A n Introdu ction to Genetic Algorithms

Josiah Poon

Outline of the lecture

- History of Evolutionary Algorithms
- Genetic Algorithm (GA)
  - Overview
  - Preliminary Considerations
  - The Basic Algorithm
- Two Examples
- Why it works? – Schema Theorem
- Advantages & Disadvantages
- Applications

The Big Picture

Sex is good

Individual  
Candidate Solution

Fitness  
Quality

Environment  
Problem

The Metaphor

Evolution  
Problem Solving

History

- Evolutionary Programming (EP) 1962  L. Fogel  San Diego, CA
- Genetic Algorithms (GA) 1962  J. Holland  Ann Arbor, MI
- Evolution Strategies (ES) 1965  I. Rechenberg  H.-P. Schwefel  Berlin, Germany
- Genetic Programming (GP) 1989  J. Koza  Palo Alto, CA
GA Definition (1)

- Grefenstette
  - A genetic algorithm is an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. During each temporal increment (called a generation), the structures in the current population are rated for their effectiveness as domain solutions, and on the basis of these evaluations, a new population of candidate solutions is formed using specific genetic operators such as reproduction, crossover and mutation.

GA Definition (2)

- Goldberg
  - They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

GA Overview

- Developed by John Holland
- Search algorithms based on the mechanics of natural evolution – survival of the fittest
- Ability to create an initial population of feasible solutions, then recombine them in such a way to direct the path to the most promising areas of the search space.
- Each solution is encoded as a chromosome (also called genotype), a fitness function is used to measure the fitness of the phenotype.
- The fitness of a phenotype determines its chances of survival.

GA Overview (continued)

- Use probabilistic rules to evolve a population from one generation to the next:
  - Biased reproduction: selection
  - Crossover
  - Mutation
- A few parameters to twist:
  - Population size
  - Crossover rate
  - Mutation rate

GA – Preliminary Considerations

- Representation
  - Can be bit string, real numbers, integers, characters, list of rules, matrices
    - Chromosomes are of the same type
  - Choice of alphabets
  - Length of chromosome
    - Chromosomes are of the same length
  - Most GAs use haploid representation as compared against human’s diploid representation
  - A representation dictates the quality of the solution
- Fitness Function

GA – Preliminary Considerations

- Population Size
  - Remain constant throughout all generations
  - Too small: premature convergence
  - Large: greater chance to find the global optimum, but higher computational cost
GA – The Basic Algorithm

t = 0
Generate initial population, P
Evaluate all individuals in P, using a fitness function
While not end of evolution
  t = t + 1
  Reproduce P, from P_{t-1}
  Perform Crossover in P_t
  Perform Mutation in P_t
  Evaluate all individuals in P_t using a fitness function

It could be number of generations, or a certain fitness value is achieved

GA – Selection for Reproduction

- Roulette Wheel
  The classical selection operator for generational GA as described by GA. Each member of the pool is assigned space on a roulette wheel proportional to its fitness. The members with the greatest fitness have the highest probability of selection.
- Ranking
- Tournament

GA – Crossover

- Mate each chromosome randomly
- In each mating,
  - randomly select the crossover positions
  - genetic materials between two parents are swapped
- Various (generic) crossover techniques
  - One-point
  - Two points
  - Uniform

GA – Crossover Example

<table>
<thead>
<tr>
<th>Single Point Crossover</th>
<th>Two Points Crossover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent1: 100 1001010</td>
<td></td>
</tr>
<tr>
<td>Parent2: 001 0110111</td>
<td></td>
</tr>
<tr>
<td>Child1: 110 0111011</td>
<td></td>
</tr>
<tr>
<td>Child2: 001 1001010</td>
<td></td>
</tr>
</tbody>
</table>

GA – Mutation

- The entire state-space is searched, given enough time
- Restore lost information or add information to the population
- Perform on a child after crossover
- Perform very infrequently, p_m, usually < 0.01

GA versus Traditional Search Algorithm
GA versus Traditional Search Algorithm

- GA works from a population of strings instead of a single point.
- Application of GA operators causes information from the previous generation to be carried over to the next.
- GA uses probabilistic transition rules, not deterministic rules.

The Search Mechanism

- A search is composed of exploration and exploitation.
- The search in GA:
  - Exploration by Recombination
  - Mutation
  - Exploration by Selection
- Properties of GA:
  - Complete? Yes
  - Optimal? No
  - Time? On-
  - Space? = chromosomes

An Example

- $f(x) = 4 \cos(x) + x + 2.5$
- $0 \leq x \leq 31$
  - Representation: a 5-bit binary string
- Parameter setting:
  - Population size = 8
  - Crossover rate = 0.75
  - Mutation rate = 0.001
  - Max. generation = 40

Example

$f(x) = 4 \cos(x) + x + 2.5$

<table>
<thead>
<tr>
<th>No.</th>
<th>Chromosome</th>
<th>x</th>
<th>f(x)</th>
<th>P(x)</th>
<th>$f(x)_{accum}$</th>
<th>$P(x)_{accum}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01011</td>
<td>19</td>
<td>25.469</td>
<td>0.180</td>
<td>25.469</td>
<td>0.180</td>
</tr>
<tr>
<td>2</td>
<td>01101</td>
<td>18</td>
<td>9.144</td>
<td>0.066</td>
<td>9.144</td>
<td>0.066</td>
</tr>
<tr>
<td>3</td>
<td>11001</td>
<td>25</td>
<td>51.480</td>
<td>0.220</td>
<td>51.480</td>
<td>0.220</td>
</tr>
<tr>
<td>4</td>
<td>00110</td>
<td>14</td>
<td>12.341</td>
<td>0.090</td>
<td>12.341</td>
<td>0.090</td>
</tr>
<tr>
<td>5</td>
<td>01111</td>
<td>11</td>
<td>13.518</td>
<td>0.098</td>
<td>13.518</td>
<td>0.098</td>
</tr>
<tr>
<td>6</td>
<td>10111</td>
<td>23</td>
<td>23.369</td>
<td>0.170</td>
<td>23.369</td>
<td>0.170</td>
</tr>
<tr>
<td>7</td>
<td>00101</td>
<td>6</td>
<td>3.885</td>
<td>0.028</td>
<td>3.885</td>
<td>0.028</td>
</tr>
<tr>
<td>8</td>
<td>00011</td>
<td>17</td>
<td>18.599</td>
<td>0.134</td>
<td>18.599</td>
<td>0.134</td>
</tr>
</tbody>
</table>

- Average $= 17.198$
- Best $= 25.469$

Selection

Roulette Wheel
Another Example – Knapsack Problem

- Try to maximize $\sum_{w} v x$ under the constraint of $\sum_{w} x < 104$
- Parameter setting:
  - Population size = 100
  - Crossover rate = 0.75
  - Mutation rate = 0.01
  - Max. generation = 100

Why it works? – Schema Theorem

- An abstract way to view the complexities of crossover
- Consider a 6-bit representation where * indicates “don’t care”
  - $0^{*****}$ represents a subset of 12 strings
  - $1^{**0**}$ represents a subset of 8 strings
- Let $H$ represent a schema such as $1**1**$
  - Order: $o(H)$, the number of fixed positions in the schema, $H$
    - $o(1^{****}) = 1$
  - Length: $d(H)$, the distance between sentinel fixed positions in $H$
    - $d(H) = 1 = 3$
  - Short schema are less likely to be disturbed by crossover than long schema

Knapsack Problem - Results

Why it works? – Schema Theorem (continued)

$$M[H, f^*] \geq M[H, f^*] \frac{1}{p_1 - p_2}$$

where
- $M[H]$ number of strings in population $i$ with the schema $H'$.
- $f(H)$ average fitness of the strings with the schema $H'$.
- $F$ average fitness of the entire population.
- $p_1$ probability of the schema being destroyed by crossover.
- $p_2$ probability of the schema being destroyed by mutation.

GA: Advantages

- A robust search technique
- No (little) knowledge (assumption) w.r.t. the problem space
- Fairly simple to develop: low development costs
- Easy to incorporate with other methods
- Solutions are interpretable
- Can be run interactively, i.e. accommodate user preference
- Provide many alternative solutions

GA: Advantages (continued)

- Acceptable performance at acceptable costs on a wide range of problems
- Intrinsic parallelism (robustness, fault tolerance)
- Superior to other techniques on complex problems with
  - Lots of data, many free parameters
  - Complex relationships between parameters
  - Many (local) optima
GA: Disadvantages

- No guarantee for optimal solution within a finite time
- Weak theoretical basis
- Interdependency of genes
- Parameter tuning is an issue
- Often computationally expensive, i.e. slow

Honours & MIT/MAIT: Neural Networks

Let’s take a break!

Honours & MIT/MAIT: Neural Networks

Evolving Artificial Neural Networks

The material of this seminar is based on the paper Evolving Artificial Neural Networks by Xin Yao, Proceedings of the IEEE, 87 (9), September 1999.

How can EA help evolve NN?

- Connection weights
- Architectures
- Learning rules

The Training Problem in NN

- Weight training in NN is usually formulated as minimization of error function
- Most training algorithms, such as BP, are based on gradient descent
- BP has its drawbacks due to its use of gradient descent – trapped in local minimum

Evolution of Connection Weights

- Reformulate the training process in NN as the evolution of connection weights
- Advantage of using EANN over gradient-descent-based training algorithms
  - Error function does not have to be differentiable or even continuous
- Two phases in the evolutionary approach
  - To decide the representation of connection weights
  - To decide the search operators, e.g. crossover and mutation
Evolution of Connection Weights:

**Representation - Binary**

- Each connection weight is represented by a number of bits with fixed-length.
- Generally difficult to apply crossover operators in evolving weight as they tend to destroy features extractors found in the evolution.
- Each connection weight is represented by 4-bits.
- 24 bits altogether.
- 0000 representing no connection.

![Binary Representation Diagram](image)

Josiah Poon © 2003

Evolution of Connection Weights:

**Advantages of Binary Representation**

- Simple and General
- Can apply classical crossover and mutation operator easily.
  - Little need to design complex and custom-made search operators.
  - Facilitate hardware implementation of NN.

Josiah Poon © 2003

Evolution of Connection Weights:

**Problems in the Evolutionary Training**

- Permutation problem.
  - Also known as competing convention problem.
- Caused by many-to-one mapping from the representation (genotype) to the NN (phenotype), i.e. two chromosomes of different values give rise to the same functional NN.

![Training Problems Diagram](image)

Josiah Poon © 2003

Evolution of Connection Weights:

**Representation – Real Number**

- Some have argued that the minimal cardinality (i.e. binary representation) might not be the best.
- Each individual row has a real vector.
- Implication of using a real vector.
  - Traditional binary crossover and mutation operators cannot be used directly.
- EP or ES are more suited to evolve real vectors (which require continuous optimization).
- **Advantages** of using mutation-based EA’s.
  - They can reduce the negative impact of the permutation problem.
  - The evolutionary process can be more efficient.

Josiah Poon © 2003

Evolution of Connection Weights:

**Mutation Operator**

- Gaussian mutation
- Implementation.
  - A set of population with \( \mu \) individuals where each individual is a pair of \((w_i, \eta_i)\).
  - \( \eta_i(j) = \eta_i(j) \exp(\tau N(0, 1) + t N(0, 1)) \)
  - \( w_i(j) = w_i(j) + \eta_i(j) N_j(0, 1) \)
  - \( \eta_i \) as the variance vectors for Gaussian mutations.
  - \( w_i(j), \eta_i(j) \) are the name component of the vectors \( w_i, \eta_i \) and \( \eta_i' \).
  - \( N(0, 1) \) indicates that the random number is generated anew for each value of \( j \).
  - The parameters \( \tau \) and \( \tau' \) are commonly set to \( \tau = \tau' = 1 \).
- Conduct pairwise comparison over the union of parents \( (w_i, \eta_i) \) and offspring \( (w_i', \eta_i') \). Select m individuals out of \( (w_i, \eta_i) \) and \( (w_i', \eta_i') \) that have most wins to form next generation (aka tournament selection scheme).

Josiah Poon © 2003

---

**GA & NN, Josiah Poon (c) 2003**
Evolution of Architectures

The architecture includes:
- Topological structure, i.e. connectivity
- Transfer function of each node
- Architecture has significant impact on a network’s information processing capability.
  - A few connections and linear nodes: may not be able to perform the task at all
  - A large number of connections and nonlinear nodes: may overfit the training data and fail to have good generalization ability

But up to now, architecture design is an expert’s gut insight and is still error-prone
Currently, no systematic way to design a near optimal architecture.

Evolution of Architecture: Direct Encoding Scheme

Two approaches
- Separates the architecture from the connection weights
- Evolves architecture and connection weights simultaneously

Evolution of Architecture: Why EA is appropriate?

- Finding an optimal architecture is a search problem.
- The performance of all architecture forms a discrete surface in the space.
- The optimal design is the highest point on this surface.
- Characteristic of the surface is:
  - Infinitely large
  - Non-differentiable
  - Complex and noisy
  - Deceptive
  - Multimodal

Evolution of Architecture: Constructive & Destructive Algorithm

Approaches
- Constructive
  - Start with a minimal network (network with minimal number of hidden layers, nodes and connections) and add new layers, nodes and connections when necessary
- Destructive
  - Start with a maximal network and delete unnecessary layers, nodes and connections during training

Criticisms:
- Angeline [149]: Such structural hill climbing methods are susceptible to becoming trapped at structural local optima
- They only investigate restricted topological subsets rather than the complete class of network architectures

Evolution of Architecture: How?

- Two phases in the evolutionary approach
  - To decide the representation of connection weights
  - To decide the search operators, e.g. crossover and mutation

Key issue:
- How much information about an architecture should be encoded in the chromosome?
  - Direct encoding: all the details
  - Indirect encoding: only the important parameters of an architecture

Evolution of Architecture: Separate the architecture from the connection weights for Feedforward Network

- Each connection weight is represented by its binary
- An $N \times N$ matrix $C = (c_{ij})_{N \times N}$
  - Has a one-to-one mapping
  - A feedforward NN will have non-zero entries only in the upper-right triangle of the matrix
- Convert the chromosome to a NN, initialize it with random weights and train.
Evolution of Architecture:
Separate the architecture from the connection weights for Recurrent Network

\[
\begin{pmatrix}
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 \\
\end{pmatrix}
\]

00110 00101 10001 00001 01000

Evolution of Architecture:
Potential Problem

- Scalability
  - Use domain knowledge to reduce the search space

Evolution of Architecture:
Indirect Encoding

- The details about each connection are
  - Either predefined according to prior knowledge
  - Or specified by a set of deterministic developmental rules

- Various Representation
  - Parametric Representation
  - Developmental Rule Representation
  - Fractal Representation
  - Other Representations

Evolution of Architecture:
Indirect Encoding - Parametric

- NN architecture is specified with a set of parameters, e.g.
  - No. of hidden layers
  - No. of hidden nodes
  - Number of connection between layers

- Encode these parameters into the chromosome
  - This approach reduces the length of binary chromosome, but the algorithm can only search a limited subset of feasible solutions
  - Example: if we encode only the no. of hidden nodes in the hidden layer, we have assumed a layered forward NN with a single hidden layer

Evolution of Architecture:
Indirect Encoding – Developmental Rule

- Optimization of developmental rules brings more compact genotypical representation.
- Destructive effect of crossover is lessened.

Evolution of Architecture:
Indirect Encoding – Developmental Rule

- Described by a recursive equation or a generation rule in the production rules.

\[
\begin{align*}
& x \rightarrow \left[ \begin{array}{c}
a \\ b \\ c \\ d \end{array} \right] \\
& z \rightarrow \left[ \begin{array}{c}
-1 \\ -1 \\ -1 \\ -1 \end{array} \right] \\
& \cdots \\
\end{align*}
\]
Evolution of Architecture: Indirect Encoding – Developmental Rule

- How can we get the set of rules to define the NN?
- By evolution, of course!
- Rules evolution
  - *Pitt approach*: encode the whole rule set as an individual
  - *Michigan approach*: encode each rule as an individual

Evolution of Learning Rules (1 of 2)

- A training algorithm may have different performance when applied to different architectures.
- The design of training algorithms (learning rules) used to adjust the connection weights depends on the type of architectures.
- To design an optimal learning rule is difficult when we know little about the NN's architecture.
- It is desirable to develop an automatic and systematic way to adapt the learning rule to an architecture and the task on hand.
  - A NN should learn its learning rule dynamically rather than a fixed and handcrafted rule.

Evolution of Learning Rules (2 of 2)

- Most of the research focus on *how learning guides evolution*.
- The outcome of this research
  - not only optimizes learning rules
  - but also models the creative process to deal with complex and dynamic environment

Evolution of Learning Rules: The Evolution of Algorithmic Parameters

- BP parameters, e.g. learning rate and momentum
- Harp et al [152] encoded the BP parameters with the architecture in the same chromosome.

Evolution of Learning Rules: The Evolution of Learning Rules

- To encode the dynamic behaviour of a learning rule into static chromosomes.
- Constraints have to be set on the type of dynamic behaviour.
- Two basic assumptions:
  - Weight updating depends only on local information
  - The learning rule is the same for all connections in a NN
- Three major issues:
  - Determination of a subset of terms
  - Representation of their real-valued coeff as chromosomes
  - The EA used for evolution

Evolution of Learning Rules: An Example by Chalmers [264]

- Defined a learning rule
  - as a linear combination of 4 local variables and their 6 pairwise products
  - No 3rd or 4th terms
  - 10 coefficients and a scale parameter were encoded in a binary string via exponential encoding
- Something fixed in the evolution
  - The architecture
  - Single layer NN
  - Number of inputs & outputs
- Results
  - After 1000 generations, the evolution discovered the well-known delta rule and some of its variants.
  - Able to demonstrate the evolutionary power to discover novel learning rules
A General Framework of EANN

Epilogue: GA & Creativity
- Evolutionary algorithms are not just search algorithms, they are frequently applied to creative design.
- Most search algorithms try to deal with what is called "routine design" where the search space is well-known and constrained.
- But the space for design is ill-defined and evolutionary algorithms are good tools to explore the space. They are applied to:
  - Cartoon Faces
    - http://www.acad.ohio-state.edu/~mlewis/AED/Faces/
    - http://budokan.cse.ohio-state.edu/~mlewis/AED/Example/
  - Music
    - http://www.designdalarna.se/projects/art_and_interactivity/living-melodies/

GA Applications
- Bin Packing
- Scheduling
- Travelling Salesman Problem
- Machine Learning
- Trajectory Planning
- Pattern Recognition
- and of course, evolving the Neural Network

Cartoon Faces
- http://www.acad.ohio-state.edu/~mlewis/AED/Faces/

Music
- http://www.designdalarna.se/projects/art_and_interactivity/living-melodies/

Books