Artificial Intelligence Lab
AL-2002

Lab 12

Instructor: Hurmat Hidayat

Semester: Spring 2023

Artificial Intelligence Lab 12

Objective

To provide an understanding of Genetic Algorithms and their application in solving optimization problems.

Learning Outcomes

1. Define what Genetic Algorithms are and describe their underlying principles.
2. Identify the types of problems for which Genetic Algorithms are suitable.
3. Describe the basic steps involved in a Genetic Algorithm and how it works.
4. Implementation of Genetic algorithm.

Table of Contents

Genetic Algorithm

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

What are Genetic Algorithms?

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on concepts of natural selection and genetics. Gas are a subset of a much larger branch of computation known as Evolutionary Computation.

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals. This is

in line with the Darwinian Theory of "Survival of the Fittest".

In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

A genetic algorithm (GA) is great for finding solutions to complex search problems. They're often used in fields such as engineering to create incredibly high quality products thanks to their ability to search a through a huge combination of parameters to find the best

match. For example, they can search through different combinations of materials and designs to find the perfect combination of both which could result in a stronger, lighter and overall, better final product. They can also be used to design computer algorithms, to schedule tasks, and to solve other optimization problems. Genetic algorithms are based on the process of evolution by natural selection which has been observed in nature. They essentially replicate the way in which life uses evolution to find solutions to real world problems. Surprisingly although genetic algorithms can be used to find solutions to incredibly complicated problems, they are themselves pretty simple to use and understand.

GA – Motivation

Genetic Algorithms have the ability to deliver a "good-enough" solution "fast-enough". This makes genetic algorithms attractive for use in solving optimization problems. The reasons why GAs are needed are as follows −

Solving Difficult Problems

In computer science, there is a large set of problems, which are NP-Hard. What this essentially means is that, even the most powerful computing systems take a very long time (even years!) to solve that problem. In such a scenario, GAs prove to be an efficient tool to provide usable near-optimal solutions in a short amount of time.

Getting a Good Solution Fast

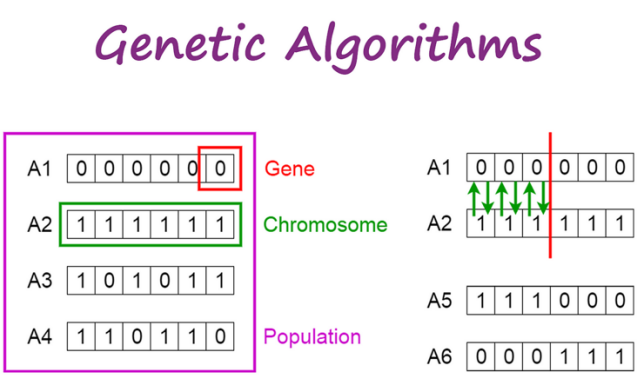
Some difficult problems like Travelling Salesperson Problem (TSP), have real-world applications like path finding and VLSI Design. Now imagine that you are using your GPS Navigation system, and it takes a few minutes (or even a few hours) to compute the "optimal" path from the source to destination. Delay in such real world applications is not acceptable and therefore a "good-enough" solution, which is delivered "fast" is what is required.

Basic Genetic Algorithm

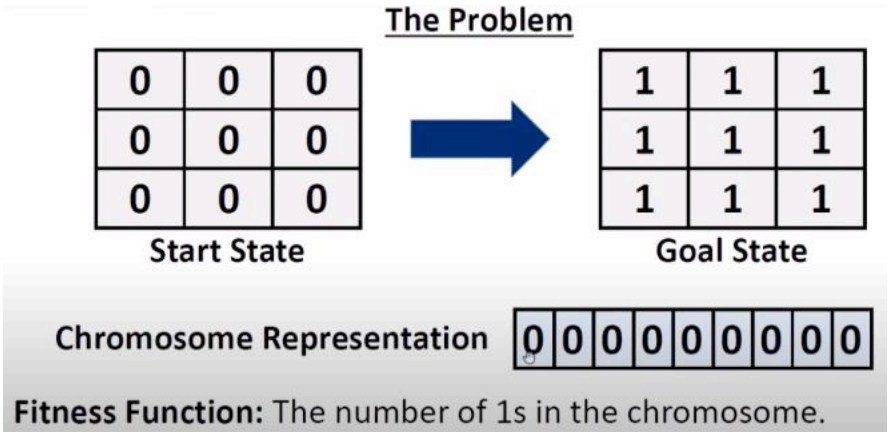
Outline of the Basic Genetic Algorithm:

1. **[Start]** Generate random population of n chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness f(x) of each chromosome x in the population
3. **[New population]** Create a new population by repeating following steps until the new population is complete

- **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
- **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
- **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
- **[Accepting]** Place new offspring in a new population
- **[Replace]** Use new generated population for a further run of algorithm
- **[Test]** If the end condition is satisfied, stop, and return the best solution in current population
- **[Loop]** Go to step 2



Example



Step 1: Initial Population

Select any random states as initial population i.e.

No. of population is a random or your choice.

Values in each population represents a random state.

P1 =	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	Fitness = 1
P2 =	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	Fitness = 2
P3 =	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	Fitness = 2
P4 =	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	Fitness = 4

Calculate the Fitness Value of each Population

Fitness (fn) = No. of 1's in a population

Step 2: Selection of Parents

1. Parents will be selected random from the population in the first step.
2. High fitness value population will have high chances of selection as a parent i.e. P4 will have high weightage to be selected as randomly

   Selection_Probability P(i) = Fitness of P(i)/Total Fitness of all Populations

Step 3: Modification

Crossover:

1. This is called convergence step because it will generate children (new states)
2. Create children from meeting of the parents
3. Generate two children/successor from two parents (new version of genetic algorithm produce one child instead)

Mutation:

1. Apply mutation on children in order to get the new/updated state (assuming it will quickly lead us to goal)
2. Mutation probability or mutation rate is fixed and chosen a very small value i.e. 0.01 means generate a random no. between (1-100), mutate the child if random no. is 1 else skip. i.e. 0.2 means generate a random no. between (1-10), mutate the child if random no. is 1 or 2. Skip otherwise

3. Mutation rate will be applied and checked for each digit/char in a population i.e. You will keep repeating this process for each array value in a single population means 16 times random no. will be generated and checked respectively.

Step 4: Evaluation

1. Compute the fitness values of newly generated children
2. Apply the goal test
3. Replace the old population with newly created population having new children
4. Repeat the steps 1-4 if goal is not found.

Advantages:

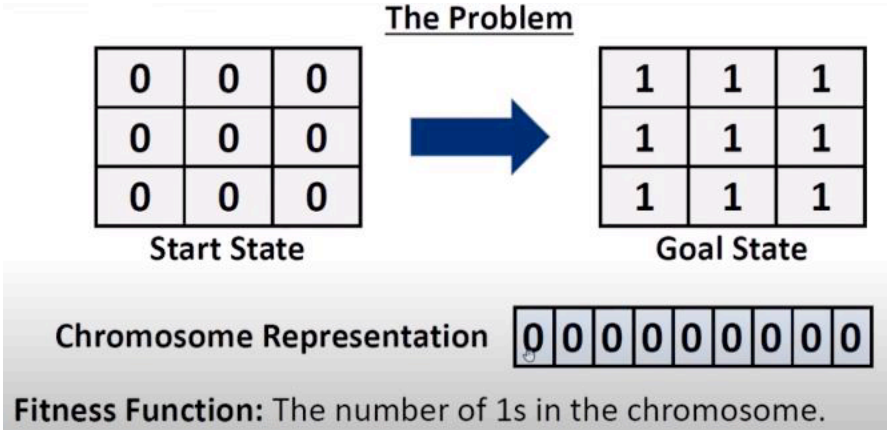
- Modular, separate from application
- Answer gets better with time
- Inherently parallel; easily distributed
- Genetic algorithms use fitness score, which is obtained from objective functions, without other derivative or auxiliary information.

Disadvantages:

- Genetic Algorithms might be costly in computational terms since the evaluation of each individual requires the training of a model.
- These algorithms can take a long time to converge since they have a stochastic nature

Lab Task

Consider the following problem and using genetic algorithm obtain



- Implement a genetic algorithm to solve a 2D array problem.
- The problem has an initial state of a 4x4 array filled with zeroes and a goal state of a 4x4 array filled with ones.
- The fitness function for each state is the number of ones in the array.
- Convert the 4x4 2D array into a 1D array of length 16 to work with chromosomes.
- Use selection, crossover, and mutation operations to generate new states.
- For selection, randomly select two parents from the population and calculate their selection probabilities based on their fitness values. (Roulette Wheel Selection)
- For crossover, perform a single-point crossover operation on the selected parents to create two children.
- For mutation, apply a mutation operator with a low probability value to introduce new genetic material.
- Calculate the fitness values of the two children and apply the goal test to check if either of the children is the goal state.
- Replace the least fit members of the population with the new children.
- Repeat the process until a solution is found or after a certain number of iterations.
- Test the algorithm to find a solution for the given problem.