Snake Game Automation: Genetic Algorithm

1.1 Introduction

1.2 Project outline

1.3 Genetic algorithm flow

1.4 Crossover, Mutation, Survivor selection

1.5 Results

Sai Ram Manohar Koya

cs.ualr.edu

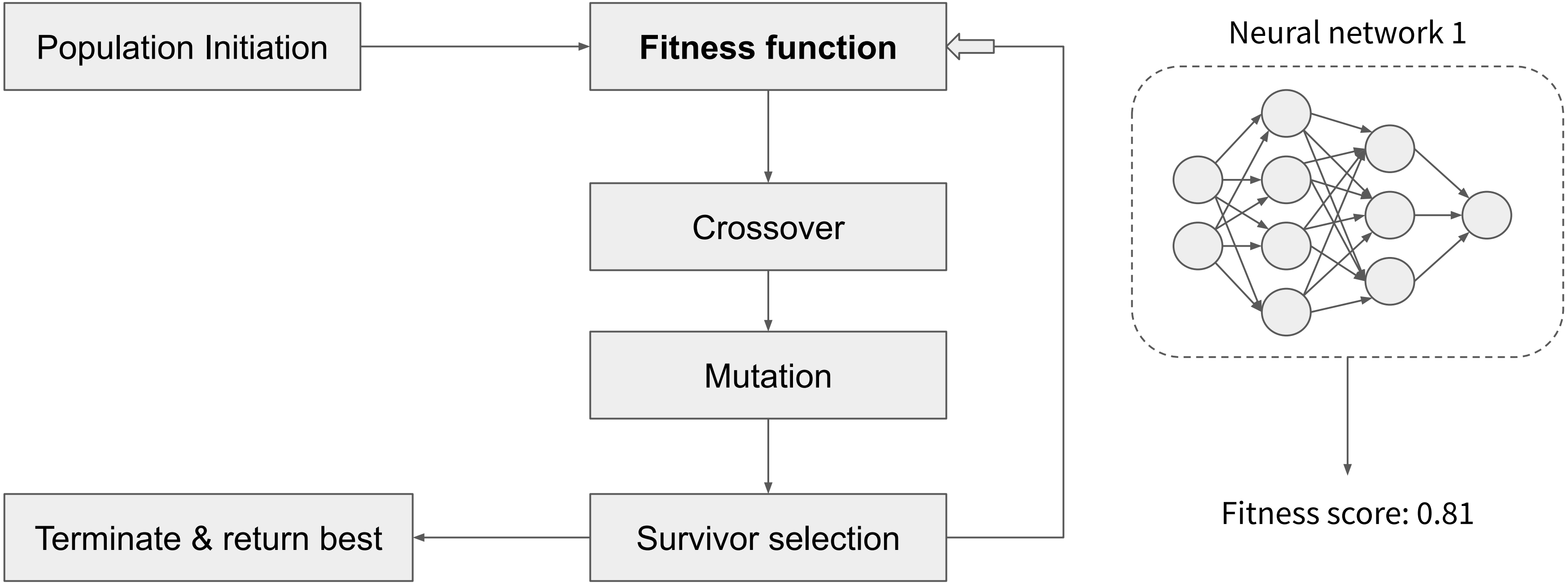
1.1 Introduction

- Genetic Algorithm
  - Genetic algorithm is a natural selection algorithm inspired by living creatures in nature.
  - Based on Darwin's theory of evolution
    - *“survival of the fittest”*
  - Any problem can be solved by using three common techniques
    - Selection
    - Crossover
    - Mutation
  - Subset of Evolutionary Computation

1.2 Project outline

- Idea
  - Game automation is a widely growing concept in the computation industry
    - Ex: Open AI 5 defeats dota-2 world champions [1]
- Objective
  - Objective of this project is automate a simple old Nokia snake game with walls in the border
  - Trying to use Genetic Algorithm for the game automation
  - Genetic algorithm is based on natural selection. Hence, we adapt is as a snake brain

1.3 Genetic algorithm flow



Population Initiation

Fitness function

Crossover

Mutation

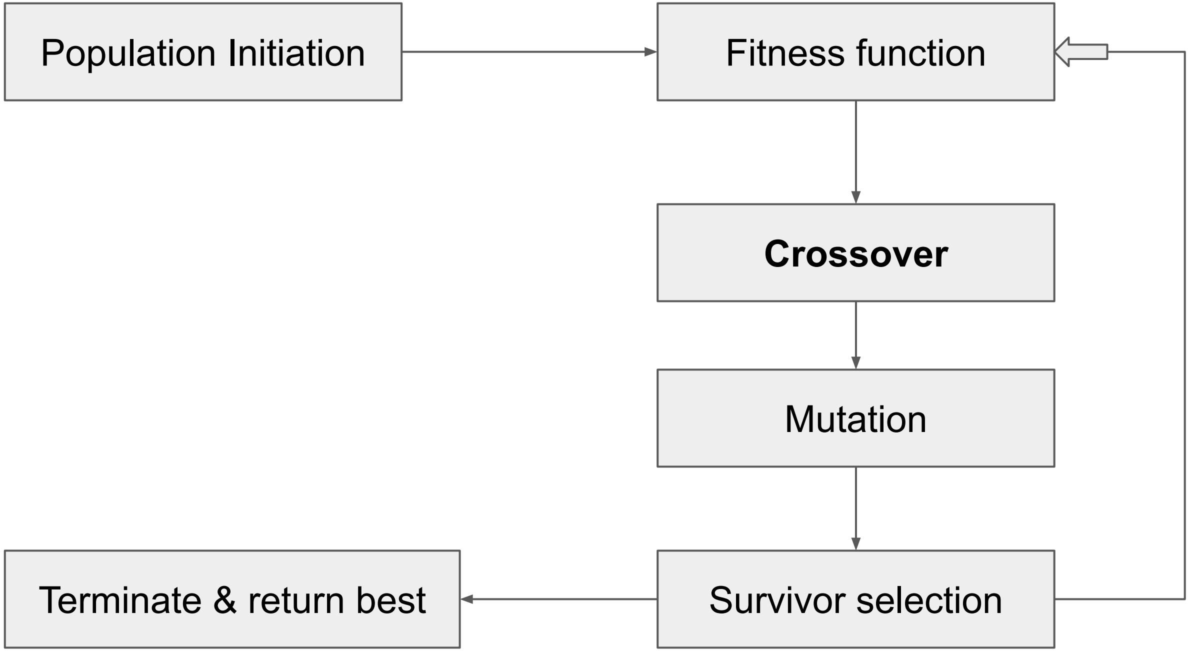
Survivor selection

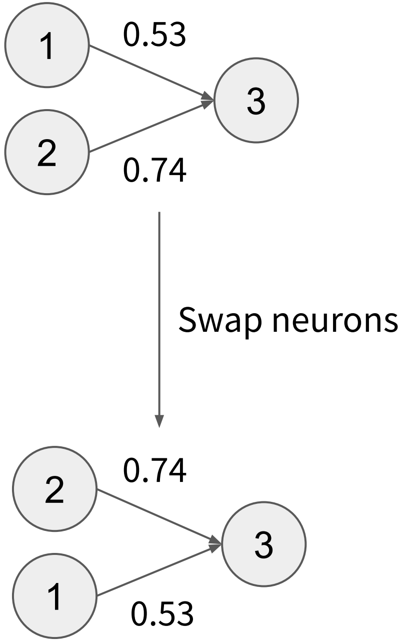
Terminate & return best

Neural network 1

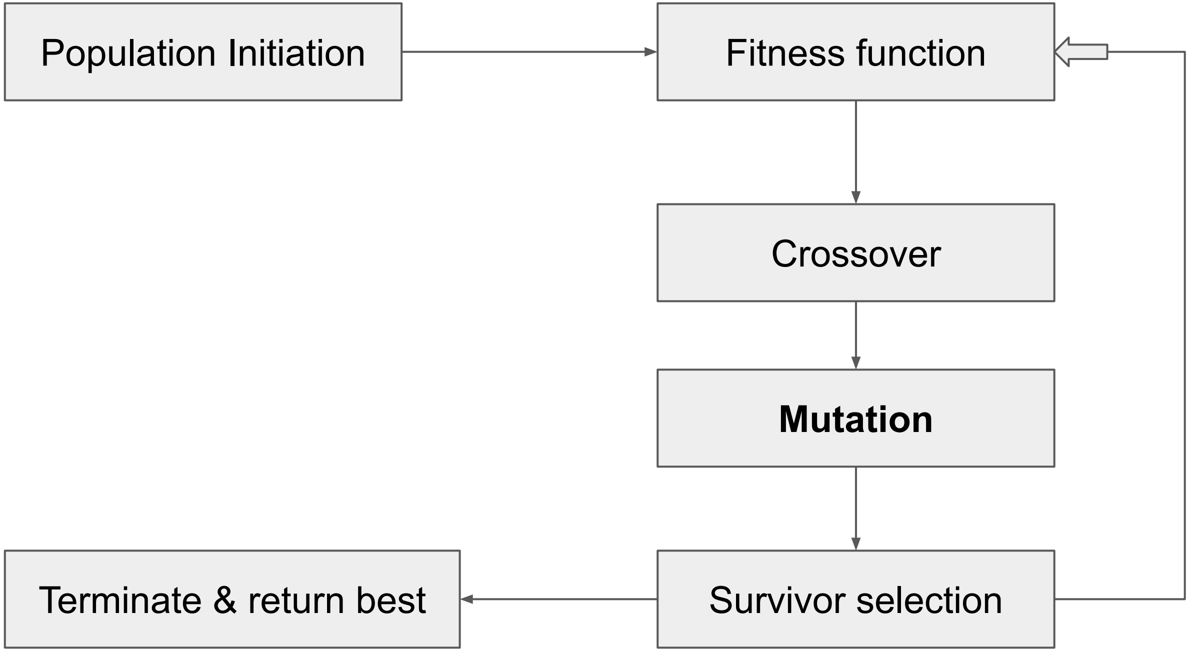
Fitness score: 0.81

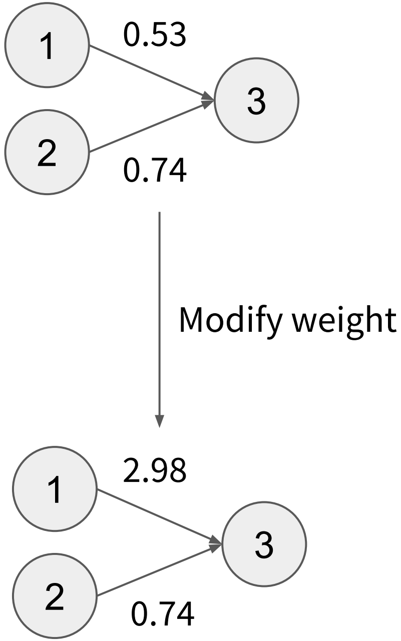
1.3 Genetic algorithm flow





1.3 Genetic algorithm flow





1.4.1 Parent selection

- Select a list of parent population for crossover and mutation
- Pseudocode

```
parent_selection(crossover_size, population):
    parents = []
    for i in range(crossover_size):
        random_population_list = get_random_list(population)
        parent = get_max_performer(random_population_list)
        parents.append(parent)
    return parents
```

1.4.2 Crossover

- Crossover the parents selected to get variation of possibilities
- Pseudocode

```
produce_children(selected_parents, crossover_size):
    Children = []
    For i in range(crossover_size):
        Child = crossover(random_parent_1, random_parent_2)
        children.append(child)
    Return children
crossover(net_1, net_2):
    new_net_1, new_net_2 = exchange_random_neuron(net_1, net_2)
    net = get_max_performer(new_net_1, new_net_2)
    return net
```

1.4.3 Mutation

- Mutate over crossover children to get better performing generation
- Pseudocode

```
mutate(neural_net, mutation_type):
    if mutation_type = “weight”:
        neural_net = change_random_weight(neural_net)
    else if mutation_type = “neuron”:
        neural_net = change_random_neuron(neural_net)
    return neural_net
```

1.4.4 Fitness function & score

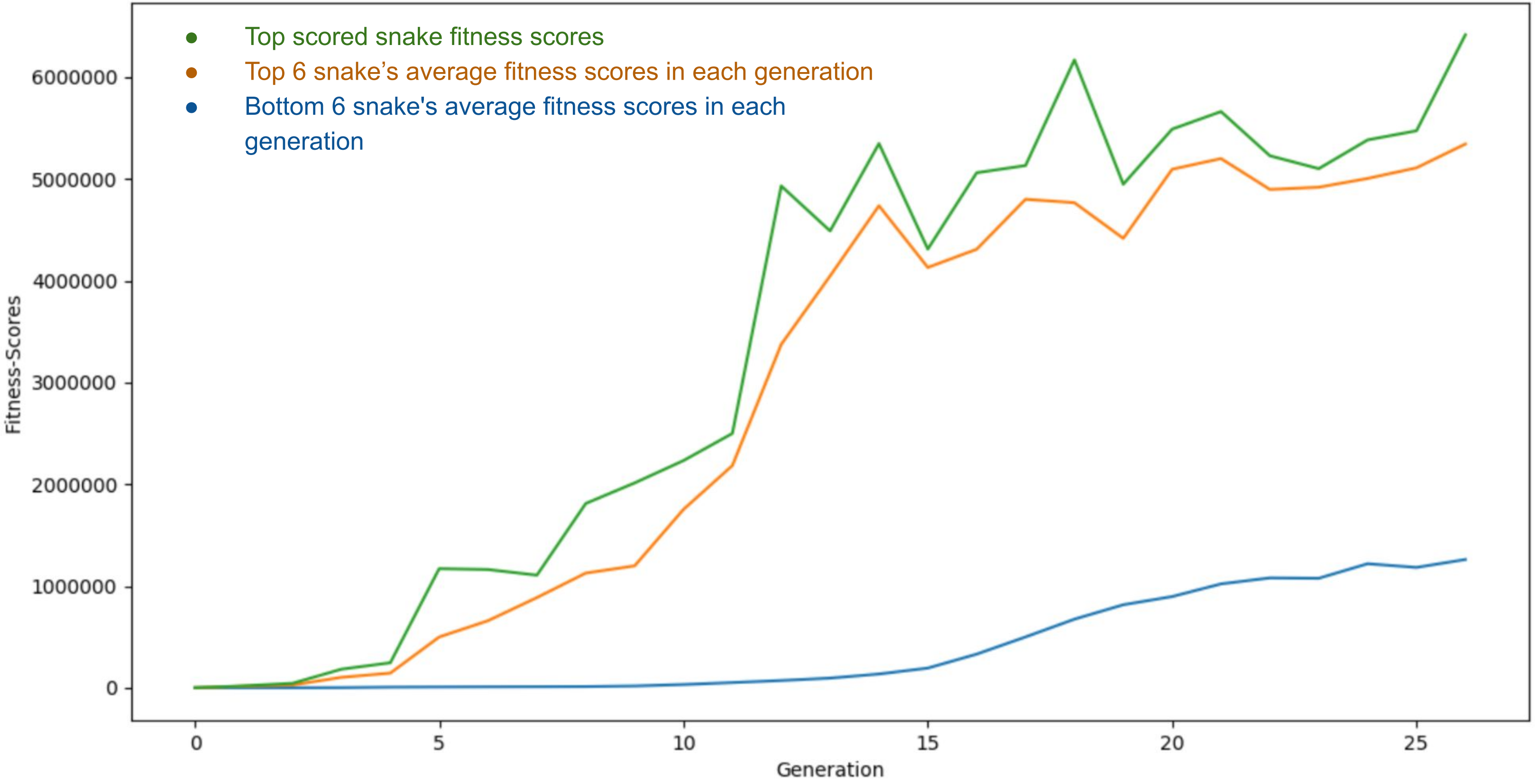
- Fitness function changes according to the requirement of the problem
- Fitness score is also a defined parameter according to the problem
- Mutated elements needs to be passed in fitness function to select the top performers for the next generation
- Pseudocode

```
fitness_function(mutated_population, generation_size):
    population_with_fitnesses = compute_fitness(mutated_population)
    sorted_population = sort_population(population_with_fitnesses)
    return sorted_population[:generation_size]
```

1.4.5 Survivor selection

- Based on the above step, fitness scores
- We sort the population according to the fitness score
- Survivor selection is the process of selecting the population which has more fitness score
- Hence, we only proceed with the population of generation size with highest fitness score
- Reiteration:
    - Reiteration is the process of repeating all the steps mentioned above (crossover, mutation, selection) over multiple generations
    - Provides the optimal results over the generations

1.5 Results



● Top scored snake fitness scores
● Top 6 snake’s average fitness scores in each generation
● Bottom 6 snake's average fitness scores in each generation
Fitness-Scores
6000000
5000000
4000000
3000000
2000000
1000000
0
0
5
10
15
20
25
Generation

1.6 Summary

- Basics of Genetic algorithm
- Terminology
- Genetic algorithm flow
- Parent selection
- Crossover, mutation, survivor selection

Questions?

Thank You

This page is intentionally left blank