## Introduction

This experiment present a local search process by using two kinds of representation method (Binary and Gray Code), and three kinds of mutation methods (Randomly, Bit Flips and Increase/Decrease in one dimension). The result shows different effects from each methods. In short, different mutation methods may cause different searching space, and different representation methods may effect searching precision.

## Method

In this experiment, I wrote a program to create six different scenarios to search the local optima in a specific space. First, I generate a random number, and extract its 20 LSB and break it into two chromosome: x and y, then I scale it in a specific range, convert it to float as input for the fitness function. Each chromosome has 10 bits length, the range is between 0~1023, so chromosome x and y can make 1024 x 1024 = 1048576 kinds of set. It's also the total space which we will need to search the local maxima.

Here I used three different methods to mutate chromosome x and y for searching the better "FITNESS". The fitness is an index of "How good it is", we can determine how good is our chromosome made by x and y. Here are the three mutation methods:

**1. Randomly Jumping**

In this method, I just keep generating random numbers. every time I generated a new number, I named it "mutated_chromosome", and break it into mutated_x and mutated_y, and use this two values to scale the fitness. If the mutated_chromosome has better fitness than previous chromosome, then I replace it. Loop this process for 10000 times, the result shows x is linearly forward to 1.0, and y to 3.0, this makes the fitness getting close to fitness 1.0.

**2. Bit Flip**

In this method, I extract the 20 LSB from input as original_chromosome, and scale and store the fitness. Then I duplicate this chromosome, and randomly flip one of its bit. This means, first I decide which bit position should I change, then flip it (0 -> 1, 1 -> 0). After flip the bit, I break it into mutated_x and mutated_y, then scale its fitness.

**3. Single Dimension Inc/Dec**

In this method, I duplicate the original_chromosome, then break it into x and y. Then randomly pick either x or y, and either +1/-1 it. So it has 4 kinds of mutation possibilities, and each one has 25% chance to be picked. After alter x or y, I reassembly it to be the mutated_chromosome, and use it to measure and search better fitness.

With these three mutation methods, I also used two different representations in this experiment. One is Binary, and the other is Gray Code. In the experiment, it shows obviously difference by different representations.

**1. Binary**

Binary representation works well for the mutation methods Randomly Jumping and Single Dimension Inc/Dec.  Both of them leads to get average fitness 0.999 in 1000 times test, but only get 0.541average fitness in 1000 times test for Bit Flip method.

**2. Gray Code**

Different as Binary representation, the Gray Code representation works well with Randomly Jumping and Bit Flip method, but not for Single Dimension Inc/Dec. For Randomly Jumping, I got 0.993 average fitness, for Bit Flip, I got 0.999, but for the Inc/Dec, I got such a bad fitness about 0.05.

| ***Average Fitness*** | Randomly Jumping | Bit Flip | Single Dim Inc/Dec |
|---|---|---|---|
| Binary | 0.993211 | 0.534123 | 0.999924 |
| Gray Code | 0.993821 | 0.999924 | 0.050902 |

by 1000 times test

## Conclusion

When I am doing searching with randomly jumping mutation method, the representation does not really effect it. This is because of we are always generate new chromosome randomly, and we don’t really care about what is the neighbor's fitness. In general,  randomly jumping search global space, so it will not stuck on the local maxima, but I guess this can only be applied on some specific situation, or it will not be a stable method for evolution.

Bit Flip works not well with Binary but not Gray Code representation. This is because of the mutation range; if we only flip one bit in the Binary code, the change is too small and easy to get stuck on the local maxima. But flip one bit in the Gray Code gives bigger changes in real number, this helps to leave the local maxima and get the chance to find the better choice. That is the reason that the fitness can be improved from 0.534 to 0.999 by just changing the representation with the Bit Flip mutation method.

Single Dimension Inc/Dec works well for the binary in this experiment. it has fairly chance to move forward or backward even it’s already on the local maxima. It means, this helps to try “bad moves”.  But it’s not good for the Gray Code representation, since when it decode from Gray Code to Binary, the real value changes without rule. So, if the initialization locate at a bad point, this method will never be able to find a better result.

This experiment shows how important of choosing representation and mutation methods for a particular question. How to find a reasonable representation and mutation method, is one of the most important topic to do "Right Evolution".