ICS674 Mini Project

Lambert Leong

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

```
1 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 corresponds 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 gets 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)
9                 child1.string = parent1.string[0:split] +
                      parent2.string[split:in_str_len]
```

```
10                child2.string = parent2.string[0:split] +
                      parent1.string[split:in_str_len]
11                offspring.append(child1)
12                offspring.append(child2)
13        individuals.extend(offspring)
14        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 susceptible 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

Individual strings are sorted based upon their fitness scores, as seen in Listing 4. Only the top 20% of the individual strings are kept for each generation, which can be seen in line 6. Lines 3-5 are for graphing purposes and not part of the selection function.

Listing 4: Selection Function

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

4 Termination Criterion

Listing 5 displays the termination code. The algorithm terminates when it has reached the last generation, seen i line 1, or when the fitness is 100, seen in line 7. Fitness scores are out of a 100 and a score of 100 equates to an identical and successful string match.

Listing 5: Termination

```
1 for generation in xrange(generations):
2                 generation_list.append(generation)
3                 individuals = fitness(individuals)
4                 individuals = selection(individuals)
5                 individuals = crossover(individuals)
6                 individuals = mutation(individuals)
7                 if any(individual.fitness >= 100 for individual in individuals):
8                         found = True
9                         break
```

5 Objective Fuction

Figure 1 indicates the max, min, and average fitness scores in each generation. During this run, the string was matched in a little over 350 generation. The blue line reaches a fitness score of 100 which is the max fitness score and indicates a successful match of the input string which was "HelloWorld". Fitness scores were calculated with the objective function shown in Listing 6

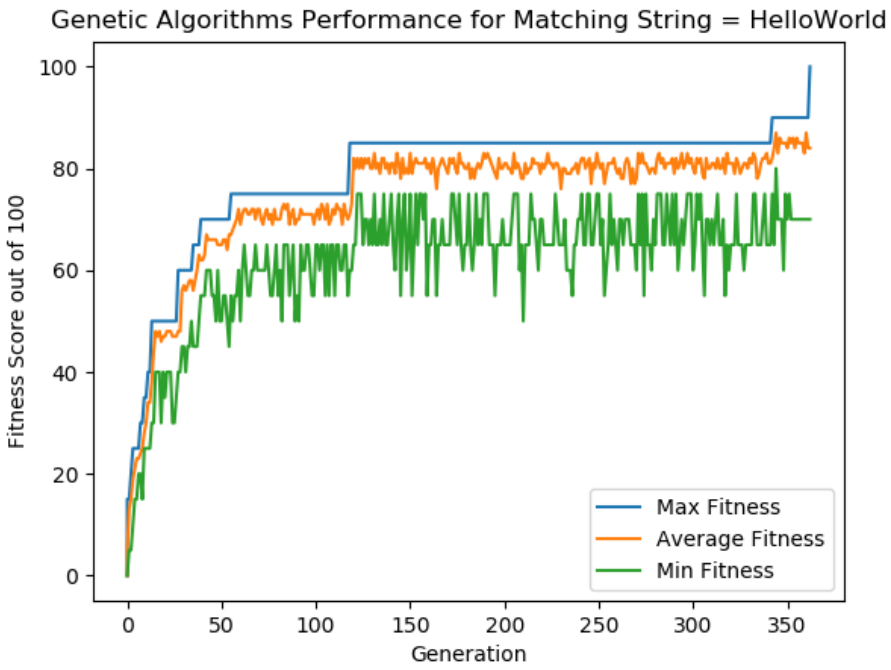

Figure 1: Max, average, and min fitness values of each individual string for each generation

As mentioned above, fitness scores are out of 100 and they are calculated as percentages. The `total` is given the value of twice the length of the string, seen in line 3. If the search/individual string contains a letter that is also contained in the input string, the fitness `score` is incremented by 1. This process occurs in lines 9-14. Once a letter from the search string is found to be in the input string, that letter is removed from the input string for future searches as to avoid duplicates and unwanted fitness score increases.

Scores are also incremented if the search string contains the right letter in the right location or at the same index as the input string, which can be seen in lines 5-7. The score is then divided by the total and multiplied by 100 to get the resulting fitness score, shown in line 15.

Listing 6: Fitness Function

```
1  def fitness(individuals):
2          for individual in individuals:
3                  total = len(in_str)*2
4                  score = 0
5                  for i, letter in enumerate(individual.string):
6                                  if in_str[i] == letter:
7                                          score += 1
8                  compare_str = in_str
9                  for a_char in individual.string:
10                         for i, in_char in enumerate(compare_str):
11                                 if a_char == in_char:
12                                         score += 1
13                                         compare_str =
                                               compare_str[:i]+compare_str[i+1:]
14                                         break
15                 individual.fitness = int((float(score)/float(total))*100)
16         return individuals
```