Problem definition:

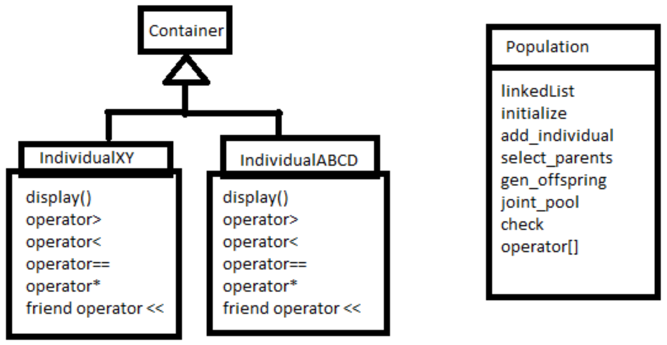
There are two problems to be solved in this final project: finding the solution with best fitness for the Ackley's function and finding the best third order curve that fits a series of points. For the first problem, the solution to the Ackley's function is known beforehand as x = 0 and y = 0 for the lowest fitness at 0 which is coupled with a long equation describing the fitness in which randomly-generated x and y can be used as input. For the second problem, there is a pre-defined cubic equation of the form:

$$a * x^3 + b * x^2 + c * x + d = y$$

For which we know the constants a, b, c and d. The purpose of the program is to generate randomly a, b, c and d and find the values for them corresponding to our known theoretical values of a, b, c and d. In order to do, the randomly generated constants are used to generate the coordinate y with pre-designated values of x and compared to theoretical values of x and y by use of the mean squared error method. For both these problems, the method for solving involves the use of population generation, selection of the fittest, generation of offspring through crossover and mutation, and checking for if the desired fitness level has been achieved.

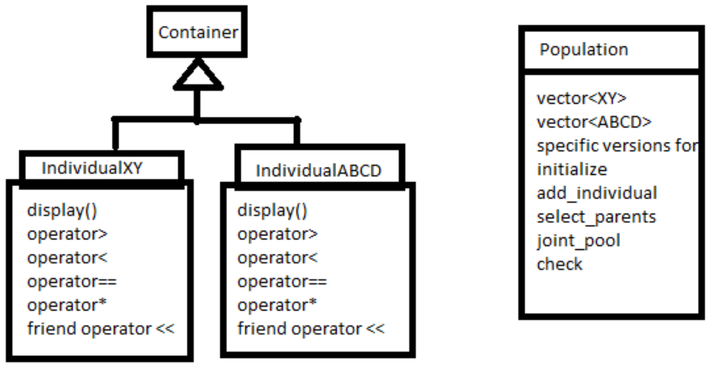
Alternatives and Recommendation

1. The first alternative would be to use linked lists instead of vectors in the program for the population class.



In general, the container takes care of the mutation and crossover using the * operator overloading while the population takes care of the tournament selection and selecting the fittest individuals by using the comparison operators to compare the fitness of different individuals of the same type. The population class tells the individual classes when to mutate and crossover, but the actual parameters for these modifications to the individual's data members reside within the operator * overloaded method.

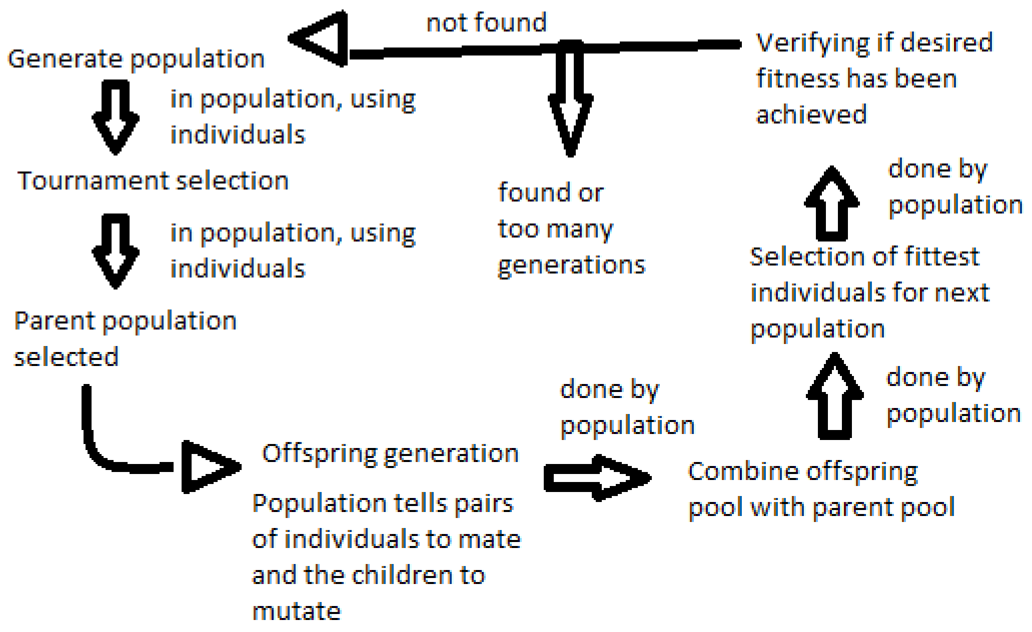
2. The second alternative would be to use class composition instead of templates which would permit to give more program more specific behaviors depending on the type of individuals present inside the population.



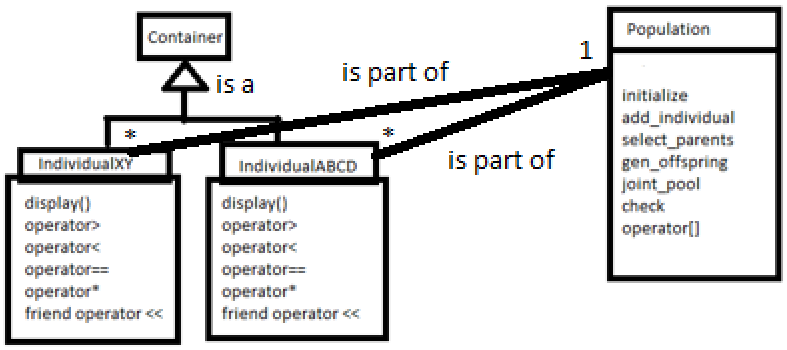
Using this alternative, the user would be able to generate points for the curve-fitting problem by passing the parameters to the population which will generate the individuals and their fitness according to the points given. As of right now due to the current implementation, the user cannot pass parameters as it would not make sense to the templated version of population. The remainder of the program will behave the same as the before but there will also be the option for the user to choose the range of the genome and other factors influencing the data members of the individuals.

Alternative 1 (linked list)		Alternative 2 (individual-specific population)	
Pros	Cons	Pros	Cons
Faster erasing	Slower access	Increases versatility	More work
Faster inserting	Needs more memory	More user-friendly	Need to check for more errors
	Need Heap space		

Design Description


not found
Generate population
Verifying if desired fitness has been achieved
in population, using individuals
Tournament selection
found or too many generations
done by population
in population, using individuals
Selection of fittest individuals for next population
Parent population selected
done by population
done by population
Offspring generation
Combine offspring pool with parent pool
Population tells pairs of individuals to mate and the children to mutate

Class Diagram with UML formatting



Inside the container, the important functions are all overloaded operators.

Operators > < == are all overloaded in order to compare individuals with each other by comparing their respective fitness which is very important to select the fittest individual.

The <= is overloaded in order to verify if the desired fitness has been reached by comparing the individual to a float representing the desired fitness, where the parameter from the individual is obviously their fitness.

The friend operator << is used to output relevant data regarding the individual, more importantly their fitness, and acts the same as the display() function.

For the population, there are several functions to process the genetic algorithm.

Initialize generates the very first population of 1000 individuals and call on the default constructor of individuals that will generate their data points and their respective fitness. Select_parents generates a parent population through the use of tournament selection between 3 random individuals of which the fittest is selected. The 3 individuals are then thrown back into the general population where they might be selected again. The parent population is 100.

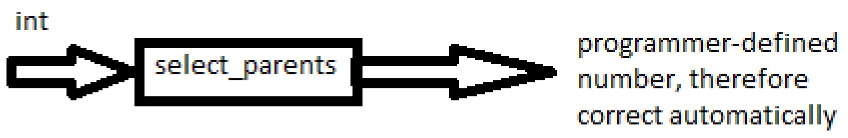
Gen_offspring generates children from the parent population. Two parents are randomly selected from the pool and will generate two most likely different children by crossover and each child will then undergo mutation in range around its crossover values. The two parents are then thrown back into the pool where they might be selected again. In total, there will be 10 more children than the original population in order to ensure a diversity of choice for the next steps.

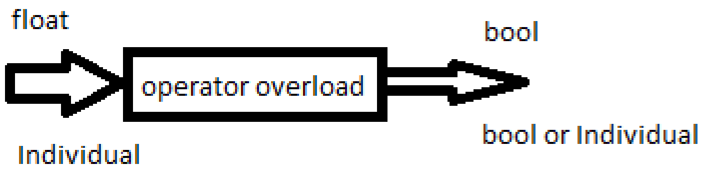
Joint_pool mixes the pool of parents and children to generate a population of 10100 that will be sorted in order to select to select the fittest 1000 individuals.

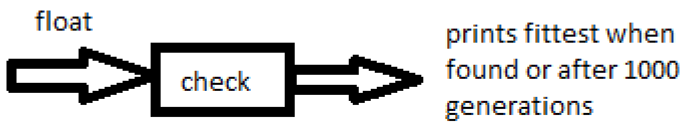
Check will verify if the new pool of 1000 individuals contains an individual that meets the desired fitness level, and if not, this pool will be the general population for the next generation. If there is an individual with the appropriate fitness level, the fittest individual of the generation will be shown on the IO stream with its generation number and its parameters.

Testing Results

None of the methods that are available to the user will generate a run-time error as they have to take parameters that are either objects, or where the value does not matter as much.







For IndividualXY, the Ackley’s function is tested by verifying the range of the genome from [-100,100] to smaller regions [-50,50] and bigger regions [-500,500], and the difference is mainly in the generation time it takes to find the desired fitness if it is not 0.
For IndividualABCD, the curve-fitting depends on the equation used and the range of the genome. The curve was tested with curves and different ranges for x, and they all work properly.