# ICS674 Mini Project

Lambert Leong

September 15, 2018

For my evolutionary computation mini project I implemented a genetic algorithm that matches an input string. Code implementations are written in Python2.7.

## 1 Search Space

The search space consists of all alphabet characters, upper and lower case. This can be seen in Listing 1 where we randomly fill the search strings with letters by calling `string.letters` from Pythons string module.

Listing 1: Search space is all letter characters, upper and lower case

```
self.string = ''.join(random.choice(string.letters) for _ in xrange(length))
```

## 2 Variation Operator

Two variation operator are implemented which include crossover and mutation.

### 2.1 Crossover

To perform crossover we we randomly select two parent strings from the previous generation, seen in lines 4&5 of Listing 2. I randomly select an integer which corrisponds to the array index at which the parent string is split for each child to inherit, seen i line 8. Child 1 gets the char from the first half from parent 1 and the second half from parent 2. Child 2 getst the first half from parent 2 and second half from parent 1.

Listing 2: Crossover Function

```
1 def crossover(individuals):
2         offspring = []
3         for _ in xrange((population - len(individuals))/2):
4                 parent1 = random.choice(individuals)
5                 parent2 = random.choice(individuals)
6                 child1 = Individual(in_str_len)
7                 child2 = Individual(in_str_len)
8                 split = random.randint(0, in_str_len)
```

```
                child1.string = parent1.string[0:split] +
                    parent2.string[split:in_str_len]
                child2.string = parent2.string[0:split] +
                    parent1.string[split:in_str_len]
                offspring.append(child1)
                offspring.append(child2)
        individuals.extend(offspring)
        return individuals
```

## 2.2 Mutation

Mutation occurs in the for of switching out a character in a search string with a random letter. All strings in the population are suseptible to mutation and multiple mutation can occur in a individual string. The mutation rate is 5% as indicated in line 4 of Listing 3.

Listing 3: Mutation Function

```
1 def mutation(individuals):
2         for individual in individuals:
3                 for i, param in enumerate(individual.string):
4                         if random.uniform(0.0, 1.0) <= 0.05:
5                                 individual.string = individual.string[0:i] +
                                      random.choice(string.letters) +
                                      individual.string[i+1:in_str_len]
6         return individuals
```

# 3 Selection Operator

```
def selection(individuals):
        individuals = sorted(individuals, key=lambda individual:
            individual.fitness, reverse=True)
        max_fit.append(max(individuals, key=lambda individual:
            individual.fitness).fitness)
        min_fit.append(min(individuals, key=lambda individual:
            individual.fitness).fitness)
        avg_fit.append(float(sum(i.fitness for i in
            individuals)//len(individuals)))
        individuals = individuals[:int(0.2*len(individuals))]
        return individuals
```

# 4 Termination Criterion

```
 for generation in xrange(generations):
                generation_list.append(generation)
```

```
        individuals = fitness(individuals)
        individuals = selection(individuals)
        individuals = crossover(individuals)
        individuals = mutation(individuals)
        if any(individual.fitness >= 100 for individual in individuals):
               found = True
               break
```

## 5 Objective Fuction

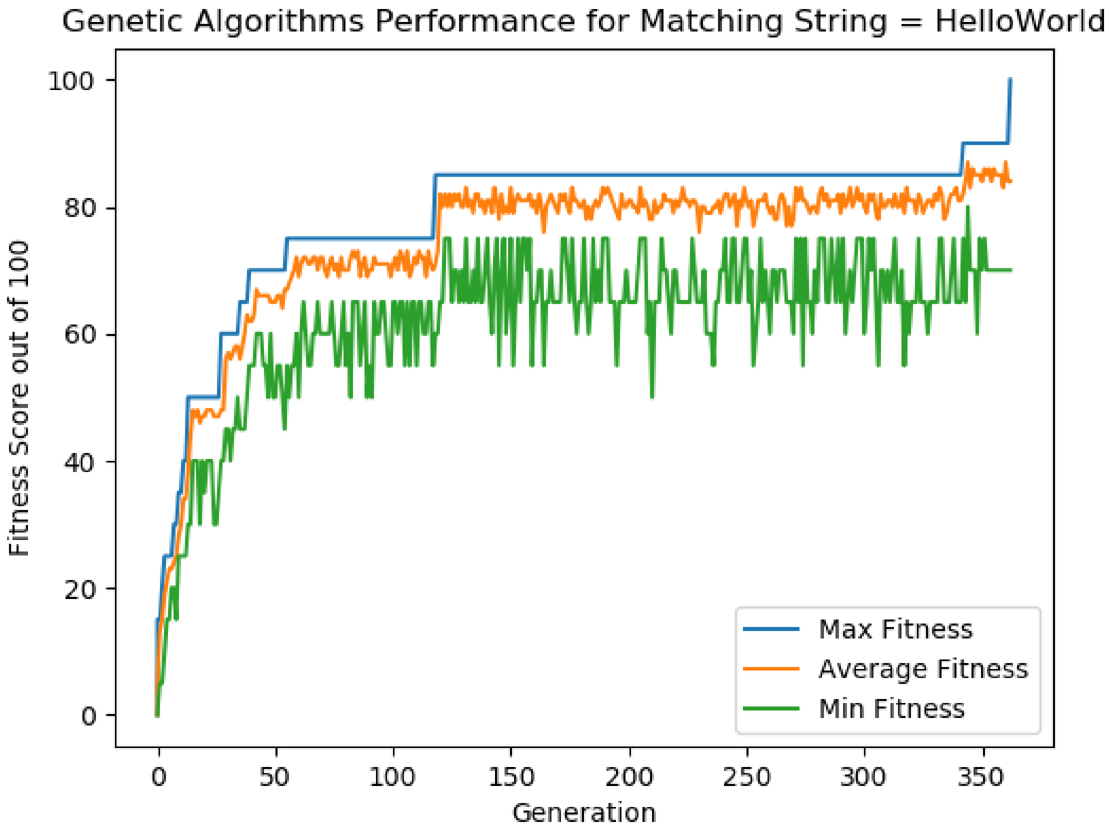

Figure 1: Max, average, and min fitness values of each indvidual string for each generation

Listing 4: Fitness Function

```python
def fitness(individuals):
    for individual in individuals:
        total = len(in_str)*2
        score = 0
        for i, letter in enumerate(individual.string):
                if in_str[i] == letter:
                    score += 1
        compare_str = in_str
        for a_char in individual.string:
            for i, in_char in enumerate(compare_str):
                if a_char == in_char:
                    score += 1
                    compare_str =
                        compare_str[:i]+compare_str[i+1:]
                    break
        individual.fitness = int((float(score)/float(total))*100)
    return individuals
```