Introduction to Genetic Algorithms

Based on Chapter 10 of Marsland
Chapter 9 of Mitchell
Genetic Algorithms - History

- Pioneered by John Holland in the 1970s
- Became popular in the late 1980s
- Based on ideas from Darwinian Evolution
- Can be used to solve a variety of problems that are not easy to solve using other techniques
Motivation: Evolution in the real world

- Each cell of a living thing contains **chromosomes** - strings of DNA
- Each chromosome contains a set of **genes** - blocks of DNA
- Each gene determines some aspect of the organism (like eye color)
- A collection of genes is sometimes called a **genotype**
- A collection of what can be “seen” (like eye colour) is sometimes called a **phenotype**
- Reproduction involves recombination of genes from parents and then small amounts of **mutation** (errors) in copying
- The **fitness** of an organism is how much it can reproduce before it dies
- Evolution based on “survival of the fittest”
## Some GA applications

<table>
<thead>
<tr>
<th>Domain</th>
<th>Application Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>gas pipeline, pole balancing, missile evasion, pursuit</td>
</tr>
<tr>
<td>Design</td>
<td>semiconductor layout, aircraft design, keyboard configuration, communication networks</td>
</tr>
<tr>
<td>Scheduling</td>
<td>manufacturing, facility scheduling, resource allocation</td>
</tr>
<tr>
<td>Robotics</td>
<td>trajectory planning</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>designing neural networks, improving classification algorithms, classifier systems</td>
</tr>
<tr>
<td>Signal Processing</td>
<td>filter design</td>
</tr>
<tr>
<td>Game Playing</td>
<td>poker, checkers, prisoner’s dilemma</td>
</tr>
<tr>
<td>Combinatorial Optimization</td>
<td>set covering, travelling salesman, routing, bin packing, graph colouring and partitioning</td>
</tr>
</tbody>
</table>
Start with a Dream…

• Suppose you have a difficult problem
• You don’t know how to solve it
• What can you do?
A dumb solution

A “blind generate and test” algorithm:

Repeat

  Generate a random possible solution, something that looks like a solution!
  Test the solution and see how good it is

Until solution is good enough
Can we use this dumb idea?

• Sometimes - yes:
  – if there are only a few possible solutions
  – and you have enough time
  – then such a method *could* be used

• For most problems - no:
  – many possible solutions
  – with no time to try them all
  – so this method *cannot* be used in general
A “less-dumb” idea (GA)

Generate a set of random solutions
Repeat
  Test each solution in the set (rank them)
  Remove some bad solutions from set
  Duplicate some good solutions
    make small changes to some of them
Until best solution is good enough
How do you encode a solution? i.e., write the solution in the program?

- Obviously this depends on the problem!
- GAs *may* encode solutions as fixed length “bitstrings” (e.g. 101110, 111111, 000101)
- Each bit represents some aspect of the proposed solution to the problem
- Decimal or other representations are also possible.
- For GAs to work, we need to be able to “test” any string and get a “score” indicating how “good” that solution is; i.e., we need a heuristic evaluation function. This is called the *fitness function*. 
Silly Example - Drilling for Oil

• Imagine you had to drill for oil somewhere along a single 1km desert road
• Problem: choose the best place on the road that produces the most oil per day
• We could represent each solution as a position on the road
• Say, a whole number between [0..1000]
Where to drill for oil?

Randomly, or with some fore knowledge, say we have two “solutions” at 300m and 900m to start with. We will start with a population of 2 solutions and try to make them better. At any “solution”, we should be able to perform some experiments/collection data to evaluate the “solution”.

Solution1 = 300

Solution2 = 900
Digging for Oil

• The set of all possible solutions [0..1000] is called the *search space* or *state space*.

• In this case it’s just one number but it could be many numbers or symbols.

• Often GAs code numbers in binary producing a bit string representing a solution.

• In our example we choose 10 bits which is enough to represent 0..1000.
In GAs these encoded strings are sometimes called "genotypes" or "chromosomes" and the individual bits are sometimes called "genes"
Drilling for Oil

Solution 1 = 300
(0100101100)

Solution 2 = 900
(1110000100)

30 and 5 are “scores” assigned to the “solutions”
Searching for Oil: One Generation at a time

- Starting population of size 2: (01001001100, 1110000100)
- In this case, we see that one of our solutions (01001001100) has a score (fitness function value) of 30 and the other (1110000100) a fitness of 5.
- So, following our algorithm, we can remove the bad solution from our population of size 2.
- We can also duplicate the good solution that has a fitness of 30.
- This leads to a population: (01001001100, 01001001100) where both solutions are the same.
- Maybe, now we can make small changes (mutate) one of the solutions (say the second) by changing one single bit.
- Our population at the beginning of the second generation becomes (say): (01001001100, 01101001100)
Summary

We have seen how to:

• Represent possible solutions as a number
• Encode a number into a binary string
• Generate a score for each number given a function of “how good” each solution is - this is often called a fitness function
• We also saw how to create a new generation of solutions.
• Our silly oil example is really optimization over a function $f(x)$ where we perform search over $x$
Search Space

- For a simple function $f(x)$ the search space is one dimensional.
- But by encoding several values into the chromosome many dimensions can be searched e.g., two dimensions $f(x,y)$
- Search space can be visualized as a surface or fitness landscape in which fitness dictates height
- Each possible genotype is a point in the space
- A GA tries to move the points to better places (higher fitness) in the space
Fitness landscapes
Search Space

• Obviously, the nature of the search space dictates how a GA will perform
• A completely random space would be bad for a GA
• Also GAs can get stuck in local maxima if search spaces contain lots of these
• Generally, spaces in which small improvements get closer to the global optimum are good
Back to the (GA) Algorithm

Generate a set of random solutions

Repeat
  Test each solution in the set (rank them)
  Remove some bad solutions from set
  Duplicate some good solutions
    make small changes to some of them

Until best solution is good enough
Adding Sex - Crossover

- Although it may work for simple search spaces our algorithm is still very simple
- It relies on random mutation to find a good solution
- It has been found that by introducing “sex” into the algorithm better results are obtained
- This is done by selecting two parents during reproduction and combining their genes to produce offspring
Adding Sex - Crossover

• Two high scoring “parent” bit strings (chromosomes) are selected and combined with some probability (crossover rate) to produce two new offspring (bit strings)
• Each offspring may then be changed randomly (mutation)
Selecting Parents

- Many schemes are possible so long as better scoring chromosomes more likely selected
- Score is often termed the \textit{fitness}
- “Roulette Wheel” selection can be used:
  - Add up the fitnesses of all chromosomes
  - Generate a random number R in that range
  - Select the first chromosome in the population that - when all previous fitnesses are added - gives you at least the value R
# Example population

<table>
<thead>
<tr>
<th>No.</th>
<th>Chromosome</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1010011010</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1111100001</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1011001100</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1010000000</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0000010000</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>1001011111</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>0101010101</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1011100111</td>
<td>2</td>
</tr>
</tbody>
</table>
Roulette Wheel Selection

Rnd[0..18] = 7
Chromosome4
Parent1

Rnd[0..18] = 12
Chromosome6
Parent2
Roulette wheel selection

The roulette wheel is divided into sections, each representing a different selection point. The size of each section indicates the probability of selection:

- Section 3: 38% (Fittest individual has largest share of the roulette wheel)
- Section 2: 5% (Weakest individual has smallest share of the roulette wheel)
- Section 1: 31%
- Section 4: 12%
- Section 5: 14%

The wheel is rotated to select an individual based on these probabilities.
Crossover - Recombination

With some high probability (crossover rate) apply crossover to the parents. (typical values are 0.8 to 0.95)
With some small probability (the mutation rate) flip each bit in the offspring (typical values between 0.1 and 0.001)
Back to the (GA) Algorithm

Generate a population of random chromosomes

Repeat (each generation)
  Calculate fitness of each chromosome
  Repeat
    Mutate some of the solutions
    Use (roulette) selection to select pairs of parents
    Generate offspring with crossover
  Until a new population has been produced

Until best solution is good enough
The GA cycle

1. Start
2. Initialize Population
3. Randomly vary individuals
4. Evaluate Fitness
5. Apply Selection
6. Stop
genetic algorithm learning

The average fitness of the population usually rises, till a limit.
Many parameters to set

- Any GA implementation needs to decide on a number of parameters: Population size (N), mutation rate (m), crossover rate (c)
- Often these have to be “tuned” based on results obtained - no general theory to deduce good values
- Typical values might be: $N = 50$, $m = 0.05$, $c = 0.9$
Applications of GAs

• We saw a toy example of GAs in this presentation.
• Gas have been used in thousands of real-life applications.
• http://neo.lcc.uma.es/TutorialEA/semEC/cap03/cap_3.html
• http://brainz.org/15-real-world-applications-genetic-algorithms/
Applications of GAs

• Optimizing list of parameters for aircraft design
• Optimize routing of telephone networks
• Planning the path of a robot arm from one point to another.
• Modeling how international actors may behave in situations like war and peace
• Finding how a simulated aircraft can evade simulated missiles.
• Designing configuration of a neural network.
• Scheduling activities in a laboratory; the activities interact in various ways.
• Designing software keyboards for a variety of languages.
Many Variants of GA

• Different kinds of selection (not roulette)
  – Tournament: Pick two random chromosomes, evaluate them, keep the better one
  – Elitism: Keep a few of the best ones from generation to generation unaltered

• Different recombination
  – Multi-point crossover
  – 3 way crossover

• Different kinds of encoding other than bit string
  – Integer values
  – Ordered set of symbols

• Different kinds of mutation: random, uniform, Gaussian