



Introduction to Artificial Intelligence

Week 7



Evolutionary Algorithms Part I



Evolutionary Algorithms

- Use the concepts of the Neo-Darwinian Synthesis or Lamarckian Evolution
 - Natural Selection
 - Inheritable Traits
 - Fitness Biased Reproduction
 - Fitness is generated based on the
 - Generational/Time Series
- Four major overarching techniques discovered about 1980
 - Genetic Algorithms - Holland
 - Genetic Programming - Koza
 - Evolutionary Programming - Fogel
 - Evolutionary Strategies – Rechenbreg/Schwefel
- Large arguments about priority of technique leads to a compromise on the title of Evolutionary Algorithms – schisms still fighting for dominance – beware ye who enter here



EA System

- Create a randomized **population** made up of **chromosomes**, data structures which encode a potential solution
- Until <Done>, based on a **stopping criteria**
 - Find an **objective/fitness score** for each member of the population
 - **Select members** to act upon using some **variation operators**
 - Apply operations on the members
 - **Crossover**
 - **Mutations**
 - Replace some members of the population with these children from the variation operators
 - Keep some members from the previous population in the new population, i.e. **elitism/inheritance**

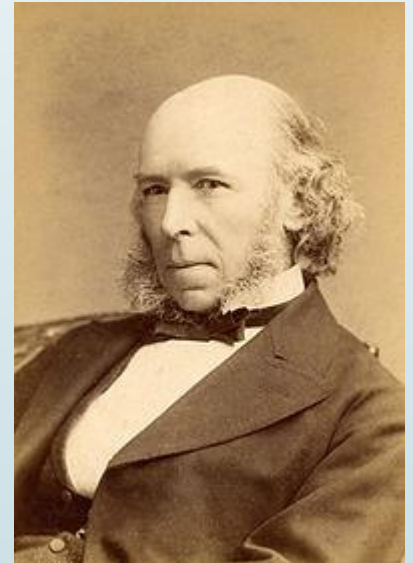
Selection

- Cartoon of the ideas of Natural Selection by Darwin
- Provides a fitness biased method of keeping good structures
 - Note Biased not based
 - We can still accept 'worst' choices
- Structures which have a higher fitness on the objective score are more likely to continue on in the population



Survival of the Fittest

- ❑ Major misconceptions in the application of this phrase
- ❑ Darwin didn't coin it – nor was it used until the 5th edition of *Origins*
- ❑ Used by Herbert Spencer in *Principles of Biology*
 - ❑ "This survival of the fittest, which I have here sought to express in mechanical terms, is that which Mr. Darwin has called 'natural selection', or the preservation of favoured races in the struggle for life."
- ❑ Darwin's use was based on the fitness of a creature to survive in a local environment



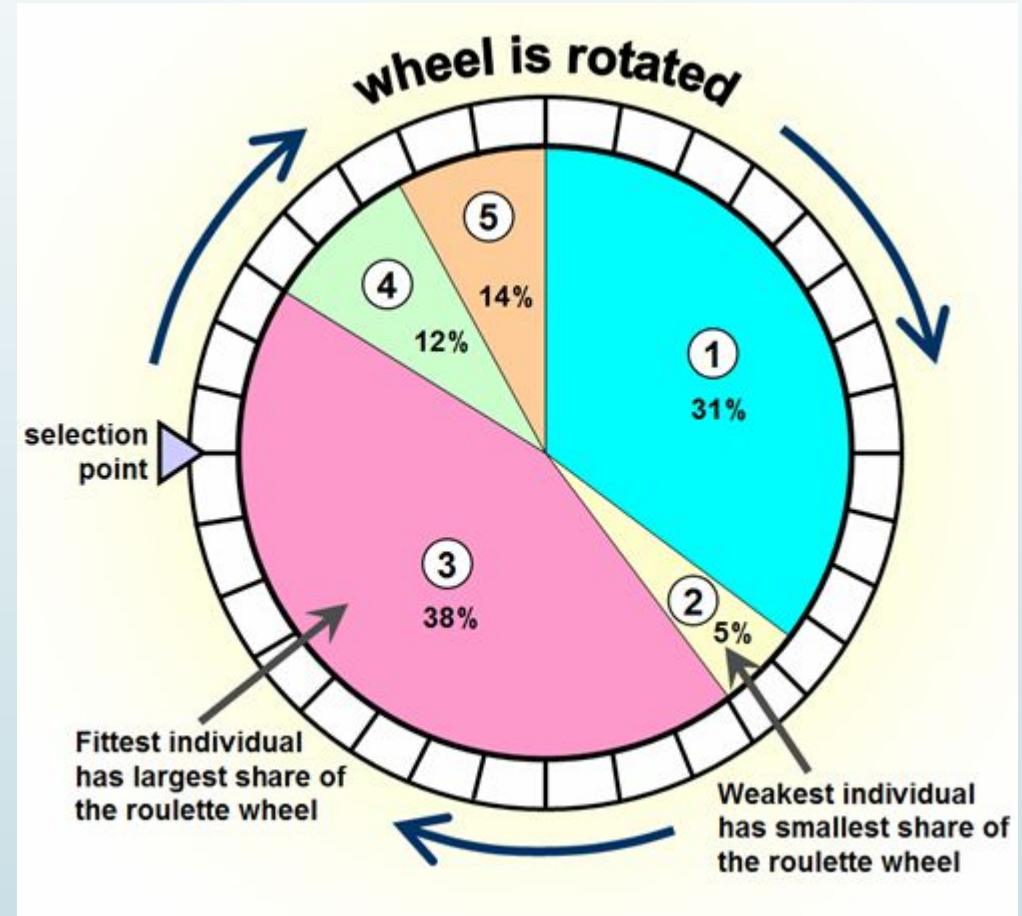


Biological Fitness

- The phrase seems to imply that there is an innate idea of what is FIT/UNFIT
- Post Hoc Ergo Proctor Hoc Fallacy
 - The creature survived as it was fit
 - The creature is fit because it has survived
- Biological Fitness is defined as the number of offspring which reach sexual maturity and are able to pass along their genes
- Evolutionary Algorithms fall under this misconception – we apply fitness as a post hoc

Fitness Proportional

- Each member is given a section of the wheel in relation to their fitness score
- Usually $\text{Fit}(\text{Member}) / \text{Sum of Fit}(\text{All Member})$
- Wheel is spun for a number of times
- Winners Breed Together





Tournament

- A number of different manners are held for the construction of the challengers
 - At Random
 - Groups of N
- Each of the structures in a tournament is compared and the most fit continues on to breed
- Fighting solutions
- Selection Pressure (the likelihood of only selecting from the higher fitness cohorts is a controllable feature)
 - Small Tournaments
 - Larger Tournaments



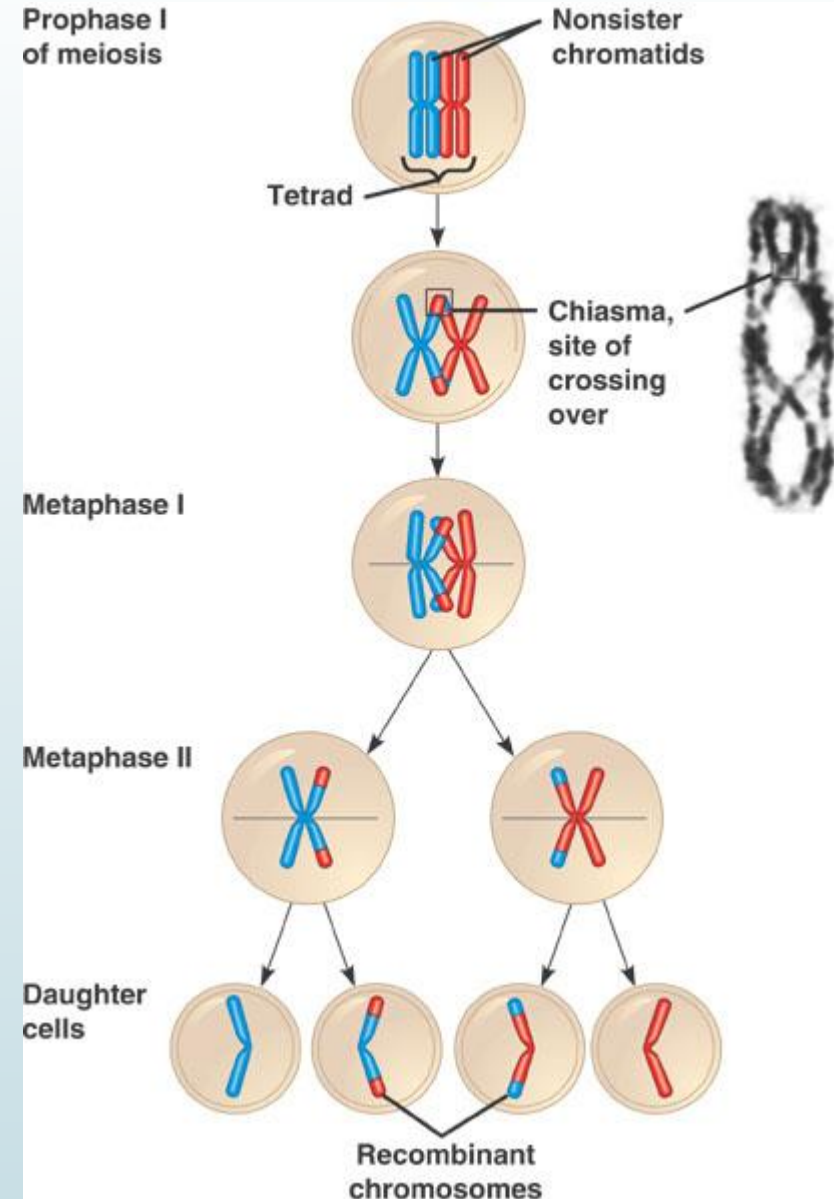
Genetic Algorithms



- Representation: Data Structure (commonly a discrete string)
- Selection: Roulette (aka Fitness Proportional) or Tournament
- Crossover: Yes. Data Structure Dependent
- Mutation: Yes. Data Structure Dependent, commonly a small change to a percentage of symbols in the string

Crossover in Biology

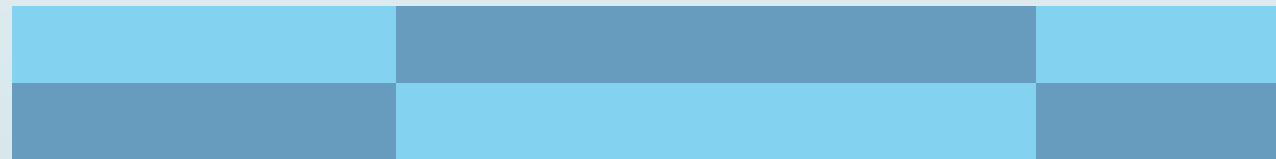
- Process of Meiosis
- Creation of **gamete** cells
 - Sex cells
 - from the Greek for wife
- Haploid creatures have chromosome pairs
- Is not a representation of the actions which happen in



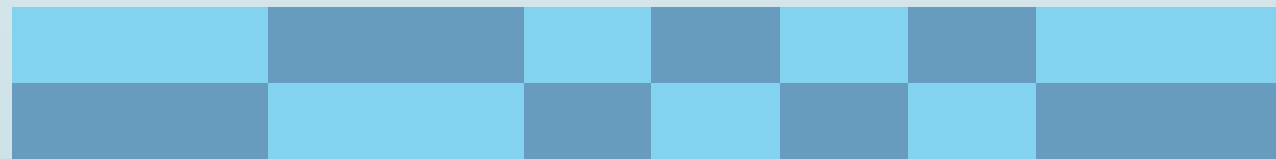
Crossover in a GA on Strings



One Point – Select One Point at Random and Swap

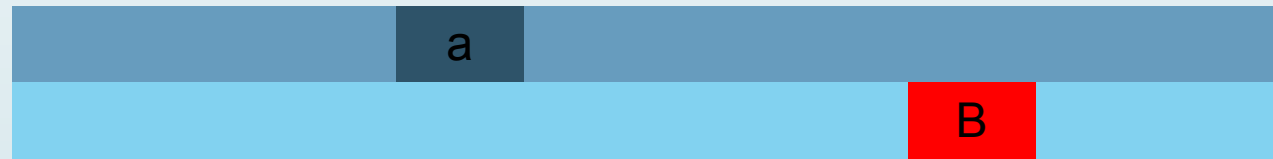


Two Point – Select Two Point at Random and Swap



Uniform Order – Swap all with Probability of .5

Mutation in a GA on Strings



Point Mutation – Change the Symbol
at a Loci to Some Other Symbol



Swap Mutation – Swap Two Loci in the String



Genetic Programming



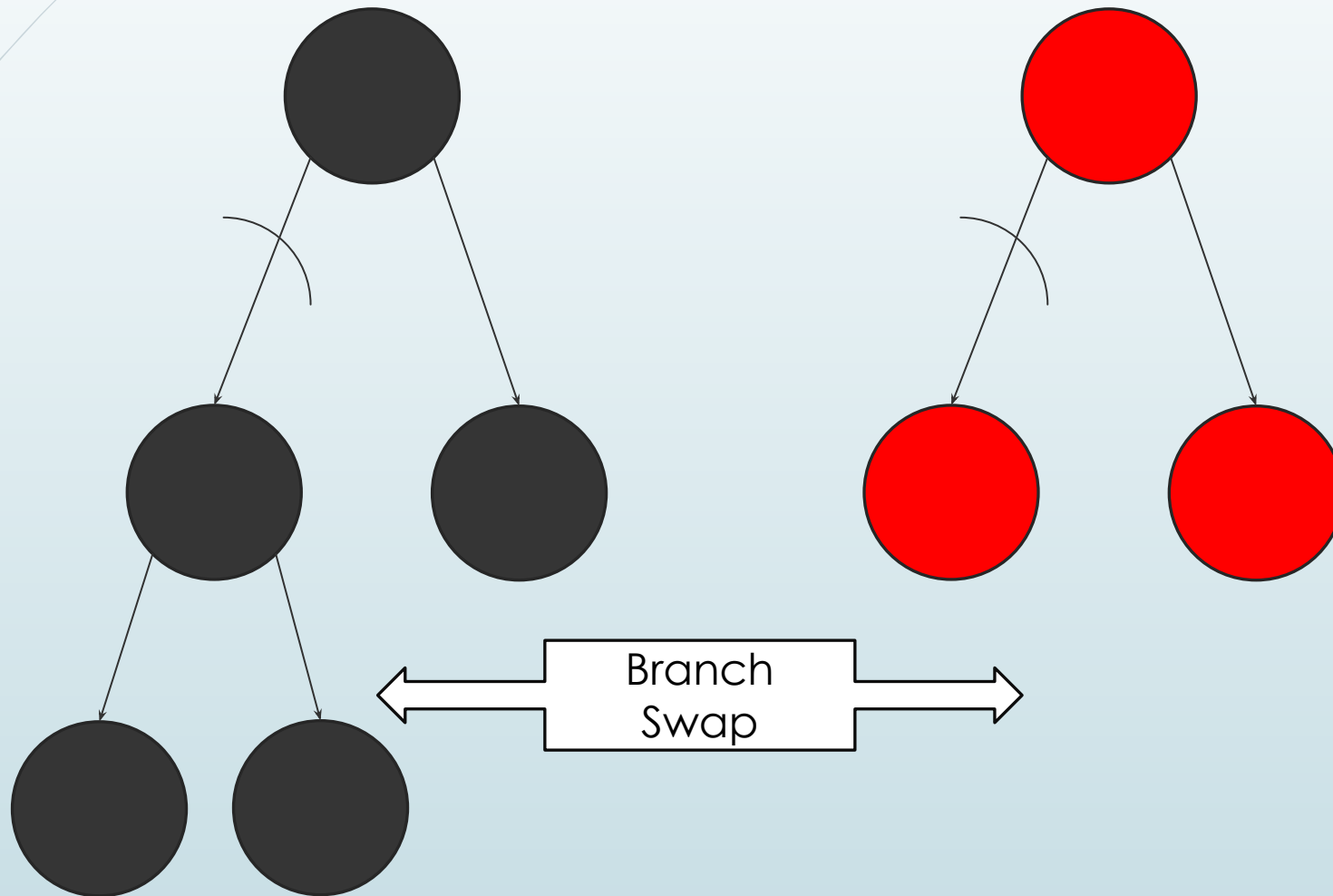
- Representation: Tree Based
- Selection: Roulette or Tournament
- Crossover: Yes. Branches of the Trees are Exchanged.
- Mutation: Yes. Leaf value/Symbol Change or Operator Change
- Special Operations: Yes. Removal of Extra Symbols called bloat. Functions may be defined as shorter symbols (ADF)



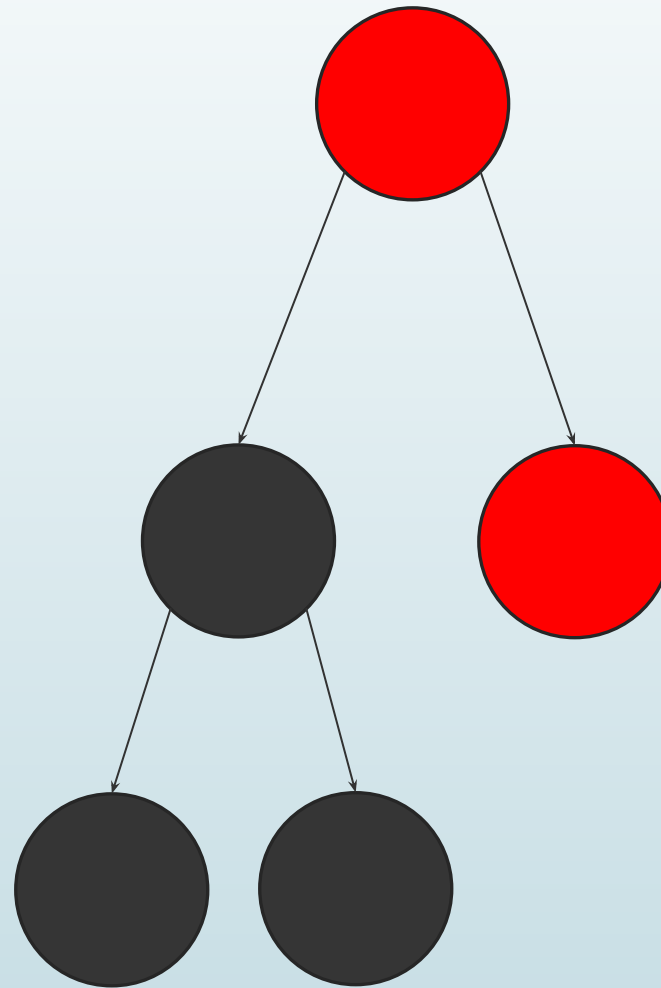
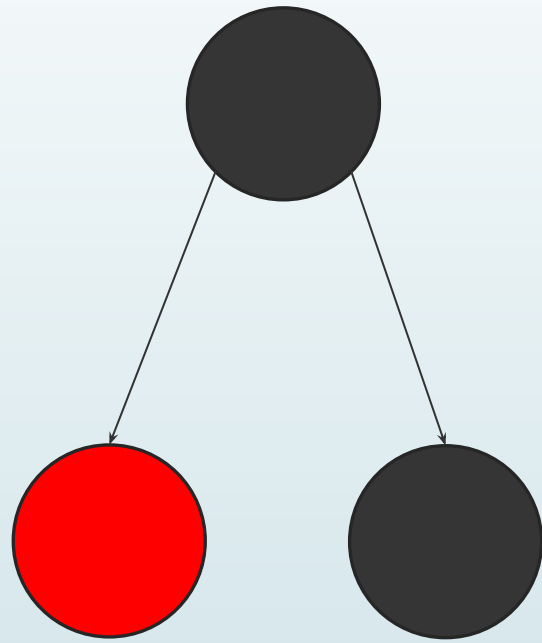
GP Parse Trees and LISP

- The idea comes from the programming language of LISP
- (function, arg1, arg2, ..., argN)
- Arguments are functions or terminals
- Terminals are literals (1, `x`) or variables (x, count)
- LISP allows for programs which manipulate code and run that code
- Other languages need to create a simulator
- Prefix notation e.g. (+ 1 (* 7 X)) is $7x+1$
- No need for order of operations – all operations are explicitly ordered by brackets

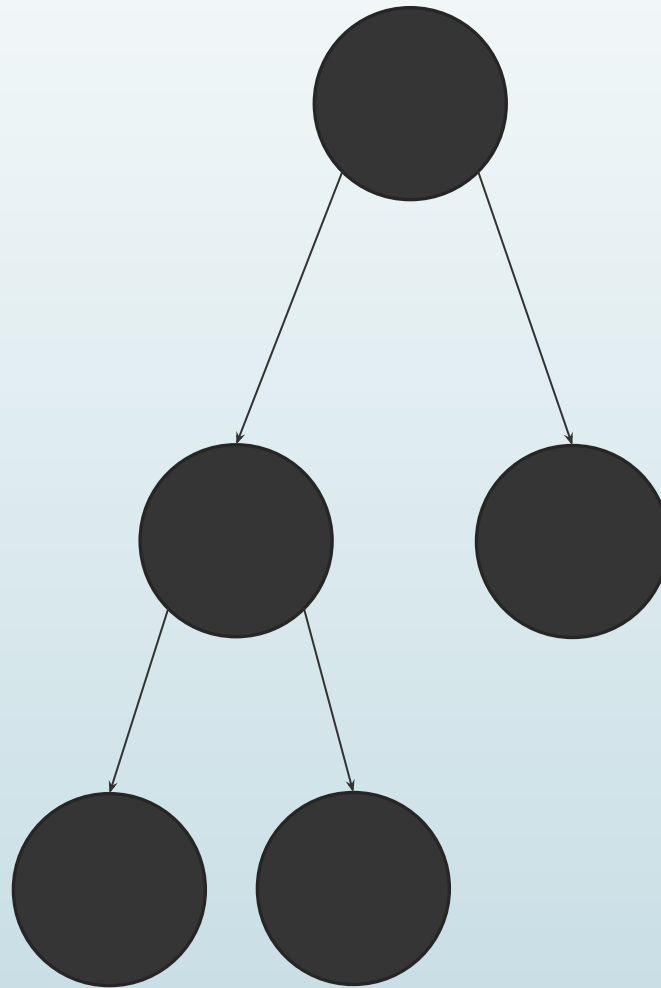
Crossover in a GP Tree



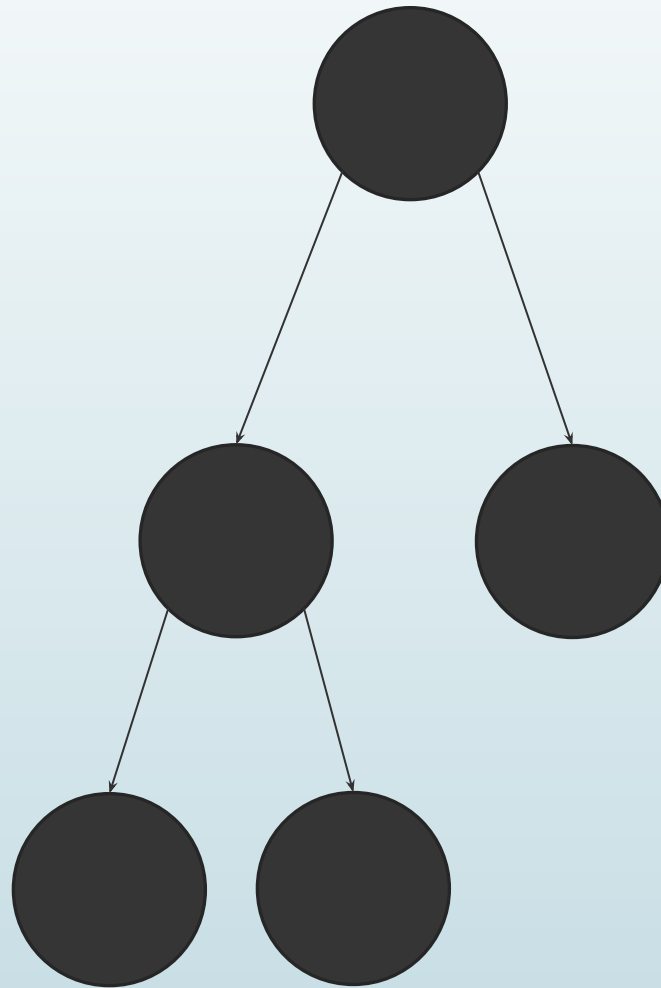
Crossover in a GP Tree



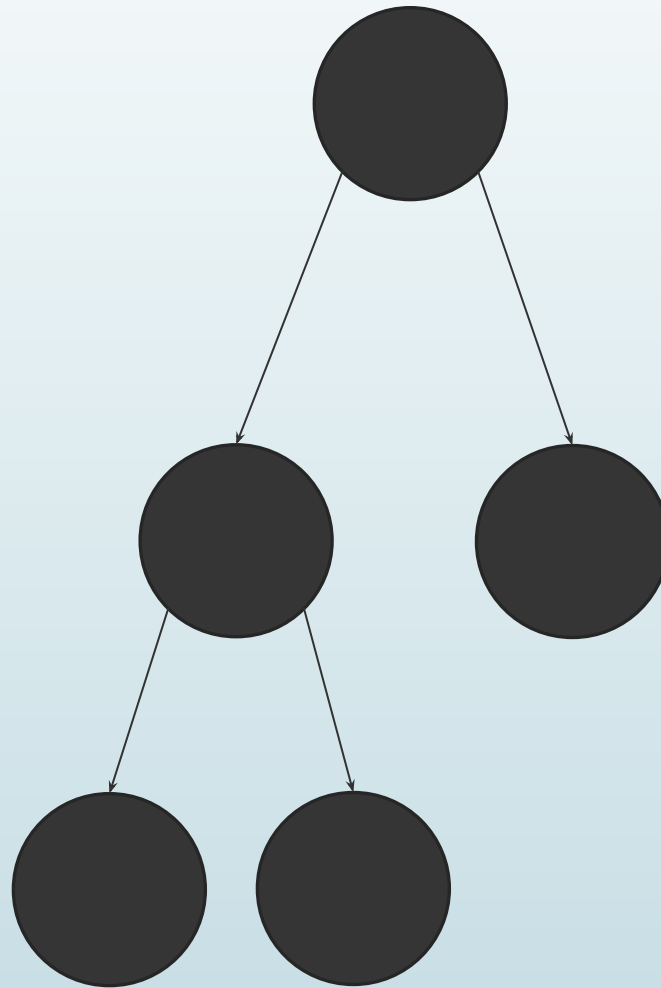
Mutation of a Terminal in a GP Tree



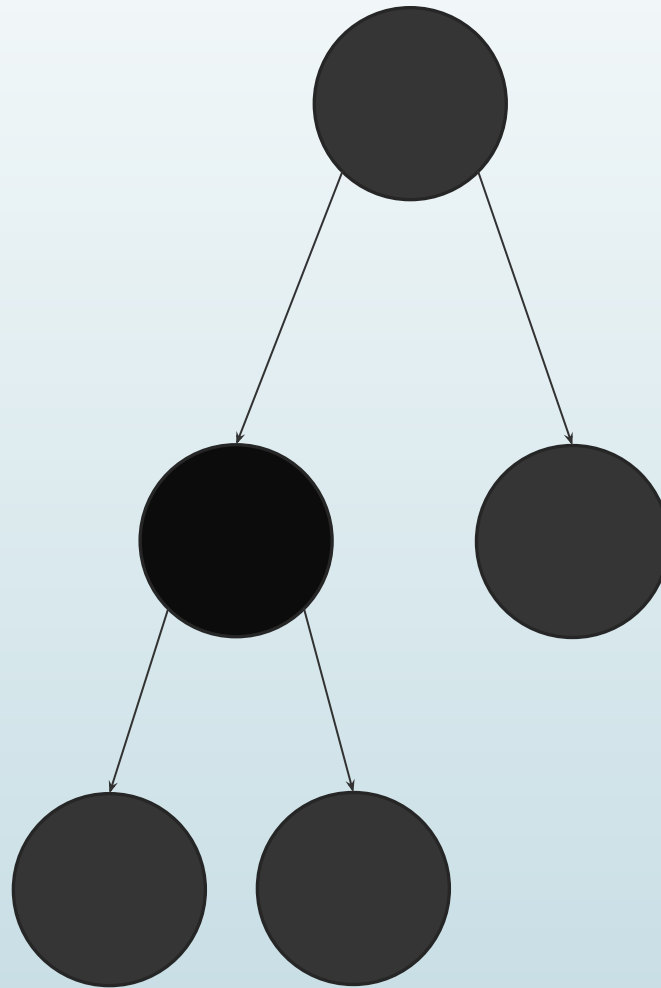
Mutation of a Operation in a GP Tree



Growing Operation in a GP Tree

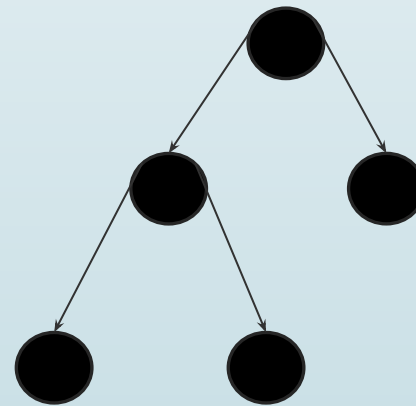
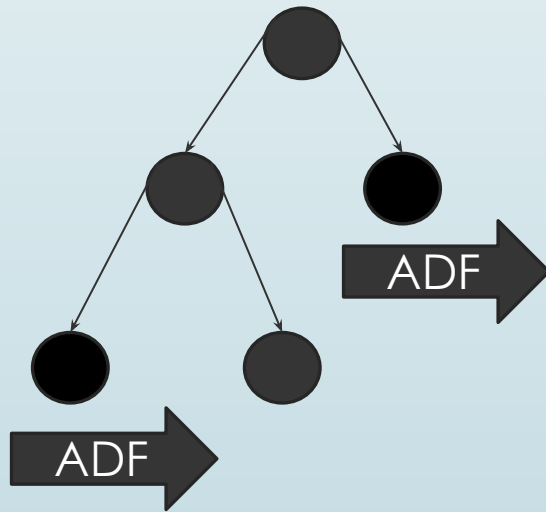


Cut Operation in a GP Tree



ADF Trees

- ADF – Automatically Defined Functions
- Many Times we have a tree computed again and again – repetition is costly
- Allow for the construction of GPs with smaller GP trees – construct a hierarchy





Rules on Functions in Trees

- All trees should produce `legal` programs
- Operations which produce common errors – such as divide by zero – should have a protected version that explicitly maps those errors to a legal input value – such as 0



Bloat



- A number of operations provide no change in the result
 - Anything multiplied by 1
 - Anything added to 0
- A number of operations cancel out parts of the tree
 - Anything multiplied by 0
 - An operation followed by its inverse
- Leads to trees which are equally as fit but are larger



Why does Bloat exist

- Imagine two trees which both add 5 to 6 the one has 3 nodes in the tree, the other has 10 nodes which add a value multiplied by 0
- You require a minimum number of 3 nodes to implement $(+ 5 6)$
 - One for each of the arguments
 - One for the operand
- 7 nodes in the second tree are bloat
- What is the probability that a mutation operation (change operand/argument) will affect the solution to the problem?



Bloat Saves Solutions

- In the first tree the changing of an operation or argument will completely change the result, 100% of the time it will change the outcome
- In the bloated tree, 3 nodes are part of our solution, one to add, and two to multiply by 0. Changing these nodes will lead to a different answer.
- Yet 4 nodes are inconsequential to the answer – 40% of the time there will be no change in fitness based on a mutation
- Heritability – A solution with more of these null mutations is likely to have its children survive as they have the same fitness



Bloat in Biology

- Repetition of genes
- Repetition of genes
- Duplication of genes
- Transposon Elements
- Repetition of genes
- Transposon Elements
- Not to be confused with redundant systems – Example Weight Loss Pill Trials

Fat Blocking Pill

- ❑ Idea – We want to create a diet pill
- ❑ Block the regulatory system in the human body which makes you gain weight
- ❑ Step 1 – find system
- ❑ Step 2 – create blocking drug
- ❑ Step 3 – Clinical Trials on Mice





Mice Got Fatter

- The clinical trial showed the mice not only gained weight – they gained more weight than the control on the same diet!
- But we Blocked the Signals
- Ah – but did you block all the signals
- Mammals have a secondary fat producing system which will come into effect when our primary system is compromised
- Issue – this secondary system is not as refined

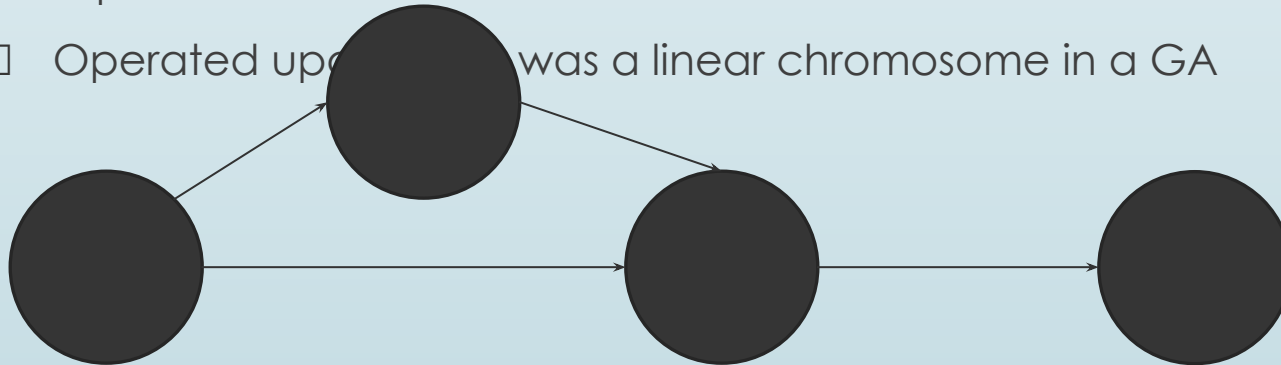


Parsimony

- We like things simple in design of solutions
 - *Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher. (Terre des Hommes, 1939).*
 - It seems that a perfect design is not one which one looks for things to add, but is one where there is nothing left to remove
- Let the trees grow but trim them at the result
- Penalize Larger Trees!
 - Reduction in fitness score
 - Less chance to Breed
- Find a method which does not use a tree based model for the representation

Other Representations

- Directed Acyclic Graphs (DAG)
 - Cartesian Genetic Programming
 - Function Stacks
- Instead of Evolving Trees – Representation is graph
- Repeated input branches are passed down the DAG representation
 - Removes the need to recompute
 - Expansion and Bloat is limited – fixed size data structure
 - Operated upon as was a linear chromosome in a GA

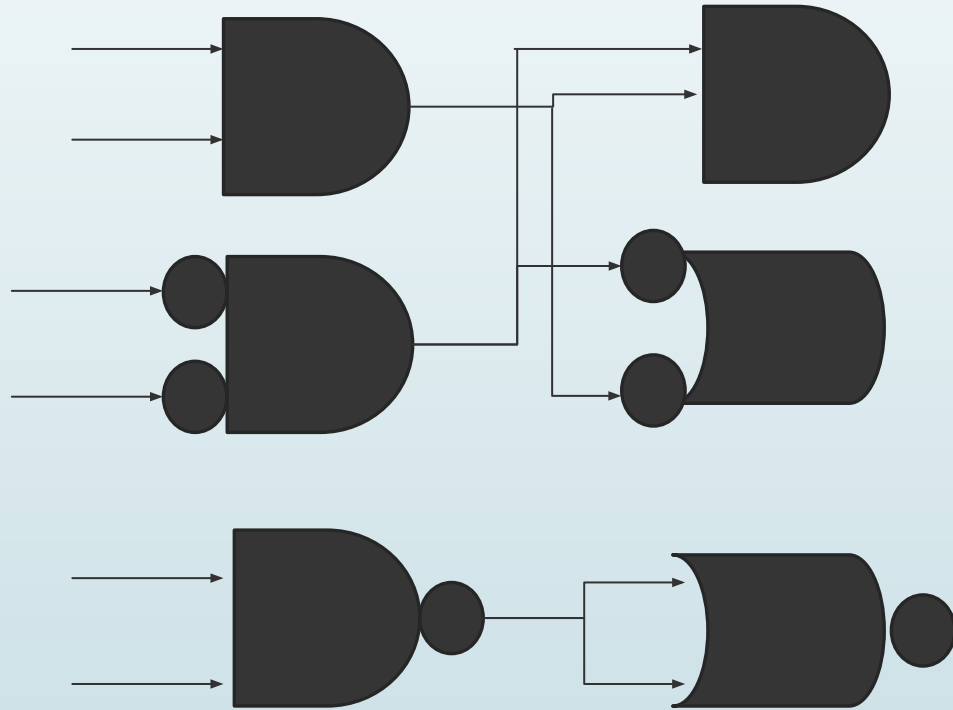


Cartesian Genetic Programming

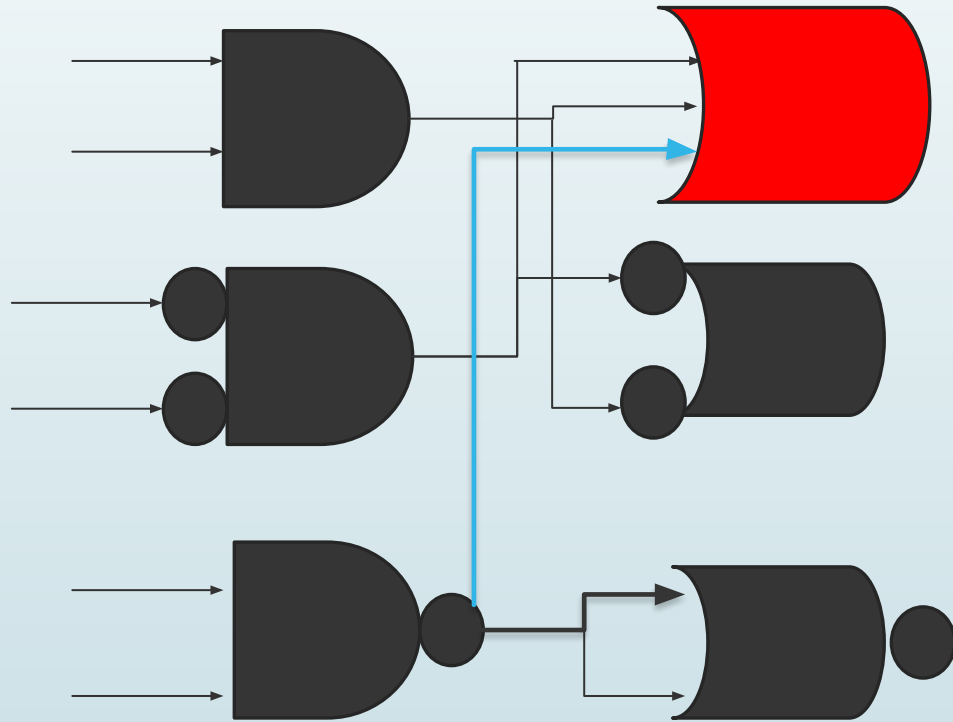
- NxM grid of Operations connected by wires
- Think Printed Circuit Boards



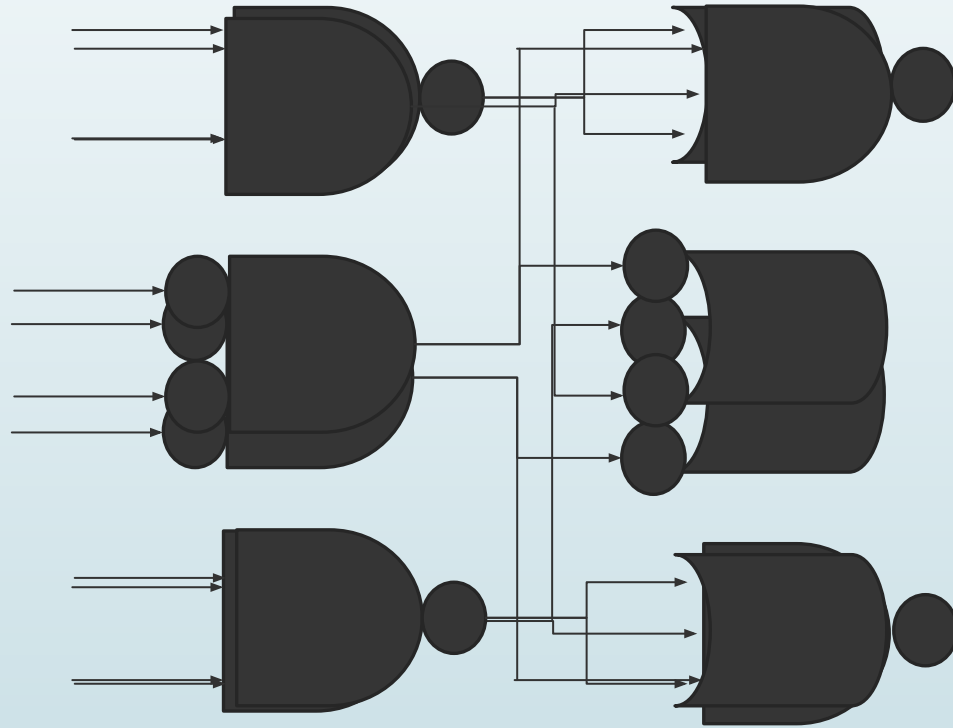
Data Structure



Mutations Can Affect Nodes and Edges



Flip Operations





Function Stack Representation

- Function Stacks have a linear chromosome consisting of nodes
- Node Contains
 - Function of 0..N inputs
 - Inputs – Either Previous Nodes in Order of the Chromosome or an input value
 - An Ephemeral Constant
- Crossover as per a linear string in a GA
- Mutations change the operation or constants



Evolutionary Programming



- Representation: Finite State Machine
- Selection: Replace with a member of a sample of mutants if better than parent
- Crossover: No.*
- Mutation: Yes. Add or Remove a node, or Change transition, output, or starting node.
- Note: Designed for use in an online setting for controller

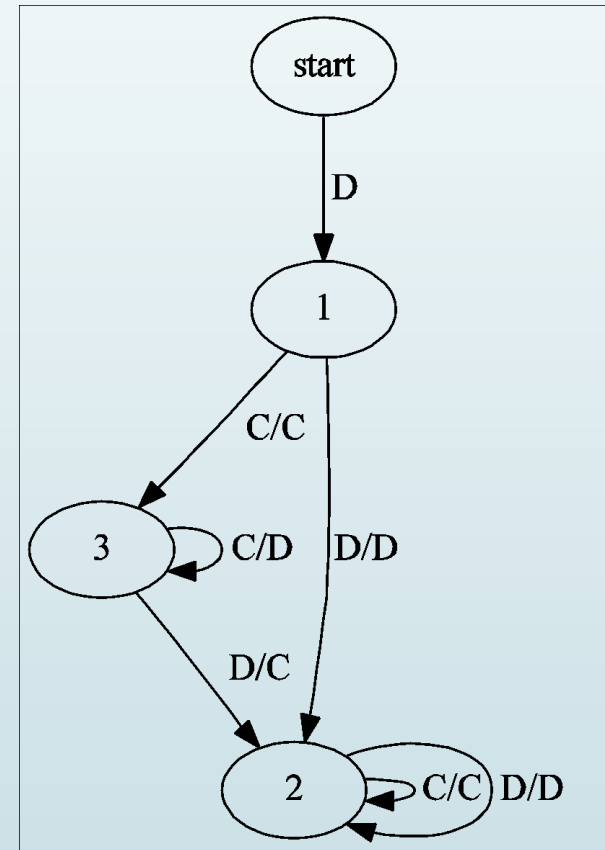
Finite State Machine

- A deterministic finite state machine is defined by a tuple $\langle Q, I, Z, O, \delta, \omega, q \rangle$ where:
 - Q – finite set of states
 - I – finite set of inputs
 - Z – finite set of outputs
 - δ – transition function $\delta: I \times Q \rightarrow Q$
 - ω – output function $\omega: I \times Q \rightarrow Z$
 - q – initial starting state where $q \in Q$
- You can also define it via a state transition diagram

Representations of a FSM

Initial 1,D

IF	C	D
1	3,C	2,D
2	2,C	2,D
3	3,D	2,C





Mutations in EP FSM

- Mutations are insertions, deletions, changes to a transition, changes to a output, change starting node
- Insertions – add a node and its connectors, find some set of random transitions to place into it (do not want it isolated)
- Deletions – select a random node, all incoming transitions sent to other nodes at random
- Change transition, change output, change initial, are self explanatory



Evolutionary Strategies

- Representation: Vector of Real Values
- Selection: Replace with a member of a sample of mutations if better than parent
- Crossover: No.*
- Mutation: Yes. Add small normally distributed parameter to a value.

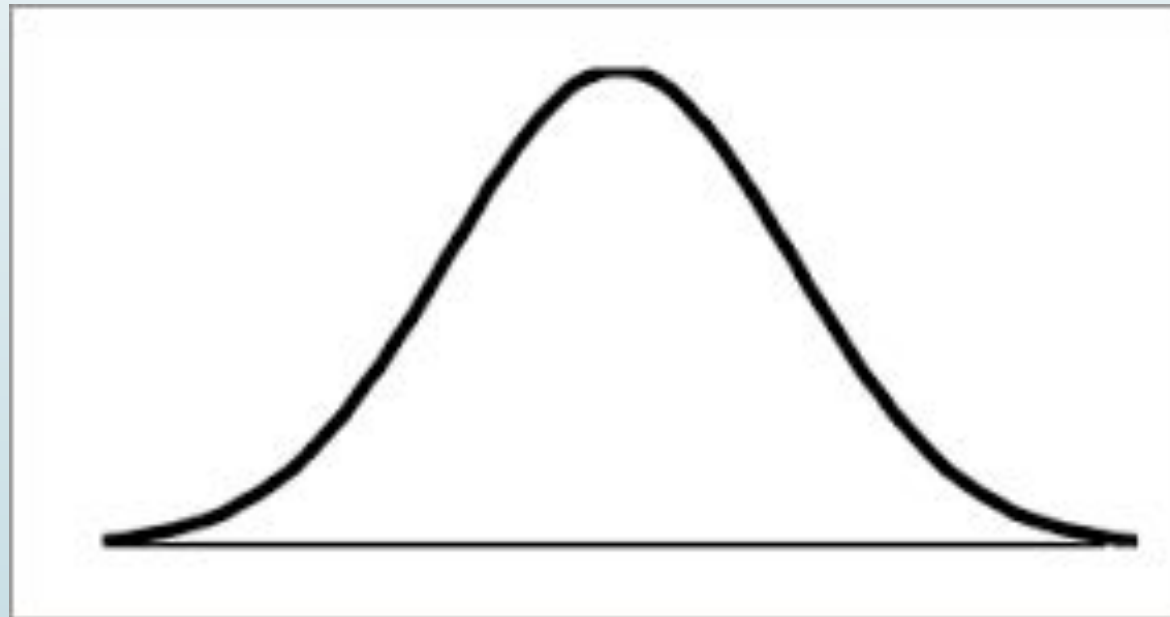
*Has been added in some variants

ES Bracketed Notation

- Normally Distributed Function of mutation is applied to the string of real numbers – some use log normal
- $(1+1)$ -ES – a mutant is tested against its parent and the fittest is retained
- $(1+\lambda)$ -ES - λ mutants are tested against their parent with the fittest remaining, the parent retained if the best
- $(1,\lambda)$ -ES - λ mutants are tested against their parent, the parent is never retained, only one of mutants will continue on
- $(\mu/\rho+, \lambda)$ -ES – A population is used where a group of mutants is made for each and compete with the set of parents, this may also have a crossover operation

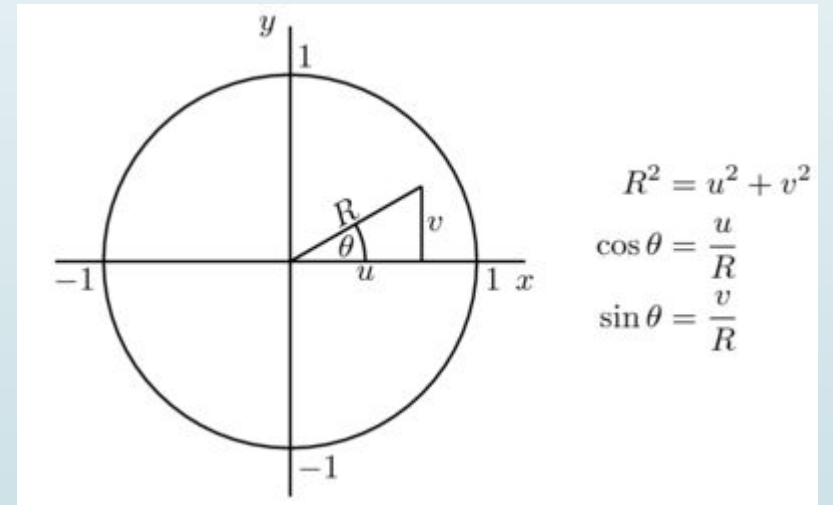
Small Mutations

- Pull from the Gaussian/Normal Distribution
- Many Mutations will make small changes in parameters, few will make large changes



Generating a Normal Random Variation

- ▣ Assume we have a Uniform RNG $[0,1]$
- Add N (larger the better) RNGs subtract $N/2$
 - Gives an approximation to the normal between $\pm N/2$
- Box-Muller Transform
 - Take two RNG numbers, u and v
 - Treat u and v as polar coordinates
 - $r^2 = u^2 + v^2$
 - $z_u = u \cdot \sqrt{\frac{-2 \ln r^2}{r^2}}; z_v = v \cdot \sqrt{\frac{-2 \ln r^2}{r^2}}$





Which Should I use?

- ❑ No Free Lunches Here
- ❑ Note the similarities between Genetic Algorithms and Programming – Key Difference is the type of representation
- ❑ Similarly Evolutionary Programming and Strategies differ based on representation
- ❑ How you can represent your problem has a big effect on which of these methods is available (more on this next time in the representation lecture)
- ❑ Offline or Online?
 - ❑ Speed of evaluation becomes a factor
 - ❑ Crossover is more expensive
 - ❑ Fitness evaluation is ALWAYS more expensive