Genetic Algorithms

Introduction

- After scientists became disillusioned with classical and neo-classical attempts at modeling intelligence, they looked in other directions.
- Two prominent fields arose, connectionism (neural networking, parallel processing) and evolutionary computing.
- It is the latter that this essay deals with - genetic algorithms and genetic programming.

What is GA

- A **genetic algorithm** (or **GA**) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- Genetic algorithms are categorized as global search heuristics.
- Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

What is GA

- Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions.

- Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

What is GA

- The evolution usually starts from a population of randomly generated individuals and happens in generations.

- In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified (recombined and possibly mutated) to form a new population.

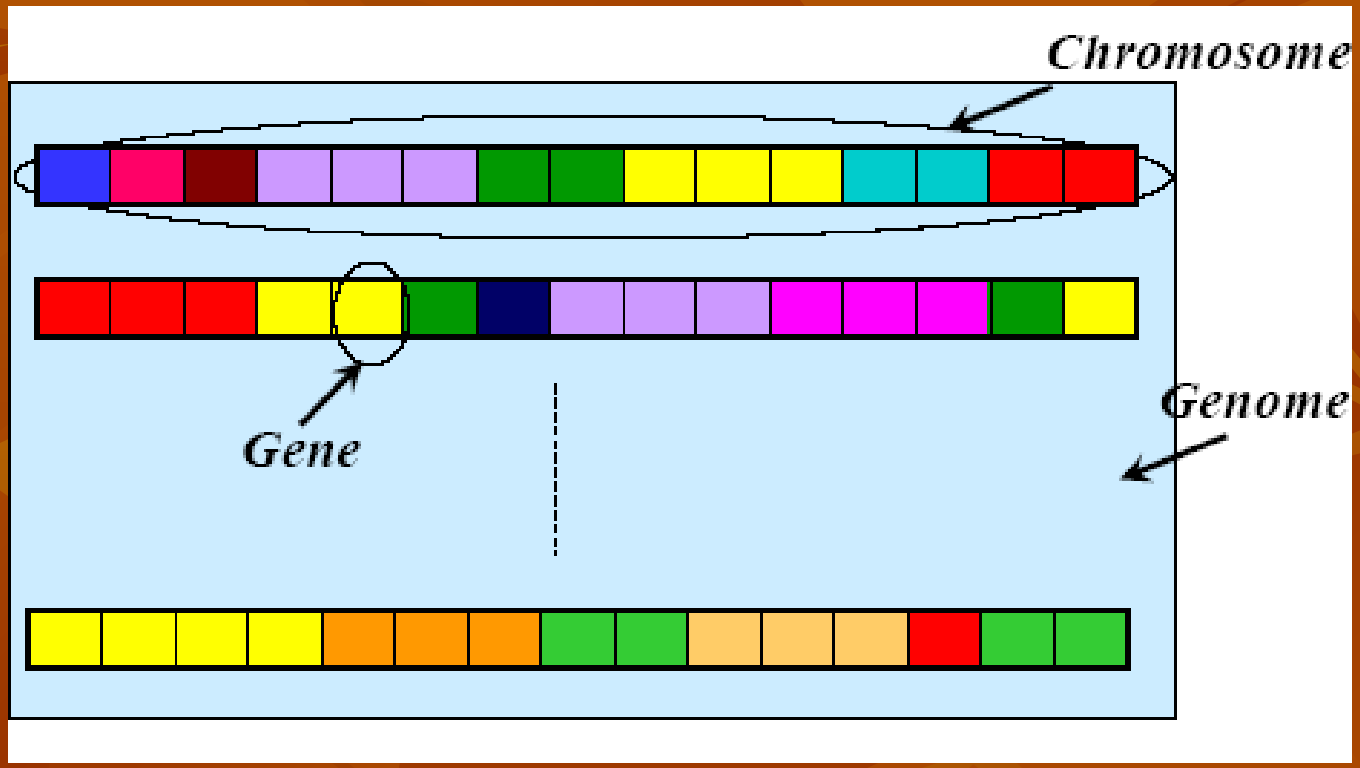
What is GA

- The new population is then used in the next iteration of the algorithm.
- Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.
- If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Key terms

- **Individual -** Any possible solution
- **Population -** Group of all *individuals*
- **Search Space -** All possible solutions to the problem
- **Chromosome -** Blueprint for an *individual*
- **Trait -** Possible aspect (*features)* of an *individual*
- **Allele -** Possible settings of trait (black, blond, etc.)
- **Locus -** The position of a *gene* on the *chromosome*
- **Genome -** Collection of all *chromosomes* for an *individual*

Chromosome, Genes and Genomes

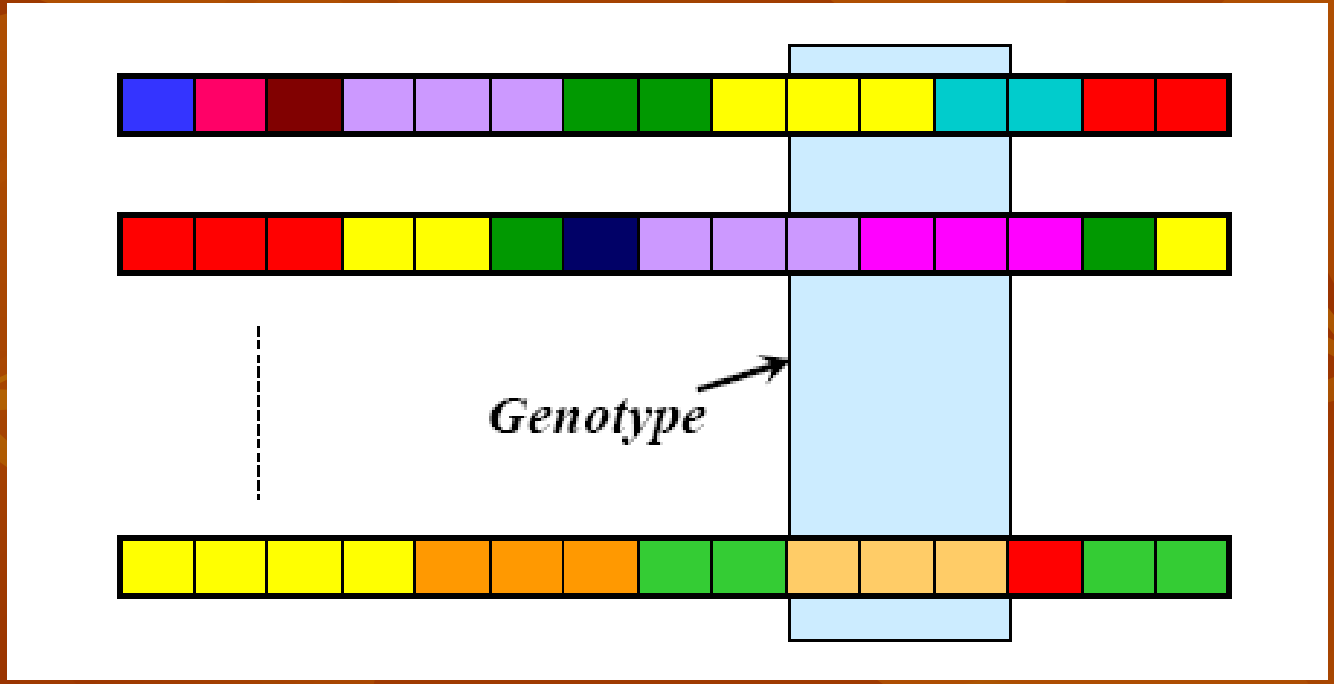


Genotype and Phenotype

- ***Genotype:***
  – Particular set of genes in a genome

- ***Phenotype:***
  – Physical characteristic of the genotype (smart, beautiful, healthy, etc.)

Genotype and Phenotype



GA Requirements

- A typical genetic algorithm requires two things to be defined:
- a genetic representation of the solution domain, and
- a fitness function to evaluate the solution domain.

- A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way.
- The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, that facilitates simple crossover operation.
- Variable length representations may also be used, but crossover implementation is more complex in this case.
- Tree-like representations are explored in Genetic programming.

Representation

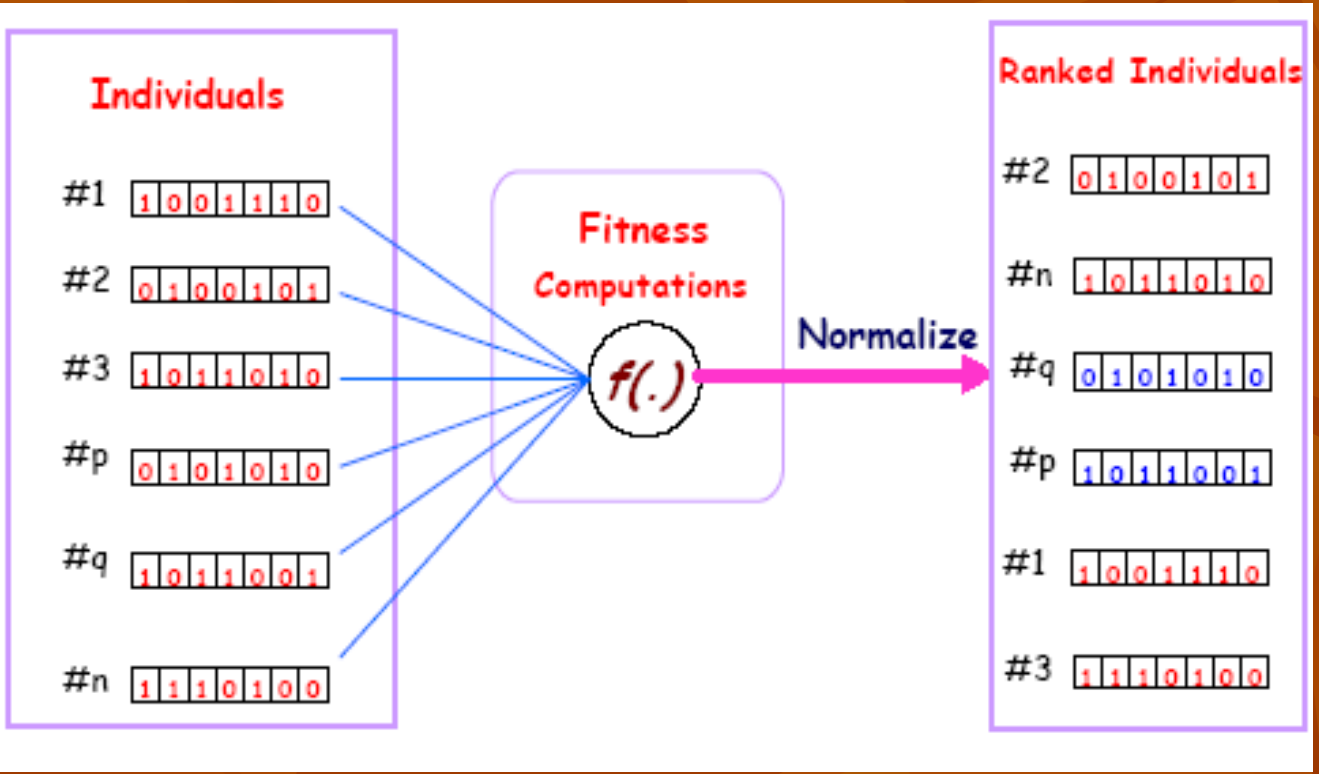
Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

GA Requirements

- The fitness function is defined over the genetic representation and measures the *quality* of the represented solution.
- The fitness function is always problem dependent.
- For instance, in the knapsack problem we want to maximize the total value of objects that we can put in a knapsack of some fixed capacity.
- A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack.
- Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack.
- The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

A fitness function



Basics of GA

- The most common type of genetic algorithm works like this:
- a population is created with a group of individuals created randomly.
- The individuals in the population are then evaluated.
- The evaluation function is provided by the programmer and gives the individuals a score based on how well they perform at the given task.
- Two individuals are then selected based on their fitness, the higher the fitness, the higher the chance of being selected.
- These individuals then "reproduce" to create one or more offspring, after which the offspring are mutated randomly.
- This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer.

General Algorithm for GA

- **Initialization**
- Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions.
- Traditionally, the population is generated randomly, covering the entire range of possible solutions (the *search space*).
- Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

General Algorithm for GA

- **Selection**
- During each successive generation, a proportion of the existing population is selected to breed a new generation.
- Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected.
- Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.
- Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection.

General Algorithm for GA

- In roulette wheel selection, individuals are given a probability of being selected that is directly proportionate to their fitness.

- Two individuals are then chosen randomly based on these probabilities and produce offspring.

General Algorithm for GA

Roulette Wheel's Selection Pseudo Code:

```
for all members of population
        sum += fitness of this individual
end for
for all members of population
        probability = sum of probabilities + (fitness / sum)
        sum of probabilities += probability
end for
loop until new population is full
        do this twice
                number = Random between 0 and 1
                for all members of population
                        if number > probability but less than next probability then
                                you have been selected
                end for
        end
        create offspring
end loop
```

General Algorithm for GA

- **Reproduction**
- The next step is to generate a second generation population of solutions from those selected through genetic operators:

  crossover (also called recombination), and/or mutation.
- For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously.
- By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each child, and the process continues until a new population of solutions of appropriate size is generated.

General Algorithm for GA

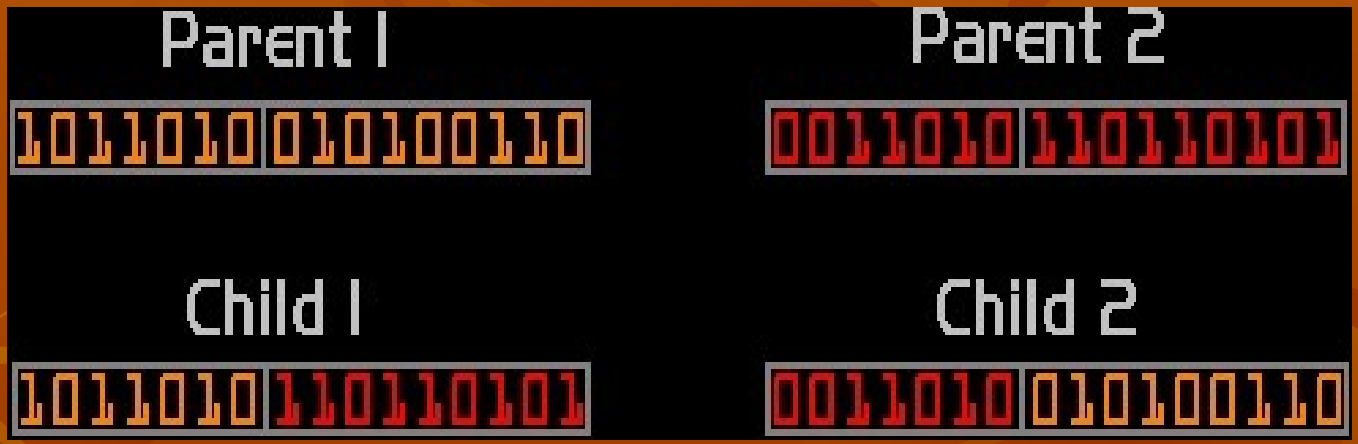
- These processes ultimately result in the next generation population of chromosomes that is different from the initial generation.

- Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

Crossover

- the most common type is single point crossover. In single point crossover, you choose a locus at which you swap the remaining alleles from on parent to the other. This is complex and is best understood visually.
- As you can see, the children take one section of the chromosome from each parent.
- The point at which the chromosome is broken depends on the randomly selected crossover point.
- This particular method is called single point crossover because only one crossover point exists. Sometimes only child 1 or child 2 is created, but oftentimes both offspring are created and put into the new population.
- Crossover does not always occur, however. Sometimes, based on a set probability, no crossover occurs and the parents are copied directly to the new population. The probability of crossover occurring is usually 60% to 70%.

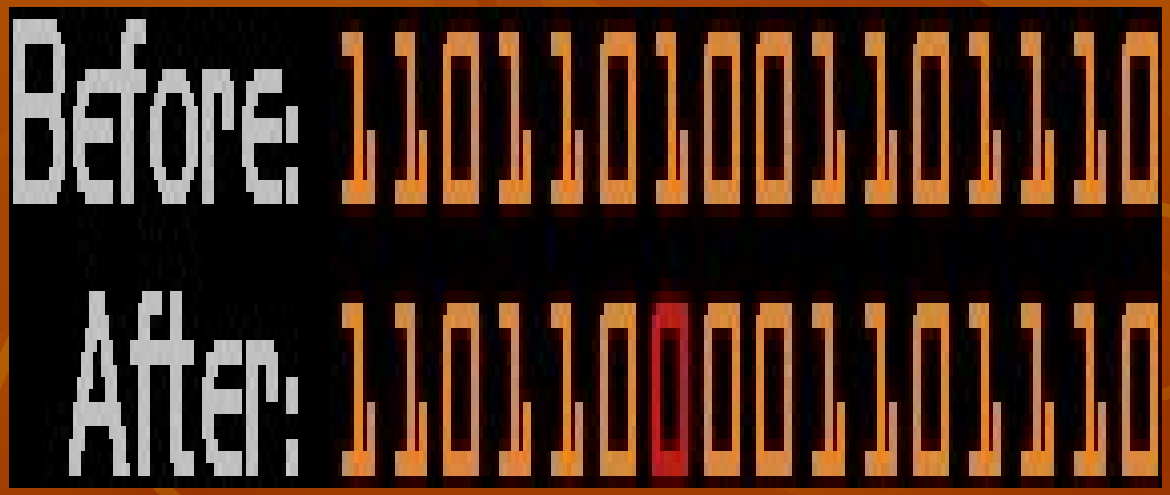
Crossover


Parent 1
10110100100110
Parent 2
00110101101101
Child 1
10110101101101
Child 2
00110100100110

Mutation

- After selection and crossover, you now have a new population full of individuals.
- Some are directly copied, and others are produced by crossover.
- In order to ensure that the individuals are not all exactly the same, you allow for a small chance of mutation.
- You loop through all the alleles of all the individuals, and if that allele is selected for mutation, you can either change it by a small amount or replace it with a new value. The probability of mutation is usually between 1 and 2 tenths of a percent.
- Mutation is fairly simple. You just change the selected alleles based on what you feel is necessary and move on. Mutation is, however, vital to ensuring genetic diversity within the population.

Mutation



General Algorithm for GA

- **Termination**
- This generational process is repeated until a termination condition has been reached.
- Common terminating conditions are:
  - A solution is found that satisfies minimum criteria
  - Fixed number of generations reached
  - Allocated budget (computation time/money) reached
  - The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
  - Manual inspection
  - Any Combinations of the above

GA Pseudo-code

Choose initial population
Evaluate the fitness of each individual in the population
Repeat

Select best-ranking individuals to reproduce

Breed new generation through crossover and mutation (genetic operations) and give birth to offspring

Evaluate the individual fitnesses of the offspring

Replace worst ranked part of population with offspring

Until <terminating condition>

Symbolic AI VS. Genetic Algorithms

- Most symbolic AI systems are very static.
- Most of them can usually only solve one given specific problem, since their architecture was designed for whatever that specific problem was in the first place.
- Thus, if the given problem were somehow to be changed, these systems could have a hard time adapting to them, since the algorithm that would originally arrive to the solution may be either incorrect or less efficient.
- Genetic algorithms (or GA) were created to combat these problems; they are basically algorithms based on natural biological evolution.

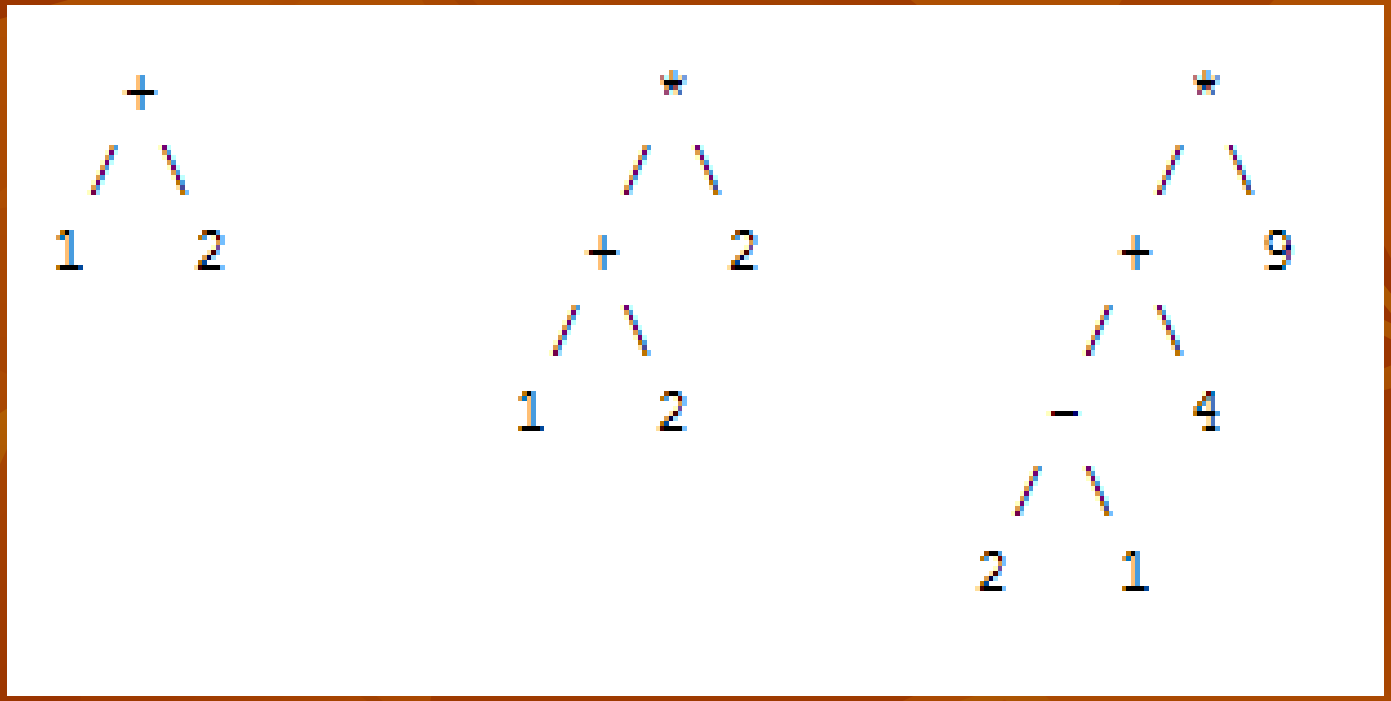
Symbolic AI VS. Genetic Algorithms

- The architecture of systems that implement genetic algorithms (or GA) are more able to adapt to a wide range of problems.
- A GA functions by generating a large set of possible solutions to a given problem.
- It then evaluates each of those solutions, and decides on a "fitness level" (you may recall the phrase: "survival of the fittest") for each solution set.
- These solutions then breed new solutions.
- The parent solutions that were more "fit" are more likely to reproduce, while those that were less "fit" are more unlikely to do so.
- In essence, solutions are evolved over time. This way you evolve your search space scope to a point where you can find the solution.
- Genetic algorithms can be incredibly efficient if programmed correctly.

Genetic Programming

- In programming languages such as LISP, the mathematical notation is not written in standard notation, but in prefix notation. Some examples of this:
- + 2 1 : 2 + 1
- * + 2 1 2 : 2 * (2+1)
- * + - 2 1 4 9 : 9 * ((2 - 1) + 4)
- Notice the difference between the left-hand side to the right? Apart from the order being different, no parenthesis! The prefix method makes it a lot easier for programmers and compilers alike, because order precedence is not an issue.
- You can build expression trees out of these strings that then can be easily evaluated, for example, here are the trees for the above three expressions.

Genetic Programming



Genetic Programming

- You can see how expression evaluation is thus a lot easier.
- What this have to do with GAs? If for example you have numerical data and 'answers', but no expression to conjoin the data with the answers.
- A genetic algorithm can be used to 'evolve' an expression tree to create a very close fit to the data.
- By 'splicing' and 'grafting' the trees and evaluating the resulting expression with the data and testing it to the answers, the fitness function can return how close the expression is.

Genetic Programming

- The limitations of genetic programming lie in the **huge** search space the GAs have to search for - an infinite number of equations.
- Therefore, normally before running a GA to search for an equation, the user tells the program which operators and numerical ranges to search under.
- Uses of genetic programming can lie in stock market prediction, advanced mathematics and military applications .

Evolving Neural Networks

- Evolving the architecture of neural network is slightly more complicated, and there have been several ways of doing it. For small nets, a simple matrix represents which neuron connects which, and then this matrix is, in turn, converted into the necessary 'genes', and various combinations of these are evolved.

Evolving Neural Networks

- Many would think that a learning function could be evolved via genetic programming. Unfortunately, genetic programming combined with neural networks could be *incredibly* slow, thus impractical.
- As with many problems, you have to constrain what you are attempting to create.
- For example, in 1990, David Chalmers attempted to evolve a function as good as the delta rule.
- He did this by creating a general equation based upon the delta rule with 8 unknowns, which the genetic algorithm then evolved.

Other Areas

- Genetic Algorithms can be applied to virtually any problem that has a large search space.
- Al Biles uses genetic algorithms to filter out 'good' and 'bad' riffs for jazz improvisation.
- The military uses GAs to evolve equations to differentiate between different radar returns.
- Stock companies use GA-powered programs to predict the stock market.

Example

- $f(x) = \{MAX(x^2): 0 <= x <= 32 \}$
- Encode Solution: Just use 5 bits (1 or 0).
- Generate initial population.

A	0	1	1	0	1
B	1	1	0	0	0
C	0	1	0	0	0
D	1	0	0	1	1

- Evaluate each solution against objective.

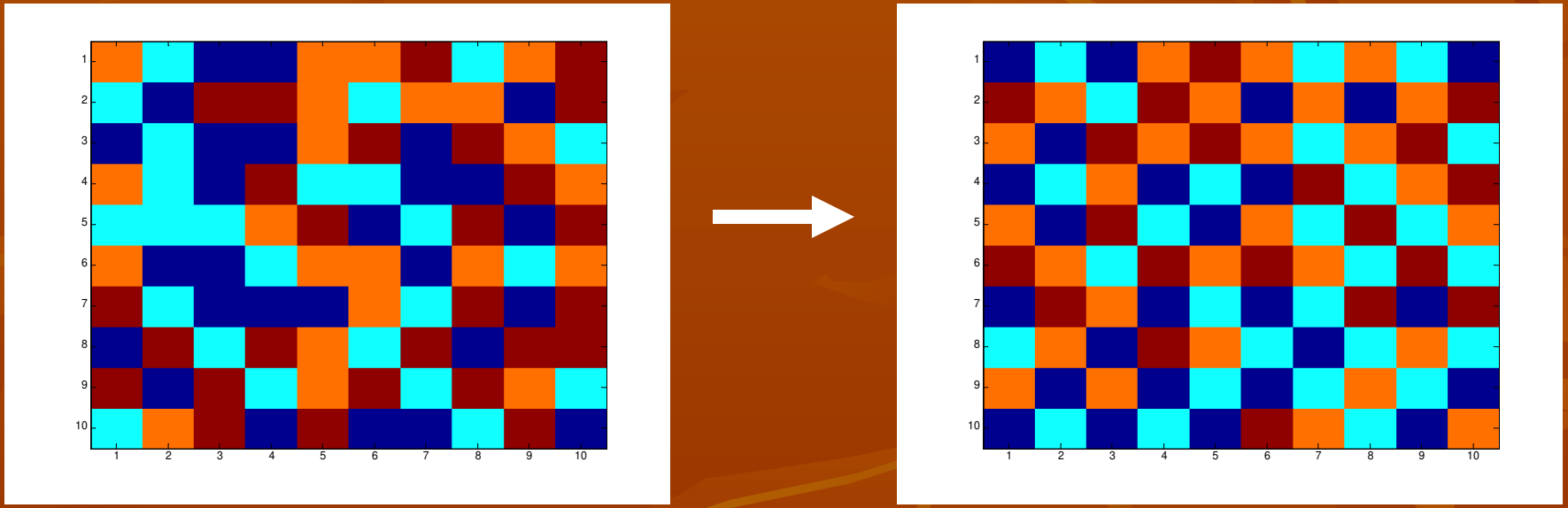
Sol.	String	Fitness	% of Total
A	01101	169	14.4
B	11000	576	49.2
C	01000	64	5.5
D	10011	361	30.9

Example Cont'd

- Create next generation of solutions
  - Probability of "being a parent" depends on the fitness.
- Ways for parents to create next generation
  - Reproduction
    - Use a string again unmodified.
  - Crossover
    - Cut and paste portions of one string to another.
  - Mutation
    - Randomly flip a bit.
  - COMBINATION of all of the above.

Checkboard example

- We are given an ***n*** by ***n*** checkboard in which every field can have a different colour from a set of four colors.
- Goal is to achieve a checkboard in a way that there are no neighbours with the same color (not diagonal)



Checkboard example Cont'd

- Chromosomes represent the way the checkboard is colored.
- Chromosomes are not represented by bitstrings but by **bitmatrices**
- The bits in the bitmatrix can have one of the four values 0, 1, 2 or 3, depending on the color.
- Crossing-over involves matrix manipulation instead of point wise operating.
- Crossing-over can be combining the parential matrices in a horizontal, vertical, triangular or square way.
- Mutation remains bitwise changing bits in either one of the other numbers.

Checkboard example Cont'd

- This problem can be seen as a graph with ***n*** nodes and ***(n-1)*** edges, so the fitness **f(x)** is defined as:

$$f(x) = 2 \cdot (n-1) \cdot n$$

Checkboard example Cont'd

- Fitnesscurves for different cross-over rules: