Genetic Algorithms

Nha Nam Ngoc Nguyen

The Genetic Algorithm

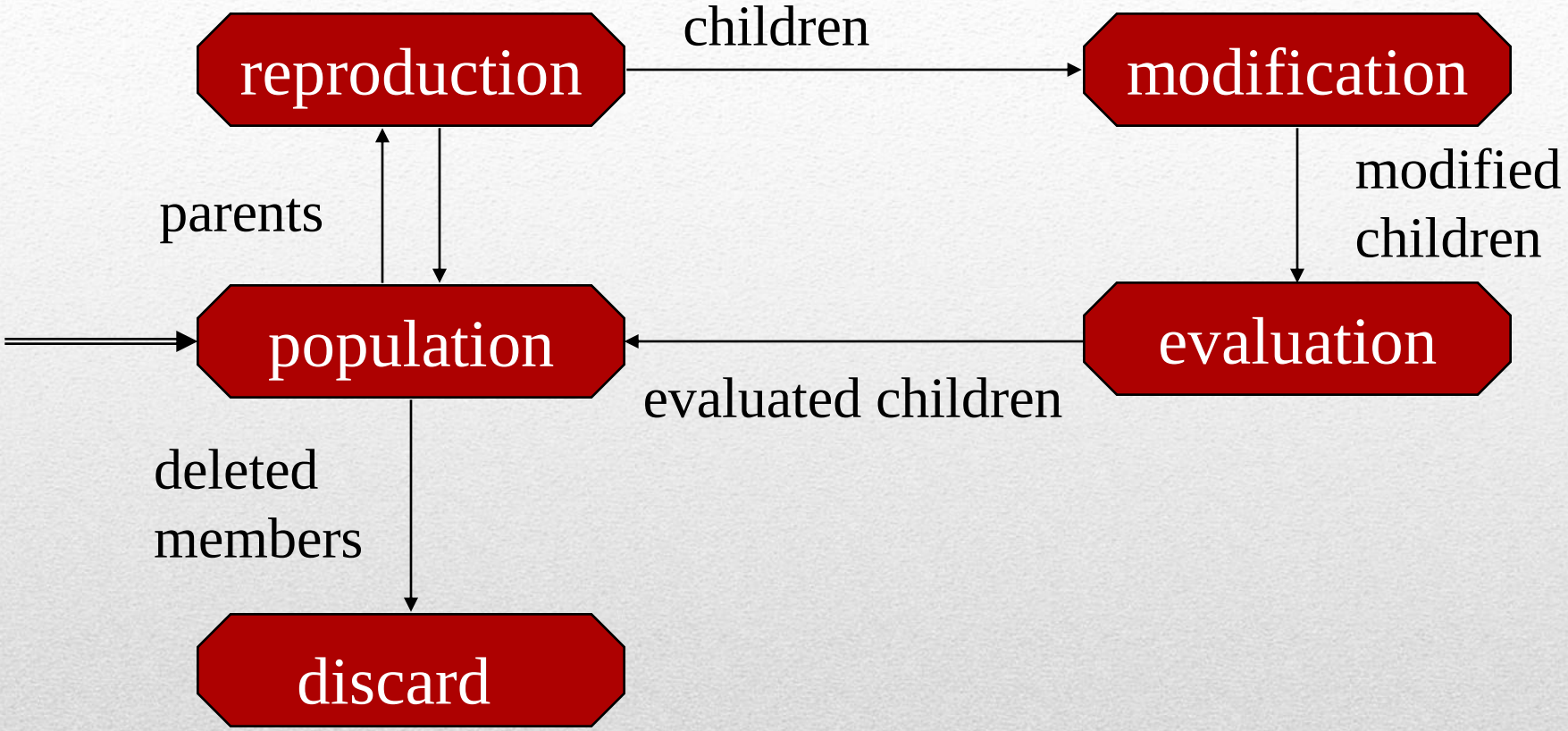
- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970's)
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems
- Provide efficient, effective techniques for optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles

Components of a GA

- A problem to solve, and ...
  - Encoding technique (*gene, chromosome*)
  - Initialization procedure *(creation)*
  - Evaluation function *(environment)*
  - Selection of parents *(reproduction)*
  - Genetic operators *(mutation, recombination)*
  - Parameter settings *(practice and art)*

```
{
    initialize population;
    evaluate population;
    while TerminationCriteriaNotSatisfied
    {
        select parents for reproduction;
        perform recombination and mutation;
        evaluate population;
    }
}
```

Simple Genetic Algorithm



The GA Cycle of Reproduction

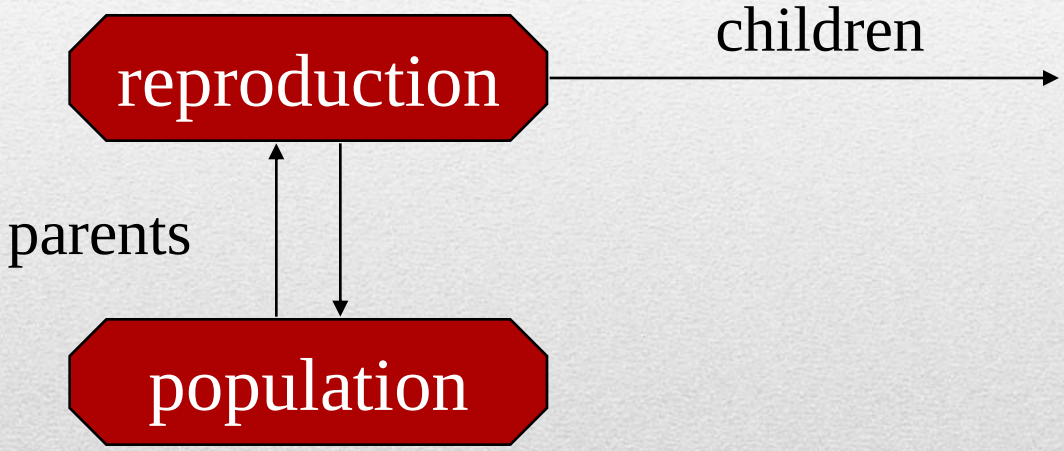
Population

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 ...

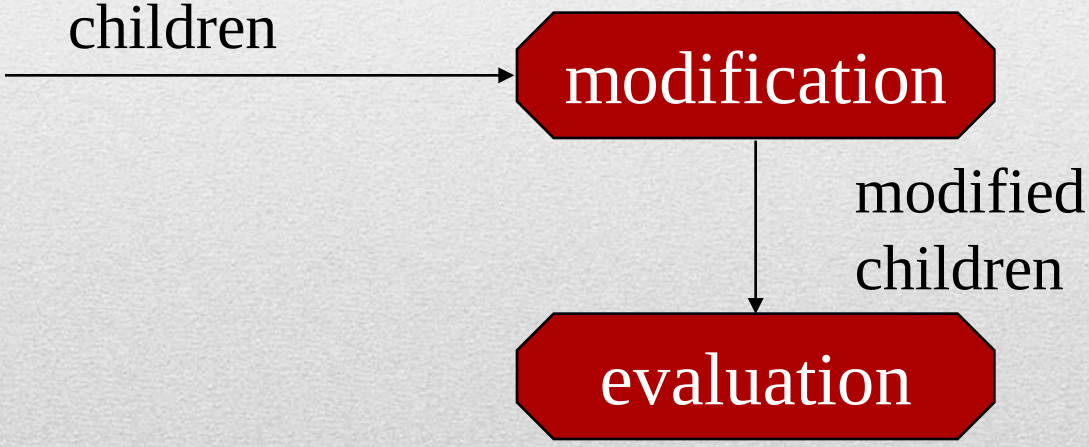
Reproduction

- Parents are selected at random with selection chances biased in relation to chromosome evaluations.



Chromosome Modification

- Modifications are stochastically triggered
- Operator types are:
  - Mutation
  - Crossover (recombination)



Mutation: Local Modification

- Before: (1 0 1 1 0 1 1 0)
- After: (0 1 1 0 0 1 1 0)

- Before: (1.38 -69.4 326.44 0.1)
- After: (1.38 -67.5 326.44 0.1)
- Causes movement in the search space (local or global)
- Restores lost information to the population

P1 (0 1 1 0 1 0 0 0)

P2 (1 1 0 1 1 0 1 0)

(0 1 0 0 1 0 0 0) C1

(1 1 1 1 1 0 1 0) C2

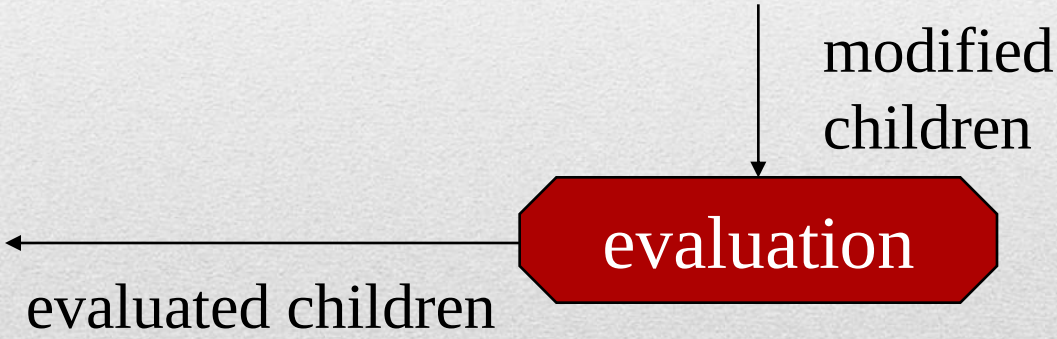
Crossover is a critical feature of genetic algorithms:

- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)

Crossover: Recombination

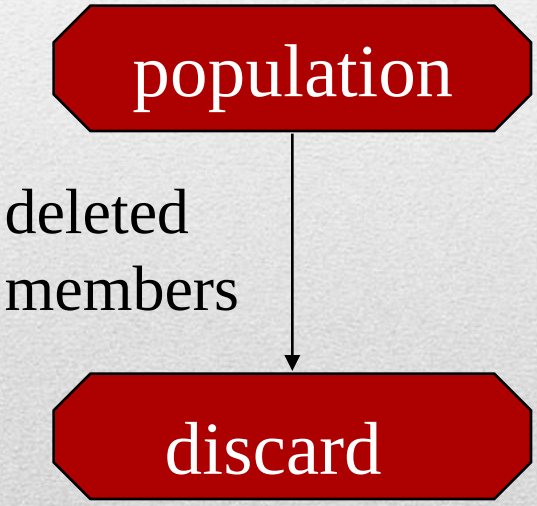
Evaluation

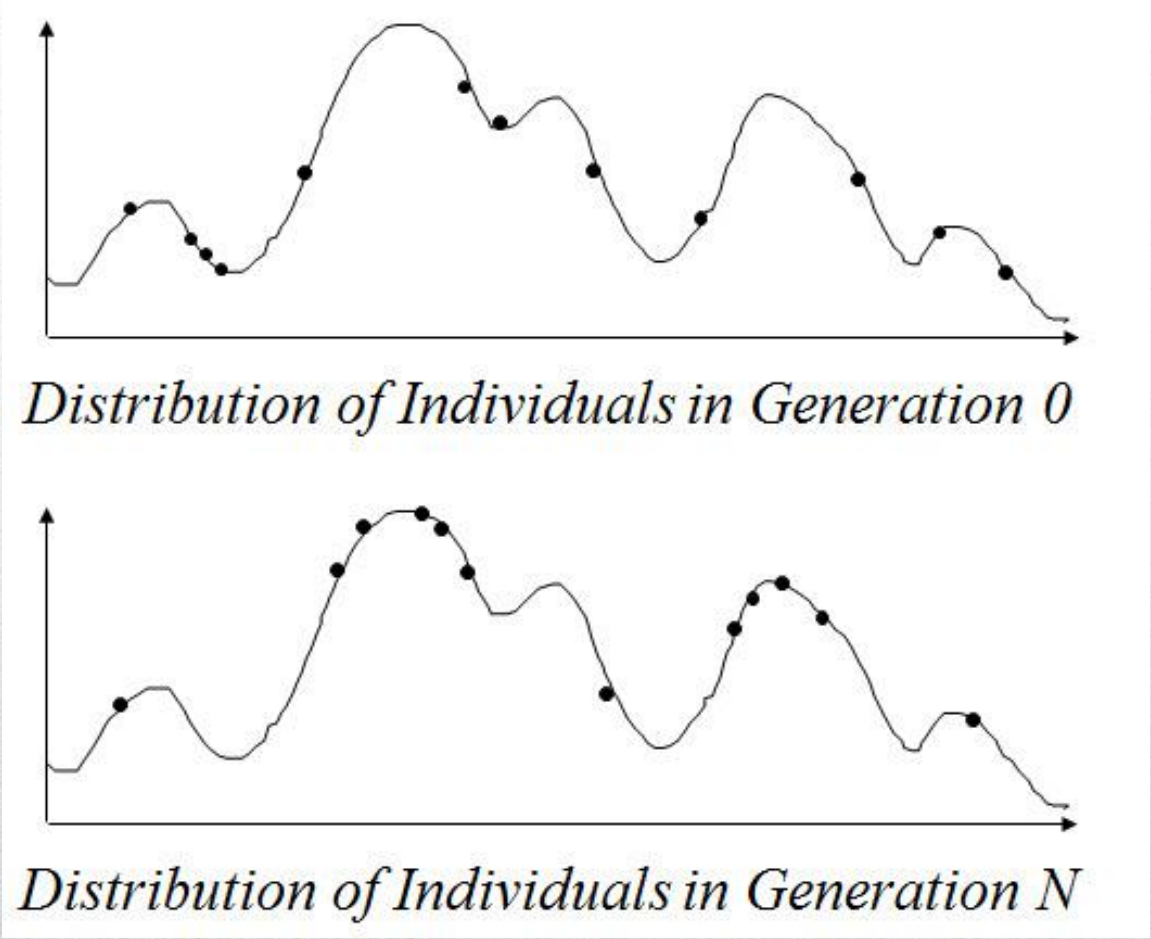
- The evaluator decodes a chromosome and assigns it a fitness measure
- The evaluator is the only link between a classical GA and the problem it is solving



Deletion

- *Generational* GA:
  entire populations replaced with each iteration
- *Steady-state* GA:
  a few members replaced each generation





An Abstract Example

A Simple Example

The Traveling Salesman Problem:

Find a tour of a given set of cities so that

- each city is visited only once
- the total distance traveled is minimized

Representation

Representation is an ordered list of city numbers known as an *order-based* GA.

1) London 3) Dunedin 5) Beijing 7) Tokyo
2) Venice 4) Singapore 6) Phoenix 8) Victoria

CityList1 (3 5 7 2 1 6 4 8)
CityList2 (2 5 7 6 8 1 3 4)

- Testing every possibility for an N city tour would be N! math additions. A 30 city tour would have to measure the total distance of be 2.65×10^{32} different tours.
- Assuming a trillion additions per second, this would take 252,333,390,232,297 years. Adding one more city would cause the time to increase by a factor of 31.
- Obviously, this is an impossible solution.

Problem ☹ !!!

- A genetic algorithm can be used to find a solution is much less time
- It might not find the best solution, it can find a near perfect solution for a 100 city tour in less than a minute

Apply GA To This Problem

Basic Steps

- First, create a group of many random tours in what is called a **population**. This algorithm uses a greedy initial population that gives preference to linking cities that are close to each other.
- Second, pick 2 of the better (shorter) tours **parents** in the population and combine them to make 2 new **child** tours. Hopefully, these children tour will be better than either parent.
- A small percentage of the time, the child tours are **mutated**. This is done to prevent all tours in the population from looking identical.
- The new child tours are inserted into the population replacing two of the longer tours. The size of the population remains the same.
- New children tours are repeatedly created until the desired goal is reached.

=>As the name implies, Genetic Algorithms mimic nature and evolution using the principles of **Survival of the Fittest**.

- The encoding of the tour
- The **crossover** algorithm that is used to combine the two parent tours to make the child tours.

Complex Issues ☹ ☹ ☹

- In a standard Genetic Algorithm, the encoding is a simple sequence of numbers and Crossover is performed by picking a random point in the parent's sequences and switching every number in the sequence after that point.
- In this example, the crossover point is between the 3rd and 4th item in the list. To create the children, every item in the parent's sequence after the crossover point is swapped.

Solution ☺

Parent 1
- F A B | E C G D

Parent 2
- D E A | C G B F

Child 1
- F A B | C G B F

Child 2
- D E A | E C G D

Solution ☺

- The difficulty with the Traveling Salesman Problem is that every city can only be used once in a tour. If the letters in the above example represented cities, this child tours created by this crossover operation would be invalid.
- **CHILD 1 GOES TO CITY F & B TWICE, AND NEVER GOES TO CITIES D OR E.**

Problem Again ☹☹☹☹

- The encoding cannot simply be the list of cities in the order they are traveled.

What is the point ?

- We store the **links** in both directions for each tour. In the above tour example, Parent 1 would be stored as:

City	First Connection	Second Connection
A	F	B
B	A	E
C	E	G
D	G	D
E	B	C
F	D	A
G	C	D

Greedy crossover

- The crossover will take every link that exists in both parents and place those links in both children.
- For Child 1 it alternates between taking links that appear in Parent 2 and then Parent 1.
- For Child 2, it alternates between Parent 2 and Parent 1 taking a different set of links.
- For either child, there is a chance that a link could create an invalid tour where instead of a single path in the tour. These links must be rejected.
- To fill in the remaining missing links, cities are chosen at random.

Parameters

- **Population Size -** The population size is the initial number of random tours that are created when the algorithm starts. A large population takes longer to find a result. A smaller population increases the chance that every tour in the population will eventually look the same. This increases the chance that the best solution will not be found.
- **Neighborhood / Group Size -** Each generation, this number of tours are randomly chosen from the population. A large group size will cause the algorithm to run faster, but it might not find the best solution.
- **Mutation % -** The percentage that each child after crossover will undergo **mutation**.When a tour is mutated, one of the cities is randomly moved from one point in the tour to another.

Parameters

- **Nearby Cities -** As part of a greedy initial population, the GA will prefer to link cities that are close to each other to make the initial tours. When creating the initial population this is the number of cities that are considered to be close.
- **Nearby City Odds % -** This is the percent chance that any one link in a random tour in the initial population will prefer to use a nearby city instead of a completely random city.
- **Maximum Generations -** How many crossovers are run before the algorithm is terminated.
- **Random Seed -** This is the seed for the random number generator. By having a fixed instead of a random seed, you can duplicate previous results as long as all other parameters are the same.

Issues for GA Practitioners

- Choosing basic implementation issues:
  - representation
  - population size, mutation rate, ...
  - selection, deletion policies
  - crossover, mutation operators
- Termination Criteria
- Performance, scalability
- Solution is only as good as the evaluation function

Benefits of Genetic Algorithms

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed

Benefits of Genetic Algorithms

- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications
- Substantial history and range of use

When to Use a GA

- Alternate solutions are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing solution
- Benefits of the GA technology meet key problem requirements

Some GA Application Types

Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning