COMP 572

Project 1b

Sanjeev Shrestha

February 06, 2014

This is the second subproject. The goal of this project is to write part of a genetic algorithm (GA) for one of the benchmark optimization problems.

For this subproject you only need to work on the Sphereical function, defined at: <http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume24/ortizboyer05a-html/node6.html#tabla:DefFunc>. (Note the first function labeled as Schwefel on this page is actually the double sum, which we are not using. We are using the Schwefel function defined immediately after the Rastigin function.)

Pay careful attention to the ranges of the functions. You will want to use those ranges both in creating intial individuals and in controlling the generation of neighbors, e.g. you don't want your GA 'wondering' out of the search space. Note that here the functions are all defined with 30 dimensions, e.g. P = 30 in the function definitions.

**Project Requirements:**

* Write a **partial** GA to find the input values (x1,...,x30) that minimizes the Spherical function.
* The GA should include the following:
  + Fitness function
  + Algorithm type: Steady state or generational
  + Selection
  + Mutation

Note: you don't need crossover for this part.

* **Project Write-up:**Write a short paper describing the results of your project that includes the following sections:
  + **Algorithm descriptions** - Description of the GA so far. Be careful to include all of the details someone would need to replicate your work.
  + **Results** - Basically, does it seem to be working.
  + **Conclusions** - If its not working, why not. And what are then next steps to complete the project.

1. **Function Overview**

* Sphere Function: The function is given below :-

|  |  |
| --- | --- |
| SN | Function Properties |
| 1 | $ f_{Sph}({\mathbf x}) = \sum_{i=1}^p x_i^2$ |
| 2 | $ x_i \in [-5.12, 5.12]$ |
| 3 | $ {\mathbf x}^*=(0,0,\ldots,0);\ f_{Sph}({\mathbf x}^*)=0$ |

1. **Algorithm Description**

Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics.GA is to use the power of evolution to solve optimization problems [1]. Typically, a genetic algorithm consists of following Genetic Operators:-

* Representations
* Mutations
* Crossovers
* Selection mechanisms[2]

This is an implementation of Genetic Algorithm so as to solve against one of the optimization functions known as **Sphere Function**.

For solving this problem, I have setup the Individual and the Population Class. I have also setup different classes for each of the functions that we will need to deal in for the later project. I have also created a Main Class which will be used to create and test the population against a particular function. Further details of the work are outlined below:-

**Individual Class**:-

The Individual class consists of an array of double values for the floating point representation, a double value for the calculated fitness Value and an instance of the RandomArrayGenerator Class that is used to create random arrays in between the range of the Optimization function. The methods associated with this class are typically to generate the Individual, calculate the fitness value of the individual and perform mutation on the Individual.

**Population Class:-**

The population class consists of a Set of Individuals, a subset population selected by selection mechanism to apply mutation and crossover, a value to store average fitness of the population, a string to get the function name as to which various calculations are branched out, the population size and offspring size and the Boolean flag to find if solution has been found and an Individual solution to the given problem. The methods in this class are:-

* generatePopulation(functionName): to generate the Population as per the functionName and range.
* selectedIndividuals(): returns the individuals who are selected for mutation and crossover.
* checkForSolution(): check if possible solution has been found as per the threshold.
* getTournamentWinner(): to get the Individual who has won the tournament.
* getRandomIndividual(population): get a random individual from the population.
* mutateAllOffSprings(individualsToMutate, functionName): mutate the Individuals as per the function given.
* appendToPopulation(populationSubSet): add the offsprings to the original population
* removeExtraPopulation(): remove the individuals with low fitness as to maintain the population size.
* populationReset(): remove all individuals from the population.
* evolve(functionName): evolve the population as per the given function name.

The evolve method evolves the population as to one generation. The evolve method is invoked from the main class up until a good solution has been found. The following code shows the implementation for evolution of the population.

**public** **void** evolve(String functionName){

populationSubSet = selectedIndividuals();

populationSubSet = mutateAllOffSprings(populationSubSet, functionName);

appendToPopulation(populationSubSet);

removeExtraPopulation();

printAll();

checkForSolution();

}

**FunctionTest Class:-**

The FunctionTest class serves as the base class for testing our population against the sphere function (or any other function). The main method invokes the testForSphere method to run the test against Sphere function. The implementation details are given below:-

**private** **void** testForSphere() {

System.*out*.println("Test for Sphere Function");

Population population = **new** Population();

population.setFunctionName(Spherical.*FUNCTION\_NAME*);

population.setOffSpringSize(6);

population.setPopulationSize(100);

population.setPopulation(population

.generatePopulation(Spherical.*FUNCTION\_NAME*));

**int** i = 0;

**for** (i = 0; !population.getSolutionFound(); i++) {

population.evolve(Spherical.*FUNCTION\_NAME*);

}

System.*out*.println("Iteration for Generation : " + i);

System.*out*.println(population.getSolution().toString());

population.populationReset();

}

**Spherical Class:-**

The Spherical Class encompasses the major details enclosing the Sphere function. I have introduced the constant FUNTION\_NAME for getting the name of the class associated with the function. The range variable describes the range the function is in between. The methods and their usage description are provided below:-

* public static Double getfitnessValueNP(Double[] varArray):- return the fitness value as per the array passed.
* public static Individual mutateIndividual(Individual individualtoMutate):-returns the mutated Individual. The mutation does a creep mutation depending on the where the fitness value increases.
* public static boolean checkLimitCondition(double val):- checks if the value generated is between the range of the Spherical function.

1. **Results**

The results generated are given below:-

|  |  |
| --- | --- |
| Function Name | Spherical |
| Population Size | 100 |
| Off Spring Size | 6 |
| Algorithm Type | Steady State |
| Number of Iterations | 890 |
| Solution List | [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.07, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] |
| Final Fitness Value | 0.0 |

1. **Conclusion**

In conclusion, Genetic Algorithm for Sphere Function was implemented and the algorithm seemed to converge to the solution very fast. No crossover mechanisms were implemented however otherwise we might have got the solution sooner. Hence, we can see that Genetic algorithm did very well in converging to the solution point.