Genetic Algorithms—Some History

Late 50’s, early 60’s—attempts to mesh computer science and evolution.


- Goldberg, gas pipeline control system (1983)
- Davis & Coombs, network design (1987)
- GE & RPI, jet engine turbine design (?)
- Koza, automatic programming (1992 ?)
• Holland, modeling ecosystems (1995)

• Aircraft design; scheduling; symbolic math; etc.
(Incomplete List of) References Available at Allegheny


Bentley, Peter J., and David W. Corne. *Creative Evolu-

Kennedy, James, and Russel C. Eberhart. *Swarm Intel-

**See also:** http://cs.felk.cvut.cz/~xobitko/ga/
Basic Algorithm

1 Generate initial Population P(0) at random, and set i=0;
2 repeat
3 Evaluate the fitness of each individual in P(i)
4 Select parents from P(i) based on their fitness in P(i)
5 Apply crossover to create offspring from parents
6 Apply mutation to the offspring
7 Select generation P(i+1) from current offspring, O(i), and parents P(i)
8 until finished
Basic Principles

Start with a problem (e.g., optimization)

Example: find \((x, y)\) that maximizes \(yx^2 - x^4\) for \(0.0 \leq x, y < 1.0\).

Define “fitness criterion” for potential solutions: value of \(f(x, y)\).
Encode \((x, y)\) as a “chromosome”: scale to integers, then use binary:

\[
(.25, .3) \rightarrow (\lfloor .25 \times 1024 \rfloor, \lfloor .3 \times 1024 \rfloor)
\]
\[
= (256, 307)
\]
\[
= 0100000000 0100110011
\]

Create random initial population; evaluate

- 0110011110 0011100010 \( f = 0.00935739 \)
- 0010100001 0001110011 \( f = 0.00216511 \)
- 1111100111 1100110000 \( f = -0.147422 \)
- 0111100101 1110101001 \( f = 0.154946 \)
- 0111011111 1001011100 \( f = 0.0811862 \)
- 1001000001 0101100101 \( f = 0.009883 \)
- 1010010000 1000111010 \( f = 0.060017 \)
- 1000000011 1011101011 \( f = 0.120539 \)

Use crossover to “mate”; fittest parents get chosen with higher probability:

\[
\begin{align*}
0111100101 & \quad 1110101001 \\
\times & \\
1000000011 & \quad 1011101011 \\
\end{align*}
\]
\[
\Rightarrow 0111100011 1010101011 \\
1000000101 1111101001
\]
Replace (a portion of) population with (more fit) offspring

Perform random mutations:

0111011111 1001011100 ⇒ 0111010111 1001011100

Reevaluate population, iterate the process

Why should it work?

*Schema* = string of “fixed” bits and “don’t cares.” Every bit string falls into many different schemata.

01001 0****, *1**1, 010*1, etc.
11010 11***, **0*1, 1****, etc.
00101 0****, **1*1, *0***, etc.
11101 **1*1, 11***, 1**01, etc.
Order of a schema = number of fixed bits. Defining length = distance between leftmost and rightmost fixed bits.

Each schema has an “average fitness” (= average fitness of all strings that match that schema).

Schemata of short defining length and high average fitness are called building blocks.

Short defining length means crossover is unlikely to disrupt a schema, so crossover tends to join building blocks together. [Schema Theorem]
Issues

How do we determine what “genes” and “chromosomes” to use, where to put them, how many, how to encode?

When should we stop the process?

What mutation rate should be used?

How many crossovers; where?

How big should the population be at any given time?

When and how should we delete an item from the population?

... and more. It is still very much an art; a current research topic is to make it more formal, more rigorous.
Related Topics

Simulated Annealing — see, e.g.,
http://www.fortunecity.com/skyscraper/quantum/1047/gtsatsp.html

or
http://www.theblueplanet.org/JSimul_readme.html

ACO — Ant Colony Optimization — see
http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html

Genetic Programming — see
http://www.genetic-programming.org/

Other nature-inspired algorithmic techniques
Senior Projects in Evolutionary Algorithms

Do you want to use EAs to study some problem?

Is problem suitable? (e.g., TSP, knapsack, scheduling, etc?)

Current methods not good enough? (e.g., no polynomial-time approximation scheme?)

Identify representation (genotype), fitness, operators (selection, crossover, mutation) parameters (population size, mutation rate, etc.).

Tune parameters

Establish basis for evaluation (metrics): Quality of solution? time? compare to existing algorithms (quality and/or time and/or . . .)? compare results to known optimal solutions? (e.g., TSPLIB)
Do you want to study some aspect of EAs?

Read some books! Know underlying principles, theory!

Parallelization; distributed implementation

Parameter tuning

Operators

Hybrid methods (EA plus problem-specific improvements)

“Exotic” EAs — ant colony, swarm, artificial immune systems, etc.

“Creative” evolutionary algorithms (Bentley)


