Evolutionary Computation

- Genetic Algorithms and Genetic Programming are prototypical examples of what is called *Evolutionary Computation*
- Evolutionary computation characterized by
  - Population consisting of multiple “individuals”
  - Fitness function evaluating individuals
  - Reproduction strategy for generating new generations of individuals
  - Ideally, fitness increases (on average) over successive generations
- Ultimately just a focused random search strategy for finding maximum-fitness individuals
Pseudo-code

Initialize generation counter
Initialize a (usually random) population of individuals
Evaluate fitness of all individuals of population
While not done (based on fitness, # generations, etc.)
  Increment the generation counter
  Select a sub-population for generating new offspring
  Generate new individuals using
    • replication of individuals
    • crossover using 2 parents
    • mutation of resulting individuals
  Evaluate fitness of all new individuals

Genetic Algorithms

• Prototypical representation: fixed length bit strings
  • “chromosomes”
• To create new individuals, select 2 “parents”
• Combine the bit strings of the 2 parents in some way to create one or more (often 2) new individuals
  • Crossover
• Also, apply small random perturbations to the “children”
  • Mutation
Crossover Operators

- **Single-point**
  
  \[
  \begin{array}{c}
  11101001000 \\
  0001010101
  \end{array}
  \rightarrow
  \begin{array}{c}
  11101010101 \\
  00001001000
  \end{array}
  
- **Two-point**
  
  \[
  \begin{array}{c}
  1101001000 \\
  0001010101
  \end{array}
  \rightarrow
  \begin{array}{c}
  00101000101 \\
  11001011000
  \end{array}
  
- **Uniform**
  
  \[
  \begin{array}{c}
  1101001000 \\
  0001010101
  \end{array}
  \rightarrow
  \begin{array}{c}
  10001000100 \\
  01101011001
  \end{array}
  

Mutation Operator

\[
\begin{array}{c}
1101001000 \\
0001010101
\end{array}
\rightarrow
\begin{array}{c}
1101011000 \\
01101011001
\end{array}
\]

With some probability, a bit is flipped.
GA Representations

- Bit strings
  - Fixed length
  - Variable length
- Strings of more general kinds of data
  - Integers
  - Reals

Representation Issues

- What if some chromosomes don’t represent valid objects in the domain of the search space? Possible approaches:
  - Give meaningless chromosomes very low fitness
  - Limit crossover or other operators so that only valid chromosomes generated
  - Follow generation of any new chromosome with a step that modifies it to make it valid
Selecting Most Fit Individuals

- Fitness proportionate selection
  \[ P(x_i) = \frac{\text{Fitness}(x_i)}{\sum_{j} \text{Fitness}(x_j)} \]

- Tournament selection
  - Pick a random subset of individuals (often 2)
  - With fixed probability \( p \), select the most fit

- Rank selection
  - Sort individuals by fitness
  - Prob. of selection based on rank

Another interesting way

- Have individuals compete head-to-head
- Appropriate where fitness defined in terms of competitive ability
- Used to evolve neural networks for evaluating checkers board positions (Fogel)
Applications

- Timetabling (e.g., exam scheduling)
- Discovering successful policies in simple dynamical systems (e.g., pole-balancing)
- Neural networks
  - Finding weights
  - Finding topology
- Other combinatorial optimization problems
  - Traveling salesman

Where do GAs fit?

- Perhaps more of a *machine discovery* method
  - A way to search spaces for “fit” individuals
  - Can be used to search for good hypotheses in more traditional machine learning applications
    - Weights/topology in neural networks
    - Rules
    - Decision trees
Genetic Programming

Population of programs represented by trees

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

E.g. \((+ (\sin x) (\sqrt{(+ (\text{expt} x 2) y)}))\)

Crossover

[Diagrams of tree structures for crossover operations]
Block Stacking Problem (Koza)

Goal: Spell UNIVERSAL vertically

Terminals:
- CS ("current stack") = name of the top block on the stack, or F
- TB ("top correct block") = name of topmost correct block on stack
- NN ("next necessary") = name of the next block needed above TB in the stack

Block Stacking: Primitive Functions

- (MS x): ("move to stack"), if block x is on the table, moves x to the top of the stack and returns T. Otherwise, does nothing and returns F.
- (MT x): ("move to table"), if block x is somewhere in the stack, moves the block at the top of the stack to the table and returns T. Otherwise, returns F.
- (EQ x y): ("equal"), returns T if x equals y, and F otherwise.
- (NOT x): returns T if x = F and returns F if x=T
- (DU x y): ("do until"), executes x repeatedly until y returns T
Learned Program

- Trained to fit 166 test problems
- Using population of 300 programs, found this after 10 generations:

\[
(\text{EQ} \ (\text{DU} \ (\text{MT} \ \text{CS}) \ (\text{NOT} \ \text{CS}))
\]

\[
(\text{DU} \ (\text{MS} \ \text{NN}) \ (\text{NOT} \ \text{NN})))
\]

Use of EQ here just a syntactically valid way to perform sequential execution

Another Example: Electronic Circuit Design

(Koza)

- Individuals are programs that transform beginning circuit to final circuit by adding or subtracting components and connections
- Use population of 640,000, run on 64 node parallel processor
- Discovers filter circuits competitive with best human designs
Evolutionary Methods: Upside

- Simple to implement
- Easily parallelized
- Less subject to local optima than more local search techniques
- Very general-purpose framework

Evolutionary Methods: Downside

- Often extremely large search spaces
  - Need to carefully handcraft
    - Fitness function
    - Representation of individuals
    - Operators
  - Can be impractically slow
- Very little theory