4]
  [2, 3, 4, 1, 6, 5]

  - Interpret as:
    - Insert 3,4,1,6,2,5 in the second genome at the second crossover point and remove those elements from wherever they occurred in the second genome.
Net effect of Crossover

- [1, 3, 2, 6, 5, 4]
- [2, 3, 4, 1, 6, 5]

- [3, 4, 1, 6, 2, 5]
- [3, 2, 6, 5, 4, 1]

- The results share some of the structure of both parents, which is desirable.

Keller’s TSP GA

- Uses same overall loop as the subset sum algorithm.
- The genome is now a permutation vector.
- Crossover is as described.
- Mutation consists of swapping two random elements of the permutation.

TSP GA in operation (1)

Traveling Salesperson Problem

- generations = 1000
- population size = 50
- retain size = 25
- immutable = 15
- mutation rate = 0.1

TSP GA in operation (2)

<table>
<thead>
<tr>
<th>Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
</tr>
<tr>
<td>2.0</td>
</tr>
<tr>
<td>3.0</td>
</tr>
<tr>
<td>4.0</td>
</tr>
<tr>
<td>5.0</td>
</tr>
<tr>
<td>6.0</td>
</tr>
<tr>
<td>7.0</td>
</tr>
<tr>
<td>8.0</td>
</tr>
<tr>
<td>9.0</td>
</tr>
</tbody>
</table>

TSP GA in operation (3)

| Improvement in generation 1: | 25: 4 6 10 14 9 11 7 12 3 5 13 1 0 8 2 |
| Improvement in generation 9: | 22: 4 6 14 9 11 7 12 10 3 5 13 1 0 8 2 |
| Improvement in generation 10: | 20: 4 6 9 14 10 11 7 12 3 5 13 1 0 8 2 |
| Improvement in generation 30: | 18: 4 6 9 14 10 12 7 11 3 5 13 1 0 8 2 |
| Improvement in generation 317: | 17: 4 6 9 14 10 12 5 13 3 11 7 1 0 8 2 |

- best after 1000 generations, 17: 4 6 9 14 10 12 5 13 3 11 7 1 0 8 2
- verification: 4 (1.0) 6 (1.0) 9 (2.0) 14 (1.0) 10 (1.0) 12 (1.0) 5 (2.0) 13 (1.0) 3 (1.0) 11 (1.0) 7 (1.0) 1 (1.0) 0 (1.0) 8 (1.0) 2 (1.0) 4

Roulette-Wheel Optimization

- Rather than keep n copies of the same individual, record the individual once, along with its % of the population.
- Then during selection, choose individuals by spinning a “roulette wheel” biased with the given % toward the individual.
Roulette-Wheel Optimization

GA Perspective

- Like gradient descent, GA’s can also get stuck in local fitness extrema.
- The space is different; for GA’s, a stuck point corresponds to a population from which crossover does not yield any better individuals.
- Mutation is one hope for leaving such an extremum. Other possibilities are simulated annealing, random restarts.

Evolution Options

- Since we are simulating using a computer, not actually evolving species, there is no reason why Lamarckian, rather than Darwinian, evolution could not be used.
- Most results to date are Darwinian.
- Lamarckian could integrate other learning models to breed new species from individuals that have learned.

Lamarckian Leads


Genetic Programming

- Genetic programming is the GA idea applied to evolving programs (as opposed to just numbers).
- The prime mover of this field is John R. Koza.

Reference

John R. Koza

Genetic Programming: On the Programming of Computers by Means of Natural Selection

MIT Press, 1996
Genetic Programming Genomes = Syntax Trees

For animated tutorial, please see:
http://www.genetic-programming.com/gpanimatedtutorial.html

Mutation of a Program

Crossover of Two Programs

Genetic Programming Demo: Symbolic Regression
(http://www.ifh.ee.ethz.ch/~gerber/approx/default.html)

The evolved program

Truck-Backer using GP
http://www.handshake.de/user/blickle/Truck/index.html

GP Caution

- Typically fitness of genetically-evolved programs is established by working on a large number of test cases.

- We know this is not completely sound.

- A possible fruitful area is to evolve a proof of correctness along with the program itself. I know of no work in this area.
Engineering Applications: Evolving Hardware (Analog & Digital)

- The evolved program is a list of instructions for constructing the circuit, rather than the graph of the circuit itself.
- This approach has also been used to construct neural networks.
- The results for circuit design are competitive with, or superior to, human engineering.
- Numerous patents have been reinvented using GP.

Generating Analog Circuits

Analog filter generated by genetic programming

A Genetically-Evolved Amplifier

Koza (striped shirt) with 70-node Beowulf cluster

Koza's 1000 node Beowulf used for genetic programming

Commercial Applications: Marketing Projections

Function Repertoire Menu

Other Opportunities

- Parallelism in Computation (local work of Beda & Margileth, and Tom Johnson, CS 152 steu)
- Parallelism optimizing transformations
- Music
- Robotics

Koza GP Video Clips

http://www.statsoft.com/textbook/stcluan.html#h