**October 23, 2019**

Biology Terms related to GA:

**Zygote**- single cell formed after fertilization

**Mitosis**- cloning process. Parent clones into multiple offspring. Cloning is safe, but there’s not enough diversity in the pool, so no new material. We need genetic diversity/variety for GA.

**Meiosis**- split into two gametes

**Ploidy** – how many strands of DNA combined into one organism

Related to ply (layers)

Humans are biploid/diploid – 2 strands of DNA

**Gametes** are haploid (half) – so 2/2=1 in humans

**Monoploid**- single strand of DNA for entire organism

**Multiploid**- multiple strands of DNA, 3 or more, exists in plant life

Introduction to The Genetic Algorithm

The Genetic Algorithm is a subset of algorithms that are modeled off natural processes. In this case, the algorithm is modeled of off genetics. In order to understand the process of The Genetic Algorithm, it is helpful to define the terminology that is associated with genetics. We will define the terms in both their biological definition and their definition in terms of The Genetic Algorithm.

The most basic term involved in genetics is the gene. The gene is a marker that is used to express a specific feature. In nature these features can range from anything from eye color to height. In terms of The Genetic Algorithm, the gene is data that represents something. This can range from a bit that represents whether an individual has a tail or not or it can be a string value that stores the eye color of an individual.

Having a single gene can only represent so much. Individual organisms have multiple features that define them. In order to achieve this, we need multiple genes. The sequence of all the genes in an organism is called the genome. The computer representation of a genome is an array. Let’s say we have an organism that has a binary genome consisting of 7 genes. On a computer this will look like:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |

There are multiple ways to look at the genome of an organism. There is the genotype, which is the act of viewing the genome from the genetic perspective. On the other hand, there is the phenotype, which is the act of viewing the genome for the expressed features perspective. For example, let’s say we have an organism that’s just a tail. The only two features are the tail color and tail length. Let’s say the tail can be red, black, white, and grey and the tail length can be 0 inches to 8 inches long. The genotype perspective for this organism would be as follows:

Using a binary genome, we need 2 bits to represent the 4 tail colors, mapped as 00 for red, 01 for black, 10 for white, and 11 for grey. We need 3 bits to represent the tail length. Let’s use an integer array of size 5 to represent the genome. The first 2 values are the tail color and the last 3 values represents the tail length. This is an arbitrary distinction; the last 2 values can be the tail color and the first 3 can be the tail length. Assuming that positions 0-1 is tail color and 2-4 is the tail length, the array will look like:

Tail Color | Tail Length

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 1 |

The phenotype view of this would be an array of values. The values can be anything but in this case it can be a string for the tail color, or it can be an integer for the tail length. It will look like:

Tail Color Tail Length

|  |  |
| --- | --- |
| Red | 3 |

Both of these views represent the same organism. The binary genome of the organism is an example of an allele. Alleles are a combination of genes variations that express a feature. The tail color feature is expressed as an allele of the first and second gene in the genome.

The natural question that this arises is that which view is better for the Genetic Algorithm. Well that depends on the problem that the algorithm is trying to solve. Genotype and phenotype both have their strength and weaknesses. The Genetic Algorithm is a way of searching for solutions to a problem. The different variations of solutions is called the search space. The Genetic algorithm picks a solution that is represented as a genome from the search space.

Let’s say that when picking a solution from the search space, the Genetic Algorithm changes every gene in the genome once. If the genome is represented as a phenotype then the tail color will change once, and the tail length will change once. If the genome is represented as a genotype, then the tail color will change twice as both the genes associated with it changes. The tail length will change three times because the three genes associated with it changes. Based on the problem, you might want either of these two scenarios. As a rule, genotype representation allows for more diversity in the solutions that your choosing. This prevents the algorithm from staying in one corner of the search space. More diversity in the area of the search space that the algorithm is searching means you’re more likely to find the right solution to the problem. However, if your close to the correct solution you don’t want the Genetic Algorithm to go to the complete opposite side the search space. This is when phenotypes are useful.

This raises the question of how the Genetic Algorithm moves through the search space. This is where the reproductive part of genetics comes in to play. To move through the search space, the algorithm uses different methods of gene crossover and mutation. In biology, crossover is the exchanging of genetic materials. For the algorithm, this means you can take two organisms with the same genome structure and make a child that has some of the genes from one parent and some of the genes from the other parent.

Before we get deeper into crossover, we need to define the concept of ply. A ply in biology is layers of genes that compose an organism. Our tail organism from before has one layer of genes so it is a monoploid. Humans are diploid and that is represented as a 2D array on a computer. There are organisms in nature that have more that 2 layers of genes and those are called multiploidy.

One way of crossover that is seen in nature is the process of meiosis. Meiosis is the process of splitting genetic material into two separate layers. So, a genome that looks like this,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |

will become,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 |

and

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 |

by the processes of meiosis. The resulting 2 layers of genes create in this example are 2 haploids, which are just a single layer of genes that was created by meiosis.

The counterpart to meiosis is mitosis. Mitosis is the process of cloning genetic materials. On a computer this would just be copying the contents of one parent to a child. This can be useful in the Genetic Algorithm because if you have a potential solution that looks correct then the correct solution might be around the same area in the search space. This isn’t true all the time though, sometimes a good-looking potential solution might be on the opposite side of the search space of the actual solution. Mitosis doesn’t create the variability that is needed for the Genetic Algorithm to give good solutions.

The other way that the Genetic Algorithm traverses the search space is by mutation. Mutation is the random changing of genes. In nature, mutation is a devastating process that can lead to illness. Mutation is only good in the small chance that changes to the genes manifest an advantageous trait in the organism. On a computer, mutation is just the random changing of the genome array value. And just like in nature, it can be devastating to finding optimal solutions. Too much mutation and the algorithm is jumping all over the search space even if it was in the vicinity of the solution. Too little mutation and the algorithm is stuck on one corner of the search space with the solution possibly being on the other side.

With the basic terms in mind, the Genetic Algorithm has some basic properties associated with it.

The first is the concept of randomness. A good Genetic Algorithm needs a good random number generator. The better the random number generator the more of the search space is available to the algorithm.

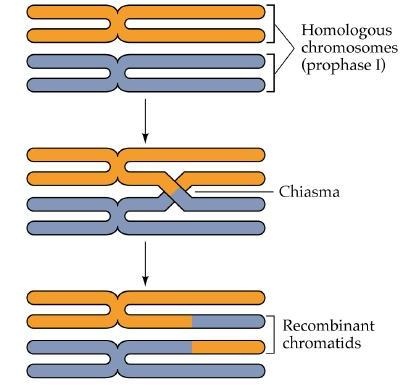
The second is the fitness function. The algorithm needs a guiding hand to point it in the right direction. Without it the algorithm would run forever without knowing if it found a good solution. The fitness function should have a concept of what is a good solution and should terminate when a potential solution is in that threshold. It should also have a cap on the number of iterations that the algorithm will take. Otherwise the function might take forever to find a solution that’s within the given threshold.

Dominant vs. Recessive genes are not user in Genetic Algorithm

crossover – exchange of genetic materials (swapping)

a genetic operator that is used to vary the programming of one or more chromosomes from one generation to the next one. It’s the process of taking at least two parent solutions and producing a child solution from them.

Source: <http://en.wikipedia.org/wiki/Crossover_%28genetic_algorithm%29>



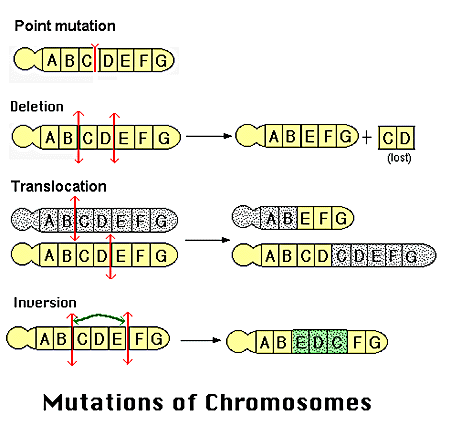
Source: <http://www.biology.duke.edu/noorlab/projects.html>

mutation – change of genetic value

A change in the genomic sequence. Mutations are random and can be defined as sudden and spontaneous changes in the cell. They can have various effects on the cells that contain them such as no effect at all, an inability for the cell to function properly or at all and also mutations can alter the product of a gene.

Source: <http://en.wikipedia.org/wiki/Mutation>

In genetic algorithms, a mutation, although random, is coded. They do not just happen by themselves in programs.



Source: <http://www.accessexcellence.org/RC/VL/GG/mutation.php>

This is just the basic form of the Genetic Algorithm. There many parts of the algorithm that can be changed to suit the problem that is being solved. There are many different areas of biology that can be useful to solve a problem. One example is of Barbara McClintock who discovered that gene position in the genome isn’t necessarily static. Even though in these notes genes are shown to be static positions in an array, some problems might call of position 0 meaning one thing but then making position 1 mean that instead. There are tons of different variations of the Genetic Algorithm ranging from ant colony to swarm optimization. The Genetic Algorithm is a purposely designed for addition based on need.

**Keywords:** Zygote, Mitosis, Meiosis, Ploidy, Biploid/Diploid, Gametes, Monoploid, Multiploid

**October 28,2019**

DAVID GOLDGERG: David Edward Goldberg is an American computer scientist, civil engineer and former professor. He was the director of the Illinois Genetic Algorithms Laboratory (IlliGAL). He is author of Genetic Algorithms in search, Optimization and Machine Learning (1989), one of most cited books in computer science.

JOHN HOLLAND: John henry Holland was an American scientist and professor of electrical engineering and computer science at the University of Michigan, Ann Arbor. He was the first person who introduce the Genetic Algorithm.

Working of Genetic Algorithm:

GA works on a population consisting of some solutions where the population size is the number of solutions. Each solution is called individual. Each individual solution has a chromosome. The chromosome is represented as a set of parameters that defines the individual. Each chromosome as a set of genes. Each gene is represented by somehow such as being represented as string of 0s and 1s.

Three prep work to do before you apply GA to a problem:

DESIGN PHASE à

Step 0: Preprocessing (before compile)

You need a data structure to :

1. Store solution of problem (dynamic data)
2. Store problem data (static data)
3. GA related data (Goldberg’s original version: Static ; Adaptive-dynamic)

Step1 : Parameters : Population size , RNG seeds , max-gen, max/min-fitness(tolerance)

( in a parameter file ). [Add preserve best solution]

If you have your GA code working, you won’t want to change it. Do not have to recompile it, we keep our parameter file.

Step 2: Generate Initial population ( Generation 0)

Step 3 : Stopping / Termination criteria

(if GA=false, don’t move on to the next step)

Step 4: Selecting (potential) mates (parents):

There are four typical approaches for selection:

**a. Random** – Can be one stage or two stages. It's the simplest version. The role of fitness is irrelevant. Just pick your pool who mates. pick your parents. In combinatorics theory, the notion of replacement. It's a simple question. If 2 parents have a child, can either of a parent have a child with someone else? Yes, it can remarry. Genetic Pressure – Practical look at the gene pool. Based on statistics of current gene pool. Natural Selection - Isolationist. No migration patterns inbreeding if limited resources. This is what Darwin meant when he said, "survival of the fittest". 1st stage, who's eligible to have children? Only top X % only are eligible to have children (typically 10%).

Mike Anshel - 1. (Pseudo) Random Number Generator (RNG)

2. Fitness function – is a percentile of individual evaluation score

3. Constraints: soft vs hard

**b. Roulette Wheel** - The point is to give everybody a chance. Sorting evaluation function of everyone from smallest to largest

1.For pedagogical purposes, assume searching for max fitness

2. Assume fitness function is meaningful (reasonable distribution)

3. Probability of selecting an individual directly correlates to its fitness.

**c. Greedy (elitism)**: The Greedy/elitist approach could improve some results in the short term. If you have a smart person marrying a smart person, or a fast person marrying another fast person then probably, their kids will have inherited those traits from both. However, this approach causes a narrowing of the genetic drift.

**d. (k-) tournament** - Randomly select an individual or 1 by 1. According to sum, you'll allow everyone or some. You must select enough pool mates. The question is, who? What's the competition score.

Step 0) One by one let every individual compete "current champion".

Step 1) Choose K contenders randomly 1 at a time. The individual with higher fitness wins the round. Your fitness score is 0-k (zero to k). individual with higher fitness wins the round. Genetic

Variation:

Genotype: The genotypes the set of genes in our DNA which is responsible for a trait.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| hair color | height | gender | ………… | ……….. | ……….. |

Phenotype: The phenotype is the physical expression, or characteristics, of that trait.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 bits | 16 bits | 2 bits | ……………. | ……………. | ……………. |

**Crossover:** Crossing over occurs between prophase 1 and metaphase 1 and is the process where homologous chromosomes pair up with each other and exchange different segments of their genetic material to form recombinant chromosomes.

**Process vs. Algorithm**

Process: A series of events to produce a result, especially as contrasted to product.

Algorithm: A precise step-by-step plan for a computational procedure that possibly begins with an input value and yields an output value in a finite number of steps.

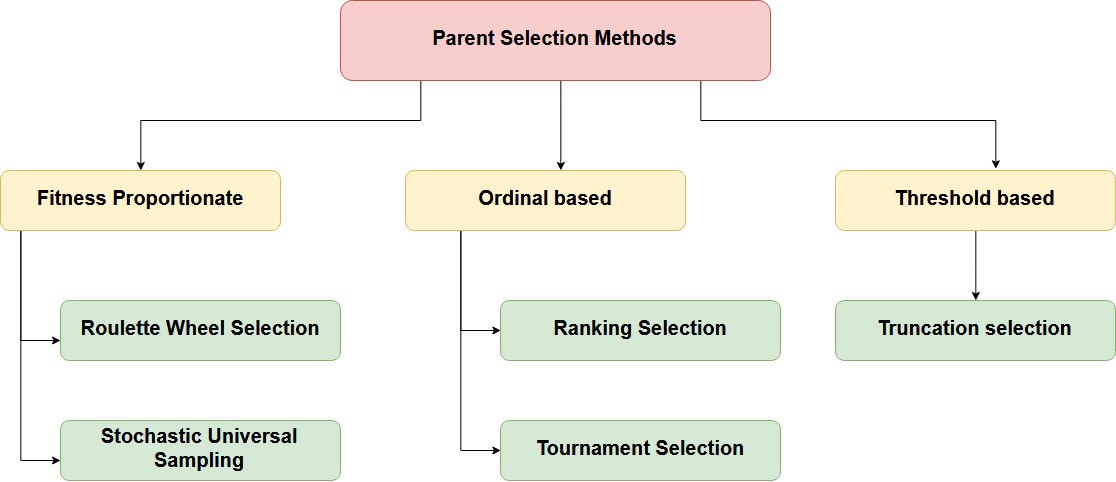
The potential role of sorting the population based on fitness.

One could differentiate subsets of population:

1. Who potentially are parents
2. Who actually are parents

**Keywords:** random, roulette wheel, greedy(elitism), k-tournament, crossover, process, algorithm

**October 30, 2019**

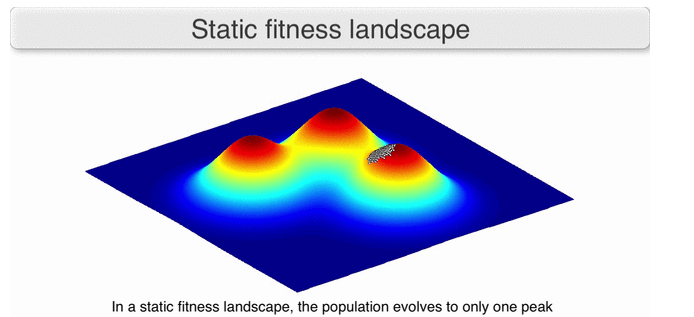


**Continuation from last Class**

**Step 4: Selecting Parents**

Parent Selection is the process of selecting parents which mate and recombine to create offsprings for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

A tool that is used to analyze the progression of a given method is the fitness landscape, which maps either fitnesses of all possible genotyped chromosomes, or fitnesses of all individuals of a given population (2d) and possibly over all generations (3d).



https://en.wikipedia.org/wiki/Fitness\_landscape

**Random Selection:**

Fitness values are ignored. Each individual of the population has equal chances of being selected. Research has not found that this approach yields the best results.

**“Roulette Wheel” method for selection:**

Fitness Proportionate Selection (“roulette wheel”) is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

Consider a circular wheel. The wheel is divided into n pie slices, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value.

In a roulette wheel selection, the circular wheel is divided as described above. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.

Roulette Wheel (used in schema theorem) “width” of slice (# of chances of selection ~ the fitness value prop.) For example, Let Ci = Chromosome (Parent) i; where i >= 1

We assume Ci values as follows,

C1 = 71 C1 = 40%

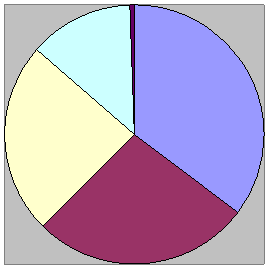
C2 = 54 C2 = 31%

C3 = 47 C3 = 27%

C4 = 3 C4 = 1.5%

C5 = 1 C5 = 0.5%

Total fitness of Chromosome(parent) = 176



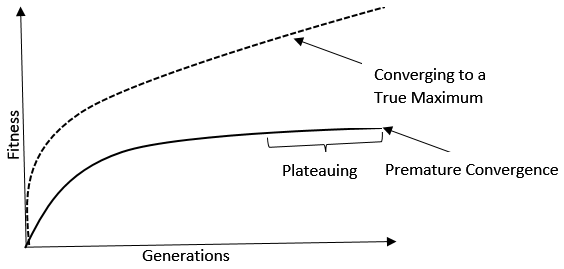
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

From the example of Roulette Wheel, we assume

1. Max is better
2. > 0, all fitnesses are greater than zero
3. (implicit) ~ linear scale of fitness value (otherwise the higher values could be way out and not comparable to the remaining values)

**Greedy (elitism):** top % (in literature, typically 10%)

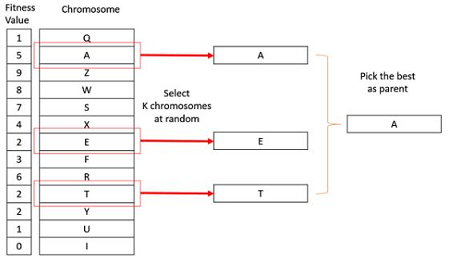
In genetic algorithms, elitism involves selecting a small proportion of the fittest candidates, to be the parent pool of the next generation. If the parents are then randomly chosen from the selected pool, this is known as “top scaling”. (Memetic approaches allow for the best individual to be cloned once into the next generation, but that is not considered greedy. It is done because a genetic algorithm can lose a good solution in subsequent generations if it is not maintained.)



In practice, greedy may cause a rise of average/max fitness in the short-term but typically will end up with a suboptimal (“premature convergence”) result because there will be little exchange of genetic material amongst the individuals selected due to the restrictive nature of elitism. The arrow in the above drawing shows the “plateauing” of the max fitness value (ie that at some point, there isn’t much improvement) where the dashed curve represents the optimal growth of the genotype.

**K-tournament selection method:**

In K-Way tournament selection, we select K individuals from the population at random and allow them to compete; typically, by selecting the best fitness out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular

in literature as it can even work with negative fitness values. 

**Source:** <https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm>

**Tournament Parameters:**

1. How many can sign-up
2. How many compete per game “K”
3. How many competitions amongst players?
4. Winning a game is based on fitness.

Winning the tournament is based on # of games won (not fitness). However, ties are won based on fitness, else random (or default champion). If lowest performers of a population all have the same fitness, then this tournament approach will encourage selection of some of these since if all performers have the same tournament value, then default champion will be selected and the default champion will be at some point one of these low performers.

**Scaling**: is technique which can be applied at initial stage of evolution and can highly affect the results. It helps to generate more optimized results by eliminating premature convergence. Fitness scaling converts the raw fitness scores that are returned by the relative fitness function to their values in a that range that is suitable for the selection function. There are a number of different types of scaling functions that can be used in genetic algorithm. Here is one of them:

**Ranked Scaling:** This is a two steps process.

* 1. All individuals are sorted by their raw fitness scores (ranked).
  2. Their new fitness is based on their percentile rank.

Some researchers calculate the ranked fitness as a function of rank, size of population and a parameter describing desired pressure such as the ratio of (best fitness/median)

**Keywords:** random selection, roulette wheel, greedy(elitism), k tournament selection method, scaling, ranked scaling

**November 04, 2019**

**Step 5:**

**Crossover**: Swapping parts of the solution with another in chromosomes or solution representations.

1. Two questions: Where is the crossover point(s)?

Ans: Crossover can have up to n points(1,2,,,,n).In one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new two off-springs.

1. Should you do it?

Ans: Our algorithm can opt out of performing crossover, and otherwise only do mutation

Crossover Rate: This is simply the chance that two chromosomes will swap their bits.

Mostly say this rate is(3% -5%) but other people say this as ( 5%-7%) as well.

1-point Crossover:

the same crossover point is selected in both parent and the tails of its two parents are swapped to get new two off-springs.

Parent1: 0 1 1 0 1 1 1 0

Parent2: 1 0 1 0 1 1 1 1

Offspring1: 0 1 1 0 1 1 1 1

Offspring2: 1 0 1 0 1 1 1 0

2-point Crossover: Two random points are chosen on the individual chromosomes (strings) and the genetic material is exchanged at these points.

Parent1: 0 1 1 0 1 1 1 0

Parent2: 1 0 1 0 1 1 1 1

Offspring1: 0 0 1 0 1 1 1 0

Offspring2: 1 1 1 0 1 1 1 1

n-point Crossover:

We don’t divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it’ll be included in the offspring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.

Parent1: 0 1 1 0 1 1 1 0

Parent2: 1 0 1 0 1 1 1 1

Child1: 0 1 1 0 1 1 1 1

In humans, crossover happens at the end or boundaries of the DNA strands, not in the middle of the genes.

Human model: Phenotype crossover.

Introns: boundaries

Exons: Meaningful materials

Transposons - Transposon, class of genetic elements that can “jump” to different locations within a genome. Although these elements are frequently called “dancing genes,” they are always maintained in an integrated site in the genome

**Step 6: Mutation**

Mutation alters one or more gene values in a chromosome from its initial state.  Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search.

1. Where should it occur?

Make it decision bit by bit.

1. Does it occur?

Mutation rate: These are the rates refer to the chance of a mutation happening, not where or when it will happen. Rate is: (1%- 3%) or (3%-5%)

Example:

Parent: 0 1 1 0 1 1 1 0

Offspring: 0 1 1 0 0 1 0 0

**Step: Place final results into next generation population**

Errors that can occur:

1. Omission - offspring is missing genetic data (e.g. parents were permutations)
2. Comission - offspring has redundant data (e.g. parents were permutations)
3. GA purists say, and they say this often, there are two ways to handle crossover and mutation errors.

* Add a penalty in the fitness function, to those offspring’s that have errors.
* Get a better random number generator

1. Hybrid GA also offers solution to errors

Add respectful operators that are application specific, to make sure every individual is closed under application constraints.

Parent1: 1 2 3 4 5 6 7 8 9 10

Parent2: 8 2 9 1 6 5 3 4 7 10

If we do one-point crossover, we will get:

Offspring1: 1 2 3 4 5 6 7 8 9 10

Offspring2: 8 2 9 1 6 6 7 8 9 10

Therefore, we can see that in offspring1

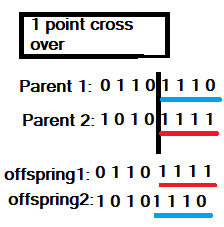
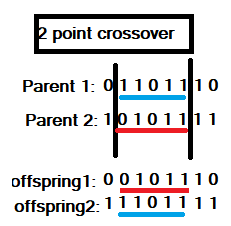
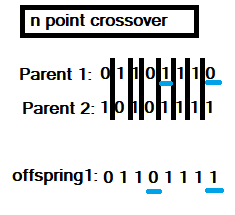
6,8,9 are missing and in offspring2

3,4,5 are missing

Respectful operators: it means result are closed under applications constraints.

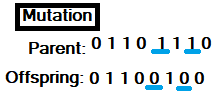
* CX- cycle crossover
* OX - order crossover
* PMX - partial match crossover

**Step 6a**: Cross Over

1. Where is the crossover joint?
   1. A cross over joint can have upto n points. A simple 1 point crossover has one point where the parent bit strings are split, and two different offsprings can be formed with those halves.
2. Should crossover be performed?
   1. Your algorithm can opt out of performing crossover, and only do mutations instead.
3. Crossover rate : (3% -5%) or ( 5%-7%)
   1. The meaning behind this will be explained in the upcoming lectures
4. Examples
   1. 
   2. 
   3. 
5. Crossover in humans happens at the ends or boundaries of the DNA strands, not between
   1. Introns: boundaries
   2. Extrons: Meaningful materials
   3. Transposons - “dancing genes”

**Step 6b**: Mutation

1. Where should it occur?
2. Should it occur?
3. Mutation Rate, (1%- 3%) or (3%-5%)
   * 1. Will be explained further in following lectures
     2. These are the rates refer to the chance of a mutation happening, not where or when it will happen.
4. Examples



**Step 7**: Place final results into next generation population

**Errors that can occur:**

1. Omission - offspring is missing genetic data
2. Co-omission - offspring has redundant data
3. GA purists say, and they say this often, there are two solutions to crossover and mutation errors.
   * 1. Add a penalty in the fitness function, to those offspring’s that have errors.
     2. Get a better random number generator
4. Hybrid GA also offers solution to errors
   * 1. Add respectful operators that are application specific, to make sure every individual is closed under application constraints

**Phenotype vs Genotype**

1. Phenotype is the trait expressed, for example, ‘John’ is a name. We can consider it as a trait. So, if we had a phenotype crossover between ‘John Doe’ and ‘Jane Shmoo’ we would offspring’s ‘John Shmoo’ and ‘Jane Doe’. Also regarded as the field name/value.
2. Genotype is the genetic material, or the bit string representation, obviously we can represent ‘John Doe’ in bit string, ASCII.
3. There is another representation, that is not bit string or field name, the char[] array. Which is debatable whether it is a phenotype or genotype.

**Keywords:** crossover, mutation, 1 point crossover, 2 point crossover, n point crossover, mutation, genotype, phenotype

**Notes 11/6/19**

Respectful operators for permutations

**Cyclic Crossover (CX)**

Parent 1: 1 2 3 4 5 6 7 8 9

Parent 2: 9 3 7 8 2 6 5 1 4

We choose position 1 from parent 1 randomly, and carry it to parent 2. Then we go to position in parent 1 with number that was replaced in parent 2. Carry that number over. Continue this until we’ve replaced the original number first copied to parent 2.

Child: **1** 3 7 **4** 2 6 5 **8 9**

**Partial Match Crossover (PMX)**

Parent 1: 1 2 3 |4 5 6 7| 8 9

Parent 2: 9 3 7 |8 2 6 5| 1 4

We choose two crossover points. Copy portion from parent 1 to parent 2. Now we have two 7’s and two 4’s: 9 3 **7** |**4** 5 6 **7**| 1 **4**

Replace the 4 outside the portion with what it replaced- which is an 8. We would replace the 7 outside the portion with what the 7 replaced which is a 5, but the 5 is already there, so we look at what the 5 replaced, which is a 2, so we replace the 7 with a 2.

Child: 9 3 **2** **4 5 6 7** 1 **8**

**Ordered Crossover (OX)**

Parent 1: 1 2 3 |4 5 6 7| 8 9

Parent 2: 9 3 7 |8 2 6 5| 1 4

We choose two crossover points. Use portion from first parent for the child. See which numbers in parent 2 are not already in the child: 9, 3, 8, 2, 1. Go to position right after copied portion in child. In parent 2, that is a 1, so start by copying 1 to child at that position and loop around: so add 1, 9 and then loop to front and add 3, 8, 2.

Child: 3 8 2 **4 5 6 7** 1 9

**Edge Recombination**

In edge recombination, we try to preserve the neighbor relationships between elements.

Parent 1 : G D M H B J F I A K E C

Parent 2: C E K A G B H I J F M D

We make chart of neighbors. If an element has the same neighbor in both parents we put a negative in front of it.

A: G, I, -K G: A, B, C, D

B: G, -H, J H: -B, I, M

C: D, -E, G I: A, F, H, J

D: C, G, -M J: B, -F, I

E: -C, -K K: -A, -E

F: I, -J, M M: -D, F, H

We start with letter with most negatives, in this case, E and K both have 2 negatives. We can start with either. Let’s say we choose E. We put E in the child. Then we look in E’s neighbors for the next element. We look to choose one of E’s neighbors that aren’t already in the child. Negatives get first priority. If there are two negatives, we choose alphabetically (closest to beginning of the alphabet). If no negatives, we just choose alphabetically. E has -C and -K, so we choose C. Then we look in C and choose D. Then look in D and choose M. This goes on until we are either done, or get stuck because all neighbors of an element are already in the child. When this happens we do what we did at the start. We look for which remaining element has the most negatives, or if tie choose alphabetically.

Child: E C D M F J B H I A K G

**Keywords:** cyclic crossover, partial match crossover, ordered crossover, edge recombination

**11/11/19**

**Mutation:**  
A gene mutation is a permanent alteration in the DNA sequence that makes up a gene, such that the sequence differs from what is found in most humans. Mutations variety in dimension; they are able to have an impact on wherever from a single DNA building block (base pair) to a huge segment of a chromosome that involves more than one gene. A mutation is an everlasting exchange somewhere in a DNA sequence. DNA is the unit of heredity of all organisms, so this means that mutations can most often be handed on to offspring. Mutations do not have got to be bad. Some are worthy, whilst others are impartial or don't have any effect.

There are different kinds of genetic mutation and they are: point mutation, inversion mutation, displacement mutation, heuristic mutation and heuristic crossover.

**Point Mutation:**

A point mutation is a mutation that only affects a single nucleotide of nucleic acid. Point mutations most commonly involve the substitution of one base for another (which changes the complementary base as well in DNA). The term point mutation also includes insertions or deletions of a single base pair. Point mutations rarely cause a huge change in the underlying organism, which is fortunate as they are by far the most common type of mutation.

**Inversion Mutation:**

An inversion is a [chromosome](https://en.wikipedia.org/wiki/Chromosome) [rearrangement](https://en.wikipedia.org/wiki/Chromosome_rearrangement) in which a segment of a chromosome is reversed end to end. An inversion occurs when a single chromosome undergoes breakage and rearrangement within itself. Inversions are of two types: paracentric and pericentric.

Paracentric inversions do not include the [centromere](https://en.wikipedia.org/wiki/Centromere) and both breaks occur in one arm of the chromosome. Pericentric inversions include the centromere and there is a break point in each arm.

**Displacement Mutation:**

Select two random points (i.e.; positions 4 and 6), grab the genes between them as a group, then reinsert the group at a random position displaced from the original.

0 1 2 3 4 5 6 7

becomes

0 3 4 5 1 2 6 7

**Heuristic Mutation:**

The heuristic information is used to guide local search, other strategies should be taken to keep the diversity of the firework swarm. Keeping a diversity of the swarm is crucial for the success of optimization procedure.

**Heuristic Crossover:**

The heuristic crossover operators use the fitness of the parents for leading the search process towards the most promising zones.

**GA vs GP:**

Genetic Algorithms (GAs) and Genetic Programming (GP) are often considered as separate but related fields. Typically, GAs use a fixed length linear representation, whereas GP uses a variable size tree representation. This paper argues that the differences are unimportant. Firstly, variable length actually means variable length up to some fixed limit, so can really be considered as fixed length. Secondly, the representations and genetic operators of GA and GP appear different, however ultimately it is a population of bit strings in the computer’s memory which is being manipulated whether it is GA or GP which is being run on the computer.

**The differences between genetic programming and genetic algorithms**

From a philosophical consideration: In genetic algorithms, you are breeding solutions from an algorithm. Whereas, in genetic programming, you are breeding the program (the actual code) or the means to the solution.

Data structure: In genetic algorithms, the default data structure is an array. Whereas, in genetic programming, the default, go-to data structure is a tree/graph since trees or graphs can be used to store logic. Earlier, flowcharts are used to store logic, through the use of visually displaying the relationships between different entities.

**What is a tree?**

* “**Abstract data type (ADT)** that represents a hierarchical tree structure, with a root value and subtrees of children with a parent node, often represented as a set of linked nodes”. – Wikipedia
* **Acyclic**: a graph without a cycle
* **Root**: unique pathway to every other node. Only one node can have that property.



Examples of graphs that are not a tree:

When referring to a node, we can assign certain attributes to it such as an indegree and outdegree. An indegree of a node is the number of edges that is directed/pointing to the node in question. An outdegree of a node is the number of edges that the node in question is pointing outwards to.

A root, mathematically, has an indegree of zero.

**Connectivity**

A graph is said to be connected if there exists a path between every pair of vertex.

Types of connectivity:

* **Strong**: (assumed by default) takes into account edge direction, where every vertex is reachable from every other vertex
* **Weak**: each vertex is connected to another vertex in either direction, where there is a subgraph that is unreachable from other nodes in the graph.
* **Uniform**: simply ignore the direction in which each vertex is pointing to.

**Directed Acyclic Graph (DAG)**: finite, directed graph with no cycles. A DAG is a directed graph that has a topological ordering, where every edge goes from earlier in the ordering to a vertex later in the ordering/sequence.

Every tree is a DAG but not every DAG is a tree

**Cycle:** contains a path from a node to itself, without using the same edge (distinct edges).

**Star graphs:** a tree with one internal node and k leaves.



**Important distinctions noted**:

In case of **undirected**, every node can be a root. A root is a convention that we use. A central property of an undirected is acyclic. Root is a corollary.

In case of **directed**, the root is the important property. Being acyclic or not does not determine whether or not a given graph is a tree (ex. DAG is not a tree). Cycle is a corollary.

In programming by default, you implement directed trees without parent node pointers. If you want to rely on parent node pointers, you use threaded trees.

**Threaded tree:** If you include parent node pointers only to access them but not because the edge exists.

Three category of trees, from an application perspective:

1. Data (assume every node has data)
2. Expression (only the leaves have data), the internal nodes have operators
3. Decision (only the leaves have data), the internal nodes have weights

**Keywords:** mutation, point mutation, inversion mutation, displacement mutation, heuristic mutation, genetic algorithm, genetic programming, abstract data type, acyclic, root, indegree, outdegree, strong, weak, uniform, directed acyclic graph, cycle, star graphs, undirected, directed, threaded tree

**November 13, 2019**

**Trees and Genetic Programming**

**Genetic Programming** is a way to breed code that solves a problem. The principle data structure for this is a tree. So, in order to understand Genetic Programming, we must first understand trees.

The nodes in a tree belong in one of two groups. There are the external nodes, which are the leaf nodes of the tree, and the internal nodes, which are the non-leaf nodes of the tree. The sequence of data in the leaf nodes listed from left to right is called the **frontier**. A disconnect set of trees is called a **forest**.

The nodes of a binary tree can also be grouped based on level, height, and depth. Level is top down grouping of the nodes in the tree starting at 1. The **root** is at level 1, its **children** are at level 2 and so forth. Height is a bottom up grouping of the nodes in the tree starting at 1. Each **leaf** node is at height 1 and its parent node is at height 2, and so forth. Depth is a top down grouping of the nodes starting at 1. The root is at depth 0, its children are at depth 1, and so forth.

Structurally, there are many different types of trees. The one that is usually encountered are called **m-ary trees.** These are trees where each node can have between 0 and m children where m is a positive integer. Binary trees are an example of m-ary trees where m is 2. The following trees are all Binary Trees:

There are 2 special types of binary trees that are useful for solving problems. The first type is a complete binary tree. This is a binary tree where each node either has 0 children or 2 children.

So this

is a **complete binary tree**, and this

isn’t.

The other type of binary trees is **balanced binary trees**. This is a binary tree where no leave node is more than one level apart. So this

is a balanced binary tree, but this isn’t.

It’s important to note that all complete binary trees are balanced binary trees but not all balanced binary trees are complete binary trees. There is also **AVL Trees** which are balanced binary trees that maintain the property of being balance by using rotations when a new node is added.

There are 3 different ways that data can be represented in a tree. Trees where every node in the tree contain data are called **Data Trees**. One example of this type of tree is search trees. A simple example is:

**Expression Trees** are trees where data is only stored in the leaf nodes of the tree and the internal nodes store operators. For example:

**Decision Trees** are trees where data is only stored in the leaf nodes and the internal nodes store weighted information about its children. For example:

The internal nodes are usually between 0 and 1 and represents some probability or frequency that is associated with the children of the internal nodes.

In order to read the info in trees, we have to traverse the tree in some order. For binary trees there are 3 values associated with a node. There is the left child, the right child, and the data/info stored in the node. The different ways of arranging these values is the different ways to traverse the trees.

D – data/info in the node

L – the left child of the node

R – the right child of the node

The traversals can be:

DLR – Preorder Traversal

DRL

LDR – Inorder Traversal

RDL

LRD – Postorder Traversal  
RLD

**In order traversal**

1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

Definitions

1. AVL – Balanced trees using rotation
2. Balanced tree – All leaves are at most n level apart
3. Complete tree – A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible

**Preorder traversal**

1. Visit root node
2. Visit all the nodes in the left subtree
3. Visit all the nodes in the right subtree

**Postorder traversal**

1. visit all the nodes in the left subtree
2. visit all the nodes in the right subtree
3. visit the root node

Level i – 2i -1 // Max number of nodes at this level, starts at level 1 which is root

Height - The height of a node is the number of edges on the longest downward path between that node and a leaf.

Depth - The depth of a node is the number of edges from the node to the tree's root node.

Breadth First Search – Level by level and left to right

**Depth first search** - is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Given bits in 3rd column, you can reach value by that traversal path.:

|  |  |  |
| --- | --- | --- |
| E | 1000 | 0 |
| R | 500 | 10 |
| S | 250 | 110 |
| M | 100 | 1110 |
| U | 50 | 11110 |
| Q | 25 | 111110 |
| X | 10 | 111111 |

**Huffman encoding** – Tree used for data compression. You can get to the value by traversing the bit pattern in the 3rd column to get to corresponding value. So, 0 will get you to e. 10 will get you to R and rest is in the chart above 3rd column.

**Keywords:** genetic programming, forest, frontier, children, root, leaf, m-ary tree, complete binary tree, balanced binary tree, AVL tree, Data trees, expression trees, decision trees,

in order traversal, preorder traversal, post order traversal, depth first search, huffman encoding