Evolutionary Computation Sort

Tyson Seto-Mook
September 16, 2017

Background

Problem

- Sort a list of integers in ascending order.
- The search space for this problem are all permutations of the given list.
    - This give us a search space of n!, where n is the size of the list to sort.

Genetic Algorithm Framework

- Common steps
    1) generate a population
    2) generate offspring
    3) reduce the population to original size

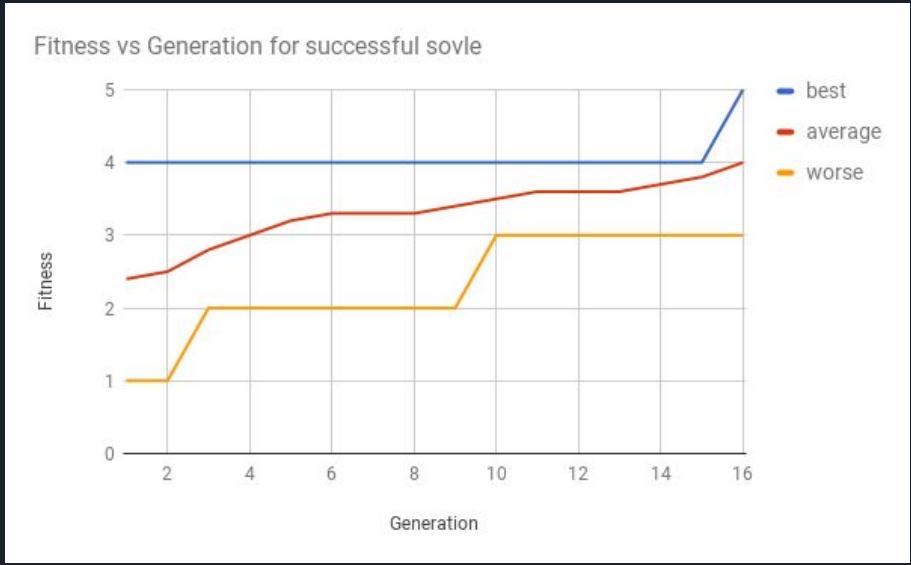
```python
#!/usr/bin/env python
import random
import itertools
from Individual import Individual

class GeneticAlgorithmSort(object):
  def sort(self):
    generation_count = self.generations
    #create initial m population
    self.createInitialPopulation()
    #keep evolving while not max generations (iterations) or sorted list found
    while (generation_count > 0) and self.population[0].fitness() < self.optimalFitness:
      #update generation count
      generation_count -= 1
      # reset offspring list
      self.offspring = []
      # generate n offspring (parents should be same size as offspring size)
      # select parent(s) (a list of individual or individual couples)
      parents = self.parentSelect(self.population, self.offspringSize)
      for parent_s in parents:
        # produce offspring (crossover, etc.) reproduce returns a list
        offspring = self.reproduce(parent_s)
        offspring = self.mutation(offspring)
        # mutate (random swap)
        self.offspring.append(Individual(offspring, self.fitness))
      #reduce population from m+n down to n
      self.population = self.selection(self.population, self.offspring, self.populationSize)
    # return sorted list if while has exited
    return self.population[0].getIndividual()

  def createInitialPopulation(self):
    # add initial list to population
    self.population = [Individual(self.original_list, self.fitness)]
    # generate permutation of original list
    permutations = list(itertools.permutations(self.original_list))
    # randomly select m individuals from the permutation list to add to the population
    for i in range(self.populationSize-1):
      rand_idx = int( random.random()*len(permutations) )
      self.population.append(Individual(permutations[rand_idx], self.fitness))
```

Tunable Methods

- Fitness Methods
  - Number of correctly sorted values based on the first value
  - E.g [3, 1, 2, 5, 4]
- Parent Selection Methods
  - Fittest individual from population
- Reproduction Methods
  - Clone parent
- Mutation Methods
  - Randomly swap 2 indices from the list
- Selection Methods
  - Truncation of fittest individuals

Results


Fitness vs Generation (close solution found)
Fitness
Generation
best
average
worse
0
1
2
3
4
5
0
100
200
300


Fitness vs Generation for successful sovle
Fitness
Generation
best
average
worse
0
1
2
3
4
5
2
4
6
8
10
12
14
16

Conclusion

- Maybe genetic algorithms are not good for sorting problems
  - genetic algorithm guarantee a close solution but not the optimal solution

- Exploring different ways of producing offspring would be the next step
  - Parent selection
  - Reproduction
  - Mutation methods