Genetic Algorithms

MSE 2400 EaLiCaRA
Dr. Tom Way

Evolution – Darwin's Natural Selection

- IF there are organisms that reproduce, and
- IF offspring inherit traits from their progenitors, and
- IF there is variability of traits, and
- IF the environment cannot support all members of a growing population,
- THEN those members of the population with less-adaptive traits (determined by the environment) will die out, and
- THEN those members with more-adaptive traits (determined by the environment) will thrive

The result is the evolution of species.

Basic Idea Of Principle Of Natural Selection

"Select The Best, Discard The Rest"

An Example of Natural Selection

- **Giraffes have long necks**.

Giraffes with slightly longer necks could feed on leaves of higher branches when all lower ones had been eaten off.
→ They had a better chance of survival.
→ Favorable characteristic propagated through generations of giraffes.
→ Now, evolved species has long necks.

NOTE: Longer necks may have been a deviant characteristic (mutation) initially but since it was favorable, was propagated over generations. Now an established trait.

So, some mutations are beneficial.

Evolution Through Natural Selection

Initial Population Of Animals
↓
Struggle For Existence-Survival Of the Fittest
↓
Surviving Individuals Reproduce, Propagate Favorable Characteristics
↓
Millions Of Years
↓
Evolved Species
(Favorable Characteristic Now A Trait Of Species)

How Genetic Algorithms Work

- Genetic Algorithms implement optimization strategies by simulating evolution of species through natural selection.
- Iteratively improve a set of possible answers to a problem by combining best parts of possible answers to form (hopefully) better answers.

Genetic Algorithms

– Invented by John Holland 1975
– Made popular by John Koza 1992

Background...

- Evolution
  - Organisms (animals or plants) produce a number of offspring which are almost, but not entirely, like themselves.
  - Extinction and Adaptation

    Some of these offspring may survive to produce offspring of their own—some won't
    - The "better adapted" offspring are more likely to survive
    - Over time, later generations become better and better adapted
  - Genes and Chromosomes
    - Genes – "instructions" for building an organism
    - Chromosomes – a sequence of genes
  - **Genetic Algorithms** use this same process to "evolve" better programs

Genetic Algorithm Concept

- Genetic algorithm (GA) introduces the principle of evolution and genetics into search among possible solutions to given problem.
- This is done by the creation within a machine of a population of individuals represented by chromosomes, in essence a set of character strings, that are analogous to the DNA, that we have in our own chromosomes.

Survival of the Fittest

- The main principle of evolution used in GAs is "survival of the fittest".
  - The good solution survive, while bad ones die.

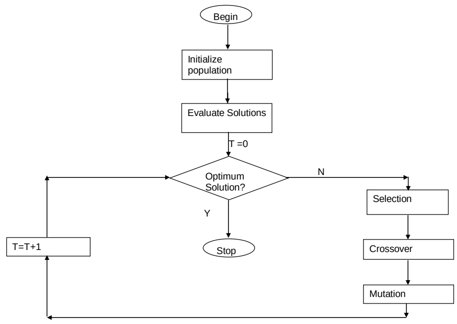
So what *is* a genetic algorithm?

- Genetic algorithms are a randomized heuristic search strategy.
- Basic idea: Simulate natural selection, where the population is composed of *candidate solutions*.
- Focus is on evolving a population from which strong and diverse candidates can emerge via mutation and crossover (mating).

Basic algorithm

- Create an initial population, either random or "blank".
- While the best candidate so far is not a solution:
  - Create new population using successor functions.
  - Evaluate the fitness of each candidate in the population.
- Return the best candidate found.

Flowchart of a Genetic Algorithm



13

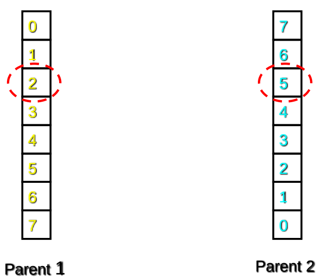
Crossover

- Crossover is the similar to natural reproduction.
- Crossover combines genetic material from two parents,
  in order to produce superior offspring.
- Few types of crossover:
  - One-point
  - Multiple point.

14

Crossover

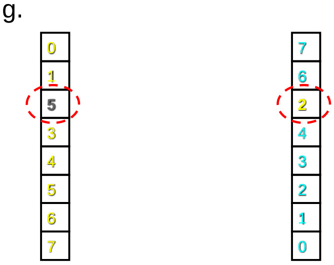
- E.g.



15

Crossover

- E.g.



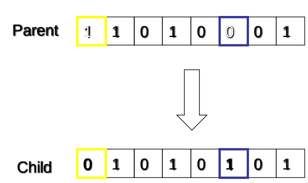
16

Mutation

- Mutation introduces randomness into the population.
- Why ‘Mutation’
  - The idea of mutation is to reintroduce divergence into a converging population.
- Mutation is performed on small part of population,
  in order to avoid entering unstable state.

17

Mutation



18

Fitness Function

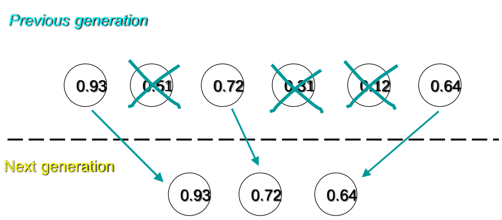
- Fitness Function is the evaluation function that is used to evaluated the solutions and find out the better solutions.
- Fitness of computed for each individual based on the fitness function and then determine what solutions are better than others.

Selection

- The selection operation copies a single individual, probabilistically selected based on fitness, into the next generation of the population.
- Several possible ways
  - Keep the strongest
  - Keep some of the weaker solutions

Selection

Survival of The Strongest



Stopping Criteria

- Final problem is to decide when to stop execution of algorithm.
- Two possible ways.
  - First approach:
    - Stop after production of definite number of generations
  - Second approach:
    - Stop when the improvement in average fitness over two generations is below a threshold

Simple example – alternating string

- Let's try to evolve a length 4 alternating string
- Initial population: C1=1000, C2=0011
- We roll the dice and end up creating C1' = cross (C1, C2) = 1011 and C2' = cross (C1, C1) = 1000.
- We mutate C1' and the fourth bit flips, giving 1010. We mutate C2' and get 1001.
- We run our solution test on each. C1' is a solution, so we return it and are done.

Basic components

- Candidate representation
  - Important to choose this well. More work here means less work on the successor functions.
- Successor function(s)
  - Mutation, crossover
- Fitness function
- Solution test
- Some parameters
  - Population size
  - Generation limit

Candidate representation

- We want to encode candidates in a way that makes mutation and crossover easy.
- The typical candidate representation is a binary string. This string can be thought of as the genetic code of a candidate – thus the term "genetic algorithm"!
  - Other representations are possible, but they make crossover and mutation harder.

Candidate representation example

- Let's say we want to represent a rule for classifying bikes as mountain bikes or hybrid, based on these attributes:
  - Make (Bridgestone, Cannondale, Nishiki, or Gary Fisher)
  - Tire type (knobby, treads)
  - Handlebar type (straight, curved)
  - Water bottle holder (*Boolean*)
- We can encode a rule as a binary string, where each bit represents whether a value is accepted.

Make				Tires		Handlebars		Water bottle	
B	C	N	G	K	T	S	C	Y	N

Candidate representation example

- The candidate will be a bit string of length 10, because we have 10 possible attribute values.
- Let's say we want a rule that will match any bike that is made by Bridgestone or Cannondale, has treaded tires, and has straight handlebars. This rule could be represented as 1100011011:

Make				Tires		Handlebars		Water bottle	
1	1	0	0	0	1	1	0	1	1
B	C	N	G	K	T	S	C	Y	N

Successor functions

- Mutation – Given a candidate, return a slightly different candidate.
- Crossover – Given two candidates, produce one that has elements of each.
- We don't always generate a successor for each candidate. Rather, we generate a successor *population* based on the candidates in the current population, weighted by fitness.

Successor functions

- If your candidate representation is just a binary string, then these are easy:
  - Mutate(c): Copy c as c'. For each bit b in c', flip b with probability p. Return c'.
  - Cross (c1, c2): Create a candidate c such that c[i] = c1[i] if i % 2 = 0, c[i] = c2[i] otherwise. Return c.
    - Alternatively, any other scheme such that c gets roughly equal information from c1 and c2.

Fitness function

- The fitness function is analogous to a heuristic that estimates how close a candidate is to being a solution.
- In general, the fitness function should be consistent for better performance. However, even if it is, there are no guarantees. This is a <u>probabilistic</u> algorithm!
- In our classification rule example, one possible fitness function would be information gain over training data.

Solution test

- Given a candidate, return whether the candidate is a solution.
- Often just answers the question “does the candidate satisfy some set of constraints?”
- Optional! Sometimes you just want to do the best you can in a given number of generations, e.g. the classification rule example.

31

New population generation

- How do we come up with a new population?
  - Given a population P, generate P’ by performing crossover |P| times, each time selecting candidates with probability proportional to their fitness.
  - Get P’’ by mutating each candidate in P’.
  - Return P’’.

32

New population generation

- That was just one approach – be creative!
  - That approach doesn’t explicitly allow candidates to survive more than one generation – this might not be optimal.
  - Crossover is not necessary, though it can be helpful in escaping local maxima. Mutation *is* necessary (why?).

33

Basic algorithm (recap)

- Create an initial population, either random or “blank”.
- While the best candidate so far is not a solution:
  - Create new population using successor functions.
  - Evaluate the fitness of each candidate in the population.
- Return the best candidate found.

34

Pros and Cons

- Pros
  - Faster (and lower memory requirements) than searching a very large search space.
  - Easy, in that if your candidate representation and fitness function are correct, a solution can be found without any explicit analytical work.
- Cons
  - Randomized – not optimal or even complete.
  - Can get stuck on local maxima, though crossover can help mitigate this.
  - It can be hard to work out how best to represent a candidate as a bit string (or otherwise).

35

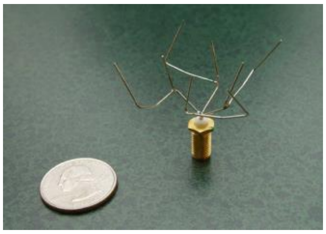
Examples - GA in the wild

- Rule set classifier generation
- I tried this as an undergrad, and it worked pretty well. Rather than classifying bikes, I was classifying Congressional representatives by party, based on their voting records.
- General approach:
  - Use GA to generate a rule for training data, with information gain for the fitness function and no solution test (just a generation limit).
  - Remove positive examples covered by the rule.
  - Repeat the above two steps until all positive training examples are covered.
  - To classify an example: iff the example is matched by any of the rules generated, consider it a positive example.

36

Examples - GA in the wild

- ST5 Antenna – NASA Evolvable Systems Group



http://ti.arc.nasa.gov/projects/esg/research/antenna.htm

37

ST5 Antenna

- Needs less power than standard antennae.
- Doesn't require a matching network nor a phasing circuit – two less steps in design and fabrication.
- More uniform coverage than standard antennae, yielding more reliable performance.
- Short design cycle – 3 person-months to prototype fabrication, vs. 5-person months on average for conventionally designed antennae.

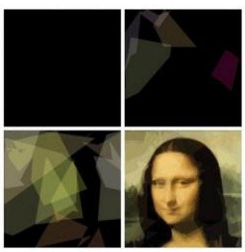
38

Examples - GA in the wild

- Image compression – evolving the Mona Lisa

Generation 1

Generation 301



Generation 2716

Generation 904314

http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/

39

Evolving the Mona Lisa

- Uses only 50 polygons of 6 vertices each.
- Population size of 1, no crossover – parent compared with child, and superior image kept.
- Assuming each polygon has 4 bytes for color (RGBA) and 2 bytes for each of 6 vertices, this image only requires 800 bytes.
- However, compression time is prohibitive and storage is cheaper than processing time. ☹

40