Random Genetic String Matcher

A Slight Biological Approach

Anson Yu
9/19/2017

Introduction

- Disclaimer
  - The Genetic Algorithm was fairly simple and randomness was indeed involved
    - Every step of the way…
- Main Objective
  - To mimic Biology and Simple Genetics

The Details

- The Search Space
  - Strings of length 16 from a pool of 60 characters
- Objective/Fitness Function
  - Scores character for character between string and target string
- Variation Operators
  - Mutations & Crossovers
- Selection Operators
  - Diseased Competition and Non-Diseased Competition

How it Works

Step-1: Generate an initial population of random strings for the next generation to spawn off

Step-2: Evolve into an N population size generation containing random mutants of the initial population and randomize the population order of the generation.

Step-3: Have the new population compete within itself in order to generate the new population and compare fitness each individual in the population.

Step-4: Repeat steps 2-3 until at least one individual of the population evolved into the target string.

Search Space

```
var poolOfCharacters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&*';

var Target = {
    target:  "AbcdeIsARealName",
};
```

Size of Search Space: 2.82×10^{28} different strings

Fitness Function

```
function fitnessFunction(phenotype) {
    var score = 0;
    var i,j;
    for (i=0;i<phenotype.length;++i) {
        if (phenotype[i] == Target['target'][i]){
            score += 1;
        }
        score += (127-Math.abs(phenotype.charCodeAt(i) - Target["target"].charCodeAt(i)))/60;
        }
    return score;
}
```

Fitness function gives full point and partial points to allow evolution to move towards target.

Mutation

```
function mutationFunction(phenotype) {
    var chance = Math.random();
    if(chance >= 0.5){
    function replaceAt(str, index, character) {
        return str.substr(0, index) + character + str.substr(index+character.length);
    }
    var i = Math.floor(Math.random()*phenotype.length);
    var res = replaceAt(phenotype, i, utils.randomString(1,poolOfCharacters));
    return res;
    }
    else{
        return phenotype;
    }
}
```

Crossover

```
function crossoverFunction(phenotypeA, phenotypeB) {
    var chance = Math.random();

    if(chance >= 0.5){
    var len = phenotypeA.length;
    var ca = Math.floor(Math.random()*len);
    var cb = Math.floor(Math.random()*len);
    if (ca > cb) {
        var tmp = cb;
        cb = ca;
        ca = tmp;
    }

    var newPhenotypeA = phenotypeB.substr(0,ca) + phenotypeA.substr(ca, cb-ca) + phenotypeB.substr(cb);
    var newPhenotypeB = phenotypeA.substr(0,ca) + phenotypeB.substr(ca, cb-ca) + phenotypeA.substr(cb);

        return [ newPhenotypeA , newPhenotypeB ];
    }
    else{
        return [phenotypeA,phenotypeB];
    }
}
```

Selection Operators

```
function compete( ) {
    var nextGeneration = []

    for( var p = 0 ; p < settings.population.length - 1 ; p+=2 ) {
        var phenotype = settings.population[p];
        var competitor = settings.population[p+1];

        nextGeneration.push(phenotype)
        if ( doesABeatB( phenotype , competitor )) {
            if ( Math.random() < 0.5 ) {
                nextGeneration.push(mutate(phenotype))
            } else {
                nextGeneration.push(crossover(phenotype))
            }
        } else {
            nextGeneration.push(competitor)
        }
    }

    settings.population = nextGeneration;
}
```

No Disease

```
function doesABeatB(a,b) {
    var doesABeatB = false;
    if ( settings.doesABeatBFunction ) {
        return settings.doesABeatBFunction(a,b)
    } else {
        return settings.fitnessFunction(a) >= settings.fitnessFunction(b)
    }
}
```

Disease

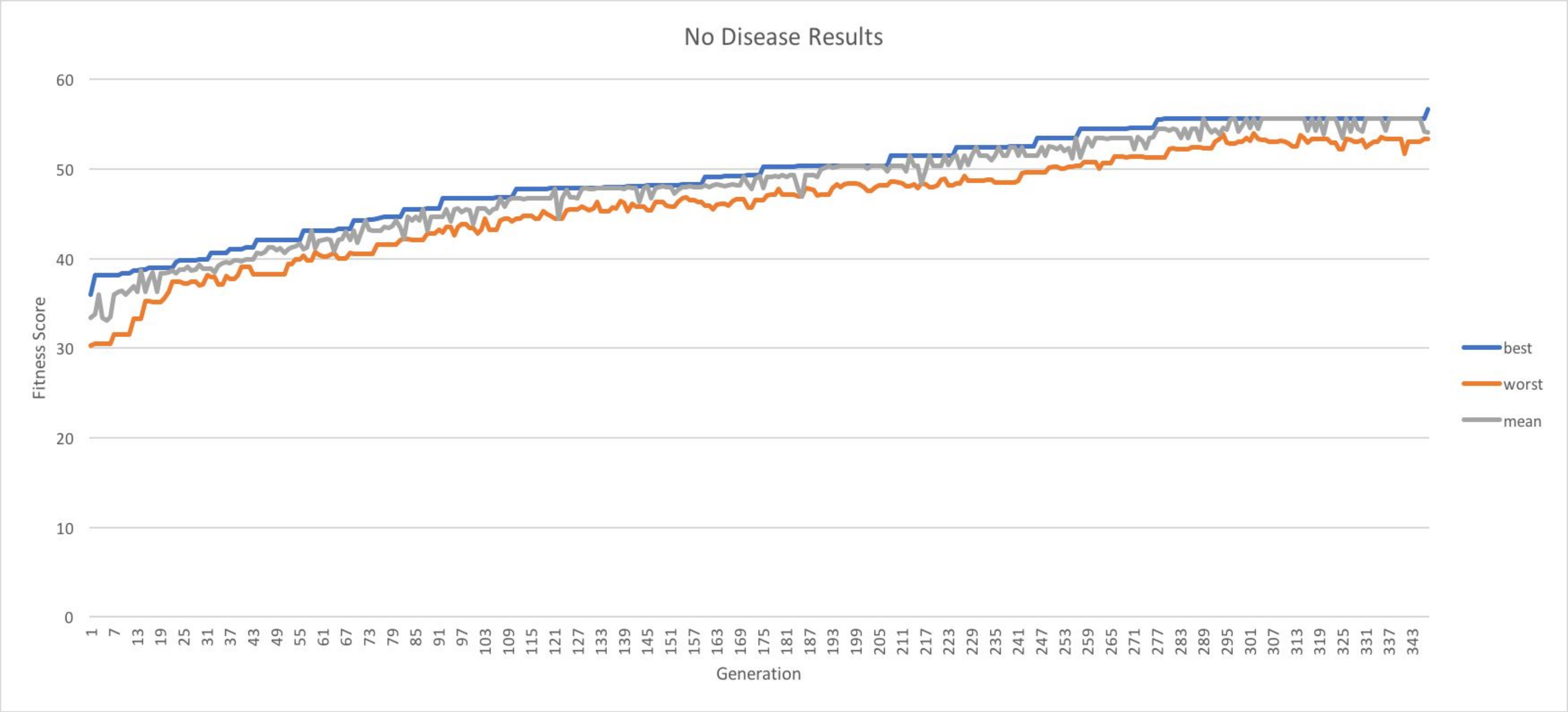
```
function diseaseCompetiton(phenotypeA,phenotypeB){
    var chance = Math.random();
    var AChance = fitnessFunction(phenotypeA);
    var BChance = fitnessFunction(phenotypeB);
    if(chance <= 0.33){
        AChance *= Math.Random();
    }
    else if(chance > 0.33 && chance <= 0.66){
        BChance *= Math.Random();
    }
    else{
        AChance *= Math.Random();
        BChance *= Math.Random();
    }

    return AChance >= BChance;
}
```

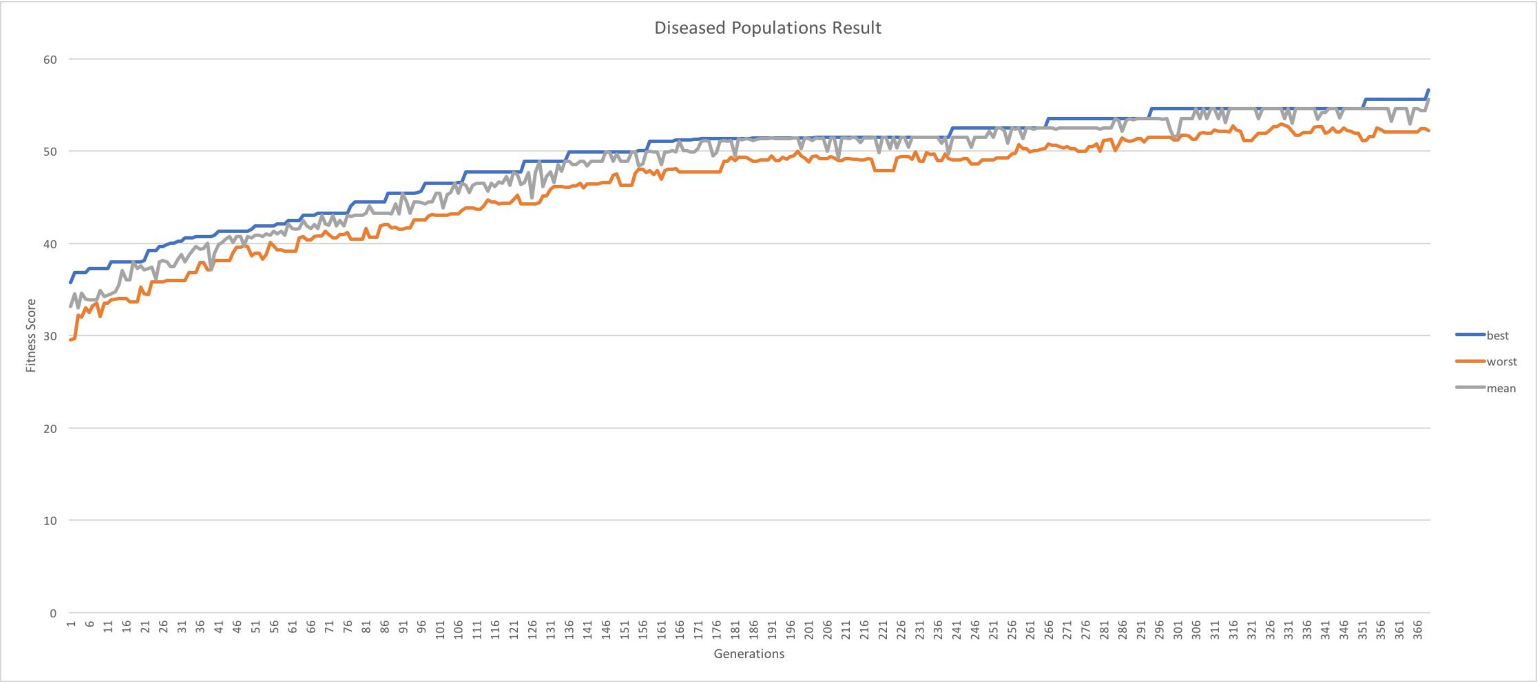
Termination

```
while(best != Target['target']){
    console.log("Generation " + gen);
    geneticAlgorithm.evolve();
    best = geneticAlgorithm.best();
    printPopulation();
    console.log("Best of This Population" + gen +":, " + geneticAlgorithm.bestScore());
    console.log("Worst of This Population"+ gen+":, "+ geneticAlgorithm.worstScore());
    console.log("Mean "+gen+":, " + geneticAlgorithm.meanPopulation());
    gen++;
}
```

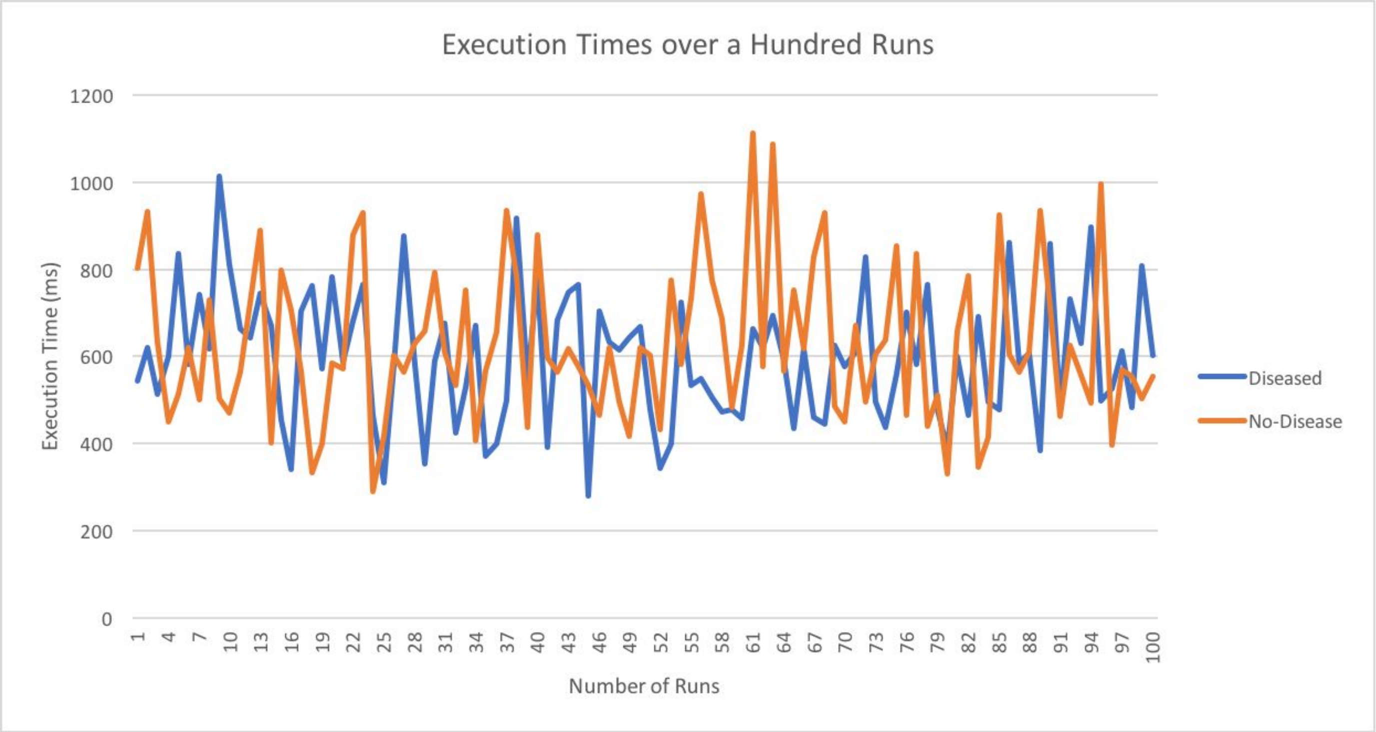
Results Without Disease



Results with Disease



Execution Runtimes



Conclusion

- The path to the objective was achieved through pure randomness.

- Throwing more and more probability doesn’t necessarily slow down the execution runtimes.

Thanks for Listening!